ABSTRACT

Title of Dissertation: KEY MANAGEMENT: TOWARDS THE DESIGN OF EFFICIENT, LIGHTWEIGHT SCHEMES FOR SECURE GROUP COMMUNICATIONS IN LARGE MOBILE AD HOC NETWORKS.

Maria Striki, Doctor of Philosophy, 2006

Directed By: Professor John S. Baras,
Department of Electrical and Computer Engineering

Securing group communications in resource constrained, infrastructure-less environments such as Mobile Ad Hoc Networks (MANETs) is a very challenging research direction in the area of wireless networking and security. This is true as MANETs are emerging as the desired environment for an increasing number of civilian, commercial and military applications, addressing an increasing number of users. Most of these applications are sensitive and require specific security guarantees. The inherent limitations of MANETs impose major difficulties in establishing a suitable secure group communications framework. Key Management (KM) is the operation that enables and supports the secure exchange of data and ensures the capability of members' secure cooperation as a group. KM protocols provide a common symmetric group key to all group members, and ensure that only legitimate members have access to a valid group key at any instance.

Our work focuses on the design of efficient, robust, novel or improved group KM schemes, capable of distributed operation where key infrastructure components are

absent or inaccessible, that accomplish the following: *(a)* better performance than this of existing schemes for similar environments, *(b)* successfully handle network dynamics and failures, in networks with large number of nodes. Our protocols address Flat or Hierarchical MANETs separately. Our solution for Flat MANETs involves the design of two new Octopus protocols, and the adaptation of the original to MANETs. We introduced algorithms for handling membership changes, disruptions and failures with low overhead for initial key establishment or steady state for all three protocols. In an effort to reduce their suboptimal performance when executed without topological considerations, underlying routing is integrated into the design by the definition of topology oriented communication schedules via lightweight heuristics. In addition, we integrated a lightweight leader election algorithm with the proposed protocols, used Elliptic Curve Cryptography to reduce certain costs further, and investigated the effects of a number of mobility models in the performance of our schemes. For Hierarchical MANETs, we designed a 2-level Hybrid scheme that supports various combinations of protocols to improve metrics of interest. This scheme exploits battlefield diversity, and links key distribution to network topology and nodes' mobility.

KEY MANAGEMENT: TOWARDS THE DESIGN OF EFFICIENT, LIGHTWEIGHT SCHEMES FOR SECURE GROUP COMMUNICATIONS IN LARGE MOBILE AD HOC NETWORKS

By

Maria Striki

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Advisory Committee:
Professor John S. Baras, Chair
Professor Virgil Gligor
Associate Professor Gang Qu
Assistant Professor Nuno Martins
Professor Nikolaos Roussopoulos

Dedications

To my parents Nikia and Lazaros,
to my sister Nana,
to my grandparents Lela and Kostas,
for an unbreakable bond of love and support everlasting
to Kyriakos,
for lighting my path during this journey…

Acknowledgements

I would like to express my gratitude to my advisor Dr. John S. Baras for giving me the opportunity to work with him. Without his guidance and support, this dissertation would not have been made possible. His broad technical skills and professional experience contributed a lot to my first significant research experience. Because of him, this journey to knowledge and expertise acquisition turned out interesting, colorful, and fruitful, in both the professional and personal level. I was able to identify and face some of my fears, and overcome them. I was brought up against situations that made me stronger and wiser. For all that, I am grateful.

I would like to thank Dr. Virgil Gligor, Dr. Gang Qu, Dr. Nikolaos Roussopoulos, and Dr. Nuno Martins, for reviewing and constructively commending on my dissertation, and for kindly consenting on serving in my defense committee. I take this opportunity to individually thank Dr. Qu, who is not only an excellent professor, but also a very humble and considerate person. I had the opportunity to verify this in many occasions, as he was the instructor in a number of classes I have taken in this department.

I am thankful to Althia, Diane, Lisa, and Kim, for all their support, understanding and valuable help with so many pending issues.

I am also thankful to a number of colleagues and mentors from Telcordia Technologies, who I met and collaborated with during my two internships there. In particular, I would like to thank Dr. Ken Young, who made these two internships possible in the first place, made sure that I was provided with a very supportive work environment at all times, and has been actively supporting me towards my career goals ever since we met. I would also like to thank Dr. Anthony McAuley and Dr. Simon Tsang, for their encouragement, their faith in my abilities, and for supporting me towards my career goals in many occasions.

I would like to acknowledge that this work was supported through the collaborative participation in the Communications & Networks Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD-10-01-2-0011.

During my studies in Maryland, I have been very fortunate to have made many friends. I would like to thank them all for standing by my side in good and bad times, but most of all for making my life richer with all the wonderful memories and laughs we share. Among all, it would be impossible not to mention Damianos Karakos, not only because he is the one that has helped me most in settling in this place and making me feel at home, but because he has been like a guardian angel, always on my side, even in the most difficult times. How can I forget our long discussions and the laughs we shared at Starbucks, or the eventful social gatherings? I would like to thank my roommates of the first years, Apostolia Schiza and Maria Tzortziou, for helping me settle in this place, and providing a relaxing, harmonious living environment, almost like home. I owe a debt of gratitude to Apostolia, for standing on my side in a period of turmoil, and for sharing the pain of regularly being up and studying during small hours. I would also like to thank Courtney Kennedy, who was my roommate for the last year and has been a great friend since then. Furthermore, it would be impossible not to express my gratitude to my dear friends from the Graduate Hellenic Student Association (DIGENIS), who provided me with support, advice, encouragement, and so many colorful moments and memories, that in retrospect are always funny and valuable. I am thankful to my dear friend Senni Perumal for standing by my side in good and bad times, for our long discussions over coffee. Although there are so many that it would have been impossible to mention each one individually, I would still like to distinguish: Kostas Bitsakos, Antonis Delligiannakis, Dimitris Tsoumakos, Nikos Fragiadakis, Gelareh Taban, and many more.

Moreover, I am very happy to acknowledge my gratitude to my best friend from high-school, and one of the strongest and kindest individuals I know, Helena Tsilidis. I do not know what to thank her most for: her continuous support and faith in me, encouraging me to go on despite adversities, her integrity, wisdom, and always accurate intuition, giving me invaluable advice and courage even at 3.00am, or for just being like a sister to me.

Also, I have been blessed to have met three people that have always been a huge source of inspiration to me: Mr. Takis and Mrs. Dimitra Anagnostopoulos, and Mr. Babis Mastroyiannis. I would like to thank them for their faith in my abilities that gave me the strength to overcome all obstacles. Mr. Takis, is not only an outstanding chemist, but an extremely gifted teacher, who showed me the magic, beauty, and harmony of chemistry and physics. Mr. Takis, Mrs. Dimitra, and Mr. Babis, always young at heart and spirit, are a continuous source of optimism to me, as they are constantly getting involved in novel, creative projects, and are not afraid to face challenges and master them.

I would like to thank Kyriakos Manousakis, one of the kindest and most positive people in my life, for his continuous support, his extreme patience, and love. He stood by my side for all the good and bad times, encouraging me to be brave and pursue without fear all challenging tasks coming in my way. His own personal example has been and will be a huge motivating force for me. I would also like to deeply thank Nina and Kostas Manousakis, for their sincere support and encouragement all this time.

Moreover, I am so happy to acknowledge my gratitude to Sia Grigoropoulou and Kostas Eustathiou that brought so much happiness, love, and harmony to my family ever since they became part of it. I thank them for the love and support they have shown in so many ways from the bottom of their heart. My special thanks to Sia, one of the most considerate and resourceful people I know, as I know that so many times my problems became her problems.

Words cannot express my most sincere gratitude to my wonderful family: my parents Nikia and Lazaros, my sister Nana, my grandparents Lela and Kostas, for their endless love, for their infinite faith in me and for their unlimited support in everything I have set out to pursue, every dream I have ever had, possible or impossible. It is this eternal, inexhaustible source of love and compassion from which I draw strength and power every moment. It is these people that taught me the power of love, the power of forgiveness, the liberation of moving on, and the beauty of the unprecedented in life, with their own examples. To me, these are the most important values in life. Sometimes it is easy to see right away, other times it takes a very long and exhausting journey to figure these out. This dissertation and all this rewarding journey that ends with this dissertation is dedicated to them.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Introduction

Securing group communications in resource constrained, infrastructure-less environments such as Mobile Ad Hoc Networks (MANETs) has become one of the most challenging research directions in the areas of wireless network security. This is especially true as MANETs are emerging as the desired environment for an increasing number of civilian, commercial and military applications, such as audio and video conferencing, visual broadcasts, military command and control, emergency/disaster situations, interactive gaming, etc., addressing also an increasing number of users. Some of these group applications are quite sensitive regarding the nature (and purpose) of the data exchanged among participants, and thus require specific security guarantees. Hence, security is becoming an indispensable requirement of our modern life for all these applications. The inherent limitations of such dynamic and resource-constraint networks impose major difficulties in establishing a suitable secure group communications framework. Securing such applications may be multipurpose. In this work, we focus on providing security from the perspective of enabling and protecting the exchange of data among peers, so that the appropriate data reaches the intended recipients and only these recipients are able to “read” it. The security challenge for multicast is providing an effective method for controlling access to the group and its information that is as efficient as the underlying multicast. A simple and efficient way to protect the exchanged data is to encrypt and decrypt it with a common secret group key, shared by legitimate group members only. A secret key shared by all group members is suitable for multicast communications because it achieves efficient bandwidth usage (same data sent to all receivers), lower computation cost (data is encrypted only once instead of multiple times), and better utilization of device and network resources as a consequence.

The Key Management (KM) operation enables and supports the secure exchange of data and ensures the capability of members' secure cooperation as a group. More formally, the KM is the set of functions, protocols, and procedures for the management of cryptographic keys, security associations, policies, keying material, and other parameters pertaining to entities communicating within a group. The role of KM is to: (*a*) enforce access control on the group key and on the group communication, and (*b*) support the establishment and maintenance of key relationships between valid parties according to a security policy being enforced on the group. The main goal of KM protocols is to provide a common symmetric key or group key to all members of a group, and ensure that only legitimate members of a secure group have access to a valid group key at any time instance during the life of a session.

The KM encompasses three equally important services: (*a*) **Key Generation (KG)**: it defines the entities and messages involved in the group key construction (i.e. logical algorithm), (*b*) **Key Distribution (KD)**: it handles the distribution of private pieces of information or the group key to group members; **Key Agreement (KA)** is a subset of the broad KD function: it imposes that all group members contribute equally to the group key, interacting among themselves, following the given KG algorithm. Finally, (*c*) **Entity Authentication (EA)**: it validates that it is indeed the owning entity that is using or deploying the owned identity in an interaction. The **Group Authentication** provides a weak form of message integrity in that the receivers can verify whether data has been modified by nonmembers in transit.

The KM schemes are built on top of the pre-existing multicast infrastructure, and it is obvious that the performance and behavior of the KM functions imposes an upper limit on the efficiency and scalability of the whole multicast communication system. Each function has its own merit towards the total cost and complexity of the multicast infrastructure. A situation such that the extra services that promote the security of group communications become also their bottleneck is highly undesirable. It is obvious that the design of suitable algorithms for each of the three KM services is of paramount importance, and even more so for resource-constrained, dynamic, and infrastructure-less environments, such as MANETs. All three functions are equally important for the desired operation of secure, integrated multicast communications system, and each one constitutes a broad and challenging research direction. In this thesis, we only focus on the study of efficient KG and KD or KA for the environment of interest. We assume that group members are already authenticated via some underlying mechanisms that appear as a black box to the user. Indeed, EA is a huge research problem that is usually studied or can be studied in isolation from the other two. On the other hand, KG and KD or KA, are studied together, as the KG logical algorithm usually determines how KD or KA is going to be executed. For simplicity, it is often the case that the design of KM protocols is mainly associated with these two functions. In this thesis we refer to KM protocols, implying the combined consideration of KG and KD and/or KA only.

The logical design of efficient KM protocols has been the main focus of the related research to-date. Such a consideration however, gives only a partial account on the feasibility and actual performance of a protocol in a real multi-hop network. This is because protocols have been evaluated only in terms of the logical KG algorithm in isolation from the underlying network functions that interact with the logical scheme and support its correct execution (i.e. routing). Logical networks are just abstractions of the physical networks, so they are free from various feasibility issues and extra sources of overhead associated with physical networks. Physical networks naturally bring about mobility, connectivity, routing, power, and other dynamic issues, which directly or indirectly affect the performance and behavior of KM protocols. Along these lines, most of the popular KD or KA schemes that exist currently in the bibliography have been developed for wire-line networks (e.g. internet). Wire-line networks in the general case provide stable infrastructure and infinite power to devices – assets that make the development of any infrastructure easier. In contrast to wire-line networks, deploying security mechanisms in MANETs is difficult due to inherent characteristics of these networks. These characteristics are the major challenge for the design of suitable KM schemes and have even more severe impact on the operation of KA. Indeed, the same protocols that have been designed for wire-line or logical networks in general, might loose even their basic efficiency characteristics when applied to MANETs, or totally fail even under a limited number of parties. Another issue to consider is that mobility plays an important role in KM, as the mobility patterns of nodes cause many of the changes in group configurations, and triggers the operation of changing the group keys (re-keying). It is worth studying how nodes' mobility models, speed, direction, transmission range, and other mobility related factors, can affect the performance of a KM scheme.

We are dealing with dynamic, infrastructure-less networks of limited bandwidth, unreliable channels where topology is changing fast. Network nodes have limited capacity, computational or transmission power. Connections are temporary (mobility changes, battery drainage, poor physical protection) and unreliable. These constraints turn most of the existing protocols inefficient in MANETs. Wire-line networks are capable of supporting an expensive KG protocol in terms of communication and computation overhead without affecting the power of nodes. In MANETs, the transmission and computational power are valuable resources for most wireless mobile nodes due to their power constraints or capacity limitations (e.g. laptops PDAs, cell-phones). Hence, it is crucial that the KM protocols addressing MANETs are as lightweight as possible. Similarly, efficient centralized schemes based on a single group leader are not feasible on large wire-less networks, as the energy resources of this leader are depleted all too fast.

Along with the continuous quest for the design of more efficient schemes than the existing ones, the need for the new KD or KA schemes to handle successfully and tolerate network dynamics and failures with low impact in a network with large number of nodes is now equally important. For example, upon failures and disruptions, it is often the case that a KA protocol must restart the group key establishment process. Whenever this event occurs (may be too often), the underlying routing is invoked once again, a significant amount of relays become involved in the exchange of large KM messages, and considerable delay burdens the network. The overall performance of the protocols degrades even more because of the indirect impact of excessive routing on additional network layers (i.e. QoS deteriorates due to more collisions at the MAC layer, the bandwidth usage and the consumption of network resources increase undesirably). There exist a number of KM proposals that address MANETs as well. However, most of them make simplistic assumptions about the capabilities of network nodes, the nature of their connectivity, etc. Some present security vulnerabilities, others do not scale over a restricted network, etc. To our knowledge, there is no single work that addresses effectively all the issues KM protocols come against when implemented on a flat multi-hop, dynamic, wire-less, infrastructure-less, and resource constrained network.

From the above, two requirements emerge: (1) the design of efficient, scalable and robust KM protocols that address a general multi-hop MANET, (2) a fair and realistic performance evaluation of KM protocols considering the underlying physical network, and the most important associated network functions (i.e. routing). Considering these, **our objectives** in this dissertation are the following: to design novel or extend existing KM schemes to achieve:

- **Efficiency:** they have better performance than existing (in wire-line or wire-less networks) w.r.t. the most significant metrics of interest (i.e. bandwidth, computation, latency, storage).

- **Scalability and Robustness**: they handle successfully or tolerate the network dynamics (**mobility**, **membership changes**) and **failures** in a large network of O(100) members/nodes.

- **Security:** the exchange of data is enabled and protected.

1.2. Contributions

In this dissertation, we contribute towards the design of new KM protocols or the extension of existing ones so that the above attributes of efficiency, scalability, and robustness are addressed and improved.

In Chapter 2 we conducted a comprehensive investigation of all major categories of KM schemes (i.e. contributory, non-contributory, hybrid, or centralized, distributed, hierarchical). We studied the behavior and performance of protocols from each category in the resource constrained, dynamic and infrastructure-less MANETs. Our objectives were to determine which of these protocols make the best candidates for such networks, and extract a number of desirable features and performance attributes that should be taken into account when designing feasible KG algorithms. This was our motivation as far as the conceptual design of schemes based on 2^d-Octopus, introduced by Becker et al. Our basic contribution at this point includes the adaptation of the original Octopus (O) to MANETs, and the modification of (O) for the design of two novel schemes (GDH.2-based modified Octopus MO and TGDH-based modified Octopus with Tree MOT) for MANETs. Our evaluation demonstrated the superiority of MOT: it outperforms (O), MO and other KA protocols, simultaneously maintaining features that make it a strong candidate for a dynamic environment with many limitations. At this point, the analysis and comparison of the schemes is done with respect to logical networks, only to measure the efficiency of their KG algorithms, as a first performance attribute. In our quest to improve the extended 2^d-Octopus schemes, we identified and provided solutions for a number of issues that have their own, significant merit of interest, related to a number of KA schemes. As a fist step, we found that the evaluation of some protocols was incomplete particularly regarding their performance at steady state (re-keyings due to membership changes). We completed such protocols by contributing either to the algorithmic level to define their behavior at steady state, or to the analytical level, or both.

However, such an evaluation of protocols, based on the “abstract logical design” gives only partial account to their actual performance and behavior. In Chapter 3, our natural next step was the consideration of the underlying routing in the design and evaluation of KA or KD schemes. We re-evaluated the most significant KA schemes that emerged from our previous study (i.e. GDH.1, GDH.2, ING, BD, TGDH, and Hypercube) subject to the underlying routing without any topology considerations. The resulting degradation in the performance of all protocols triggered our next step. We introduced a number of new algorithms and techniques (lightweight heuristics for the manipulation of structures based on spanning trees), to efficiently merge the logical design of KA protocols with the underlying routing and topology of the network as well. The protocols were executed w.r.t. a newly defined topologically aware communications schedule, and this new feature alone resulted in the substantial improvement of one or more metrics of interest. Our basic objective was to improve the scheme used within our Octopus-based extensions, namely Hypercube, GDH.2, and TGDH, but also determine from the results of a comparative performance evaluation if there are more potential candidates to be incorporated in the Octopus schemes.

In Chapter 4, we focused on issues of robustness and fault-tolerance for all schemes studied, and in particular for the new topologically aware versions of the protocols that are even more sensitive to the dynamic changes. Dynamic changes (i.e. mobility, resources depletion, etc.) may cause network partition or merging, member or link failures. These changes are reflected on KM protocols as membership updates in the best case, and are handled by re-keyings if the protocols provide such algorithms. If such changes occur during the key establishment phase, it turns out that in most cases the protocol must start over. This is even more so with KA protocols, where members interact among themselves to produce the group key: dynamic changes have serious impact on the KG algorithm and on the connectivity of members and may cause the protocols to stall. So, in this section we investigate if we can do better: (*a*) for the Octopus schemes, in essence for the KA protocols they include, and (*b*) for all the KA schemes studied, individually. We methodically studied the impact of a number of realistic scenarios of disruptions on the discussed schemes in a MANET, during key establishment or at steady state. For every protocol and each case, we introduced algorithms and techniques to go around the failures caused by network dynamics or remedy those with very low extra overhead. So, we have “enriched” KA protocols to be more robust to dynamic changes. In certain cases, we were able to extend certain protocols and modify their KG algorithm or communication regulation (Hypercube, TGDH), to make them stealthier to failures and dynamic changes. Hence, our contribution towards efficient, scalable, and robust KM protocols for MANETs is built step by step through Chapters 2, 3, 4.

In Chapter 5, we shifted the focus of our study from totally flat to hierarchical MANETs with a specific configuration that resembles typical scenarios encountered in civilian cases or most often in the battlefield. We designed a two-level hybrid KM scheme (2HH) that models these environmental variations and exploits the diversity of the battlefield. It uses a combination of protocols in the two levels to achieve improvement in one or more of the metrics of interest. We analytically explored how combinations of a wide range of protocols affect the metrics of interest for 2HH, starting from two diverse centralized protocols: GKMP and LKH, and expanding to many others. After a thorough analysis, we were able to provide theory and design a tool that automatically determines the most beneficial protocol combinations for 2HH, w.r.t. the metrics we want to improve, given the specific values or ranges of the network parameters. In addition, we looked into the effects of mobility at steady state of potentially any KM protocol through 2HH. For that, we simulated the impact of three mobility models (individual or group) on the connectivity among members on an arbitrary hierarchical protocol. We measured the frequency of member evictions from their original subgroup, given the mobility model and parameters. Then, conducting the required analysis, we worked to combine our results with the probability of evictions in 2HH, and provide this new metric as input to the 2HH theoretical and simulation tool. Our aim was to be able to determine the exact variations in the re-keying (mainly) overhead (in bits or packets) resulting from varying basic mobility parameters.

Last but not least, still exploring ways to improve the efficiency of our schemes, in Chapter 6 we investigate the efficiency of the implementation of KM protocols, and in particular Octopus-based schemes, under Elliptic Curve Cryptography (ECC). We substituted EC in protocols that were originally based on RSA or DH public key systems. Our aim was to gain insight on how all the metrics of interest are affected, and examine if further improvement in certain metrics can be achieved by this substitution. In particular, we substituted all DH-based with ECDH operations in the three Octopus-based schemes, and the RSA operations with EC-El Gamal in the centralized OFT. We conducted a comparative analytical evaluation of the EC-equivalents of the schemes *vs.* the original. As a result, we were able to determine which protocols and for which KM operations and metrics "gained" from this substitution. Indeed, the performance of all Octopus schemes was significantly improved in all aspects.

Prototyping: We implemented (O) and MOT on a test-bed. Our implementation includes both the basic functionality of these protocols for the establishment of a multicast group key (including the group formation) and the response of the protocols in dynamic membership changes in the multicast group. The presentation of the protocols has been supported from a GUI and a user level multicast application which encrypts, transmits, receives and decrypts multicast data (i.e. pictures). The application encrypts and decrypts data by utilizing the established multicast group keys from the application of each one of the implemented protocols. The subgroups are dynamically generated, and the leaders are sensing the subgroup changes and re-keying is triggered. We demonstrated: (*a*) how participants of the secure group establish a common session key, (*b*) the dynamic addition and deletion of a member to the secure group, and (*c*) nodes that no longer belong to the secure group cannot receive the transmitted encrypted data. The protocols were implemented on Linux (kernel-2.4.18).

Chapter 2: Efficient Scalable Key Agreement Protocols for Secure Multicast Communications in MANETs

2.1. Introduction

In this chapter we focus on the development and performance evaluation of efficient, robust and scalable Key Management (KM) protocols for multi-hop wire-less ad hoc networks. As a first step, we perform a comprehensive study of the characteristic and performance of a number of representative Key Generation (KG) protocols that exist in the literature. Towards this end, we classify KM protocols in major categories (i.e. contributory, non-contributory, hybrid, or centralized, distributed, hierarchical, etc.) Our objectives are: (*a*) to determine which of these protocols make the best candidates to apply in dynamic, infrastructure-less, resource-constraint networks, and (*b*) extract a number of desirable features and performance attributes that should be taken into account when designing KG algorithms feasible for the same environments. Towards this end, we identified, studied and provided solutions for a number of research issues that have their own, significant merit of interest. More specifically, we performed an analytical evaluation of some among the most popular contributory or Key Agreement (KA) protocols and a few centralized ones for the KM operations of initial key establishment (initialization) and re-keyings (steady state), with respect to the following metrics of interest: communication, computational, storage cost, number of rounds. We found that the evaluation of some protocols was incomplete particularly in terms of their performance at steady state (re-keyings due to membership changes). We completed the evaluation of such protocols by contributing either to the algorithmic level (KG) to define their behavior at steady state, or to their performance analysis level, or both, whenever required. As an auxiliary step, we analytically estimated the bit-complexities of a number of cryptographic functions, essential for our analysis and integrated the results in the analytical cost functions of the discussed protocols.

As a next step, we present our main contribution towards secure, lightweight, scalable and efficient group communications for MANETs, which is based on 2^d–Octopus hybrid KA scheme, introduced by Becker et al. in [1]. Our contribution includes the adaptation of the *original Octopus* (O) to MANETs, and the modification of the *original Octopus* (O) for the design of two novel schemes (GDH.2-*based modified Octopus* MO and TGDH-*based modified Octopus with Tree* MOT) for MANETs as well.

The introduction of (O) [1] is limited to an overview of the protocol operation for the initial group key establishment only. At this point, the analysis is conducted by considering only the efficiency of the KG function, isolated from the underlying network functions that are directly or indirectly involved. In other words, the analysis and comparison of the schemes is done only to measure the efficiency of the KG algorithm, as a first performance attribute. A limited analysis was presented only for the initial state. No algorithms that handle membership updates or failures at steady state are addressed in [1]. In this chapter, we complete the original work, by introducing efficient join and leave procedures, and we introduce the two extended schemes: MO and MOT. Furthermore, we provide analytical formulae for all KM operations with respect to all metrics of interest and we conduct a comparative performance evaluation of all Octopus schemes, including also a representative and very efficient centralized scheme - OFT. Although OFT has been designed for a wire-line network and is infeasible for the environment of interest, it is included in this comparison in order to gain insight on the extra overhead required to render KG protocols robust and scalable in MANETs. Through the overview, the analysis and the evaluation, the feasibility of Octopus schemes in MANETs will be shown. Moreover, the superiority of MOT will be demonstrated: it outperforms (O), MO and other KA or even KD protocols. For example, MOT reduces the incurred communication overhead of (O) in half, "approaching" communication-wise the very efficient OFT, simultaneously maintaining features that make it a strong candidate for a dynamic environment with many limitations. Our main objectives in this chapter are that: (a) the extra communication, computation, storage costs and latency incurred to the network for the key establishment (initial deployment) and maintenance (re-keying) of our security schemes is as low as possible, and (*b*) the scheme maintains an acceptable performance under a highly dynamic topology and an increasing number of users.

The Octopus protocols acquire some very alluring features which have motivated our interest to explore them: they introduce hierarchy through the partition of a large group to 2^d subgroups of smaller size, each having their own subgroup leader (as opposed to a single group leader handling a large group). This way, they can tolerate network dynamics and be more scalable in the first place. Disruptions can be handled locally at first within the subgroup, and then reflected to the whole group via the execution of Hypercube among subgroup leaders. Hypercube is an efficient KA protocol with the property that it minimizes the amount of rounds (latency) required for the group key establishment. By further exploring the Octopus and Hypercube schemes, we found that this combination can lend itself to low cost re-keying algorithms, by modifying the execution of Hypercube appropriately. This has been the main reason for maintaining the Hypercube core intact in all Octopus versions we introduce. In addition, this scheme allows great flexibility as within each subgroup, a variety of novel or existing KA or KD can be potentially applied, depending on the performance metrics we wish to optimize, on the network specifications defined locally (i.e. density, etc). Later in this chapter, we are going to justify the selection of the two particular protocols: GDH.2 and TGDH for our extended Octopus-schemes. For example, GDH.2 may be substituted with GDH.1 as well, without significant change in the flow of the algorithm. GDH.1 may be preferable if the network does not support broadcasting. However, we would still prefer to use GDH.2, mainly because only GDH.2 (and GDH.3) lends itself to more efficient construction algorithms that handle membership changes (addition/deletion).

In the course of this and the following two chapters, we will gradually: (*a*) show the feasibility of Octopus-based protocols and point out its more beneficial assets over other schemes, and (*b*) address its weaknesses and introduce new schemes and algorithms to eliminate them or go around them with low impact and cost.

2.2 Previous Work

Securing multicast communications in MANETs has become one of the most challenging research directions in the areas of wireless networking and security. Some group applications are quite sensitive regarding the nature (and purpose) of the data exchanged among parties and require specific security guarantees. Securing such application may be multipurpose. The most significant requirement is this of enabling and protecting the exchange of data among peers. The security challenge for multicast is providing an effective method for controlling access to the group and its information that is as efficient as the underlying multicast. A simple and efficient way to protect the exchanged data is to encrypt and decrypt it with a common group key, shared by legitimate group members only. The purpose of KM is to enable and support the secure exchange of data in insecure environments, and ensure the capability of members' secure cooperation as a group. The KM operation encompasses the three services reported in the introduction: KG, KD or KA, and EA.

For the scope of our work, we focus on ensuring that the proper data reaches the intended recipients and only these recipients are able to "read" it. The design of efficient, scalable, and robust algorithms and mechanisms for each of the above services, for resource-constrained, infrastructure-less environments, such as MANETs, is of paramount importance, since the performance of the corresponding KM functions imposes an upper limit on the efficiency and scalability of the whole secure group communication system. In this work we address only the first two functions of KM, namely KG and KD or KA, as previously discussed. In the rest of this work, for simplicity, when we refer to KM, we will be implying only the latter functions. On the other hand, it is common that the topic of EA is addressed separately.

Security and Performance attributes of KM: KM protocols in addition to efficiently accomplishing security for the secure group start-up, they must efficiently accomplish **group re-key and maintenance** at **steady state**. At steady state, a **new group key** is required in the following cases: (*a*) to handle membership updates (addition of new members or eviction of existing ones), and (*b*) periodically, to refresh keys that reach the end of their cryptographic lifetime. In (*a*), the re-key operation is needed to ensure that messages sent to the group cannot be accessed by a former member whose membership has been revoked (**Forward Re-key**). It is often required that a member who joins be denied access to messages that were sent to the group prior to its membership (**Backward Re-key**). At any case, the group key should be generated and distributed in such a way that it becomes computationally infeasible for an adversary to recover it (**Group Key Secrecy**) or use it to recover subsequent or preceding group keys (**Key Independence**). Hence, the main security requirement of **member addition** is the secrecy of the **preceding group keys** with respect to both outsiders and subsequent group members, while this of **member deletion** is the secrecy of the **subsequent group keys** with respect to both outsiders and former group members:

Forward (Backward) Secrecy: A passive adversary who knows a subset of group keys cannot discover subsequent (preceding) group keys.

Adversarial Model: In this work we deal with passive adversaries, who are only allowed to eavesdrop on the entire communication along the network. We aim to show that our KG schemes are secure in the following sense: two or more parties are allowed to exchange information over an insecure channel and agree on a common group key, while it is computationally infeasible for a node that is not part of the group to recover the group key even after collecting all messages exchanged in the clear.

There exist quite a few KM proposals for secure group communications in the literature. Most of them address wire-line networks and cannot operate as such in MANETs. Based on the diverse nature of KG, KD or KA algorithms, we classify them in two major categories: the *contributory* and the *centralized* schemes.

Contributory: all members contribute equally (with partial shares) to the construction of the group key by interacting among themselves as follows: each node receives contributions from other nodes, and combines them with its own to compute the same group key. They are not robust to node failures (GDH.1.2.3, etc.)

Non-contributory: a group leader alone generates and distributes the group key to members, which do not contribute to group KG. These are centralized protocols, with simple KG algorithm. They are robust to member failures, but the leader is a single point of failure (GKMP, CBT, LKH, OFT, ELK, etc.)

In distributed architectures the group key can be either generated in a contributory fashion or generated by one member. They simply do not have group controllers. Next, we provide a brief overview of existing work on KM, following these two categories.

Contributory Schemes.

Becker et al. [1], derived lower bounds for contributory KG systems for the gossip problem and proved them realistic for **Diffie-Hellman** (**DH**) based protocols. The DH protocol is used for two parties to agree on a common key. Becker et al. used the basic DH distribution extended to groups from the work of Steiner et al.[3]. Steiner et al. extended DH [4] to three protocols that support group operations: **GDH.1, GDH.2, GDH.3.** GDH.2 is the most efficient representative: it achieves minimization of the total number of message exchanges, and provides efficient algorithms for re-keying. **Tree Group Diffie Hellman** (**TGDH**) by Perrig [6] and Kim et al. [7], is a new hybrid, efficient protocol that blends binary key trees (where members are represented by the leaves in these trees) with DH key exchanges

(DHKEs). Becker in [1], introduced the **Hypercube** protocol as one requiring the minimum number of rounds. It is based on successive pair-wise DHKEs between different pairs of nodes in each round. The direction of the exchanges follows the dimension of a logical Hypercube. In [2], Asokan et al. added to Hypercube an algorithm to assist to its operation if failures occur during key establishment. This was an important step, as most contributory protocols are not designed to handle failures at all. Becker et al.[1] introduced the **Octopus** protocol that yields minimum number of total messages, and then presented the **2^d-Octopus** that combined Octopus with Hypercube to a very efficient protocol that worked for arbitrary number of nodes. Burmester et al. [9] proposed **BD**, a very efficient protocol that executes in three broadcast logical rounds. The drawback is the requirement of $2n$ broadcast messages. Ingemarsson et al. [8] introduced another DH-based scheme **- ING -** that requires all parties to be connected into a logical ring. We will analyze all these protocols in detail later in our work. Boyd [24] introduced another protocol for conference KA (CKA). The group key is generated with a function that combines the hashed contributions of all but one member, with the contribution of that one member. However, the non-hashed contribution must be encrypted with the public key of each of the rest of members, (and later decrypted from each member) and this makes CKA impractical. A distributed approach based on the Logical Key Hierarchy D-LKH is suggested by Rodeh et al. [25], in which sub-trees already sharing common keys with their members, agree on a mutual group key. The leader of one sub-tree communicates securely to the leader of the other sub-tree (they share a common key) its selection of a new key. Then, each sub-tree sends that key to its members, encrypted with the secret key known only to that sub-tree. The algorithm takes $\log_2 n$ rounds to complete. Another approach D-OFT using distributed LKH was proposed by Dondeti et al. [26]. This protocol uses OFT [10] without centralized entity. Every member is responsible for generating its own key and sending the blinded version of this key to its sibling. In one of the breakthrough advancement in KA protocols based on Elliptic Curves (ECs), Joux proposed a single-round three party KA protocol that uses bilinear pairings. Barua et al. [27] presented a ternary KA protocol by extending the basic Joux's protocol to multi-party setting with a proof of security against a passive adversary. From another perspective, a number of new or extended KA protocols have sprung from the design of secure and efficient authenticated group KA that has received much attention in current research. For example, Katz et al. [28] proposed a scalable, constant round (three), authenticated group KA protocol with forward secrecy. This protocol is a slight variant of BD, accompanied with a detailed proof of security. Li et al. [29] proposed a KA protocol based on secret sharing techniques. They use the well-known polynomial secret sharing to reconstruct a session key. Their work leads to a constant-round protocol which is also very costly.

Non-contributory Schemes - Centralized Architectures.

The proposed solutions are based on a simple key distribution center. The simplest and most fundamental such scheme is the **Group Key Management Protocol** (**GKMP**) [12], in which a group leader shares a secret key with each member and uses it to communicate the secret group key to the associated member. When a new member wants to join the group, the leader generates a new group key and encrypts it with the current group key. As all members know the group key, there is no solution for keeping the forward secrecy when a member leaves the group except to recreate an entirely new group without that member. Another proposal is the **Logical Tree Hierarchy protocol** (**LKH**) [11], where a key distribution center creates a tree of keys and distributes a hierarchy of key encryption keys (KEKs) for each member, associated with a leaf in the tree. Each group member is secretly given one of the keys at the bottom of the hierarchy and can decrypt the keys along its path from the leaf to the root. The key held by the root is the group key. A joining member is associated with a (new) leaf in the tree. All KEKs in the nodes from the new leaf's parent in the path to the root should be changed. A re-key message is generated containing each of the new KEKs encrypted with the

KEKs of the node's offspring. The size of the message produced is at most 2(log$_2n$). Removing a member follows a similar process. Waldvogel et al. [30] proposed LKH+, which is a variation of LKH. Instead of generating fresh keys and sending them to members during a re-key, all keys affected by membership change are passed through a one-way function. Every member that already knew the old key can calculate the new one. An improvement of LKH is the **One Way Function Tree** (**OFT**), that minimizes the number of bits broadcast to members after a membership change, proposed by McGrew et al. [10]. This scheme reduces the size of the re-keying message from 2(log$_2n$) to only (log$_2n$). Here, a node's KEK is generated, not distributed. The KEKs held by a node's children are blinded with a one-way function and then mixed together with a mixing function. The result is the KEK held by that node. Each node receives its own key, and the blinded keys corresponding to each node in its sibling set. Canetti et al. [31] proposed a slightly different approach, known as One-way Function Chain Tree that achieves the same communication overhead. This scheme uses a pseudo-random generator instead of a one-way function to generate the new KEKs and is applied only on users' removal. The pseudorandom generator doubles the size of its input. The process for the re-key is similar to OFT. Rafaeli et al. [32] presented a variation called **efficient hierarchical binary tree** (**EHBT**). Perrig et al. [13] proposed the **Efficient Large-group Key** (**ELK**), which is similar to OFT in the sense that a parent node key is generated from its children keys. ELK uses pseudo-random functions to build and manipulate the keys in the tree and no multicast messages are sent during a join operation. When members are deleted, new keys have to be generated for those nodes in the path from the removed node to the root. ELK also introduces the idea of *hints*. A *hint* is smaller than a key update message, and can be used to recover possible lost re-key message updates. It is provided to improve the reliability of the re-key operation and it is conveyed in data messages.

Decentralized Architectures.

In the decentralized approach, a large group is split into smaller subgroups. Different controllers are used to manage each subgroup minimizing the problem of concentrating the work on a single entity. In this approach, more entities are allowed to fail before the whole group is affected. Among attributes that are important in order to evaluate the efficiency of these frameworks are: independence of keys, locality of the re-key operation, backward and forward secrecy, the existence of decentralized controllers, keys *vs*. data.

RFC1949 proposes the use of trees built by the **Core Based Tree** (**CBT**) multicast routing protocol to deliver keys to a multicast group. This scheme does not provide solution for forward secrecy other than to recreate an entirely new group without the leaving members. Mitra, in [34], proposed a **Iolus**, a framework with a hierarchy of agents: a Group Security Agent (GSA) manages each subgroup. The GSAs are grouped in a top-level group that is managed by a Group Security Controller (GSC). Iolus uses independent keys for each subgroup and the absence of a general group key means that subgroup membership changes are only treated locally. In addition, the system is fault-tolerant because if a GSA fails, only its subgroup is affected. Although Iolus is scalable, it has the drawback of affecting the data path. There is the need for translating the data that goes from one subgroup to another. Since the GSA has to manage the group and in addition perform the translations needed, it may become a bottleneck. Dondeti et al. proposed a **dual-encryption protocol** (**DEP**) to overcome the problem of trusting third parties [35]. They suggest a hierarchical sub-grouping of the members where a subgroup manager (SGM) controls each subgroup, by sharing three kinds of KEKs. The group key is encrypted with more than one of these KEKs each time, as in the “onion approach”. This protocol is effective but costly and also compromises forward secrecy. Setia et al. [36] proposed **Kronos**, an approach driven by periodic re-keys rather than membership changes. Each GSA independently generates the same group key and transmits it to its members at the end of a predetermined period. Hence, the GSAs must have their clocks synchronized and agree on a re-key period. Although Kronos does not use a GSC and the

GSAs can generate the new keys independently, it compromises the group security because new key is based on the previous one. DeCleene et al. [37] present an intra-region group KM protocol (**IGKMP**). IGKMP is divided in administratively scoped areas. There is a Domain Key Distributor (DKD) and many Area Key Distributors (AKDs), responsible for one area each. The group key is generated by the DKD and distributed to members through the AKDs. The DKD keeps track of all AKDs, and each AKD keeps track of the members in its area. The protocol presents the undesirable feature of allowing the whole group to be disrupted if the DKD is compromised, and if an AKD is unavailable no members in that area are able to access the group communication. Rafaeli et al. proposed **Hydra** [33], which is a decentralized scheme without a central subgroup controller. If one or more Hydra Servers (HSs) become unavailable, the rest won't be affected. In order to have the group key distributed to all HSs a synchronized group KD protocol (**SGKDP**) is employed, that ensures that only a single valid HS generates the new group key at every time.

Recent proposals for wire-less ad-hoc networks.

There are numerous recent proposals for wireless ad-hoc networks. Even these schemes, do not seem to scale well or handle successfully the dynamics of the network. Furthermore, some of these approaches rely on public key cryptography, which is very expensive for resource constrained nodes, or on threshold cryptography [16, 17, 18, 19], which results in high communication overhead. Protocols that utilize threshold cryptography cannot efficiently accommodate an increasing number of members, and have potential security vulnerabilities, mainly due to the mobility of nodes. A very different approach this of probabilistic key pre-distribution, by Eschenauer et al. [20], or Perrig et al.[21], which is a very lightweight method, designed for sensor networks, but presents certain security vulnerabilities, and requires essentially some basic infrastructure to handle mobility and membership changes (mainly revocations). Amir et al. [14, 15], focus on robust contributory

KA, and extend GDH protocols to be made fault-tolerant to asynchronous network events. However, their scheme is designed for the Internet, and also requires an underlying reliable group communication service and ordering of messages, so that preservation of virtual semantics is guaranteed. Lazos et al. [22] address the problem of minimizing the communication overhead (in terms of energy expenditure) for secure group communications in energy-constrained wireless ad hoc networks. A cross-layer algorithm is used to build an "energy-optimal" routing multicast tree that will be used by the source, on top of which a KD tree is constructed. Their clustering algorithms cannot be implemented without basic infrastructure. The network topology is static, and it is shown that the optimal solution of their formulation for KD over an optimal tree does not scale with the group size.

It can be seen that there is no unique solution that can achieve all performance requirements. While centralized KM schemes are easy to implement, they tend to impose an overhead on a single entity. Protocols based on hierarchical sub-grouping are harder to implement, and raise other issues, such as interfering with the data path or imposing security hazards on the group. Distributed KM, by design, is not scalable.

2.3. Problem Statement: Octopus-based Schemes for MANETs

In this section, we provide a brief overview of the operation of our proposed KM protocol for multicast communication in MANETs, describe the network model, and our assumptions. Finally, we contrast the two main KM protocol categories with respect to a number of attributes that play a key role for their feasibility in a MANET, point out the difficulty of designing a suitable KM protocol for the network of study, and justify our proposed solution.

2.3.1. 2^d – Octopus Protocol Operation.

Hierarchy is supported through the partition of a large KA group to 2^d subgroups of smaller size. The KM operations are now applied per subgroup. In that fashion, each instance of the protocol operates on a smaller subset of nodes and as a consequence it can better capture its dynamics. Topological changes are easier to monitor and tolerate. Hierarchy enhances also the robustness of the network, reduces the control overhead and the communication information, and favors the spatial reuse. Distributed schemes benefit from hierarchical frameworks that permit data manipulation in a localized manner. All operations in our schemes can be executed from subgroup members without the need for centralized entities, pre-existing information, or fixed infrastructure. We note here that the simple Octopus (d=2) has the property of minimizing the total number of messages required for the group key.

First Step: Each subgroup agrees on its own subgroup key, and elects a subgroup leader to handle KA locally within its own subgroup. In MO and MOT, each subgroup executes a non-centralized (GDH.2) or a hybrid (TGDH) subgroup KG protocol, as opposed to (O), where each subgroup executes a centralized scheme that resembles GKMP.

Second Step: The subgroup leaders, representing their subgroups, interact among themselves and use the previously generated subgroup keys to agree on a global group key via another KA protocol, Hypercube, efficient in the number of rounds required for the group KG.

Third Step: The subgroup leaders distribute securely the group key to their subgroup, using the KG algorithm designated by the subgroup protocols used during the first step.

2.3.2. Network Model

The entities of the secure multicast group are represented as the vertices V in a (not necessarily connected) logical graph G: $\{V, E\}$. An edge (u, v) is assigned between any two vertices u, v, that correspond to members that communicate according to the KG algorithm. This bi-directional edge represents a communication link, associated with a weight $w(u, v) = 1$ (for logical graphs).

Octopus-extensions: The n nodes deployed in the field are separated into $2^d < n$ subgroups, where d is pre-selected. Each subgroup obtains a subgroup leader selected among the members. The subgroup leaders are either pre-defined or are the outcome of a leader election.

The subgroup leaders use link state information to transmit "advertising messages" to network nodes. Nodes that get multiple "advertising" signals and wish to subscribe to the group select as their subgroup leader the one whose signal prevails with respect to certain criteria (signal strength, relative direction or distance of two nodes, SNR etc.). The subgroup member notifies the leaders in the vicinity of its selection, and the designated leader subscribes this member to its subgroup members list. The subgroup leader probes its list at regular intervals in order to keep track of the membership changes. Therefore, the number of members in each subgroup depends on the network topology, density, etc.

2.3.3. Assumptions

1. All entities participating to the secure group are assumed to be authenticated prior to the key establishment phase.

2. No assumptions are made about secure routing. Our KG scheme, as any scheme that relies on non-secured routing, is susceptible to routing attacks. However, we rely on the redundancy of the routing to ensure that the exchanged messages are delivered in a timely manner.

3. Members are randomly deployed within the network and have only local view of the network (initially at least). Through an underlying discovery process that uses link state information ("Hello" messages and leaders' advertisements) members eventually obtain the required group membership information.

4. We initially assume a pre-selection of subgroup leaders that covers the entire field. These leaders know the identities of the rest of subgroup leaders at deployment.

5. As time progresses, the topology of the network changes and parts of the network may become unreachable. In these cases, either re-clustering or new leader election is required to accommodate the largest possible subset of group members and improve certain network metrics. As clustering is a broad and challenging research problem out of the scope of this work, we focus on the operation of Octopus-based protocols between two re-clustering

instances. Furthermore, we assume an underlying clustering mechanism that takes care of the initial separation of the original group to 2^d subgroups.

6. In this chapter the design and evaluation of all protocols is conducted at instances in between failures in the following sense: (*a*) no leader failures occur, (*b*) no member failures occur during the key establishment phase. At steady state, members' failures are handled as evictions, without further affecting the configuration of the subgroups within the network.

7. For the purpose of Octopus schemes, it is assumed that a sufficient number of nodes that acquire the appropriate processing and bandwidth capabilities to participate as tentative subgroup leaders can be found.

2.3.4. Centralized *vs.* Contributory Schemes in MANETs: Octopus Solution

Efficiency and Scalability: Centralized schemes can potentially provide simpler and more efficient key establishment. For example, in tree-based centralized protocols, in the event of a node failure, a new group key is computed, updating only a restricted number of keys.

Trust: Contributory protocols are preferred when no previous common secrets exist among parties, and no single node is trusted to generate the group key alone and distribute it to the rest of the parties. Such a group key could be compromised more easily.

Robustness: In a centralized scheme no node other than the leader itself constitutes a single point of failure, meaning that unless the leader fails, the group key can be established despite the failure of other members at any point during key establishment. This is the strongest asset of centralized protocols. Contributory protocols on the other hand are not robust to member failures. Members interact among themselves in complex predefined patterns and if a member does not respond when expected, the whole procedure comes to a standstill as all further actions depend now on the lost contribution and the process of the initial key establishment must start all over. Making contributory protocols stealthy to node failures is a very challenging problem. For example, the mechanism proposed to make Hypercube fault tolerant in [2], as well our own fault-tolerant version of the same protocol for MANET are far more complex than the mechanism inherent in centralized structures. In the recent work [14, 15], authors handle synchronization issues of GDH.2 and attribute robust characteristics to the scheme, by executing it on top of a secure routing tool. Still, the environment of deployment is the wire-line network. In a contributory protocol, the impact from the failure of any node (including a leader) is probably less severe than this of a group leader in a centralized protocol. This is because in the case of contributory protocols, it is easier to find another node that could undertake the role of a group leader. Even if the faulty leader can be replaced in a centralized protocol, the new leader will probably have to obtain secret keys with each one of the members before the group key establishment. For that, a linear number of asymmetric encryptions would be required, leading to considerable overhead in bandwidth and computation. In contributory protocols, failure of any member during group key establishment may result in restarting the initial key establishment phase, which is expensive. The overhead incurred on any single node is less than this incurred on the new leader of a centralized protocol. Moreover, the probability of such critical failures is much higher in a contributory protocol, since the latter will be affected by the failure of almost any member.

Group Leader: In centralized protocols, a node powerful enough, to execute the heavy tasks of a group leader might not even exist. Nevertheless, a multitasking leader becomes a single point of failure. Such a leader, apart from coordinating and synchronizing keying events, must have the capability to generate a session key alone, encrypt it separately with the public key of each individual member, etc. Furthermore, in a wire-less ad-hoc network potential leaders are running the risk of becoming resource drained, increasing also the latency for the group key establishment. If the leader fails, the centralized protocol must first initiate a leader election process. Provided that a member capable of undertaking the role of a leader exists in the first place, then this new leader becomes overloaded with the following tasks among others: it must trigger the key establishment phase from scratch, and must synchronize and coordinate its members. Every time a leader fails, the overhead to replace it and resume the KG is substantial. It would be preferred that either the leader addresses to a group of restricted size or that its role is rather coordinating, as in certain contributory schemes. Contributory protocols reflect the totally distributed nature of a group. The notion of a group leader usually exists in these protocols as well, but a leader in this case has a rather coordinating role, for which the required bandwidth and processing are less significant. From this perspective, contributory protocols appear more feasible than centralized, in a resource-constrained, infrastructure-less environment.

Conclusion: Based on these observations and comparisons, our motivation for proposing solutions based on Octopus schemes can be justified. The following features of Octopus schemes have motivated our interest to extend them: *a*) their **hierarchical framework** through which they can tolerate network dynamics and be more scalable, *b*) a **subgroup leader** handling a relatively **small subgroup** is a feasible option in MANETs, as opposed to a single leader in centralized schemes handling a very large group, *c*) **disruptions** are **handled locally** at first within the subgroup, and then reflected to the whole group via low cost re-keying algorithms over Hypercube, *d)* **flexibility**: within each subgroup we can apply a variety of novel or existing contributory or centralized schemes.

2.4. Key Agreement (KA) Schemes over a Logical Network Graph

In this section, we provide an overview and complexity analysis on protocols that are essential to our investigation and design of the extended Octopus protocols. We mostly focus on a number of KA protocols, but also provide an overview of a few centralized schemes (i.e. OFT, GKMP) that either help towards the analysis of certain KA schemes, or are used merely for performance evaluation purposes. Most of the protocols that we describe below are well documented in [3, 6, 7, 8, 9, 10, 12], so we provide a brief description here. However, the documentations for several of these protocols lack a comprehensive analytical evaluation of crucial metrics associated with both the initial group key establishment phase and steady state (re-keyings). Our contribution with respect to the documented studied protocols lies also in completing the existing analysis and yielding additional results on their performance. This analysis has its own merit of interest as in our effort to design efficient Octopus-based schemes, we have further individually improved and extended a number of candidate schemes to meet some or all of our objectives (i.e. performance, robustness, scalability).

Notation1: We use similar notation as in [3], for reasons of consistency:

n: number of participants in the protocol.

N_i: random exponent generated by i-th group member M_i.

q: order of algebraic group.

a: exponentiation base; generator in the algebraic group of order q.

G: cyclic group

S: subset of $\{N_1, N_2, \ldots, N_n\}$.

$\Pi(S)$: product of all elements is subset S.

K_n: group key shared among n members, we assume $K_n = K$ for simplicity.

Notation2: All the exponentiations pertained to the GDH-based protocols are in fact modular exponentiations (MEs), i.e. $H(X, q) = a^X \bmod q$. For simplicity, in all the subsequent sections, we refer to the MEs simply as exponentiations, and we write $F(X) = a^X$.

Notation3: Let $B(x) = a^x$ be the **blinding** of value x (ME of x under base a).

Notation 4: Let $\varphi(x) = \varphi_n(x) = x \bmod n$.

Definition 2.1. A **group** consists of a set G together with a binary operation: $G \times G \rightarrow G$. We denote the binary operation between elements A and B as $A*B$ (or AB). A group must satisfy the following axioms:

1. ***Closure:*** If A and B are two elements in G, then the product AB is also in G.

2. ***Associativity:*** For all $A, B, C \in G$: $(AB)C = A(BC)$.

3. ***Identity:*** There is an identity element I such that $IA = AI = A$ for every element $A \in G$.

4. ***Inverse:*** There must be an inverse or reciprocal of each element. Therefore, the set must contain an element $B = A^{-1}$ such that $AA^{-1} = A^{-1}A = I$ for each element of G.

If in addition, multiplication in G is commutative, that is, $AB = BA$, then the group is *abelian*. In a **finite** group, the number of elements is called **group order**. For n>1, g^n represents multiplication of g with itself n times, and g^0 is the identity element. If $G = \langle g \rangle$ for some $g \in G$, then G is **cyclic** with *generator g*. This means that every element in G is of the form g^k for some integer k. All **cyclic groups** are ***abelian*.** A cyclic group is associative, closed under addition, and has unique inverses. The numbers from 0 to n-1 represent its elements, 0 is the identity element, and the inverse of i is represented by n-i.

Definition 2.2. Discrete Logarithm Problem (DLP): Given an element g in a finite group G and another element $h \in G$, find an integer x such that $g^x = h$.

Like the factoring problem, DLP is believed to be difficult and also to be the hard direction of a one-way function. For this reason, it has been the basis of several public-key cryptosystems. The security of these systems rests on the assumption that DLs are hard to compute.

2.4.1. Group Diffie Hellman Protocols (GDH)

GDH protocols are the natural extension of the original 2-party DH key exchange (**DHKE**) to n parties. Having proven the security of a generic protocol of this class, a number of secure protocols of this class can be derived. In [3], three new GDH protocols are introduced, each of which is optimal with respect to certain metrics of interest. The security of DH and of the various versions of the generic GDH relies on the hardness of DLP.

DHKE: It allows two nodes to securely agree on the same secret key at the presence of "eavesdroppers" even though they exchange contributions in the clear. A DHKE for parties v, t with secret values r_v, r_t respectively includes the following steps: they both blind their secret values and exchange them. In the end, party v possesses (r_v, a^{r_t}), and party t possesses values (r_t, a^{r_v}), respectively. Party v raises its own secret to the received value and generates the next secret value as follows: $K_v = (a^{r_t})^{r_v} = a^{r_v r_t}$. Similarly, party t generates the secret value $K_t = (a^{r_v})^{r_t} = K_v$. So, through a DHKE the parties agree on the same secret value.

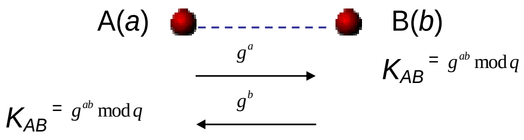
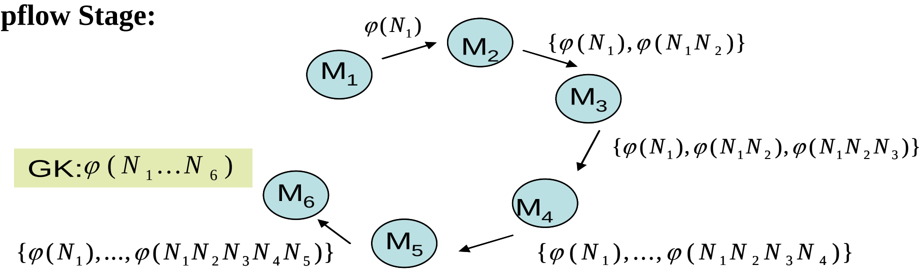


Figure 2.1. Illustration of a pair-wise DH Key Exchange (DHKE).

2.4.1.1 Overview and Cost Evaluation for Contributory GDH.1

Initial Key Establishment: GDH.1 consists of two stages: **upflow** and **downflow**.

Upflow Stage:



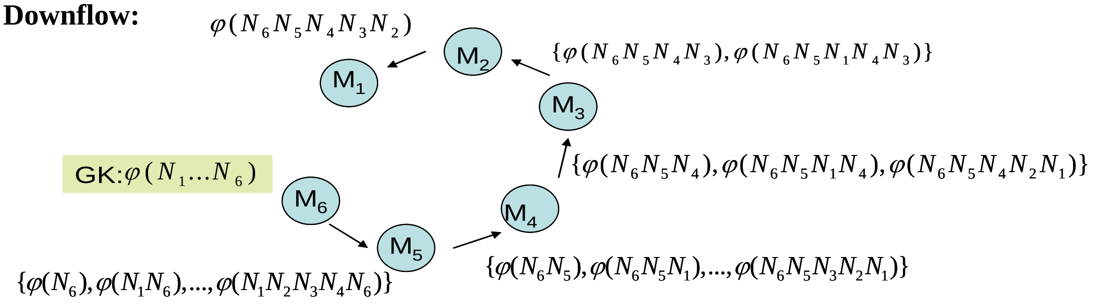
$\varphi(N_1)$
M_1
M_2
$\{\varphi(N_1), \varphi(N_1N_2)\}$
M_3
$\{\varphi(N_1), \varphi(N_1N_2), \varphi(N_1N_2N_3)\}$
GK: $\varphi(N_1 \ldots N_6)$
M_6
M_4
M_5
$\{\varphi(N_1), \ldots, \varphi(N_1N_2N_3N_4N_5)\}$
$\{\varphi(N_1), \ldots, \varphi(N_1N_2N_3N_4)\}$

Figure 2.2. Illustration of GDH.1 Upflow Stage.

The purpose of this stage is to collect contributions from all members. Each member M_i receives a collection of intermediate values. The task of each member M_i on the upflow is to compute $a^{N_1 \ldots N_i}$ by raising $a^{N_1 \ldots N_{i-1}}$ - the highest numbered incoming value – to the power of N_i, append it to the incoming flow and forward **all** to M_{i+1}. For example, M_4 receives the set $\{a^{N_1}, a^{N_1N_2}, a^{N_1N_2N_3}\}$ and forwards to M_5 the set: $\{a^{N_1}, a^{N_1N_2}, a^{N_1N_2N_3}, a^{N_1N_2N_3N_4}\}$. In the upflow stage, each member performs one ME and an upflow message between M_i and M_{i+1} contains i intermediate values. The final transaction in the upflow stage takes place when the highest-numbered member M$_n$ receives the upflow message and computes $(a^{N_1 \ldots N_{n-1}})^{N_n}$ which is the intended group key K_n.

The formula of the **upflow stage at round *i*** is: $M_i \rightarrow \{a^{\Pi\{N_K | k \in [1, j]\}} \mid j \in [1, i]\} \rightarrow M_{i+1}$.

Downflow:



$\varphi(N_6N_5N_4N_3N_2)$
M$_1$
M$_2$
$\{\varphi(N_6N_5N_4N_3), \varphi(N_6N_5N_1N_4N_3)\}$
M$_3$
$\{\varphi(N_6N_5N_4), \varphi(N_6N_5N_1N_4), \varphi(N_6N_5N_4N_2N_1)\}$
GK:$\varphi(N_1 \ldots N_6)$
M$_6$
M$_4$
M$_5$
$\{\varphi(N_6N_5), \varphi(N_6N_5N_1), \ldots, \varphi(N_6N_5N_3N_2N_1)\}$
$\{\varphi(N_6), \varphi(N_1N_6), \ldots, \varphi(N_1N_2N_3N_4N_6)\}$

Figure 2.3. Illustration of GDH.1 Down-Flow Stage.

After obtaining K_n, M_n initiates the downflow stage. In this stage, each M_i performs *i* MEs: one to compute K_n and (*i*-1) to provide intermediate values to subsequent (lower-indexed) group members.For example, assuming *n*=5, M_4 receives a downflow message: $\{ a^{N_5}, a^{N_1N_5}, a^{N_1N_2N_5}, a^{N_1N_2N_3N_5} \}$. First, it uses the last intermediate value to compute K_n. Then, it raises all intermediate values to the power of N_4 and forwards the resulting set: $\{ a^{N_5N_4}, a^{N_1N_5N_4}, a^{N_1N_2N_5}, a^{N_1N_2N_5N_4} \}$ to M_3. In general, the size of a downflow message decreases on each link; a message between M_{i+1} and M_i includes *i* intermediate values. The formula for the **downflow at round (*n*-1+*i*)** is: $M_n \rightarrow \{a^{\Pi\{N_K | k \notin [i, j]\}} \mid j \in [1, i]\} \rightarrow M_{n-i+1}$.

The **main drawback** of GDH.1 is its relatively large number of rounds. At the same time, GDH.1 imposes no special communication requirements, i.e., **no broadcasting or synchronization is necessary**.

Membership Changes:

Member Addition: We assume that member M_n saves the contents of the upflow message (stage 1, round (*n*-1)). M_n generates a new exponent $\widehat{N}_n$ and computes a new upflow message (using $\widehat{N}_n$, not N_n). For example, the new cardinal value now becomes: $a^{N_1 \dots N_{n-1}\widehat{N}_n}$. The new upflow message is sent to the new member M_{n+1}. M_{n+1} generates its own exponent and computes the new key $K = K_{n+1} = a^{N_1 \dots N_{n-1}\widehat{N}_n N_{n+1}}$. Finally, as in the normal protocol M_{n+1} computes the new downflow message and sends it to member M_n, and so on and so forth. It is obvious that while this algorithm spares the need to re-run anew the upflow message, it burdens all members with repeating all the computations of the downflow stage of the initial group key establishment.

Member Deletion: GDH.1 **cannot** handle the member deletion operation in a simple way, just by slightly modifying the flow of the original key establishment, as is the case for the member addition operation. Let M_p be the member slated for removal of the group, with $p \in [1, n\text{-}1]$, $p \neq n$. Even if M_n plays still a special role by generating a new exponent $\widehat{N}_n$, the nature of the intermediate messages collected from the previous state, makes it impossible to exclude the contribution of M_p without excluding also other members from the group key. That means that M_p will be still capable of reconstructing the new group key, if it gets a few particular messages that are transmitted in the clear. Besides, the downflow is such that only M_p can contribute to the lower indexed member $M_{p\text{-}1}$ the intermediate messages this member requires. So, there is no straightforward algorithm to exclude M_p alone from the group key without disrupting the operation of the whole protocol or without the need to reconstruct a totally different group key and change the initial algorithm in a non-trivial way. In the event that M_n is to be removed from the group, $M_{n\text{-}1}$ assumes the special role of generating a new exponent, and then the downflow stage can be re-run as described for the initial case. So, in this case, M_n can be excluded successfully from the computation of the new group key in a very simple way. In general however, GDH.1 cannot handle this operation efficiently.

2.4.1.2 Overview and Cost Evaluation for Contributory GDH.2

Initial Key Establishment: GDH.2 consists of two stages: **upflow** and **downflow**.

Upflow Stage:

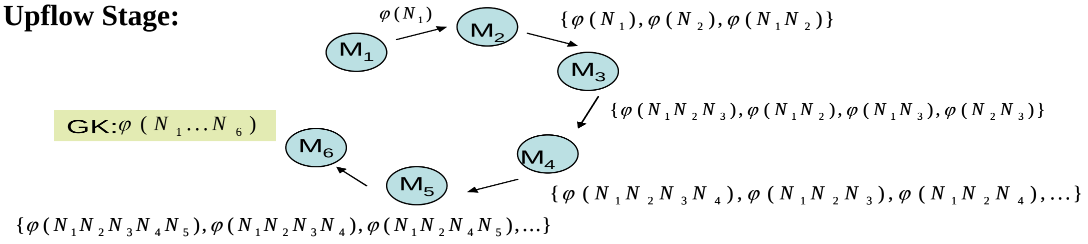

$\varphi(N_1)$
M1
M2
$\{\varphi(N_1),\varphi(N_2),\varphi(N_1N_2)\}$
M3
$\{\varphi(N_1N_2N_3),\varphi(N_1N_2),\varphi(N_1N_3),\varphi(N_2N_3)\}$
GK:$\varphi(N_1\ldots N_6)$
M6
M4
M5
$\{\varphi(N_1N_2N_3N_4),\varphi(N_1N_2N_3),\varphi(N_1N_2N_4),\ldots\}$
$\{\varphi(N_1N_2N_3N_4N_5),\varphi(N_1N_2N_3N_4),\varphi(N_1N_2N_4N_5),\ldots\}$

Figure 2.4. Illustration of GDH.2 UpFlow Stage.

The purpose of this stage is to collect contributions from all members. Member M_i receives a collection of intermediate values as illustrated in the scheme below. Each member M_i must compose i intermediate values (of i-1 exponents each), and one cardinal value containing i exponents and send them to M_{i+1}. For example, M_4 receives: $\{a^{N_1N_2N_3}, a^{N_1N_2}, a^{N_1N_3}, a^{N_3N_2}\}$ and outputs: $\{a^{N_1N_2N_3N_4}, a^{N_1N_2N_3}, a^{N_1N_2N_4}, a^{N_1N_3N_4}, a^{N_3N_2N_4}\}$. The cardinal value in this example is: $a^{N_1N_2N_3N_4}$. By the time the upflow reaches M_n, the cardinal value becomes: $a^{N_1 \ldots N_{n-1}N_n}$. M_n is thus the first group member to compute the key K_n.

The **upflow stage formula for round *i*** is: $M_i \rightarrow \{a^{\Pi\{N_K | k\in[1,i]\wedge k\neq j\}} \mid j\in[1,i]\} \rightarrow M_{i+1}$.

Downflow Stage:

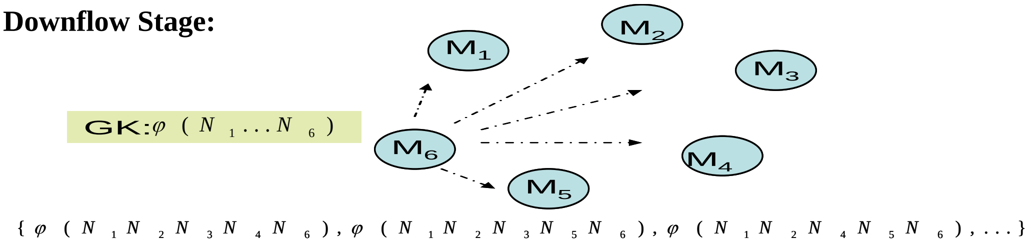

M1
M2
M3
GK:$\varphi(N_1\ldots N_6)$
M6
M4
M5
$\{\varphi(N_1N_2N_3N_4N_6),\varphi(N_1N_2N_3N_5N_6),\varphi(N_1N_2N_4N_5N_6),\ldots\}$

Figure 2.5. Illustration of GDH.2 DownFlow Stage.

The highest-indexed member M_n plays a special role by having to broadcasts the intermediate values to all group members. Each member M_i raises a corresponding intermediate value to its own secret value N_i and reconstructs the group key. For instance, M_2 receives: $a^{N_1 N_3 N_4}$, and computes the following group key: K_n= $(a^{N_1 N_3 N_4})^{N_2} = a^{N_1 N_2 N_3 N_4}$.

The formula of the **broadcast stage** is: $M_n \rightarrow \{a^{\Pi\{N_K | k \in [1,n] \wedge k \neq i\}} \mid i \in [1,n]\} \rightarrow M_i$.

Observation: The main advantage of GDH.2 is the low number of protocol rounds (*n*). Also, GDH.2 involves the least overhead w.r.t. the communication infrastructure: each node sends one message and receives two (except M_1 and M_n which receive only one message).

Membership Changes

Member Addition: We assume that member M_n saves the contents of the upflow message (stage 1, round (*n*-1)). M_n generates a new exponent $\widehat{N}_n$ and computes a new upflow message (using $\widehat{N}_n$, not N_n). For example, the new cardinal value now becomes: $a^{N_1 \ldots N_{n-1} \widehat{N}_n}$. The new upflow message is sent to the new member M_{n+1}. M_{n+1} generates its own exponent and computes the new key $K = K_{n+1} = a^{N_1 \ldots N_{n-1} \widehat{N}_n N_{n+1}}$. Finally, as in the normal protocol M_{n+1} computes *n* sub-keys of the form { $a^{\prod\{N_k | k \in [1,i] \wedge k \neq j}$ | $j \in [1,n]$ } and broadcasts to the other group members. This extension is straight-forward and requires only two additional rounds per each new member. The new key, K_{n+1} is easily computable by all parties and retains the same secrecy properties as K_n. However, while all other members compute K_{n+1} with a single ME, M_n is required to perform *n* MEs in addition to generating a new exponent.

Member Deletion: The protocol extension is similar to this for the member addition. Let M_p be the member slated for removal of the group, with $p \in [1, n\text{-}1]$, $p \neq n$. M_n plays a special role by generating a new exponent $\widehat{N}_n$. This time M_n computes a new set of (*n*-2) sub-keys: { $a^{\prod\{N_k | k \in [1,i] \wedge k \neq j}$ | $j \in [1, n-1]$ } and broadcasts them to all group members. Note that, since $a^{N_1*....*N_{p-1}*N_{p+1}*....N_{n-1}*\widehat{N}_n}$ is missing from the set of broadcasted sub-keys, the newly excluded M_p is unable to compute the new group key. In the event that M_n is to be removed from the group, $M_{n\text{-}1}$ assumes the special role as described above.

Since GDH.1 and GDH.2 both require (i+1) MEs from every M_i, the computational burden on every member increases as the group size grows. The same is true for message sizes. To address this concern, the authors of [3] have constructed GDH.3, which is different from GDH.1, GDH.2. We omit its description here to avoid redundancies.

2.4.1.3. Performance Evaluation for GDH.1-2-3 over Logical Network

GDH.1, GDH.2, and GDH.3 offer the **following advantages** (a proof outline is provided for each of the following claims [3]): **no a priori ordering of members, no synchronization**, **small number of MEs**, **minimal total number of messages (GDH.2)**, **minimal number of rounds for asynchronous operation (GDH.2), minimal number of messages sent/ received** by each **party (GDH.2), security** level equivalent to **2-party DH, implementation simplicity:** Just like 2-party DH, GDH.1, GDH.2, GDH.3 require only the **ME** operation and the random number generator for the protocol execution.

	Rounds	Mssgs	Combined Message Size	Exp/s per M_i	Total Exp/s
GDH.1	2(n-1)	2(n-1)	(n-1)n	(i+1) for i<n, n for M_n	$\frac{(n+3)n}{2}$ -1
GDH.2	n	N	$(n-1)(\frac{n}{2}+2)-1$	(i+1) for i<n, n for M_n	$\frac{(n+3)n}{2}$ -1
GDH.3	n+1	2n-1	3(n-1)	4 for i<(n-1), 2 for $M_{n\text{-}1}$, n for M_n	5n-6

Table 2.1. Performance of GDH.1, GDH.2, GDH.3 over Logical networks for Initial state.

2.4.2. Overview, Evaluation, and Extensions of Ingemarsson (ING)

2.4.2.1. Overview and Cost Evaluation for Contributory Ingemarsson (ING)

ING requires a synchronous start-up, requires that all parties are connected according to a logical ring, and completes in (n-1) rounds. In any given round, every participant raises the previously-received intermediate key value to the power of its own random exponent and forwards the result to the next party. Hence, in every round, *n* message exchanges are being performed. After (*n*-1) rounds and *n*(*n*-1) message exchanges, everyone computes the same key $K_n = a^{N_1 N_2 \ldots N_n}$. Since ING falls into the class of “natural” extensions of the 2-party DH protocol, it inherits the same security properties, and the proof of security presented in [1] for the generic GDH applies to ING as well. The following formula shows the **algorithmic flow of ING** for round *k*, where $k \in [1, n\text{-}1]$: $M_i \rightarrow a^{\Pi\{N_j | j \in [(i-k) \bmod n, i]\}} \rightarrow M_{i+1}$.

ING has the following characteristics: (*a*) **Per member Computation** (MEs): *n*, (*b*) **Total Processing** (MEs): n^2, (*c*) **Total # of Message Exchanges**: *n*(*n*-1), (*d*) **Latency:** (*n*-1).

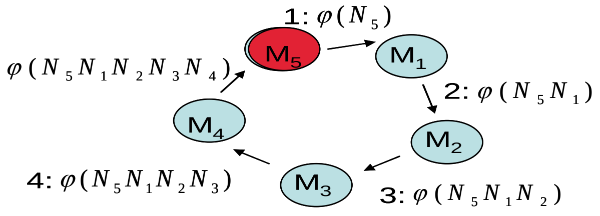

1: $\varphi(N_5)$
M_5
M_1
2: $\varphi(N_5 N_1)$
M_2
3: $\varphi(N_5 N_1 N_2)$
M_3
4: $\varphi(N_5 N_1 N_2 N_3)$
M_4
$\varphi(N_5 N_1 N_2 N_3 N_4)$

Figure 2.6. Illustration of ING Ring Protocol.

2.4.2.2. Our Complementary Analysis of ING: Introduce Re-Keying Algorithms

Member Additions/Deletions: ING does not lend itself to efficient construction of protocols for member additions or deletions. However, with careful inspection of the algorithmic flow and the intermediate values, some savings with respect to communication and computation can still be achieved.

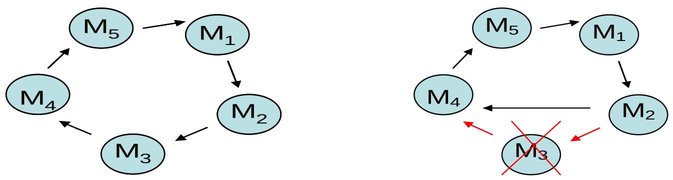


Figure 2.7. Illustration of Member Deletion Operation in ING Ring Protocol.

Member Deletion: We illustrate our study with an example of a 5 member ring. Assume that member M_3 is being evicted, and member M_5 alters its exponent (from N_5 to $\widehat{N}_5$) to preserve the required property of forward secrecy. There is not a simple, straight-forward way to exclude N_5 or N_3 from the previous group key or the intermediate values that the remaining members possess, and construct the new group key $K'_5 = a^{N_1N_2N_4N'_5}$. Therefore, the main KG algorithm must be repeated for the new group key to be established. However, since the exponents N_1, N_2, and N_4 remain the same, the message exchanges and computations for the rounds that involve only these exponents need not be repeated. The latency for the establishment of the new group key is still equal to the size of the ring (it is 3 rounds in this example), because there is at least one member for which all the intermediate values need to change. The idle rounds for each member can be thought of as "dummy" rounds. The new message exchanges and computations per round are depicted in the following table. It can be seen that both the communication exchanges and the computations can be reduced to half when we execute the algorithm for member deletion with savings.

The **overall number of MEs** now becomes: $n-1+\sum_{i=1}^{n-2} i = \frac{1}{2}(n^2-3n+4)$

The **overall number of message exchanges** now becomes: $\sum_{i=1}^{n-2} i = \frac{1}{2}(n^2-5n+6)$.

	Round 1	Round 2	Round 3	Group Key Computation
M_1	B($\widehat{N}_5$)	B($N_4\widehat{N}_5$)	B($N_2N_4\widehat{N}_5$)	B($N_1N_2N_4\widehat{N}_5$)
M_2	---	B($\widehat{N}_5N_1$)	B($N_4\widehat{N}_5N_1$)	B($N_1N_2N_4\widehat{N}_5$)
M_4	---	---	B($\widehat{N}_5N_1N_2$)	B($N_1N_2N_4\widehat{N}_5$)
M_5	---	---	---	B($N_1N_2N_4\widehat{N}_5$)

Table 2.2. New Message Exchanges for Member Deletion Operation in ING.

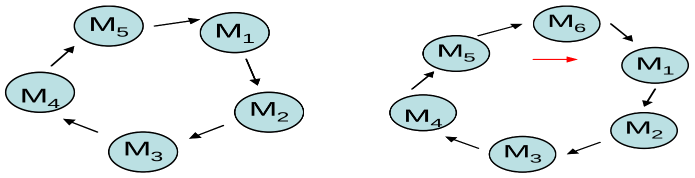


Figure 2.8. Illustration of Member Deletion Operation in ING Ring Protocol.

Member Addition: The addition operation is executed in a similar manner. We illustrate our study with the 5 member ring as well. Assume that node M_6 is added to the group, and assume further that member M_5 alters its exponent (from N_5 to $\widehat{N}_5$) to preserve the required property of backward secrecy. There is not a simple, straight-forward way to exclude N_5 and add N_6 in the previous intermediate values that the existing members possess, in order to construct the new group key $K_6' = a^{N_1 N_2 N_3 N_4 N_5' N_6}$. Hence, the main KG algorithm must be repeated for the new group key to be established. However, since the exponents N_1, N_2, N_3 and N_4 remain the same, the message exchanges and computations for the rounds that involve only these exponents need not be repeated. The **latency** for the establishment of the new group key **is still equal to the size of the ring** (5 rounds in this example), because there is at least one member for which all the intermediate values need to change. The idle rounds for each member can be thought of as "dummy" rounds. The new message exchanges and computations per round are depicted in the following table (Table 2.3).

The **overall number of MEs** now becomes: $(n+1) + \sum_{i=1}^{n} (i+1) = \frac{1}{2}(n^2 + 5n + 2)$.

The **overall number of message exchanges** now becomes: $n + \sum_{i=1}^{n} i = \frac{1}{2}(n^2 + 3n)$.

	Round1	Round 2	Round 3	Round 4	Round 5	Group Key
M_1	B(N_6)	B($\widehat{N}_5 N_6$)	B($N_4 \widehat{N}_5 N_6$)	B($N_3 ... N_6$)	B($N_2 ... N_6$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)
M_2	---	B($N_6 N_1$)	B($\widehat{N}_5 N_6 N_1$)	B($N_4 ... N_1$)	B($N_3 ... N_1$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)
M_3	---	---	B($N_6 N_1 N_2$)	B($\hat{N}_5 ... N_2$)	B($N_4 ... N_2$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)
M_4	---	---	---	B($N_6 ... N_3$)	B($\hat{N}_5 ... N_3$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)
M_5	---	---	---	---	B($N_6 ... N_4$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)
M_6	B($\widehat{N}_5$)	B($N_4 \widehat{N}_5$)	B($N_3 N_4 \widehat{N}_5$)	B($N_2 ... \widehat{N}_5$)	B($N_1 ... \widehat{N}_5$)	B($N_1 ... N_4 \widehat{N}_5 N_6$)

Table 2.3. Message Exchanges for Reduced Member Addition algorithm in ING.

2.4.3. Overview, Evaluation, and Extensions of Burmester/Desmedt (BD)

2.4.3.1. Overview and Cost Evaluation for Contributory Burmester/Desmedt (BD)

This scheme only requires 2 logical rounds and can be divided into three phases:

(1) Member M_i generates its random exponent N_i and broadcasts $z_i = a^{N_i} \bmod p$,

(2) Every member M_i computes and broadcasts $X_i = ({z_{i+1}}/{z_{i-1}})^{N_i} \bmod p$,

(3) Member M_i can compute the key $K_n = z_{i-1}^{nN_i} X_i^{n-1} X_{i+1}^{n-2} \ldots X_{i-2} \bmod p$.

The key defined by this scheme is different from the previous protocols, namely $K_n = a^{N_1N_2+N_2N_3+\ldots+N_nN_1}$. BD, as ING, is a totally symmetric protocol, as opposed to the GDH schemes. BD is superior to the rest of the contributory protocols discussed in terms of MEs since almost all operations (all but one) involve relatively small exponents (exponents of the order of O(n) – actually, the average exponent gets the value $n/2$ - as opposed to random exponents $N_i \in$ G). This makes for big savings in computation. Similar is the case in terms of **latency**. BD requires **only two rounds** of **simultaneous broadcasts** as opposed to linear number of rounds in the other protocols.

Observation 1: BD assumes that parties do n simultaneous broadcasts. This is not a feature available in most networks. So, it may be worthwhile to compare the other protocols with BD* - a version of BD without the simultaneous broadcast feature. Since BD* would require 2n-1 rounds, it does not compare with the rest as favorably as plain BD. The extra rounds in BD* are due to nodes waiting for a chance to win the medium for transmission. This is in contrast to GDH.1/2/3 where rounds are mostly triggered by message arrival. Thus, a broadcast round in BD* is shorter than a round in GDH.1/2/3. Moreover, in most non-specialized networks architectures a node cannot receive multiple messages simultaneously.

Observation 2: In terms of **total bandwidth overhead** BD/BD* come out clear winners, with the least total amount of information exchanged. However, in terms of **the messages**

received and sent by each party, BD/BD* do not compare favorably to the rest of the protocols, as **each party must receive** (*n*-1) messages and must broadcast 2 messages.

BD has the following **characteristics:** (*a*) **Latency (rounds)**: 2, (*b*) **Combined message size**: 2*n*, (*c*) **MEs per member:** (*n*+1), **Total MEs**: (*n*+1)*n*, (*d*) **Divisions per member**: 1.

2.4.3.2. Our Complementary Analysis on BD (Membership Changes)

Member Deletion: Let member M_n be removed from the group, and member M_{n-1} changes its own exponent from N_{n-1} to $\hat{N}_{n-1}$, to preserve the property of forward secrecy. The protocol is re-run but not all three steps are triggered for all members:

(1) Member M_{n-1} generates the random exponent $\hat{N}_{n-1}$ and broadcasts $\hat{z}_{n-1} = a^{\hat{N}_{n-1}} \bmod p$.

(2) Only members: M_{n-2}, M_{n-1}, M_1 compute and broadcast: $\hat{X}_{n-2} = (\hat{z}_{n-1}/z_{n-3})^{N_{n-2}} \bmod p$,

$\hat{X}_{n-1} = (z_1/z_{n-2})^{\hat{N}_{n-1}} \bmod p$, $\hat{X}_1 = (z_2/z_{n-1})^{N_1} \bmod p$, respectively.

(3) Member M_i, $i<n$, can compute the key $K_{n-1} = z_{i-1}^{(n-1)N_i} X_i^{n-2} X_{i+1}^{n-3} \ldots X_{i-2} \bmod p$.

Member M_n cannot compute the new key even if it gets all X_{is} , because it cannot derive the factor $z_{i-1}^{(n-1)N_i}$ (in essence the exponent N_i) for any other member than itself, which is however excluded from the new key, since it does not appear in the formula of the new group key K_{n-1}. Hence, forward secrecy is preserved. BD lends itself to a more efficient algorithm for member deletion than just starting over the key establishment. We present the analytical metrics of the algorithm below:

(*a*) **Latency (rounds)**: It remains the same, since at least one node has to go through all three steps. Hence, the protocol takes 2 rounds of message exchanges.

(*b*) **Combined message size**: During the 1st step, only 1 message is broadcast. During the 2nd step, 3 messages from members M_{n-2}, M_{n-1}, M_1 are broadcast. The combined message size is 4.

(*c*) **MEs per member:** During the 1st step, only member M_{n-1} performs 1 ME. During the 2nd step, members M_{n-2}, M_{n-1}, and M_1 perform 1 ME and 1 division each to compute the values they broadcast to the rest of members. During the 3d step, each member M_i computes K_{n-1} from K_n a. It is easy to see how K_{n+1} can be computed from K_n, if we analyze the formulae of the two group keys as illustrated below:

$$K_{n+1} = z_{i-1}^{(n-1)N_i} X_i^{n-2} X_{i+1}^{n-3} ... X_{n-2}^{t} \hat{X}_{n-1}^{t-1} \hat{X}_1^{t-2} \hat{X}_2^{t-3} X_3^{t-4} ... X_{i-2} \bmod q \text{ , and}$$

$$K_n = z_{i-1}^{nN_i} X_i^{n-1} X_{i+1}^{n-2} ... X_{n-2}^{t+1} X_{n-1}^{t} X_n^{t-1} X_1^{t-2} X_2^{t-3} X_3^{t-4} ... X_{i-2} \bmod q \text{ .}$$

Assuming that each member M_i stores each factor X_{i+j}^{n-j-1} used to compute K_n, it can easily compute K_{n-1} as follows: (*a*) it derives the factor $(z_{i-1}^{N_i} X_i \, X_{i+1} ... X_{n-2})^{-1} \bmod q$, by performing 1 ME and (*n*-2-*i*) modular multiplications (MMs), and 1 modular division (MD), (*b*) it raises the newly computed X_{is} to the appropriate exponents designated by the protocol and multiplies them into the formula computed in (*a*), that is, it does 3 MEs and 3 MMs, (*c*) it multiplies the rest of X_{is} (as stored from the previous key establishment) into the combined formula computed in (*b*), that is, it performs (*i*-2) MMs. Hence, during the 3d step, each member performs 4 MEs, (*n*-1) MMs, and 1 MD.

Overall, all members except for M_{n-2}, M_{n-1}, M_1, perform 4 MEs and 1 MD. Only members M_{n-2}, and M_1 perform 5 MEs and 2 MDs, and only member M_{n-1} performs 6 MEs and 2 MDs.

(*d*) **Total MEs**: (n-4)×4+2×5+6 = n.

(*e*) **Divisions:** (n-4)×1+3×2 = n+2.

Member Addition: Let member M_{n+1} be added to the group and generate its random exponent N_{n+1}. We further assume that member M_n changes its own exponent from N_n to $\hat{N}_n$, to preserve backward secrecy. The protocol is re-run but not all three steps are triggered for all members, as we will see next:

(1) Members M_n and M_{n+1} generate their random exponents $\hat{N}_n$, N_{n+1} and broadcast $\hat{z}_n$ = $a^{\hat{N}_n} \bmod p$, and z_{n+1} = $a^{N_{n+1}} \bmod p$.

(2) Only members: M_{n-1}, M_n, M_{n+1}, M_1, compute and broadcast: X_{n-1}= $(\hat{z}_n / z_{n-2})^{N_{n-1}} \bmod p$, X_n= $(z_{n+1} / z_{n-1})^{N_n} \bmod p$, X_{n+1}= $(z_1 / \hat{z}_n)^{N_{n+1}} \bmod p$, X_1= $(z_2 / z_{n+1})^{N_1} \bmod p$, respectively. The new member however will get al the rest of X_is used before except for X_n = $(z_{n+1} / z_{n-1})^{N_n} \bmod p$ from member M_n (or any other member) through a simple unicast.

(3) Any member M_i will compute the key:

$$K_{n+1} = z_{i-1}^{(n+1)N_i} X_i^{\,n} X_{i+1}^{\,n-1} ... X_{i-2} \bmod p \; = a^{N_1 N_2 + N_2 N_3 + ... + N_{n-1}\hat{N}_n + \hat{N}_n N_{n+1} + N_{n+1} N_1} \bmod p.$$

However, member M_{n+1} cannot compute the old key because it does not possess X_n and consequently, the exponent N_n would not appear in the formula of the old group key K_n.

It can be seen that BD lends itself to a more efficient algorithm for member addition than just starting over the key establishment. We present the analytical metrics of the algorithm below:

(*a*) **Latency (rounds)**: It remains the same, since at least one node has to go through all three steps. Hence, the protocol takes 2 rounds of message exchanges.

(*b*) **Combined message size**: During STEP 1, only 2 messages are broadcast. During STEP 2, 4 messages from members M_{n-1}, M_n, M_{n+1}, M_1 are broadcast, but one member unicasts all the rest of X_is to the new member (n messages). Hence, the combined messages size is (n+2).

(*c*) **MEs per member:** During STEP 1, M_n and M_{n+1} do 1 ME. During STEP 2, M_{n-1}, M_n, M_{n+1}, M_1, do 1 ME and 1 division each to compute the values they broadcast to the rest of members. During STEP 3, each member M_i computes K_{n+1} from K_n. It is easy to see how K_{n+1} is computed from K_n, if we analyze the formulae of the two group keys as illustrated below:

$$K_{n+1} = z_{i-1}^{(n+1)N_i} X_i^{\,n} X_{i+1}^{\,n-1} ... X_{n-2}^{\,t+2} \hat{X}_{n-1}^{\,t+1} \hat{X}_n^{\,t} \hat{X}_{n+1}^{\,t-1} \hat{X}_1^{\,t-2} X_2^{\,t-3} ... X_{i-2} \bmod q \text{, and}$$

$K_n =$ $z_{i-1}^{nN_i} X_i^{n-1} X_{i+1}^{n-2} ... X_{n-2}^{t+1} X_{n-1}^{t} X_n^{t-1} X_1^{t-2} X_2^{t-3} ... X_{i-2} \bmod q$.

Assuming that each member M_i stores each factor X_{i+j}^{n-j-1} used to compute K_n, it can easily compute K_{n+1} as follows: (*a*) it derives the factor $z_{i-1}^{N_i} X_i \, X_{i+1} ... X_{n-2} \bmod q$, by 1 ME and (*n*-2-*i*) MMs, (*b*) it raises the newly computed X_{is} to the appropriate exponents designated by the protocol and multiplies them into the formula computed in (*a*), that is, it does 4 MEs and 4 MMs, (*c*) it multiplies the rest of X_{is} (as stored from the previous key establishment) into the combined formula computed in (*b*), that is, it performs (*i*-2) MMs. Hence, during the 3^d step, each member performs 5 MEs and *n* MMs.

Overall, all members except for M_{n-1}, M_n, M_{n+1}, M_1, do 5 MEs. Only members M_{n-1}, and M_1 do 6 MEs and 1 MD, and only members M_n and M_{n+1} do 7 MEs and 1 MD.

(*d*) **Total MEs**: $(n+1-4)\times 5+2\times 6+2\times 7 = 5n-11$.

(*e*) **Divisions:** 1 MD per member for members M_{n-1}, M_n, M_{n+1}, M_1, namely 4 MDs overall.

2.4.4. Overview, Evaluation, and Extensions for Hypercube Protocol

2.4.4.1. Overview and Cost Evaluation for Contributory Hypercube Protocol

Overview: Hypercube is designed with the objective to minimize the total number of simple rounds required for the establishment of the group key. In general, 2^d parties agree upon a key within *d* simple rounds by performing pair-wise DHKEs on the edges of a logical *d*-dimensional cube. 2-party DHKEs constitute the basic Hypercube module. Every member is involved in exactly one DHKE per round. Each member uses the intermediate secret key K_{i-1} generated at the end of its DHKE in round (*i*-1), to compute through the current DHKE, the intermediate secret key K_i for round *i*. During a DHKE at round *i*, each member processes K_{i-1} and sends its peer in the clear the value $B(\varphi(K_{i-1}))$.

We identify the 2^d participants on the *d*-dimensional space $GF(2)^d$ and choose a basis $b_1, \ldots,$ b_d of $GF(2)^d$. In every round, the participants communicate on a maximum number of parallel edges of the d-dimensional cube (round i, direction b_i). Furthermore, all parties share a common key at the end of this protocol because the vectors b_1, …, b_d, form a basis of the vector space $GF(2)^d$. Now the protocol may be performed in d rounds as follows:

1.In the first round, every participant $v \in GF(2)^d$ generates a random number r_v and performs a DHKE with participant $v+b_1$ using the values r_v and r_{v+b1}, respectively.

2.In the i-th round, every participant $v \in GF(2)^d$ performs a DHKE with participant $v+b_i$, where both parties use the value generated in round i-1 as the secret value for the DHKE.

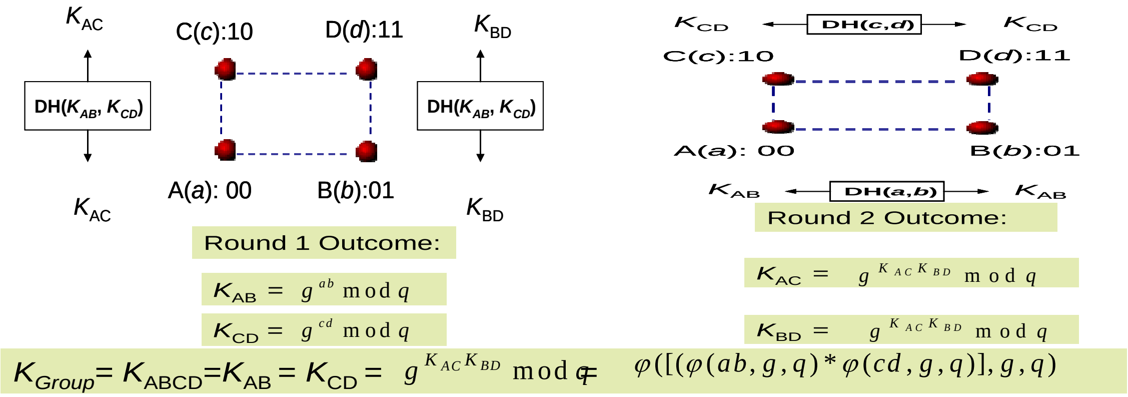
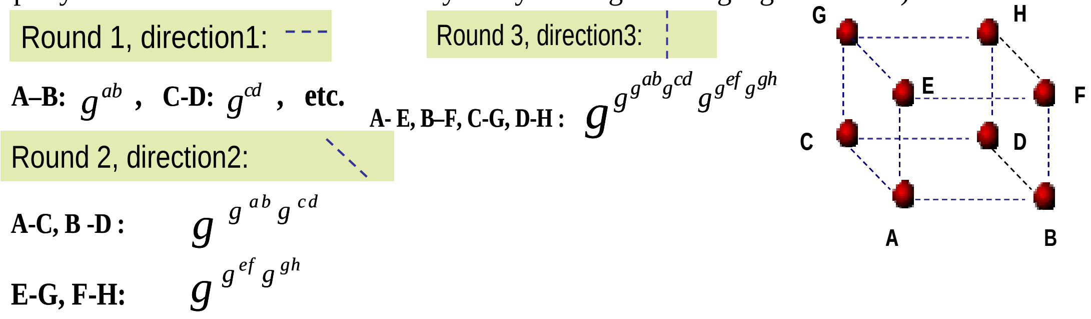


Figure 2.9. Illustration of Hypercube Protocol Operation with d=2.

Hypercube Performance Characteristics: (*a*) Total Number of Messages: $n \times d$, (*b*) Total Number of MEs: $n \times d$, (*c*) Total Exchanges: $n \times d/2$, (*d*) Simple Rounds (every party sends and/or receives at most one message per round)**: *d*, (*e*) Synchronous Rounds** (every party sends and/or receives arbitrarily many messages during a given round)**: *d*.**



Round 1, direction1:
A–B: g^{ab} , C-D: g^{cd} , etc.
Round 2, direction2:
A-C, B -D : $g^{g^{ab}g^{cd}}$
E-G, F-H: $g^{g^{ef}g^{gh}}$
Round 3, direction3:
A- E, B–F, C-G, D-H : $g^{g^{g^{ab}g^{cd}}g^{g^{ef}g^{gh}}}$
G
H
E
F
C
D
A
B

Figure 2.10. Illustration of Hypercube Protocol Operation with d=2

2.4.4.2. Our Complementary Analysis on Re-Keying Algorithms for Hypercube

No efficient re-keying algorithms have been anticipated for the original Hypercube in the event of membership changes. Actually, membership changes are difficult to handle compared to other KA protocols, because: (*a*) they have direct impact on multiple other members (i.e. up to *d*), and (*b*) they modify the number of members while the protocol requires that this number is a power of 2. In this section, we look into efficient re-keying algorithms in order to preserve the properties of forward or backward secrecy without re-starting the protocol from scratch when membership changes occur. Below, we introduce our own re-keying algorithms for eviction and addition and we analytically evaluate both.

Member Deletion: Through successive member deletions the number *n* of parties takes values in $[2^{d-1}, 2^d)$. Even if $2^{d-1}<n<2^d$ Hypercube still requires *d* rounds and 2^d message exchanges to compute a group key for its *n* parties. Hence, in order to keep Hypercube running, we must maintain the virtual *d*-cube of communications, despite the eviction of potentially multiple members. We re-establish the Hypercube structure as follows, ensuring that the pre-established schedule of communications does not change significantly:

- **Case 1:** The absence of any of the initial 2^d members – say *B* - can be accommodated by "replicating" its peer of the 1st round – say *A* - and placing it on the vertex previously occupied by the slated party *B*. Peer - *A* - substitutes *B* in all rounds, if Hypercube is executed anew. Node *A* may or may not generate a different blind value, depending on how we agree to construct the new group key. In every round, the nodes that were scheduled to perform a message exchange with *B*, communicate with *A* instead.
- **Case 2:** If two peers of the 1st round – say *A* and *B* – are evicted simultaneously, then another pair of nodes of the 1st round logically splits – say *C* and *D* – to accommodate the evicted pair. Then, *C* remains on the edge occupied by *C* and *D*, logically replicating itself and replacing *D*, and *D* handles the edge previously occupied by *A*, *B*, as shown in case 1.
- **Generalized Case:** If an *i*-cube of 2^i peers formed during round *i* is evicted simultaneously, then another *i*-cube of nodes formed during round *i* as well logically splits to accommodate the evicted *i*-cube. The rest is handled as in cases 1, 2. If the number of remaining parties *n*

becomes a power of 2 again, so that $n=2^X$, where $X<d$, then the communications schedule is reconfigured to X rounds only, and the performance of the scheme shifts back to the optimal.

To preserve forward secrecy, we want to remove the contribution of the evicted member from the intermediate keys, and consequently from the group key. From this perspective, a member that is no longer legitimate according to the group policy must not be able to compute any subsequent group keys, even if it still receives all the future blinded values or blinded keys (BKs) exchanged. We want the remaining members to compute a new group key so that: (*a*) *B* is unable to reconstruct it (forward secrecy), while (*b*) most of the messages already exchanged still remain valid for the new group key (re-use KM data to limit the re-keying overhead). The extreme approach to exclude member *B* from the new group key is to start over Hypercube from scratch, and have the remaining members contribute new BKs.

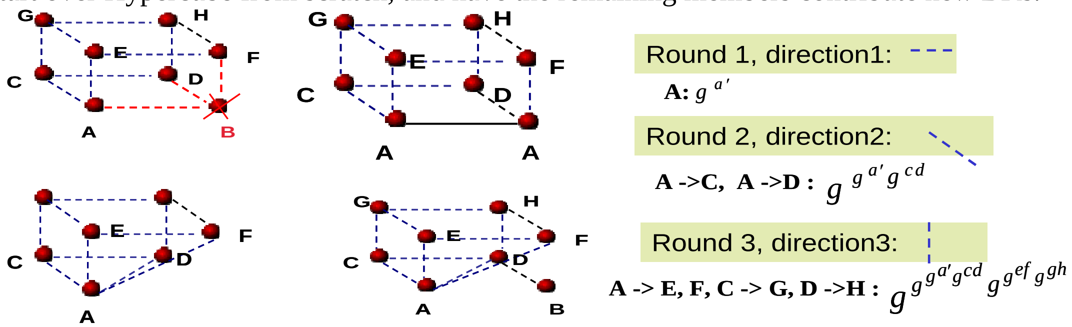


Figure 2.11. Illustration of Hypercube Re-Keying after eviction of member B (d=3).

Our Approach – Reduced Hypercube Execution: A reduced version of Hypercube re-starts from round 1 as follows: The communication of messages in every round is one way instead of bi-directional. In every round, only the members whose previous intermediate values were “polluted” with the contribution of the evicted member become senders of new BKs to their peers (they re-use however the BKs sent previously from these peers). Their peers become only receivers of the new BKs. In any round, all members that become either senders or receivers of new intermediate values re-compute new BKs for the following round, and trigger the peers of the following round, and so on and so forth.

"Polluted" are the parties whose exchanges during a given round have been affected by the secret value of the evicted member, namely the "polluted" members comprise the secret value of the evicted member in their exponents. We need to remove the contribution of the evicted member from all exponents, and re-do the computations in such a way that the evicted member cannot benefit from the intermediate key messages it may receive in the clear. If member *B* is evicted, member *A* substitutes it and generates a new secret value *a'*. At the end of round 1, member *A* generates the "un-polluted" BK: $a^{a'}$, and will use this for round 2 (free from the contribution of member B). Only member *A* is affected from the eviction of *B* in round 1. In round 2, *A* keeps the value $a^{a^{cd}}$ received during the same round of the previous key establishment by members *C* or *D*, but communicates the new BK $a^{a'}$ to both *C* and *D*. In round 3, members *A*, *C*, *D* keep the BK $a^{a^{a^{ef}a^{gh}}}$ received during the previous key establishment by members *E*, *F*, *G*, *H* respectively, but each (*A*, *C*, *D*) communicates the new BK $a^{a^{a'}a^{cd}}$ to those members (*E*, *F*, *G*, *H*). So, for the new key establishment after the eviction of a member, $2^{i\text{-}1}$-1 members (the "polluted" ones that got cleaned up before round *i*) communicate their newly computed BKs to the $2^{i\text{-}1}$ members as designated by their communication schedule during round *i*. Hence, through successive communication of the re-computed BKs in one direction, a virtual cube of increasing size is restored per round (*i*-cube after round *i*). At the end, all members compute new group key. The evicted member, even though it may get all BKs sent, it is unable to derive the new group key. This approach yields the following performance characteristics:

(*a*) Overall # MEs: All parties do 1 ME per round to re-compute their intermediate values, but only the members that become senders in each round ($2^{i\text{-}1}$-1 members at round *i*) do 1 more ME to compute the BKs they need to send out. Hence, the overall number of MEs (as opposed to $n \times d$ of the initial Hypercube) is: $\sum_{i=1}^{d}(2^{i-1}-1)+2^{d}$ = $2 \times (2^{d}\text{-}1)$–(*d*-1) =2*n*-(*d*-1), (***b***) **Overall Bandwidth**: Only $2^{i\text{-}1}$ members become senders of a single message at round *i*. Hence, the overall bandwidth becomes $\sum_{i=1}^{d} 2^{i-1} = 2^d - 1$ = *n* (as opposed to $n \times d$ if Hypercube starts all over), **(*c*) Number of Rounds**: *d*.

Member Addition: Through successive member additions the number *n* of members takes values in (2^d, 2^{d+1}]. If 2^{d+1}>*n*>2^d executing Hypercube requires *d*+1 rounds and 2^{d+1} message exchanges. However, a reconfiguration of the schedule must take place, to accommodate the new member. Furthermore, we must also maintain the virtual (*d*+1)-cube of communications. One way to do that is to replicate members from the previous *d*-Cube to fill the empty spaces in the (*d*+1)-Cube produced by the addition of a number of members. This can be handled according to *case1*, *case2*, and the *general case* described for the case of *member deletion*. This means that most of the peers of the previous *d*-Cube must split and replicate themselves in order to fill the empty gaps so that a (*d*+1)-Cube is generated. It is obvious that this method is very inefficient in terms of communication and computation overhead, as well as the overhead and latency required for the reconfiguration of the communication schedule. This is even more so, if the number of the members added to the group is low. Hence, the original Hypercube as such does not lend itself for efficient algorithms for members' addition.

However, the *Octopus* protocol, that we are going to introduce later in this section, is an evolution of Hypercube, that provides a very flexible framework to execute *member additions*: 2^d members become the vertices of a *d*-cube, and all the additional members, become members of the subgroup with leader any of the 2^d Hypercube members. The same approach can be adopted for the case of member additions in Hypercube. Each additional member is attached to any of the previous 2^d members that act as subgroup leaders. If the overall number of Hypercube members becomes a power of 2 again, then the communication schedule can be reconfigured and a virtual Hypercube can be restored.

The property of the backward secrecy must be preserved in our re-keying scheme when a member addition ocurs. The subgroup of the added member(s) can apply any KG protocol and generate a subgroup key. The subgroup leader (Hypercube member) is responsible to generate the BK of that key, and participate to the Reduced Hypercube execution as previously described. The nature of a suitable subgroup protocol will be studied later in the context of Octopus schemes. For now, we can as well assume that the added member and its leader perform a simple DHKE: the leader participates to the exchange with a new secret value, so that backward secrecy is preserved. Let the newly added member *J* be attached to member *A*. After the DHKE, member *A* triggers Hypercube using the BK: $a^{a''} = a^{a^{a'j}}$ .From this point, the case is handled similarly to the *member deletion* algorithm. Hence, through successive communication of the re-computed BKs in one direction, a virtual cube of increasing size is restored per round (*i*-cube after round *i*). At the end round, all members compute a new group key.

This approach yields the following performance characteristics: **(*a*) Overall number of MEs**: All parties must do 1 ME per round to re-compute their intermediate values, but only the members that become senders in each round (2^{i-1}-1 members at round *i*) do 1 extra ME to compute the BK they need to send out. Hence, the overall number of MEs (as opposed to $n \times d$ MEs of the initial Hypercube) is: $\sum_{i=1}^{d}(2^{i-1}-1)+2^{d}$ +4 = 2×(2^d+1) – (*d*-3) = 2*n*-(*d*-3), **(*b*) Overall bandwidth**: Only 2^{i-1} members become senders of a single message at round *i*. Hence, the overall bandwidth becomes: $\sum_{i=1}^{d}2^{i-1}+2=2^{d}+1$ = *n*. (as opposed to $n \times d$ of the original Hypercube), **(*c*) Number of Rounds**: *d*+1, with the additional round being the DHKE between the subgroup leader and the newly added member.

2.4.5. Overview and Cost Evaluation of One-Way Function Tree (OFT)

OFT uses one-way functions to compute a binary tree of keys. The keys are computed up the tree from the leaves to the root. This approach reduces re-keying broadcasts to about $\lg_2 n$, n being the number of members of the group. The group leader maintains a binary tree, each node x of which is associated with two cryptographic keys, the **un-blinded key** k_x and **blinded key (BK)** $k_x' = g(k_x)$, where g is a one-way function (e.g. hash function, i.e. MD5, SHA-1). The key is blinded in the sense that an adversary of limited computation may know k_x' and yet cannot find k_x. Each member is associated with a leaf in the tree. The leader communicates securely via symmetric encryption with subsets of members after they have obtained their session key. Prior to this, it communicates asymmetrically a randomly chosen key to each member. Interior keys are defined by the rule: $k_x = f(g(k_{left(x)}), g(k_{right(x)}))$, where f is a mixing function (e.g. XOR), and *left*, *right* denote the left and right children of a node. The key associated with the root serves as the group key. ***System Invariant:*** *Each member knows only the un-blinded node keys on the path from its node to the root, and the BKs of the siblings to the nodes that lie on its path to the root.* This "path" of siblings is also denoted as co-path. Those BKs are computed and sent by the leader. Each member maintains the un-blinded key associated with its leaf, and a list of all BKs of its co-path. This enables the member to compute the un-blinded keys along its path to the root, including the group key. If one BK changes and members get the new value, they re-compute the keys on their path and find the new group key. The addition of the one way function g to blind internal nodes gains an important functionality: each internal key is used as a communications subgroup key for the subgroup of all descendant members.

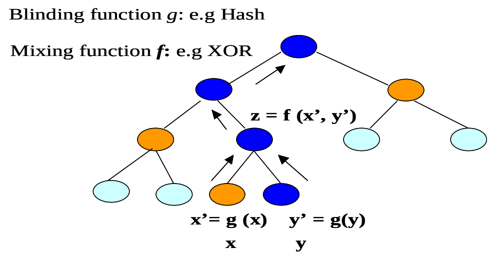


Figure 2.12. Illustration of centralized OFT Key Generation.

Member Addition or Eviction: After a member is added (evicted), the keys along the path from its node to the root change (its sibling gets a new un-blinded key). Because of the system invariant the added (evicted) member only knows the BKs of the siblings to the path from its parent to the root. These BKs are insufficient to compute directly any un-blinded keys. The added (evicted) member does not know the un-blinded key that is on the sibling node, so it cannot compute any un-blinded node keys and cannot compute the old (new) group key. This way, the backward (forward) secrecy is preserved. The performance analysis of OFT is well documented in [10]. Here, we just quote the main results.

Notation: We denote the cost for: (*a*) generating RSA random keys as C_r, (*b*) blinding a key as C_g, and (*c*) symmetrically encrypting a key as C_{SE}, h as the tree height.

OFT	# Broadcast bits	# Unicast bits	Leader Computation	Max Member Computation
Initial	$2nK$ or $3nK$	0	$2n(C_{SE}+C_r)+nC_r$	hC_E
Add	hK	hK	$h(C_{SE}+C_g)+2C_r$	$H(C_{SE}+C_g)$
Evict	hK	0	$h(C_{SE}+2C_g)+C_r$	$H(C_{SE}+C_g)$

Table 2.4. Analytical Evaluation of OFT

We also provide the analytical evaluation results for two important centralized protocols, GKMP and LKH, to contrast with OFT. Overall, OFT presents superior performance.

LKH, GKMP	LKH Bcast	GKMP Bcast	LKH Leader Computation	GKMP Leader Computation	LKH Max Member Computation	GKMP Max Member Computation
Initial	$2nK/3nK$	nK	$2n(C_{SE}+C_r)$	$n(C_{SE}+C_r)$	hC_{SE}	C_{SE}
Add	$2hK$	nK	$h(2C_{SE}+C_r)$	$nC_{SE}+C_r$	hC_{SE}	C_{SE}
Evict	$2hK$	nK	$h(2C_{SE}+C_r)$	nC_{SE}	hC_{SE}	C_{SE}

Table 2.5. Analytical Evaluation of LKH, GKMP.

2.4.6.1. Cost Evaluation for the Tree Group Diffie-Hellman protocol (TGDH)

Overview: In TGDH binary key trees are combined with DHKEs. Any member should be ready to become "sponsor" and assume the duties of a leader. TGDH resembles OFT in its structure and its operation. The basic *differences* are the following: (*a*) any member can act as a leader, (*b*) a member knows **all BKs of the tree** in addition to the secret (un-blinded) keys in its path from the leaf to the root, and (*c*) the merging function *f* is the 2-party DH, and the blinding function *g* is the ME. The secret key *x* of an internal node *s* results from a DHKE between offspring *left*(*s*) and *right*(*s*) with secret keys *y* and *z*. Then, $x = a^{yz}$, and a^x is the BK of node *s*. TGDH is a hybrid KA protocol: all members contribute equally to the group key, but only the sponsor undertakes the communication regulation.

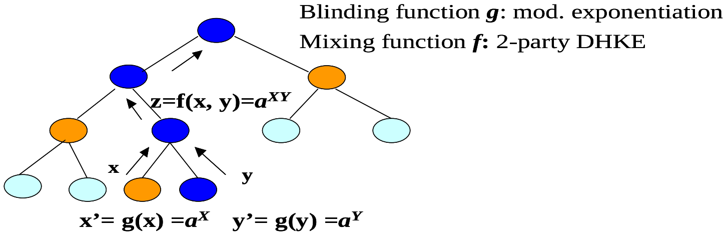

Blinding function g: mod. exponentiation
Mixing function f: 2-party DHKE
z=f(x, y)=a^{XY}
x
y
x'= g(x) =a^X
y'= g(y) =a^Y

Figure 2.13. Illustration of hybrid TGDH Key Generation.

2.4.6.2. Our Complementary Evaluation of the TGDH performance

1. Performance of TGDH for centralized initialization

The sponsor in TGDH is required only for communication regulation. At any given tree level, a member communicates to the sponsor the BK computed for that level. The sponsor waits to collect the BKs for that level computed by all members. Then, it combines them together to a single broadcast to all members. Each member gets the broadcast, extracts the required BK, combines it with its own secret, and generates the secret key of the next level. This process is repeated until the root key is computed, and requires $h = \log_2 n$ steps.

(*a*) Initial Sponsor Communication: $n \times h \times K$, **(*b*) Initial Member Communication:** $h \times K$, **(*c*) Total Initial Communication:**$\{n \times h \times K,\ (2n\text{-}1) \times K\ \} + (n\text{-}1) \times h \times K\ =\ \{(2\ n\text{-}1) \times h \times K,\ (n \times (2+h)\text{-}1\text{-}h)\ \times K\}$, **(*d*) Initial Member/Sponsor Computation (MEs):** $2 \times \log_2 n = 2h$ (1 ME to compute the secret key and 1 ME to blind it, at any tree level), **(*e*) Add/Delete Member/Sponsor Computation:** The new member does *h* MEs (using the BKs of its co-path). The rest of members do 1 to (*h*-1) MEs, as not the same number of BKs changes for each. In fact, $n/2$ members do only 1, $n/4$ members do 2, $n/2^h$ members do h MEs. In average, each member does: $\frac{1}{n}\sum_{i=1}^{\log_2 n=h}\frac{n}{2^i}\times i \approx 2$ MEs.

2. Performance of TGDH for distributed initialization

Initial Member/Sponsor Computation/Communication: Initially, every member becomes a "sponsor": it computes and broadcasts to the group the BK of the internal that corresponds to a given level. For every successive tree level, the number of sponsors is reduced to half (i.e. at the 2nd level ($n/2$) members remain sponsors etc.). The sponsor that corresponds to an internal node broadcasts its BK to all members, computes the secret key of its parent node and blinds it. So, as long as a member is sponsor, it does 2 MEs and 1 broadcast at any level. (***a***) Total number of **broadcast messages**: $\sum_{i=1}^{\log_2 n=h}\frac{n}{2^{i-1}}$ = (2n-1)<2n, and hence total **MEs**<4n, (***b***) **Per Sponsor/Member Communication:** $\log_2 n$ = h broadcasts at maximum, (***c***) **Per Sponsor/Member Processing:** At maximum, each member does $2\times\log_2 n = 2h$ MEs, (***d***) **Initial Member/Sponsor Storage:** Every member stores all (2n-1) BKs of the tree, and h keys for all the un-blinded keys in its path to the root. The values of the remaining metrics are extracted exactly as these of the centralized version.

2.4.7. Overview and Cost Evaluation of the Original Octopus Protocol (O)

Overview: Octopus uses a DH key computed in one round as a random input for the subsequent round. It is further assumed that there is a bijection from generator ***G*** into the field Z_q from which the parties choose their random secrets. Four parties *A, B, C, D* generate a group key using only four exchanges. Parties *A* and *B* execute a DHKE generating key α^{ab}, and simultaneously parties *C* and *D* execute a DHKE generating key α^{cd}. Then, the direction of DHKE changes, so that now *A* and *C* as well as *B* and *D* do a DHKE using as secret values the keys generated in the first step. *A*(*B*) sends $\alpha^{\phi(a^{ab})}$ to *C*(*D*) while *C*(*D*) sends $\alpha^{\phi(a^{cd})}$ to *A*(*B*) so that *A* and *C* (*B* and *D*) can generate the joint key $\alpha^{\phi(a^{cd})\phi(a^{ab})}$. Parties P_1, P_2,…, P_n generate a common group key by first dividing themselves into five groups. Four parties P_{n-3}, P_{n-2}, P_{n-1}, P_n take charge of the central control, denoted as *A*, *B*, *C*, *D*. The remaining parties distribute themselves into four groups: $\{P_i \mid i \in I_A\}$, $\{P_i \mid i \in I_B\}$, $\{P_i \mid i \in I_C\}$, $\{P_i \mid i \in I_D\}$, where I_A, I_B, I_C, I_D are pair-wise disjoint, possibly of equal size, and $I_A \cup I_B \cup I_C \cup I_D = \{1,\ldots, n\text{-}4\}$. Now $P_1,\ldots, P_n$ generate a group key as follows:

1. $\forall\ X \in \{A, B, C, D\}, \forall\ i \in I_X$, *X* generates joint key k_i with P_i via a DHKE.

2. Parties *A*, *B*, *C*, *D* do the 4-party DHKE described above using the values: $a=K(I_A)$, $b=K(I_B)$, $c=K(I_C)$, $d=K(I_D)$, where $K(J):=\prod_{i\in J} \phi(k_i)$ for $J \subseteq \{1,\ldots, n\text{-}4\}$. Thereafter, *A*, *B*, *C*, *D* hold the joint and later group key $K= a^{\phi(a^{K(I_A\cup I_B)})\phi(a^{K(I_C\cup I_D)})}$.

3. The step is described only for *A*. Parties *B*, *C*, *D* act accordingly. $\forall\, j \in I_A$, *A* sends the following two values to P_j: $a^{K(I_B\cup I_A\setminus\{j\})}$ and $a^{\phi(a^{K(I_C\cup I_D)})}$. P_j generate *K* now; first P_j calculates $(a^{K(I_B\cup I_A\setminus\{j\})})^{\phi(k_j)} = a^{K(I_A\cup I_B)}$ and then $K= a^{\phi(a^{K(I_C\cup I_D)})\phi(a^{K(I_A\cup I_B)})}$.

This protocol requires (*n*-4) exchanges to generate the DH keys k_i, 4 exchanges for the KA between *A*, *B*, *C*, *D* and (*n*-4) messages to be sent from *A*, *B*, *C*, *D* to P_1, P_2,…, P_{n-4}. Hence it performs a minimum number of (2*n*-4) exchanges.

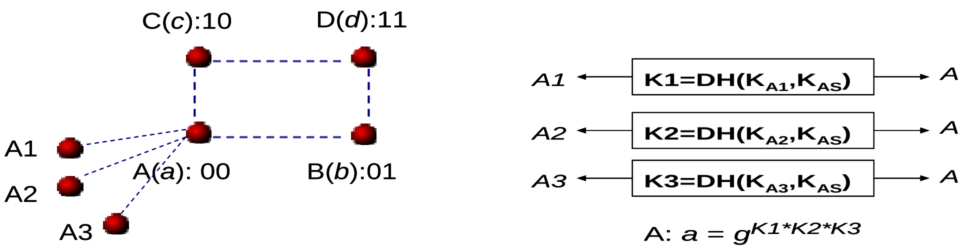


Figure 2.14. Illustration of the Initial Key Establishment for Octopus Protocol (d=2)

protocol	messages	exchanges	simple rounds	Synchr. Rounds	broadcasts

Octopus	$3n$-4	$2n$-4	$2\times\left\lceil \frac{n-4}{4} \right\rceil+2$	4	-

Table 2.6. Cost of Parameters in Octopus Protocol.

2.4.8. 2^d-Octopus Protocol

Overview: The number of simple rounds can be minimized by generalizing the idea of the 4-party KA described above. 2^d parties can agree upon a key within d simple rounds by performing DHKEs on the edges of a d-dimensional cube (Hypercube). In order to formulate a protocol for an arbitrary number of participants (<>2^d), that requires a low number of simple rounds, the idea of the Octopus protocol can be adopted again. In the 2^d–Octopus the parties act as in the simple Octopus. However, 2^d instead of four parties are distinguished to take charge of the central control, whereas the remaining (n-2^d) ones divide into 2^d groups. In steps 1 and 3, 2^d parties manage communication with the rest and in step 2 these 2^d parties execute Hypercube among themselves. We obtain the following complexities for 2^d-Octopus:

Total number of messages: During step 1 (n-2^d) members perform a DHKE with their subgroup leader (2 messages). During step 2 Hypercube is executed among 2^d subgroup leaders. During step3, each of the (n-2^d) members receives a single message from its leader.

Total number of Rounds: (d+2), (i.e. d for the Hypercube execution, 1 for the DHKEs of step 1, and 1 for the partial KD during step 3).

Protocol	Messages	Exchanges	Simple Rounds	Syn. Rounds
Hypercube	$n\times d$	$n\times d/2$	D	D
2^d- Octopus	$3\times(n-2^d) + 2^d \times d$	$2\times (n-2^d) + 2^{d-1} \times d$	$2\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil+d$	$2+d$

Table 2.7. Cost of Parameters in 2^d - Octopus Protocol.

2^d–Octopus provides a tradeoff between the total number of messages/exchanges and the number of simple rounds. For d = 2 (Octopus) the number of exchanges is optimal, whereas the number of simple rounds is comparatively high. Becker introduced Octopus as one that requires minimum number of total messages and then derived the 2^d-Octopus that combined Octopus with Hypercube to a very efficient scheme that works for arbitrary number of nodes.

2^d-Octopus protocols provide a class of KD systems without broadcasting which matches the lower bound - $\log_2 n$ - for the total number of synchronous rounds if n is a power of 2.

2.5. Our Extensions of the Original 2^d-Octopus on a Logical Network Graph

We are extending the original 2^d-Octopus (O) scheme by (*a*) providing efficient re-keying algorithms to it, (*b*) providing detailed analysis of the overall performance of the protocol, and (*c*) making it feasible for MANETs. The design of the original scheme does not provide for members evictions/additions, failures, partitions and merges that may occur all too often in MANETs. We also designed two new 2^d-Octopus based protocols, denoted as:

(MO): GDH.2 modified 2^d-Octopus (Modified Octopus MO), and

(MOT): TGDH modified 2^d-Octopus (Modified Octopus with Tree MOT).

We have derived the analytical formulae for Communication, Computation, Storage Costs, and Latency for all three Octopus-based schemes (O), MO, MOT, for the operation of initial key establishment and re-keying. We have also compared these protocols with the centralized OFT scheme. OFT is considered among the most efficient schemes with respect to the communication overhead incurred due to the re-keying operations at steady state. Even though schemes like OFT are not directly applicable to a large MANET, we want to compare our schemes with OFT to gain insight about the extra overhead required to render KM protocols scalable and applicable in such networks.

Our analytical evaluation has been conducted in between failures: leaders' and members' failures, as well as leader election schemes are not considered in the analysis conducted in this chapter. Such events and operations will be added to the basic framework gradually, as we progress our work in the following chapters. Through our description and analysis we have been able to demonstrate the superiority of the new protocol MOT over MO and (O), and the improvements compared to the original scheme (O). We derive MO and MOT by modifying the 1st and 3d steps of (O) accordingly, while maintaining the 2nd step, the core of all Octopus-based protocols -Hypercube -, intact. The three Octopus-based steps can be considered independent modules that are executed sequentially. Therefore, in what follows, we give an overview of each step individually, for the cases of initial key establishment and re-keying at steady state, for all three protocols, and discuss on the security level provided for each step. As discussed, we study the initial and steady state of KM protocols on groups in between re-clustering instances. Membership changes, including leader failures, are handled by the generation of new subgroup and/or group keys.

Why Octopus? In addition to the hierarchical structure of the Octopus which brings the discussed benefits of scalability, localization, etc., we were able to find that Octopus protocols lend themselves to simple and very efficient algorithms for handling membership changes, and failures. In particular, we have modified the core protocol of Octopus-based schemes - Hypercube - in such a way that: (*a*) the re-keying algorithms it executes substantially reduce the communication and computation costs incurred, compared to other schemes, (*b*) we integrate algorithms for handling any possible case of subgroup leader failures, including scenarios that allow leaders to resume, with low extra communication and computation cost. We also introduce an algorithm for handling failures that reduces the latency incurred in order to overcome these failures (Chapter 3, 4). Hypercube is not the most appropriate protocol to use within the subgroups. For the most efficient performance, Hypercube requires that the number of parties is a power of 2, and clearly the number of members within a subgroup depends must be flexible to accommodate membership changes and dynamic changes leading to failures. This number is also determined by factors such as the network configuration, the topology, the density of a region, etc. Therefore, the number of subgroup members cannot be controlled or pre-determined. In this case, other protocols may prove to be more efficient, such as centralized, or tree-based schemes, etc. that do not have requirements on the number of participants. On the other hand, within the subgroup, we may be interested in improving various and possibly different metrics.

2.5.1. Initial Key Establishment Operation for (O), MO, MOT

1st STEP of Initial Key Establishment for (O), MO, MOT

(O) includes only 2-party DHKEs between each member and the subgroup leader. Then, the subgroup leader combines all these contributions by multiplying them, and blinding the product. MO and MOT establish a subgroup key by applying GDH.2 and TGDH within each subgroup respectively. GDH.2 and TGDH are proven to be secure, as extensions of the generic GDH protocol, proven to be secure under the assumption that the DDH problem is hard. Therefore, during STEP 1 of MO and MOT, all members obtain their subgroup key in a secure way. Each of the 2^d subgroups establishes the subgroup keys that will be used for initiating step 2. In (O), each member generates a common key with the subgroup leader via a 2-party DHKE. Then, at a later instance (STEP 3), the subgroup leader will distribute individually to each member parts of the common group key that will be processed with the secret key between individual member and subgroup leader. This star-based scheme in the subgroup of (O) resembles GKMP, with the difference that here the subgroup key is built from contributions of all members, and is not arbitrarily selected by the leader. Membership changes in the subgroup of (O) are handled similarly to GKMP, not in a very efficient way in terms of communication and computation. Providing more efficient subgroup protocols to the original 2^d-Octopus scheme has been a basic motivation for the design of MO and MOT. In MO and MOT, GDH.2 and TGDH are applied instead, to improve one or more of the metrics of interest, particularly after the protocol reaches steady state. The subgroup leader, which can be for example the first member to compute the GDH.2 subgroup key in MO or the TGDH sponsor in MOT, will participate to STEP 2, using a function of the newly derived subgroup key. In contrast to (O), subgroup members in MO or MOT get the subgroup key already from the 1st STEP. In (O), simple members do not get a subgroup key at any step of the protocol execution. They get the group key at the 3d STEP, as in MO and MOT. This is yet another difference between the original protocol and our modified versions. A subgroup key can be used to handle and coordinate the subgroup in a localized manner, for exchange of local information within the subgroup, for handling dynamic changes within the subgroup. As we are going to prove with our analytical evaluation, the extra cost to generate a subgroup key in the 1st STEP of the protocol in MO and particularly in MOT is compensated at the 3d STEP, where the existence of subgroup keys facilitates the communication of the group key and lowers the associated cost for this operation. Furthermore, we are going to show that modifying the original (O) so that the subgroup members share a common subgroup key after the 1st STEP does not degrade the security level already provided in (O).

2nd STEP of Initial Key Establishment for (O), MO, MOT

This step is the same for all three protocols. The subgroup leaders only execute Hypercube, using as initial secret values the subgroup keys computed at the end of the 1st STEP. Hypercube is proven to be secure under the passive adversarial model. It reduces to a generic GDH protocol, which is secure under the assumption that the DLP is hard.

3d STEP of Initial Key Establishment for (O), MO, MOT

The subgroup leader of each subgroup S securely distributes to its members parts of the group key that has been computed at the end of Hypercube, in a way that each member that collects the corresponding parts can itself reconstruct the actual group key. This step is quite different in MO and MOT, as opposed to the (O). We will present an overview for all three protocols (O), MO, MOT, focusing on any arbitrary subgroup S_A. We denote the associated subgroup leader as A, and the associated subgroup key as $K(A)$. $K(A)$ is blinded and used by A to initiate Hypercube in step 2. Assume that A interacts with leader B during the 1st Hypercube round. Below, we demonstrate the resulting values computed at the end of the first few Hypercube rounds for the subgroup leaders involved in each round.

1st round ($A \leftrightarrow B$)**:**

$B\rightarrow A$: $\alpha^{\Pi_{i\in B}\phi(k_i)} = a^{K(I_B)} = a^{K_B} = H_B$, $A\rightarrow B$: $\alpha^{\Pi_{i\in A}\phi(k_i)} = a^{K(I_A)} = a^{K_A} = H_A$,

A, *B* do a DHKE and compute**:** $a^{K_A\cup K_B} = a^{K(I_A\cup I_B)} = \alpha^{\Pi_{i\in A,B}\phi(k_i)} = a^{K_{AB}} = H_{AB}$.

2nd round ($A\leftrightarrow C$):

$C\rightarrow A$: $a^{\phi(a^{K(I_C\cup I_D)})} = a^{\phi(H_{CD})}$. $A\rightarrow C$: $a^{\phi(a^{K(I_A\cup I_B)})} = a^{\phi(H_{AB})}$.

A, *C* (previously interacted with *B*, *D* respectively), do a DHKE and compute**:**

$a^{\phi(a^{K(I_A\cup I_B)})\phi(a^{K(I_C\cup I_D)})} = a^{\varphi(a^{K_{AB}})\varphi(a^{K_{CD}})} = a^{\phi(H_{AB})\phi(H_{CD})} = H_{ABCD}$.

3^d round ($A\leftrightarrow E$):

$E\rightarrow A$: $a^{\phi(a^{\varphi(a^{K_{EF}})\varphi(a^{K_{GJ}})})} = a^{\phi(a^{\varphi(H_{EF})\varphi(H_{GJ})})} = a^{\phi(H_{EFGJ})}$.

$A\rightarrow E$: $a^{\phi(a^{\varphi(a^{K_{AB}})\varphi(a^{K_{CD}})})} = a^{\phi(a^{\varphi(H_{AB})\varphi(H_{CD})})} = a^{\phi(H_{ABCD})}$.

A, *E* (previously interacted with *C*, *G* respectively), do a DHKE and compute:

$a^{\varphi(H_{ABCD})\varphi(H_{EFGJ})} = H_{ABCDEFGJ}$, etc.

The remaining rounds continue in a similar way. The final value is computed from all subgroup leaders and is the same for all subgroup leaders participating to the Hypercube.

Notation: Function $\varphi : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q$, maps the resulting values of a DHKE from $\mathbb{Z}_p^*$, to an order *q* subgroup $G\in \mathbb{Z}_p^*$ for the next Hypercube round, e.g. $x = \varphi(a^{AB})$, where *x, A, B* $\in \mathbb{Z}_q$, and $a^{AB} \in \mathbb{Z}_p^*$. We selected function φ as modulo *q* of a value: $\varphi(x) = x \; mod \; q$.

Lemma 2.4.1. For this instance of φ, the following is true: $\varphi(a^{AB}) = \varphi(\varphi((\varphi(a))^A)^B)$.

Proof: $\varphi(a^{AB}) = \varphi(\underbrace{a\times a\times ...a}_{A\times B\_times})$ (2.1),

where *a* can also be expressed in terms of modulo *q*: $a = t\times q + \varphi(a)$ (2.2).

Substituting (2.2) for value a in (1), we get:

$$\varphi(a^{AB}) = \varphi\,((t \times q + \varphi\,(a)) \times (t \times q + \varphi\,(a)) \times \ldots \times (\,t \times q + \varphi\,(a))) \qquad (2.3).$$

$$\varphi(a^{AB}) = \varphi\,\left(q \times F\,(q, t,\ \varphi\,(a)) + \prod_{1}^{A \times B} \varphi(a)\right), \qquad (2.4).$$

$F\,(q, t,\ \varphi\,(a))$ is the factor of (2.3) divided by q. Clearly, $\varphi\,\left(q \times F\,(q, t,\ \varphi\,(a))\right) = 0$ (2.5).

So, $\varphi(a^{AB}) = \varphi\,(\prod_{1}^{A \times B} \varphi(a)) = \varphi\,((\varphi\,(a))^{AB}) = \varphi\,((((\varphi\,(a))^{A})^{B}) = \varphi\,((X)^{B})$, (2.6).

where $X = (\varphi\,(a))^{A}$ can be expressed as in (2.2), in terms of modulo q: $X = r \times q + \varphi\,(X)$, s.t. similarly to (2.4), (2.5), we obtain: $\varphi(a^{AB}) = \varphi\,((\varphi\,(X))^{B}), = \varphi\,((\varphi\,((\varphi\,(a))^{A}))^{B})$.

Using leader A as our example, we see that leader A must collect the following values during the first three Hypercube rounds: $H_B = a^{K_B}$, $a^{\phi(H_{CD})}$, $a^{\phi(H_{EFGJ})}$, etc., respectively.

These values can be obtained from any node, but without knowing the proper initial secret (subgroup key K_A for leader A), they are of no use. K_A will be combined with the BK received at the 1st round, to construct the secret value of the 2nd round, and so on and so forth. At STEP 3, leader A communicates to its members all the intermediate BKs received during all d Hypercube rounds. Then each member uses K_A, shared with leader A at the end of STEP 1, to reconstruct the same intermediate values with A until it generates the group key itself.

Leader *A*	Prev. Rnd secret	Value Sent	Value Received	Next Rnd Secret	Secret Processed
Rnd #1	K_A	$H_A = a^{K_A}$	$H_B = a^{K_B}$	$H_{AB} = a^{K_{AB}}$	$\varphi\,(H_{AB})$
Rnd #2	$\varphi\,(H_{AB})$	$a^{\phi(H_{AB})}$	$a^{\phi(H_{CD})}$	$H_{ABCD} = a^{\phi(H_{AB})\phi(H_{CD})}$	$\varphi\,(H_{ABCD})$
Rnd #3	$\varphi\,(H_{ABCD})$	$a^{\phi(H_{ABCD})}$	$a^{\phi(H_{EFGJ})}$	$H_{ABCDEFGJ} = a^{\varphi(H_{ABCD})\varphi(H_{EFGJ}}$	$\varphi\,(H_{ABCDEFGJ})$

Table 2.8. Illustration of values pertaining to leader A in this Hypercube example.

(O): The subgroup key owned only by subgroup leader *A* is computed as follows: *K*(*A*):= $\prod_{i \in A} \phi(k_i)$, where k_i is the private key that member *i* has established with *A* as a result of their DHKE during the 1st step. *K*(*A*) is blinded and used by *A* to initiate Hypercube at the 2nd step. The secret value computed by *A* and *B* at the end of the 1st round is: $a^{K_A \cup K_B}$ = $a^{K(I_A \cup I_B)} = \alpha^{\Pi_{i \in A,B} \phi(k_i)} = a^{K_{AB}} = H_{AB}$. A subgroup member can use H_{AB} instead of the subgroup key, to successfully reconstruct the final group key. H_{AB} is secret and hence not communicated in the clear: leader *A* distributes H_{AB} partially to each member $j \in$ S$_A$, sending to it only the value $X_j = \alpha^{(\Pi_{i \in A,B} \phi(k_i))/\phi(k_j))}$ = a^{y_j}. Only leader *A* and member *j* hold the secret value $\phi(k_j)$. Member *j* can then raise $\phi(k_j)$ to the base X_j and compute: $(\alpha^{(\Pi_{i \in A,B} \phi(k_i))/\phi(k_j))})^{\phi(k_j)} = X_j^{\phi(k_j)} = \alpha^{\Pi_{i \in A,B} \phi(k_i)} = H_{AB}$. The remaining values that *A* sends to its members are the same for all members (i.e. illustrated in Table 2.8).

MO, MOT: The subgroup key *K*(*A*) = K_A is the result of the subgroup KG protocol applied to all subgroup members, including leader *A* during the 1st STEP. *K*(*A*) is blinded and used by *A* to initiate the Hypercube. The secret value computed by *A* and *B* at the end of the 1st round is: $a^{K_A \cup K_B} = a^{K_{AB}} = H_{AB}$. A subgroup member must also compute H_{AB}, in order to successfully reconstruct the final group key. Since H_{AB} is secret and therefore cannot be communicated to the subgroup members in the clear, leader *A* needs only to send the value it gets from leader *B* after the 1st Hypercube round: H_B = a^{K_B}. So, each member gets from leader *A* the same value H_B, and raises it to the power of the secret subgroup key, that is used as the exponent, to finally compute: H_{AB} = $(H_B)^{K_A}$ = $(a^{K_B})^{K_A} = a^{K_A K_B}$. The difference with (O) is that here leader *A* needs only communicate (broadcast) the **same single**

value to all its subgroup members. This approach results in the substantial reduction of the communication and computation cost for the 3^{d} STEP. In the case of MO and MOT, leader *A* does not need to compute $|S_A|$ base values as is the case in (O). It simply forwards value H_B received during the 1st Hypercube round. In fact, during the 3^{d} STEP, each subgroup leader communicates to its subgroup members only the BKs received during the *d* Hypercube rounds. Subgroup members in MO and MOT need only perform *d* MEs to derive the group key. All members start with the same secret value – subgroup key – and since they all receive identical values from leader *A*, the resulting intermediate values are the same for all members.

An issue to be investigated is whether this essential difference in the design of (O), MO and MOT affects the level of security in MO or MOT, compared to (O). In the following lemma we show that the deviations from (O) during the 1st and 3^{d} STEP of MO, MOT do not deteriorate the security level previously achieved in (O).

Lemma 2.2. *The distribution of the group key from leader to subgroup members at the 3^{d} STEP of MO or MOT protocols is not less secure that the corresponding operation in (O).*

Proof: It has been shown in (O) that the group KD from the subgroup leader *A* to its members satisfies our security objectives. In MO and MOT, GDH.2 and TGDH used for the establishment of the common subgroup key are proven to be secure against passive adversaries. What remains to be seen is whether the existence (3^{d} STEP of MO, MOT) of a common subgroup key affects the overall security of the schemes. In MO and MOT, the leader *A* communicates the same *d* values to all its members. Each member does *d* successive MEs, to generate the group key. Since **all members** receive and compute the **same** values, they can be logically abstracted and viewed as one **super-member**. Under this perspective, the protocol can be simulated as if there is only one member *j* subscribed to the subgroup leader *A*. The subgroup key established at STEP 1 in MO and MOT can be simulated as the secret key established between member *j* and leader *A* at STEP 1 in (O). Again, based on the hardness of DLP, an adversary can learn nothing about this secret key K_A by obtaining H_A, even if the same H_A is transmitted multiple times, as in MO and MOT. In contrast, the adversary in (O) gets more possibilities, as it may collect distinct versions of H_A, combine them and attempt to solve DLP or the factorization problem to obtain either the group key, the subgroup key or a pair-wise secret if possible (which is still infeasible). It is obvious that in the case of MO or MOT it is even harder for an adversary to extract any kind of secret information from the exchange messages during the 3^d step of the protocol. Thus, MO and MOT provide equal or better level of security than the original (O).

However, one may argue that sharing a subgroup key among all subgroup members from the 1st STEP in MO or MOT makes these protocols more vulnerable (e.g. to Byzantine attacks) compared to (O). In (O), an adversary may need to compromise most if not all nodes or learn the secret member-leader pair-wise keys of most subgroup members to generate the subgroup key after the 1st STEP. However, it only needs to obtain the pair-wise secret key of one member at any step to generate the group key. In MO and MOT, an adversary needs to compromise any subgroup member to learn initially the subgroup key, and subsequently generate the group key at the 3^d STEP. Thus, in all three protocols, an adversary needs to capture **any single** subgroup member or obtain the pair-wise secret or subgroup key of any single subgroup member to be able to generate finally the group key. From this perspective, all three protocols present the same vulnerability. In this work we do not investigate attacks from malicious insiders. In (O), an adversary needs to capture the subgroup leader to obtain the subgroup key, whereas in MO or MOT, it could get the subgroup key from any subgroup member. Getting the subgroup key however, is just an intermediate step, not the ultimate goal. In all three protocols, an adversary can generate the group key by getting either any pair-wise secret (in (O)) or the subgroup key (in all cases) of a subgroup member.

2.5.2 Initial Group Key Establishment: Analytical Evaluation of (O), MO, MOT

Before we proceed to the analytical evaluation of all three protocols, we illustrate in Table 2.9 an approximate estimation of the bit-complexity of a number of cryptographic parameters, required for the analysis of the protocols. Their derivation is described in our appendix in detail. During this auxiliary step, some of the results are quoted from the existing literature, and others are derived in this work, as the corresponding existing results are either platform or hardware/software implementation driven, or do not provide the bit accuracy required for our purposes. We are interested in the bit-complexities of these parameters to complete our analytical evaluation and produce results that are as generic as possible.

We provide estimation of the following parameters: public encryption/decryption C_{PE} and C_{PD}, symmetric encryption /decryption C_{SE} and C_{SD}, ME operation C_E, hashing operation C_g, KG functions C_r, and C_{rr}. We use the pseudo-random number generator C_{rr}, to generate secret shares for all protocols. This is different and simpler from the one we use to generate random numbers for the RSA system, C_r.

C_{rr}	$K + 1.5\ (1/k)\times K^3$, $1< k< \log_2 K$ pseudo- random number generation
C_r	$0.5\times K^4 + 0.35\times K^3 + 0.35\times K^2$, pseudo-random generation for primes (used for RSA keys)
C_{PE}	$3.5\times K_e\times K^2 + 2\times K_e \times K$ (RSA method, Common selections for K_e and K_d are: K_e=3, $K_d < K$)
C_{PD}	$(3.5/25)\times K_d \times K^2 + (2/25)\times K_d\times K$ (RSA method)
C_{SE} , C_{SD}	$\mathrm{C}_{\mathrm{DES}} \times K$, $35<C_{DES}<60$ (DES method)
C_E	$3.5\times l\times K^2 + 2\times l\times K$ (Montgomery complexity for mod multiplications, l: bit-size of exponent)
C_g	$30\times K$

Table 2.9. Bit-complexity of cryptographic parameters used in the protocols.

STEP 1: Each Subgroup establishes its own Subgroup Key

ORIGINAL OCTOPUS - (O):

The members of each subgroup run a hybrid KA protocol similar to this of GKMP. Each subgroup member is engaged in a 2-party DHKE with its leader. All (n-2^d) members of all subgroups send one message to their leaders, and the leaders broadcast to members 2^d messages in total. This is so because we assume that the leader sends the same share a^x to it own members. Each member j of course creates its own secret share λ_j, and the resulting secret key shared with its leader becomes: $y_j = a^{x\lambda_j}$. Hence, all leaders (2^d) and all members (n-2^d) together contribute (n-2^d+2^d) = n communication messages for the initial DHKE between member-leader. Each leader performs ($\left\lceil {n-2^d}/{2^d} \right\rceil$+1) MEs and the member performs 2 MEs. The leader stores the $\left\lceil {n-2^d}/{2^d} \right\rceil$ partial keys shared with each member, and also multiplies those keys to generate the subgroup key it is will use at the 2nd step. In the original documentation of (O), these multiplications are non-modular. However, taking into consideration Lemma 2.1, the leader can equivalently compute directly (a) the modular values of the elements it sends to each member, and (b) the modular value of the subgroup key. That is, the leader computes directly the values: $\varphi(a^{K(I_A)})$ instead of $K(I_A)$, and $\varphi(a^{K(I_B \cup I_A \backslash \{\varphi(k_j)\})})$ instead of $a^{K(I_B \cup I_A \backslash \{\varphi(k_j)\})}$, for a given member j. The leader computes the partial key $K_{A,B}$ as: $K_{A,B} = (a^{\varphi(a^{K(I_A)})})^{\varphi(a^{K(I_B)})} = (a^{\varphi(a^{K(I_A \cup I_B)})})$. Similarly, member j receives $\varphi(a^{K(I_B \cup I_A \backslash \{\varphi(k_j)\})})$ and computes: $\varphi((\varphi(a^{K(I_B \cup I_A \backslash \{\varphi(k_j)\})}))^{\varphi(k_j)})$ = $\varphi((\varphi(a^{K(I_B \cup I_A \backslash \{\varphi(k_j)\}) \cup \{\varphi(k_j)\}}) = \varphi(a^{K(I_B \cup I_A)})$, which is what member j must obtain in order to generate the final group key.

We will now compute the processing cost incurred at the subgroup leader for both options: (a) follow the original Octopus protocol and perform non-modular multiplications, and (b) spare substantial processing cost, by computing the moduli of the designated values, as discussed above. We start by exploring option (a). The following two lemmas contribute to the computation of the processing cost incurred by the leader to generate the intermediate subgroup key and the intermediate values at the end of STEP 1.

Lemma 2.3. *The processing cost for the non-modular multiplication of* $\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil$ *elements of size K (in bits) each is approximately:* $M = \left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil(\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil - \frac{1}{2})\times K^2$.

Proof: The leader must multiply x=$\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil$ elements of length K. The product of these non-modular multiplications is the subgroup key. The processing cost for this operation can be computed as follows: the x elements are pair-wise multiplied. The products of these multiplications have size $2\times K$ bits and are pair-wise multiplied as well resulting in products of size $4\times K$ bits, and so on and so forth. The processing cost for the non modular multiplication of two elements of bit size K each is approximately K^2. As the number of elements after each round of multiplications is reduced to half, the total number of multiplication rounds is $y = \log_2 x$. For example, at round j the number of multiplications is $x/2^j$, and the size of each element is: $2^{j-1}\times K$. The total processing cost is:

$$M = \sum_{j=1}^{y} \frac{x}{2^j} \times (2^{j-1} \times K)^2 = x \times K^2 \times \sum_{j=1}^{y} \frac{(2^{j-1})^2}{2^j} = x \times K^2 \times \sum_{j=1}^{y} 2^{j-2} = \frac{x}{2} \times K^2 \times \sum_{j=0}^{y} 2^j$$

$$M = \frac{x}{2} \times K^2 \times (2^{y+1} - 1) = x^2 \times K^2 - \frac{x}{2} \times K^2 = (x^2 - \frac{x}{2}) \times K^2 = \left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil(\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil - \frac{1}{2})\times K^2 .$$

The leader processes the subgroup key again before the beginning of the 3^d STEP, to generate the partial contribution to be sent to each of the x subgroup members during STEP 3. Hence it stores another x values. These values are generated for each member by dividing the subgroup key computed only at the leader (bit size $x\times K$) with the secret key that corresponds to the member in discussion (bit size K), erasing thus the contribution of that member from the subgroup key. A simple way to execute the division operation with multiplications and subtractions only and with less complexity, particularly in the number of subgroup members is large, is to use the "*divide and conquer algorithm*". Otherwise, the leader can re-do x multiplications of (x-1) shares to generate the shares of all x members. In the latter case, under the worst case scenario, the processing at the leader becomes: $\left\lceil \frac{n-2^d}{2^d} \right\rceil^3 \times K^2$.

We now explore option (b). The leader does $\left\lceil \frac{n-2^d}{2^d} \right\rceil$ MEs to compute the subgroup key, and $\left\lceil \frac{n-2^d}{2^d} \right\rceil \times (\left\lceil \frac{n-2^d}{2^d} \right\rceil$ -1) MEs to compute the partial keys of all its subgroup members.

Below, we summarize the most significant computation costs for the Initialization case at the 1st step, under the worst case scenario (under option (a)):

(*a*) Total Communication Cost: $n \times K$ bits, **(*b*) Leader Computation Cost:** C_{rr} + $(\left\lceil \frac{n-2^d}{2^d} \right\rceil + 1) \times C_E + \left\lceil \frac{n-2^d}{2^d} \right\rceil^3 \times K^2$, **(*c*) Member Computation Cost:** C_{rr} + $2 \times C_E$, **(*d*) Leader Storage Cost:** $\left\lceil \frac{n-2^d}{2^d} \right\rceil \times K + \left\lceil \frac{n-2^d}{2^d} \right\rceil \times (\left\lceil \frac{n-2^d}{2^d} \right\rceil + 1) \times K$ bits.

MODIFIED OCTOPUS WITH GDH.2 - (MO) and TGDH – (MOT):

The sponsor of TGDH for MOT, and the M_n member of GDH.2 for MO, becomes the subgroup leader. In fact, any member of MO, and MOT may become subgroup leader and represent its subgroup during STEP 2 and 3, because all members obtain the subgroup key at the end of STEP 1. The required operations to derive the group key are exactly the same for these members in MO and MOT. We assume that each subgroup contains $\left\lceil \frac{n}{2^d} \right\rceil$ members.

GDH.2 related Costs for the 1st Step of the Initial Phase of MO.

(*a*) Total message length for the 2^d leaders is: $2^{d-1} \times (\left\lceil \frac{n}{2^d} \right\rceil^2 + 3\left\lceil \frac{n}{2^d} \right\rceil - 4) \times K$ bits, **(*b*) MEs per leader**: $\left\lceil \frac{n}{2^d} \right\rceil$, with cost C_E, **(*c*) avg. MEs per member**: $(\left\lceil \frac{n}{2^{d+1}} \right\rceil + 1)$, with cost C_E.

TGDH related Costs for the 1st Step of the Initial Phase of MOT.

(***a***) **Initial Sponsor Bandwidth:** $(\sum_{i=1}^{h} \frac{\lceil n/2^d \rceil}{2^{i-1}}) \times K$ = $(2\lceil n/2^d \rceil\text{-}1) \times K$, (***b***) **Initial Member Bandwidth:** $h \times K$, (***c***) **Total Initial Bandwidth**: $(\lceil n/2^d \rceil \times (2+h)\text{-}1\text{-}h) \ \times K$, (***d***) **Initial Member/Sponsor Computation Cost:** Each member/sponsor does 2*h* MEs.

STEP 2: The subgroup leaders of (O), MO, and MOT execute Hypercube.

This step is identical for all three protocols (O), MO, and MOT. For the initial derivation of the group key the 2^d leaders perform Hypercube with initial values the blinded subgroup keys they have obtained from STEP 1. Hypercube requires *d* rounds, in each of which we have an exchange of 2^d messages ($2^{d\text{-}1}$ DH pairs/round). Each leader performs $2 \times d$ MEs in total. The combined message length is $2^d \times d \times K$ bits.

STEP3: Leaders of (O), MO, MOT, distribute the group key to their subgroup.

In the example for the simple (O) the authors have used *d* = 2. For *d* > 2, STEP 3 should be modified as follows: each of the 2^d subgroup leaders must communicate now *d* values to its group, in analogy to the two values derived for the case where *d*=2. For example if *d*=3, we have eight leaders noted as: *A, B, C, D, E, F, G, H*), and the following operations are executed:

Keys derived after 1st round: a^{AB}, a^{CD}, a^{EF}, a^{GH} .

Keys derived after 2nd round: $a^{a^{AB} a^{CD}}$, $a^{a^{EF} a^{GH}}$

Key derived after 3d round: $a^{a^{a^{AB} a^{CD}} a^{a^{EF} a^{GH}}}$ (final group key).

The subgroup leader *A* communicates the following *d* parts of the group key to its member *j*:

(O): $a^{a^{a^{EF} a^{GH}}}$, $a^{a^{CD}}$, a^{AB/K_j}, **(MO), (MOT):** $a^{a^{a^{EF} a^{GH}}}$, $a^{a^{CD}}$, a^{B}.

What differentiates (O) from MO and MOT, is the last value communicated to members by the subgroup leader. This value differs for every member in (O), but is the same for the members the same subgroup in MO or MOT. The rest (d-1) values are the same for all members in the subgroups of all three protocols. The leader broadcast to its members the d values and the members do d MEs to compute the final group key. Each member raises the value that corresponds to the 1st Hypercube round the to the power of its own secret and gets the first outcome, then raises the value that corresponds to the 2nd Hypercube round to the first outcome, and computes the second outcome etc. It finally raises the dth part of the key that corresponds to the dth Hypercube round to the (d-1)th outcome and gets the group key.

Example: Member j execute MOT and obtains subgroup key X at the end of STEP 1. At the end of STEP 3, it receives from A the following values: a^{B} (outcome of 1st HCube round), $a^{a^{CD}}$ (outcome of 2nd HCube round), $a^{a^{a^{EF}a^{GH}}}$ (outcome of 3d HCube round). Member j computes the group key through the following d MEs:

1: $(a^{B})^{X} = a^{BX}$, 2. $(a^{a^{CD}})^{a^{XB}} = a^{a^{CD}a^{XB}}$, 3. $(a^{a^{a^{EF}a^{GH}}})^{a^{a^{CD}a^{XB}}}$ = GK.

(*a*) Leader Bandwidth for MO: (d-1+$\left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil$)×$K$, **(*b*) Leader Bandwidth for MOT:** d×K, **(*c*) Member Computation Cost for MO and MOT:** d MEs with cost C_E.

2.5.3. Re-Keying Operations at Steady state for protocols: (O), MO, and MOT.

STEP 1: Each Subgroup Re-Keys within its own Subgroup

Each subgroup in (MO) or (MOT) handles member additions/evictions exactly as indicated by GDH.2 or TGDH protocols. The case of member additions/evictions has not been anticipated for (O), so we present our own algorithm for handling the membership changes at steady state. The key observation in the case of membership changes is to initially re-compute the subgroup key only for this subgroup in which the membership change has occurred. Hence, during STEP 1, only the subgroup(s) that have witnessed a membership change become active, initiate and execute STEP 1.

ORIGINAL OCTOPUS - (O):

Important Security Constraint: The re-keying algorithms of GDH.2 or TGDH preserve the properties of forward and backward secrecy. It is essential that the hybrid star protocol used in the subgroups of (O) provides re-keying algorithms that ensure these properties. We modify the original hybrid star scheme to meet the security requirements, in a similar manner as most contributory schemes handle membership changes: another member of the same subgroup modifies its secret share and establishes a new DH key with its leader. So, the leader in the case of join (evict) performs two (one) DHKEs. Then, the leader must compute the new group key. Let member *j* be the one that modifies its shared key from K_j to $K_j^{/}$ with the leader in either case (addition or eviction).

Addition: Let the new member *m* establish with the leader the shared key K_m. The old subgroup key was: $K = a^{K_1K_2...K_j...K_l}$, while the new subgroup key is: $K' = a^{K_1K_2...K'_j...K_lK_m}$. The partial keys the members will receive from their leader will contain the secret exponent $K_j^{/}$ and/or the secret exponent K_m. Hence, the new member won’t be able to derive the old key and backward secrecy will be preserved. Now, the leader must compute the new partial keys that will be communicated to its subgroup. Assuming that it has stored all partial keys that correspond to the previous subgroup key for each member individually, it is simple to compute the new partial keys. The subgroup leader computes K' from K as follows, referring to cases (*a*) and (*b*) distinguished during the analysis of the initial key establishment operation:

Case (*a*) (non-modular multiplications): We multiply the previous exponent by: $K_j^{-1}K'_jK_m$,

Case (*b*) (modular multiplications): We do three successive MEs as follows: $((K^{K_j^{-1}})^{K'_j})^{K_m}$.

The partial keys of members are computed in a similar fashion, i.e. for member *j*, the previous partial key stored is raised to K_m, and for member m the previous subgroup key stored is raised to $K_j^{-1}K'_j$.

We now calculate the total computation cost incurred at the leader when either of the two versions (*a*), (*b*) is used. Let $x=\left\lceil \frac{n-2^d}{2^d} \right\rceil$, for ease of the notation.

Version (*a*):

1) (x-1) multiplications of an element of size (x-1)K (the previous partial keys of members already stored), with an element of size 3K (with processing cost of approximately 2K^2),

2) 1 multiplication of an element of size xK (the previous subgroup key already stored) with an element of size 3K (with processing cost of approximately 2K^2),

3) 1 multiplication of an element of size (x-1)K (the previous partial key of member j already stored) with an element of size K, and

4) 1 multiplication of an element of size xK (the partial key that corresponds to the new member m) with an element of size 2K, (with processing cost of approximately K^2).

In total, the **processing cost** incurred at the leader becomes: $(x-1)[(x-1)3K^2+2K^2]+[3xK^2+2K^2]+[(x-1)K^2]+[2xK^2+K^2]=(3x^2+2x+5)K^2=(3(\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2+2\left\lceil \frac{n-2^d}{2^d} \right\rceil+5)K^2$.

Subsequently, the total **storage cost** for the leader becomes: $(x-1)[(x-1)K+3K]+[xK+3K]+[(x-1)K+K]+[xK+2K]=(x^2+4x+3)K=((\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2+4\left\lceil \frac{n-2^d}{2^d} \right\rceil+3)K$.

Version (*b*):

1) 3 MEs for (x-1) elements (the previous partial keys of members already stored),

2) 3 MEs for 1 element (the previous subgroup key already stored),

3) 1 ME for 1 element (the previous partial key of member j already stored)

4) 2 MEs for 1 element (the partial key that corresponds to the new member m).

The **processing cost** incurred at a leader becomes: $(3(x-1)+3+1+2)C_E = 3(x+1)C_E = 3\left\lceil \frac{n}{2^d} \right\rceil C_E$.

Subsequently, the total **storage cost** for the leader becomes: $(x+2)K = (\left\lceil \frac{n}{2^d} \right\rceil+1)K$.

For the case of member addition at STEP 1, we summarize the most significant costs:

(*a*) **Leader Bandwidth**: 2K, (***b***) **Member Bandwidth:** K (for two members), 0 for the rest, (***c***) **Leader Computation:** $(3(\lceil {}^{n-2^d}\!/_{2^d} \rceil)^2+2\lceil {}^{n-2^d}\!/_{2^d} \rceil+5)K^2+2C_E$ (version *a*), $(3\lceil {}^{n}\!/_{2^d} \rceil+2)C_E$ (version *b*), (***d***) **Member Computation:** $C_{rr}+2C_E$ (two members only), 0 for the rest, (***e***) **Leader Storage:** $((\lceil {}^{n-2^d}\!/_{2^d} \rceil)^2 +4\lceil {}^{n-2^d}\!/_{2^d} \rceil+3)K$ (version *a*), $(\lceil {}^{n}\!/_{2^d} \rceil+1)K$ (version *b*).

Deletion: Let member *l* be the evicted one.

The old subgroup key was: $K= a^{K_1K_2...K_j...K_tK_l}$, while the new subgroup key is: $K^{'} = a^{K_1K_2...K_j^{'}...K_m}$. The subgroup members will receive new partial keys from their leader that contain the secret exponent $K_j^{/}$. Hence, the evicted member won't be able to derive the new group key, and the forward secrecy will be preserved. Now, the subgroup leader must compute the new partial keys that will be communicated to its subgroup. Assuming that it has stored all partial keys that correspond to the previous subgroup key for each member, it is simple to compute the new partial keys. The subgroup leader computes $K^{'}$ from K as follows:

(*a*) It multiplies the previously stored exponent with $K_j^{-1}K_j^{'}$,

(*b*) It performs two MEs as follows: $(K^{K_j^{-1}})^{K_j^{'}}$.

For the computation of the partial keys of all members $x \neq j$, *l* the process is similar to this described for the subgroup key. For member *j*, no additional partial key needs to be sent.We calculate the total computation incurred at the leader when any of the two versions is used.

Version (*a*):

1) (x-1) multiplications of an element of size (x-1)K (the previous partial keys of members already stored), with an element of size 2K (with processing cost of approximately K^2),

2) 1 multiplication of an element of size xK (the previous subgroup key already stored) with an element of size 2K (with processing cost of approximately K^2).

In total, the **processing cost** incurred at the leader becomes: $(x\text{-}1)[(x\text{-}1)2K^2+K^2]+[2xK^2+K^2]$ = $(2x^2\text{-}x+2)K^2 = (2(\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2 - \left\lceil \frac{n-2^d}{2^d} \right\rceil+2)K^2$.

Subsequently, the total **storage cost** for the leader becomes: $(x\text{-}1)[(x\text{-}1)K+2K] + [xK+2K]$ = $(x^2+x+1)K = ((\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2 + \left\lceil \frac{n-2^d}{2^d} \right\rceil+1)K$.

Version (*b*):

1) 2 MEs for (x-1) elements (the previous partial keys of members already stored),

2) 2 MEs for 1 element (the previous subgroup key already stored).

The **processing cost** incurred at the leader becomes: $(2(x\text{-}1)+2)C_E = 2xC_E = 2\left\lceil \frac{n-2^d}{2^d} \right\rceil C_E$.

Subsequently, the total **storage cost** for the leader becomes: $xK = \left\lceil \frac{n-2^d}{2^d} \right\rceil K$.

For member deletion case at STEP 1, we summarize the most significant costs incurred:

(*a*) Leader Bandwidth: K, **(*b*) Member Bandwidth:** K (for one member only), 0 for the rest, **(*c*) Leader Computation:** $(2(\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2\text{-}\left\lceil \frac{n-2^d}{2^d} \right\rceil+2)K^2 + C_E$ (version *a*), $(2\left\lceil \frac{n-2^d}{2^d} \right\rceil+1)C_E$ (version *b*), **(*d*) Member Computation:** $C_{rr}+2\mathrm{C}_\mathrm{E}$ (one member only), 0 for the rest, **(*e*) Leader Storage:** $((\left\lceil \frac{n-2^d}{2^d} \right\rceil)^2 + \left\lceil \frac{n-2^d}{2^d} \right\rceil+1)K$ (version *a*), $\left\lceil \frac{n-2^d}{2^d} \right\rceil K$ (version *b*).

Modified Octopus (MO), (MOT):

The subgroup members handle membership changes exactly as indicated by GDH.2 or TGDH re-keying algorithms respectively. We summarize the main results for member addition/deletion in Table 2.10, and we omit the computations here to avoid redundancies.

STEP 2: All subgroup leaders execute Hypercube (Re-keying version)

We execute a “reduced Hypercube version”, to propagate to the whole group the membership changes that occurred in one or more subgroups. However, the subgroups that have witnessed no membership changes during this period, may participate with their old subgroup keys.

Security Aspect: The properties of backward and forward secrecy are not violated. This is so, because once the subgroup key of the subgroup that has witnessed a membership change is altered in a way that the backward and forward secrecy are preserved, the new group key established via Hypercube is such, that the evicted (new) member would require the new (old) subgroup key to reconstruct the new (old) group key. For example, assume that Octopus consists of four leaders, denoted as *A*, *B*, *C*, *D*, with subgroup keys *KA*, *KB*, *KC*, and *KD* respectively. These leaders execute Hypercube with *d*=2 at STEP 2, and produce the following group key $K = a^{a^{(KA)(KB)} a^{(KC)(KD)}}$. Assume now that a membership change occurs in the subgroup of leader *A*, and its current members compute a new subgroup key: *KX*. If Hypercube is executed minimally, that is, using the previous shares of the subgroups that witness no membership changes, the new group key becomes: $K^{'} = a^{a^{(KX)(KB)} a^{(KC)(KD)}}$. The subgroup members of leader *A*, use the previously stored intermediate shares received during the previous execution of step 3, $(a^{KB}, a^{a^{(KC)(KD)}})$ and combine them with their newly computed subgroup key *KX* to compute $K^{'}$. It is obvious that a member that knows *K* cannot compute $K^{'}$ without knowledge of *KX* (forward secrecy), and a member that knows $K^{'}$ cannot compute *K* without knowledge of *KA* (backward secrecy). Hence, subgroups that do not experience membership changes can participate to subsequent executions of Hypercube with the same subgroup keys, without affecting the security level of the overall scheme.

Performance Characteristics: Not all calculations need to be done anew during the Hypercube execution after re-keying. This is so, since at any round, those subgroup leaders for which the BKs to be communicated to their peers have not been modified during STEP 1 or during the previous rounds, are already stored in their peers and need not be communicated to them again. During any given round, we distinguish the following cases for a leade:

(*a*) it needs not send its BK to its peer and it does not receive an updated BK from it either. They both do 0 MEs and will use for the next round the stored BK of the previous execution.

(*b*) it needs not send its BK to its peer, but receives an updated BK from it. They both do 2 MEs, 1e to compute the new secret key, and 1 to compute the updated BK for the next round.

(*c*) it sends its BK to its peer, but does not receive an updated BK from it. They both do 2 MEs, 1 to compute the new secret key, and 1 to compute the updated BK for the next round.

(*d*) it sends its BK to its peer, and receives an updated BK from its peer. They both do 2 MEs, 1 to compute the new secret key, and 1 to compute the updated BK for the next round.

In this case it can be seen that 2^{i-1} messages are transmitted per round: in the 1st round one leader with modified BK sends an update to its peer, in the 2nd round 2 leaders with modified content will send updates to its peers, and in general, the leaders with modified content are doubled in every round, and so are the updated messages. Each subgroup leader that either sends or receives an updated BK performs 2 MEs: 1 to compute the updated secret key of the subsequent round, and 1 to blind that secret key. In the 1st round, 2 members are active (senders or recipients of updated values), in the 2nd round, 4 members are active, in the *i*th round 2^i members are active, and perform 2^{i+1} MEs, and so on and so forth.

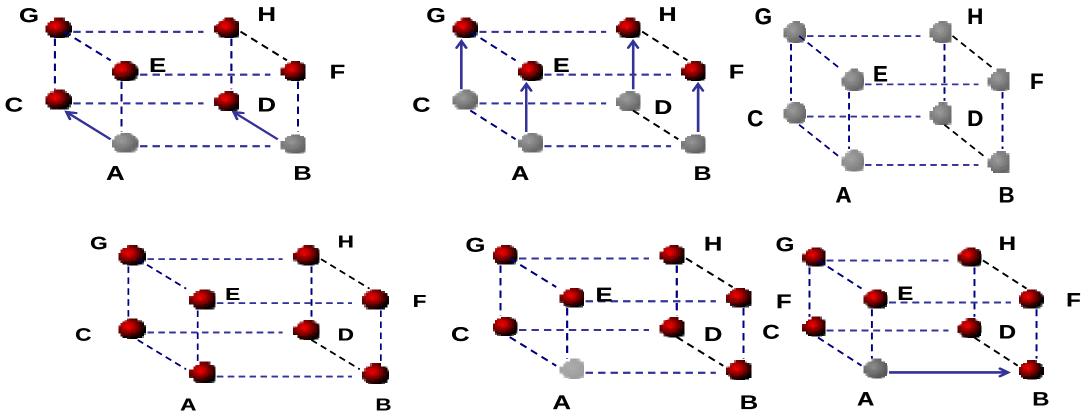


Figure 2.15. (a) Previous HCube, (b) A changes partial key (PK), (c) Rnd1: A sends PK to B, (d) Rnd2: Only A, B send PKs to C, D, (e) Rnd3: only A, B, C, D send PKs to E, F, G, H, (f) End of re-keying, group key created only with: 7 messages vs. 24, 14 MEs vs. 48.

Total Communication Cost: $\sum_{i=1}^{d} 2^{i-1}$ = (2^d-1), as opposed to $d \times 2^d$ in the original Hypercube.

Total MEs Cost: $\sum_{i=1}^{d} 2^{i+1}$ = 4(2^d-1), as opposed to $d \times 2^{d+1}$ in the original Hypercube.

It is obvious that Hypercube is re-run with the minimum cost computed above, only if membership changes occur: (*a*) in one subgroup only, or (*b*) in two subgroups that happen to be peers during the 1st Hypercube round. If membership changes occur within multiple subgroups, then the resulting communication and computation cost to run Hypercube is higher, but also depends on the relative logical placement of these subgroups (i.e. in which Hypercube round they become peers). After two subgroups become peers in Hypercube, they are combined, as if the one peer is absorbed to the other, and they propagate the updated blinded information, as if the one peer was just activated by the other. Before that however, each subgroup updates its peers individually, resulting in a more expensive execution. Referring to the previous example, membership changes in the subgroups of *A*, *B*, *G* result in the following communication exchanges:

Rnd1: *A*<->*B*, *G*->*H* (3), **Rnd2**: *A*->*C*, *B*->*D*, G->*E*, *H*->*F* (4), **Rnd3:** all subgroups participate (8). Hence, subgroups of *A*, *B*, *G* results in a total of 15 messages.

Membership changes in the subgroups of *A*, *C*, *G* result in the following exchanges:

Rnd1: *A*->*B*, *C*->*D*, *G*->*H* (3), **Rnd2**: *A*<->*C*, *B*<->*D*, G->*E*, *H*->*F* (6), **Rnd3:** all subgroups participate (8). Hence, subgroups of *A*, *B*, *G* results in a total of 17 messages.

We will now compute the overall cost required for running Hypercube after re-keying, when re-keying occurs within *X* subgroups. In any given Hypercube round *i*, we denote as $T_i(Z)$ the subset of peers that correspond to a given subset *Z*, where T_i is the mapping function of a node to its peer, according to the Hypercube schedule in round *i*. We start with the subset *S*=*X*, and after each Hypercube round *j* this subset is expanded as follows: $S \rightarrow S \cup T_j(\mathrm{S})$. In other words, after round *j*, the subset *S* will expand to include those peers of *S* during round *j* that do not already belong to *S*. The cardinality of *S* before each round *j*, represents the number of messages that will be communicated during round *j*, the overall number of messages communicated can be computed by adding the cardinality of *S* for every round. We denote the instance of S before any given round j as S_j. Then, we compute the **total communication cost** as: $\sum_{i=1}^{d} | S_i |$, and the **total number of MEs** as: $2\sum_{i=1}^{d} | S_i |$.

If before the start of round $(k+1)<d$, S includes all Hypercube parties 2^d, then the remaining communication cost until round d is this of the original execution, that is: $(d\text{-}k)\times 2^d$. To provide an upper limit for the above formulae, we assume that the subset S is doubled in every round (the pairings are not taken into account), until round k, after which $|S| = 2^d$. That occurs if $2^k \times X \geq 2^d$, while $k<d$, or if: $k \geq d\text{-}\log_2 X$.

Then, $\sum_{i=1}^{d} | S_i | \leq \sum_{i=1}^{k} 2^{i-1} X + (d\text{-}k)\times 2^d = (2^k\text{-}1)X+(d\text{-}k)\times 2^d, \quad d\text{-}\log_2 X \leq k \leq d,$

without considering the pairings, that would reduce the resulting communication cost even further. **The reduction in the communication cost** reduction (from original Hypercube) is:

$d\times 2^d - (2^k\text{-}1)\times X - (d\text{-}k)\times 2^d = k\times 2^d - (2^k\text{-}1)X = 2^k\times(2^{d\text{-}k}\times k\text{-}X) + X, \; 0 \leq k<d.$

Indeed, for k=0, the reduction in the communication cost is 0, as expected. If we take the pairings into account, then we can provide a more stringent upper limit for the latter formulae. Assume that round 1 includes $y_1 \leq min\{|S_1|=X, 2^{d\text{-}1}\}$ peers (that is $y_1/2$ pairs), round 2 includes $y_2 \leq min\{|S_2|, 2^{d\text{-}1}\}$ peers, …, round $(d\text{-}1)$ includes $y_{d\text{-}1} \leq min\{|S_{d\text{-}1}|, 2^{d\text{-}1}\}$ peers, etc.

Round 1: X messages are produced from the X leaders. Since there are $y_1/2$ pairs in this round, expansion of one of the two peers for the subsequent rounds will include the other peer. For the next round we need to consider the expansion of $X_1 = (X\text{-}\ y_1/2)$ leaders, to avoid the consideration of duplicate messages. The expansion includes $2\times X_1 = (2\times X - y_1)$ leaders.

Round 2: The $2X_1$ leaders send messages to their peers of the current round (hence $2X_1$ messages), and the expansion includes $2\times(2\times X_1 - y_2/2) = (4\times X_1 - y_2) = 4\times X\text{-}2\times y_1\text{-}y_2$.

Round *i*: The $2X_{i-1}$ leaders send messages to their peers of the current round ($2X_{i-1}$ messages), and the expansion includes $2\times(2\times X_{i-1} - y_i/2) = (4\times X_{i-1} - y_i) = 2^i\times X - 2^{i-1}\times y_1 - 2^{i-2}\times y_2 - \ldots - y_i$. Working similarly, after the end of round d, **the number of messages sent** becomes:

$$\sum_{i=1}^{d} | S_i | \le (2^l-1)\times X + (d-l)\times 2^d - (2^{l-1}-1)\times y_1 - (2^{l-2}-1)\times y_2 - \ldots - y_{l-1}, \quad (d\text{-}\log_2 X) \le k \le l < d.$$

The **reduction in the communication cost** (as opposed to the original Hypercube) is:

$d\times 2^d - ((2^l-1)\times X - (2^{l-1}-1)\times y_1 - (2^{l-2}-1)\times y_2 - \ldots - y_{l-1} + (d-l)\times 2^d) = 2^l\times(2^{d-l}\times l - X) + X + (2^{l-1}-1)\times y_1 + (2^{l-2}-1)\times y_2 + \ldots + y_{l-1} \;\ge\; 2^l\times(2^{d-l}\times l - X) + X + (2^{l-1}-1)\times y_1$.

From the latter inequalities it is obvious that if pairings are considered, the reduction in the communication cost becomes even more significant. Hence, we have **provided a formal algorithm that computes the associated costs for executing Hypercube after re-keying**, for a general configuration, and we have shown how significant the overhead reduction due to our algorithm is, as opposed to executing the original protocol from scratch. Of course, the more the leaders that compute a new group key, the less the overhead reduction, which is expected. If (2^d-1) or 2^d leaders re-key, our algorithm converges to the original Hypercube.

STEP 3: Subgroup leaders distribute the new group key to their members.

Now, all subgroup leaders possess the same group key, and they are going to distribute it to their subgroup members at STEP 3. For $d>2$, STEP 3 is modified as follows: each of the 2^d leaders communicates now to its subgroup only those partial BKs that have changed since the previous Hypercube execution. Hence, a leader may communicate 1 to d partial intermediate keys, as opposed to the d values communicated in the the initial group key establishment.

Example with Re-Keying in 1 Subgroup Leader: For the case of $d=3$, we denote the eight leaders as: *A, B, C, D, E, F, G, H*. In this example we illustrate the operations performed and the values communicated during a **re-keying operation for STEPS 2 and 3**, when leader *A* witnesses membership changes and modifies its subgroup key from *KA* to *X*.

After round 1 the BKs derived and/or stored are: a^{BX}, a^{CD}, a^{EF}, a^{GH} .

After round 2 the BKs derived and/or stored are: $a^{a^{XB}a^{CD}}$, $a^{a^{XB}a^{CD}}$, $a^{a^{EF}a^{GH}}$, $a^{a^{EF}a^{GH}}$.

After round 3 the new group key derived is: $a^{a^{a^{XB}a^{CD}}a^{a^{EF}a^{GH}}}$ for all four pairs.

Subgroup Leader *A*: After the end of the 3 rounds *A* has stored the following values:

1: *X*, 2: a^{B}, 3: $a^{a^{CD}}$, 4: $a^{a^{a^{EF}a^{GH}}}$, 5: $a^{a^{a^{XB}a^{CD}}a^{a^{EF}a^{GH}}}$.

The intermediate values 2, 3, 4, remain unchanged from the previous time. The members of subgroup *A* will also use the same intermediate values they have stored from the previous time. We now show exactly where protocol (O) differentiates from MO, and MOT:

(O): Leader *A* needs only communicate the following part of the group key, denoted as 1', (in the place of values 1,2) to each member *j*: a^{XB/K_j} . Each member raises its secret key (K_j) to (1'), and uses the outcome as the exponent to which (3) will be raised, the outcome will be used as the exponent to which value (4) will be raised, and the group key (5) will be generated. Hence, each member does 2 MEs in this example, or *d* in the case of a *d*-Cube.

Communication Cost for Leader: $\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil \times K$, **MEs for Members:** *d*, with cost C_E each.

(MO), (MOT): Now, the members already have all the required intermediate values as well as the subgroup key *X*. Hence, leader *A* needs to communicate nothing to its subgroup at STEP 3. Each member will use (1) as the exponent to which (2) will be raised, the outcome will be used as the exponent to which value (3) will be raised, the outcome will be used as the exponent to which value (4) will be raised, and the group key (5) will be generated. Hence, each member performs 3 MEs in this example, or *d* in the case of a *d*-Cube.

Total Communication Cost for Leader or Members: 0.

Total MEs for Members: *d*, with cost C_E each.

Subgroup Leader *B*: After the end of the 3 rounds *B* has stored the following values:

1: β, 2: a^{X}, 3: $a^{a^{CD}}$, 4: $a^{a^{a^{EF}a^{GH}}}$, 5: $a^{a^{a^{XB}a^{CD}}a^{a^{EF}a^{GH}}}$.

The intermediate values (1), (3), (4), stored remain the same but for one: (2): a^{X} . Leader *B* will only need to communicate to its members messages related to this value, depending on which of the three protocols it executes. The members of subgroup *B* will also use some of the intermediate values they have stored from the previous time. We now show exactly where protocol (O) differentiates from MO, and MOT:

(O): Leader *B* needs only communicate the following part of the group key, denoted as 1', (in the place of values 1,2) to each member *j*: a^{XB/K_j} . Each member will raise its secret key (K_j) to (1'), and use the outcome as the exponent to which (3) will be raised, the outcome will be used as the exponent to which value (4) will be raised, and the group key (5) will be generated. Hence, each member does 2 MEs in this example, or *d* in the case of a *d*-Cube.

Total Communication for Leader: $\left\lceil {n-2^d}/{2^d} \right\rceil \times K$. **Total MEs for Members:** *d*, with cost C_E.

(MO), (MOT): Members already possess all the required intermediate values but one: (2): a^{X} . Members also possess the subgroup key β . Hence, leader *B* needs to communicate (2) to its subgroup at step 3. Each member will use (1) as the exponent to which (2) will be raised, the outcome will be used as the exponent to which (3) will be raised, the outcome will be used as the exponent to which (4) will be raised, and the group key (5) will be generated. Hence, a member needs to perform 3 MEs in this example, or *d* MEs in the case of a *d*-Cube.

Total Communication for Leader-Members: *K*, **Total MEs for Members:** *d*, with cost C_E.

Subgroup Leader *F*: After the three Hypercube rounds *F* has stored the following values:

(1): *f*, (2): α^{E} , (3): $a^{a^{GH}}$, (4): $a^{a^{a^{EF}a^{GH}}}$, (5): $a^{a^{a^{XB}a^{CD}}}$, (6): $a^{a^{a^{XB}a^{CD}}a^{a^{EF}a^{GH}}}$.

Observe that all intermediate values stored remain the same but for one: (5): $a^{a^{a^{XB}a^{CD}}}$. This is the only value that the leader needs to communicate to the members of its subgroup after the derivation of the group key. Then, these members will be able to reconstruct the group key themselves by performing one ME. In this case, (O), MO, and MOT do not differentiate:

(O), MO, MOT: Leader *F* needs only communicate the blinded intermediate value (5): $a^{a^{a^{XB_{a}CD}}}$ to all its members. Each member knows the secret value that corresponds to the intermediate BK (4), and will use that secret value as the exponent to which value (5) will be raised, and the group key (6) will be generated. Hence, a member does 1 ME in this example.

Total Communication for Leader: *K*, **Total MEs for Members:** 1, with cost C_E each.

Example Generalization: It can be seen that not all members do *d* MEs as in the initial case: members of 2^{d-1} leaders (i.e. *E*, *F*, *G*, *H* in this example) do only 1 ME, members of 2^{d-2} leaders (i.e. *C*, *D* in this example) do 2 MEs, …, members of 2^{d-x} leaders do *x* MEs, and so on and so forth. Members of the leader that witness membership changes and its 1st Hypercube round peer (i.e. *A*, *B*) do *d* MEs to reconstruct the new group key. In average, members of each participant do 2 MEs to reconstruct the new group key in the case of addition/deletion.

The leader in the subgroup of which a membership change occurred needs to communicate to its members no parts for MO or MOT, and one part only for (O). All the rest of the leaders need to communicate one intermediate BK as well for any of the three protocols (O), MO, MOT. So, the leader communicates only one value to the whole subgroup, the value each member requires to reconstruct the group key, which is the same for every member in the subgroup. We denote the number of members in each subgroup as $x = \left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil$.

Total # MEs in (O), MO, MOT for all members: $x\sum_{i=1}^{d}2^{i-1}(d+1-i)$=$x[(2^d\text{-}1)\times(d+1)\text{-}\frac{1}{2}\sum_{i=1}^{d}2^i\times i\,]$

$$=x[(2^d\text{-}1)\times(d+1)\text{-}\tfrac{1}{2}\sum_{i=1}^{d}(\sum_{i}^{d}2^i)\,]=x[(2^d\text{-}1)\times(d+1)\text{-}\tfrac{1}{2}\sum_{i=1}^{d}(2^i(2^{d+1-i}-1))=(2^d\text{-}1)\times(d+1)\text{-}\tfrac{1}{2}\sum_{i=1}^{d}(2^{d+1}-2^i)] \quad =$$

$x[(2^d\text{-}1)\times(d+1)\text{-}2^d\times d+\frac{1}{2}\sum_{i=1}^{d}2^i\ ]=x[(2^d\text{-}d\text{-}1)+(2^d\text{-}1)]=x(2^{d+1}\text{-}d\text{-}2)=\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil(2^{d+1}\text{-}d\text{-}2)=(n\text{-}2^d)\times 2\text{-}\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil$ $\times(d\text{-}2)$, with cost C_E each, as opposed to $(n\text{-}2^d)\times d$ for the initial establishment.

Note: $\frac{1}{2}\sum_{i=1}^{d}2^i\times i = 2^d\times d\text{ - }(2^d - 1) = 2^d\times(d\text{-}1) + 1$.

Total Communication Cost in (O) for all subgroup leaders: $(2\times\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil + (2^d\text{-}2))\times K$, (as opposed to $[(n\text{-}2^d) + (2^d\times(d\text{-}1)]\times K$ for the initial case).

Total Communication Cost in MO, MOT for all subgroup leaders: $(2^d\text{-}1)\times K$, (as opposed to $(2^d\times d)\times K$ for the initial case).

Multiple Subgroup Leaders Re-Keying Case: Assume now that X subgroup leaders witness membership changes within their subgroup and modify their subgroup key. From the subset of the X modified subgroup keys we want to compute how many and which intermediate BKs stored by the leaders at the end of Hypercube will be modified as well. Referring to the previous example, if the subgroup keys of leaders C, F are modified, the intermediate BKs that will be affected, stored at a subset of the leaders, will be those that contain any of the modified keys in any rounds: c (stored at leader C), f (stored at leader F), a^C (stored at leader D), a^F (stored at leader E), $a^{a^{CD}}$ (stored at leaders A, B), $a^{a^{EF}}$ (stored at leaders G, H), $a^{a^{a^{AB}a^{CD}}}$ (stored at leaders E, F, G, H), $a^{a^{a^{EF}a^{GH}}}$ (stored at leaders A, B, C, D).

Notation 1: In any given round i, we denote as T_i a function that maps an element generated at round i to its peer element, according to Hypercube schedule in round i. Then, $X_{i+1} = X_i \cup T_i(X_i)$, where X_{i+1} is the element generated for the subsequent round (i+1). Accordingly, $S_{i+1}=S_i\cup T_i(S_i)$, where S_{i+1} is the subset of elements in round (i+1), produced from the subset of elements S_i in round i and their peers of round i, $T_i(S_i)$.

Notation 2: For round i, we denote the peer element $T_i(X_i)$ as "repeated" *iff* $T_i(X_i)\in S_i$. We denote the number of "repeated" pairs of elements in round i as y_i. Then, $y_i<|S_i|$, $|S_i| = |T_i(S_i)|$.

The modification of X subgroup keys (i.e. x_1, x_2, …, x_x) that corresponds to 2^0 subgroup leaders each, results in:

(1) The modification of X BKs received after the end of the 1st Hypercube round (i.e. a^{x_1}, a^{x_2} ,…., a^{x_x}), corresponding to 2^0 subgroup leader each.

(2) The modification of X_1 BKs received after the end of the 2nd Hypercube round that contain either 1 or 2 keys from $\{X\}$, i.e. $a^{a^{X_k Y_t}}$, $a^{a^{X_k X_t}}$, etc., without repeating of course the same value twice: i.e. $a^{a^{X_t X_k}}$ and $a^{a^{X_k X_t}}$, which occurs only when we have "repeated" pairs of elements. In this case, if we have y_1 repeated pairs, we need to remove y_1 elements to avoid duplicates. Hence, $X_1 = X- y_1$, $X_1 \leq X$. Each BK in X_1 is stored in 2^1 leaders, and of course corresponds to 2^1 leaders.

(3) The modification of X_2 BKs received after the end of the 3d round that contain either 1 or 2 elements from $\{X_1\}$, so that $X_2 = X_1 - y_2$, $X_2 \leq X_1$, using the same reasoning as before. Each BK in X_2 is stored in 2^2 leaders, and of course corresponds to 2^2 leaders.

(4) The modification of X_{i-1} BKs received after the end of the ith round that contain either 1 or 2 elements from $\{X_{i-2}\}$, so that $X_{i-1} = X_{i-2} - y_{i-1}$, $X_{i-1} \leq X_{i-2}$, using the same reasoning as before. Each BK in X_{i-1} is stored in 2^{i-1} leaders, and corresponds to 2^{i-1} leaders, and so on and so forth.

(5) The modification of X_{d-1} BKs received after the end of the dth round that contain either 1 or 2 elements from $\{X_{d-2}\}$, so that $X_{d-1} = X_{d-2} - y_{d-1}$, $X_{d-1} \leq X_{d-2}$, using the same reasoning as before. Each BK in X_{d-1} is stored in 2^{d-1} leaders, and corresponds to 2^{d-1} leaders.

The total number of BKs that will change for all subgroup leaders (TB) becomes:

$$TB = X + \sum_{i=2}^{d} 2^{i-1} \times X_{i-1} = X + \sum_{i=2}^{d} 2^{i-1} \times (X - \sum_{j=1}^{i-1} y_j) = \sum_{i=1}^{d} 2^{i-1} \times X \text{ - } \sum_{i=2}^{d} 2^{i-1} \times \sum_{j=1}^{i-1} y_j \; =$$

$$= (2^d\text{-}1)X\text{-} \sum_{j=1}^{d-1} y_j (\sum_{i=j}^{d-1} 2^i) = (2^d\text{-}1)X\text{-} \sum_{j=1}^{d-1} y_j (2^d - 2^j) < X\,(2^d\text{-}1)$$

Since values: X, y_1, y_2, …, y_{d-1} can all be pre-computed, TB can be determined precisely.

Following the same reasoning as in the previous example that deals with the modification in a single subgroup key, in terms of the total communication cost, protocol (O) differentiates from MO and MOT as follows:

Communication in (O) for all leaders: $(X\left\lceil \frac{n-2^d}{2^d} \right\rceil+(TB\text{-}X))\times K$ = $(X\left\lceil \frac{n-2^{d+1}}{2^d} \right\rceil+TB)\times K$, as opposed to $[(n\text{-}2^d) + (2^d\times(d\text{-}1)]\times K$ (initial case).

Communication in MO, MOT for all leaders: $TB\times K$, as opposed to $2^d\times d\times K$ (initial case).

Now, we compute the total number of MEs done by all members in order to compute the group key at STEP 3, in the event that X subgroup leaders have modified their subgroup keys. Regarding the previous example that deals with the modification in a single subgroup key with d=3, we repeat the BKs stored at an arbitrary subgroup leader, for example F. After the three Hypercube rounds F has stored the following blinded values:

(0): f, (1): α^E, (2): $a^{a^{GH}}$, (3): $a^{a^{a^{XB}a^{CD}}}$, (4): $a^{a^{a^{a^{XB}a^{CD}}a^{a^{EF}a^{GH}}}}$ (group key).

As generalized earlier, if values (1), or (2), or (3) change, the leader must do 3, or 2, or 1 MEs respectively to reconstruct the group key. In addition to Notation 1, 2, we also use Notation 3.

Notation 3: By S we denote the initial set of elements that correspond to the X updated subgroup keys.

As discussed in the example illustrated, a member for which its subgroup key of is modified will perform d MEs to reconstruct the group key from the intermediate BKs either received or stored. In fact, the number of the ME for these members does not change even if they also receive updated intermediate BKs. From the previous example as well, we can claim the following **invariant** for our scheme: **Invariant:** *The lowest intermediate value that is altered is the one that determines the number of MEs performed by a member.*

We will now compute the total number of MEs performed by the rest of the members for which one or more intermediate BKs (1), (2), …, (d) are updated. The members for which (1) is updated will do d MEs, members for which (2) is the first value to be updated will do (d-1) MEs, etc. Hence, we want to compute how many members belong to each of the discussed categories, given that: (*a*) X subgroups modify their subgroup keys, (*b*) the schedule at every round i is known (T_i), and so is the subset of per round "repeated" elements $\{y_i\}$.

First intermediate updated value (1): We denote the set of elements that correspond to members for which the subgroup key does not change as $S_0 = \{X_0\}$, and the set of elements that correspond to the updated values (1) as $S_1 = \{X_1\}$. This case corresponds to a number -N_1- of subgroups for which the subgroup key does not change (the leaders of these elements - X_0 - do not belong to S) but they receive peer value (1) from leaders that belong to S. We want to determine the number N_1: We know that: $\{X_1\} \in X$, $\{X_0\} \notin X$, and $T_1(\{X_1\}) \notin X$. However, $\exists$ y_1 pairs of elements in S s.t. $T_1(\{X\}) \in \{X\}$, or $|T_1(\{X\}) \cap \{X\}| = y_1(X) \neq \varnothing$. Hence, since $\{X_1\} \in X$ and $T_1(\{X_1\}) \notin X$, then: $N_1 = X\text{-}2y_1(X)$. (y_1 is multiplied by 2 because each "repeated" pair (A_1, A_2) implies that 2 "repeated" elements: A_1, and A_2 should be excluded from $\{X\}$). The corresponding number of members that belong to this category is in effect xN_1.

First intermediate updated value (2): We denote the set of elements that correspond to members (or subgroups) for which the subgroup key does not change as $S_0 = \{X_0\}$, the set of elements that correspond to the stored values (1) as $S_1 = \{X_1\}$, and the set of elements that correspond to the updated values (2) as $S_2 = \{X_2\}$, where $X_2 = (A_1 \cup A_0)$, and $(A_1 \cup A_0) \in \{X\}$, (either $A_1 \in \{X\}$, or $A_0 \in \{X\}$, or both A_1, $A_0 \in \{X\}$). This case corresponds to a number of subgroups for which values (0) and (1) remain the same (the leaders of these elements that correspond to (0) and (1) do not belong to S) but they receive updated intermediate BKs (2). We want to determine the number of such elements N_2: We know that: S_2 includes$\{X\}$, but S_0, $S_1 \notin \{X\}$. From construction, observe that: $\forall X_2 \in S_2$, $\exists$ $X_1 \in S_1$ and $X_0 \in S_0$, s.t. $T_2(X_2) = (X_1 \cup X_0)$. For example, in the case illustrated for the members of subgroup leader F when d=3, elements G and H s.t. $(G \cup H) \in S_2$ compose value (2): $a^{a^{GH}}$. However T_2 $(G \cup H) = (E$

$\cup F$). As shown before: (0): f, and (1): α^{E}. In other words, $F \in S_0$, and $E \in S_1$. We are looking to find elements $X_2 \in S_2 \in S$ s.t. $T_2(X_2) \notin S_2$. However, $\exists$ y_2 pairs of elements in S_2 s.t. $T_2(\{X\}) \in S_2$, or $|T_2(\{X\}) \cap \{X\}| = y_2$ $(X) \neq \varnothing$. Hence, since $\{X_2\} \in X$ and T_2 $(\{X_2\}) \notin X$, $2y_2$ elements of S_2 must be removed (for example each repeated pair ($X_{1k} \cup A_k$, $X_{1m} \cup A_m$) is included 2 times in S_2: once associated with element X_{1k}, and once associated with element X_{1m}). Furthermore, since we are trying to identify the distinct elements of S_2, an additional number of y_1 elements must be removed from S. As discussed in the previous case, each of the y_1 elements contains a pair of the type (X_{1k}, X_{0m}). Even if $\exists$ t s.t. $X_{2t} = (X_{1k} \cup X_{0m}) \in S_2$, it is repeated two times, once associated with element X_{1k}, and once associated with element X_{0m}.For our intermediate computation, we only need to include this element once. Hence, the distinct values that belong to this category are: X-y_1-$2y_2$. However, each such value in S_2 consists of 2 unary elements, i.e. $X_2 = (X_1 \cup X_0)$. From Hypercube symmetry, each such value is assigned to two subgroups, since it will be received from two members. In conclusion, $N_2 = 2(X\text{-}y_1\text{-}2y_2)$. The number of members that belong to this category is now: xN_2.

First intermediate updated value (3): Working similarly, we compute: $N_3 = 2^2(X\text{-}y_1\text{-}y_2\text{-}2y_3)$. This formula is extracted using exactly the same reasoning as in the previous two cases.

First intermediate updated value (*j*): We compute: $N_j = 2^{j\text{-}1}(X\text{-}y_1\text{-}y_2\text{-}\ldots\text{-}\ y_{j\text{-}1}\ \text{-}2y_j)$.

First intermediate updated value (*d*): We compute: $N_d = 2^{d\text{-}1}(X\text{-}y_1\text{-}y_2\text{-}\ldots\text{-}\ y_{d\text{-}1}\ \text{-}2y_d)$.

Total number of MEs in (O), MO, MOT for all members:

$$Z=x\sum_{i=1}^{d}2^{i-1}(d+1-i)(X\text{-}y_1\text{-}y_2\text{-}\ldots\text{-}2y_i)= x\sum_{i=1}^{d}2^{i-1}X(d+1-i)\text{ - }x\sum_{i=1}^{d}2^{i-1}(\sum_{j=1}^{i}y_j+y_i)(d+1-i)\ = A\text{-}B$$

$$A = x\ \sum_{i=1}^{d}2^{i-1}X(d+1-i) = xX(2^{d+1}\text{- }d\text{-}2) = \left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil(2^{d+1}\text{-}d\text{-}2)\ X = [2(n\text{-}2^d)\text{ - }\left\lceil {}^{n-2^d}\!/_{2^d} \right\rceil(d\text{-}2)]\ X.$$

$$B=x\sum_{i=1}^{d}2^{i-1}(\sum_{j=1}^{i}y_j+y_i)(d+1-i) = \tfrac{1}{2}(d+1)x\sum_{i=1}^{d}2^{i}(\sum_{j=1}^{i}y_j+y_i)\text{ - }\tfrac{1}{2}x\sum_{i=1}^{d}2^{i}\times i(\sum_{j=1}^{i}y_j+y_i) = C\text{-}D$$

$$C = \tfrac{1}{2}(d+1)x\sum_{i=1}^{d} 2^{i}(\sum_{j=1}^{i} y_j + y_i) = \tfrac{1}{2}(d+1)x\sum_{i=1}^{d}(2^{i}\times 2y_i + 2^{i+1}(\sum_{j=1}^{d-1-i} 2^{j-1} y_i))$$

$$C = \tfrac{1}{2}(d+1)x\sum_{i=1}^{d} y_i(2^{i}\times 2 + 2^{i+1}(2^{d-i}-1)) = \tfrac{1}{2}(d+1)x\sum_{i=1}^{d} 2^{d+1}\times y_i = \tfrac{1}{2}(d+1)x2^{d+1}\sum_{i=1}^{d} y_i = (d+1)(n\text{-}2^{d}) \sum_{i=1}^{d} y_i$$

$$D = \tfrac{1}{2}x\sum_{i=1}^{d} 2^{i}\times i(\sum_{j=1}^{i} y_j + y_i) = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i}\times i\times 2y_i + (\sum_{j=1}^{d-1-i} 2^{i+j}\times(i+j)\times y_i))$$

$$D = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i}\times i\times 2y_i + (\sum_{j=1}^{d-1-i} 2^{i+j}\times(i+j)\times y_i)) = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i}\times i\times 2y_i + y_i(\sum_{j=1}^{d-1-i} 2^{i+j}\times(i+j))$$

$$D = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i}\times i\times 2y_i + y_i(\sum_{j=1}^{d-1} 2^{j}\times j - \sum_{k=1}^{i} 2^{k}\times k) = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i}\times i\times 2y_i + y_i(2^{d}(d-2)+2-2^{i+1}(i-1)-2))$$

$$D = \tfrac{1}{2}x\sum_{i=1}^{d}(2^{i+1}\times y_i + y_i\times(2^{d}(d-2))) = \left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil \sum_{i=1}^{d} 2^{i}\times y_i + \tfrac{1}{2}(n\text{-}2^{d})\,(d\text{-}2)\sum_{i=1}^{d} y_i$$

We now derive B=(C-D)=(d+1)(n-2^{d})$\sum_{i=1}^{d} y_i$ - $\left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil \sum_{i=1}^{d} 2^{i}\times y_i$ - $\tfrac{1}{2}$(n-2^{d})(d-2)$\sum_{i=1}^{d} y_i$ =($\tfrac{d}{2}$+2)(n-2^{d})

$$Z = (\text{A-B}) = [2(n\text{-}2^{d}) - \left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil(d\text{-}2)]\,X - (\tfrac{d}{2}+2)(n\text{-}2^{d})\sum_{i=1}^{d} y_i + \left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil \sum_{i=1}^{d} 2^{i}\times y_i$$

Z = (n-2^{d}) (2X-($\tfrac{d}{2}$+2)$\sum_{i=1}^{d} y_i$) + $\left\lceil {}^{n-2^{d}}\!/_{2^{d}} \right\rceil$ ($\sum_{i=1}^{d} 2^{i}\times y_i$ - (d-2)X), MEs with cost C_E each.

Note: $\tfrac{1}{2}\sum_{i=1}^{d} 2^{i}\times i = 2^{d}\times d - (2^{d}-1) = 2^{d}\times(d\text{-}1) + 1$.

2.6. Performance Evaluation and Comparative Evaluation Results

Table 2.10, as well as Figures 2.16-2.19, illustrate some indicative comparison results of the three Octopus-based protocols *vs.* the very efficient centralized OFT scheme. The figures represent the costs of a number of metrics required to measure the performance of the discussed protocols *vs*. the group size, while the rest of the parameters remain constant. These costs represent the number of bits required for communication, computation, storage. For the leader Add/Evict Computation, (O) is the worst, MO and mainly MOT performs much better when *d* is small. As for the Initial Communication, MO is the worst, MOT and (O) slightly outperform OFT. For the Addition/Eviction Communication (critical in MANETs), (O) is outperformed by MO, MOT, and MOT gets closer to OFT than any other protocol.

Cost	2^d-Octopus (O)	Mod. 2^d- Octopus (GDH.2)- (MO)	Mod.2^d-Octopus (TGDH)-(MOT)
GSC Storage	$K(\lceil {}^{n-2^d}/_{2^d} \rceil + d) / K(2\lceil {}^{n-2^d}/_{2^d} \rceil + d)$	$K(\lceil {}^{n-2^d}/_{2^d} \rceil + d)$	$(\lceil {}^{n}/_{2^d} \rceil + \log\lceil {}^{n}/_{2^d} \rceil + d)K$
Member Storage	$(2+d)K$	$(d+1)K$	$(\lceil {}^{n}/_{2^d} \rceil + \log\lceil {}^{n}/_{2^d} \rceil + d)K$
Initial GSC Computation	$(3\lceil {}^{n-2^d}/_{2^d} \rceil + 2d)C_E + (\lceil {}^{n-2^d}/_{2^d} \rceil)^{4/3} + 1.25($ $\lceil {}^{n-2^d}/_{2^d} \rceil)K^2 + \lceil {}^{n}/_{2^d} \rceil C_{rr}$	$(\lceil {}^{n}/_{2^d} \rceil + 2d)C_E + \lceil {}^{n}/_{2^d} \rceil C_{rr}$	$(2\log\lceil {}^{n}/_{2^d} \rceil + 2d)C_E + \lceil {}^{n}/_{2^d} \rceil C_{rr}$ at max.
Initial Members Computation	$(d+2)C_E$	$((1/2)\lceil {}^{n}/_{2^d} \rceil + d)C_E$	$(2\log\lceil {}^{n}/_{2^d} \rceil + d)C_E$ at max
Initial Comm/tion	$(2n+(d-1)2^{d+1})K$	$(2^{d-1}(\lceil {}^{n}/_{2^d} \rceil^2 + 3\lceil {}^{n}/_{2^d} \rceil - 2) + 2^{d+1}d)K$	$(2^d 2\lceil {}^{n}/_{2^d} \rceil + 2^{d+1}d)K$
Add GSC Computation	$(3\lceil {}^{n-2^d}/_{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil + 4)C_E + 2C_{rr}$ $+2(\lceil {}^{n-2^d}/_{2^d} \rceil - 1)K^2$,one $(2\lceil {}^{n-2^d}/_{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil)C_E$ rest	$C_E(\lceil \frac{n+1}{2^d} \rceil + 1 + 2\lceil \frac{d+1}{2} \rceil) + C_{rr}$, one $C_E(2\lceil \frac{d+1}{2} \rceil)$, rest	$C_E(2\log\lceil \frac{n+1}{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil) + 2C_{rr}$, one GSC $C_E(2\lceil \frac{d+1}{2} \rceil)$, rest
Add Members Computation	$4C_E$, two $(2+d)C_E$ max. $2C_E$, the rest dC_E max.	$3C_E$, one subgroup $(1+d)C_E$ max. $2C_E$, rest dC_E max.	$4C_E$, one member $(h+d)C_E$ max $2C_E$, rest dC_E max
Add Comm/tion	$(4+2(2^d-1)+(2^d-2)+2\lceil {}^{n-2^d}/_{2^d} \rceil)K$	$(2\lceil {}^{n+1}/_{2^d} \rceil + 2(2^d-1)+(2^d-1))K$	$(\log\lceil {}^{n+1}/_{2^d} \rceil + \lceil {}^{n+1}/_{2^d} \rceil + 2(2^d-1)+(2^d-1))K$.
Delete GSC Computation	$(3\lceil {}^{n-2^d}/_{2^d} \rceil + 2 + 2\lceil \frac{d+1}{2} \rceil)C_E + C_{rr} +$ $+(2\lceil {}^{n-2^d}/_{2^d} \rceil - 5)K^2$, one $(2\lceil {}^{n-2^d}/_{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil)$ rest	$C_E(\lceil \frac{n-1}{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil) + C_{rr}$, one $C_E(2\lceil \frac{d+1}{2} \rceil)$, rest	$C_E(2\log\lceil \frac{n-1}{2^d} \rceil + 2\lceil \frac{d+1}{2} \rceil) + C_{rr}$, one $C_E(2\lceil \frac{d+1}{2} \rceil)$, rest
Del. Members Computation	$3C_E$, two $(1+d)C_E$ max. $2C_E$, the rest dC_E max.	$3C_E$, one subgroup or $(1+d)C_E$ max. $2C_E$, rest dC_E max.	$4C_E$, one member $(h+d)C_E$ max $2C_E$, rest dC_E max
Delete Comm/tion	$(2+2(2^d-1)+(2^d-2)+2\lceil {}^{n-2^d}/_{2^d} \rceil)K$	$((\lceil \frac{n-1}{2^d} \rceil - 1)+2(2^d-1)+(2^d+1))K$	$(\log\lceil \frac{n-1}{2^d} \rceil + 2(2^d-1)+(2^d-1))K$

Table 2.10. Analytical Performance Metrics for KA Protocols (O), MO, MOT

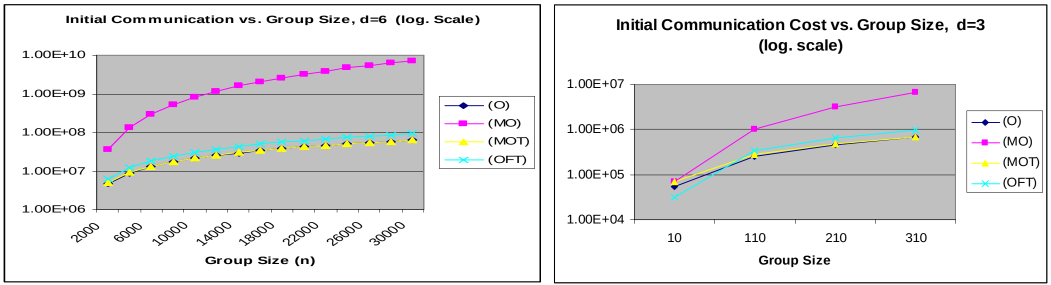


Figure 2.17. Initial Communication Cost for (O), MO, MOT, OFT in terms of Group Size. MO is by far the worst. MOT along with (O) are the clear winners followed closely by OFT

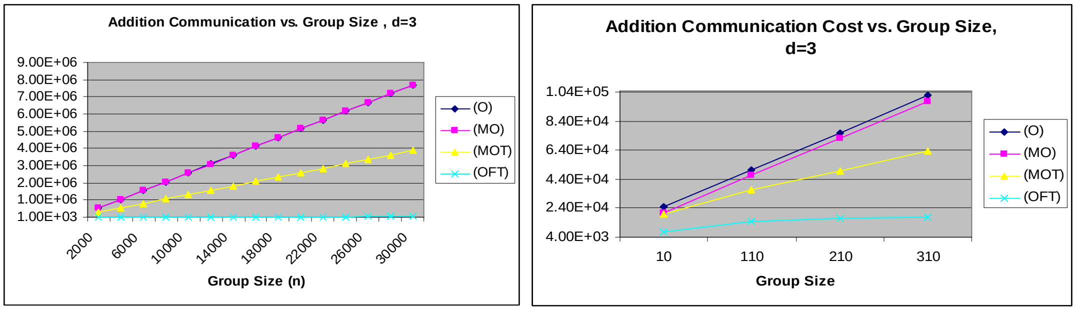


Figure 2.18. Member Addition Communication Cost for (O), MO, MOT, OFT in terms of Group Size. MO and (O) are by far the worst. MOT reduces overhead to almost half w.r.t. OFT. Similar is the graph for the Deletion Communication Cost.

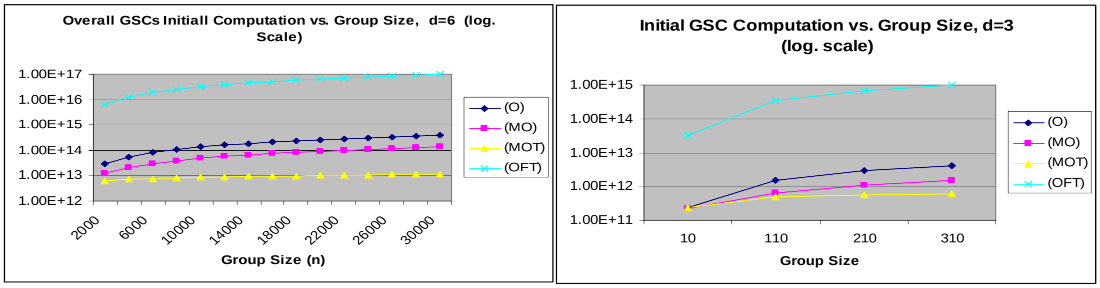


Figure 2.19. Overall Initial Leader Computation for (O), MO, MOT, OFT w.r.t Group Size. OFT is by far the worst. MOT has the best performance overall, followed by MO, then (O).

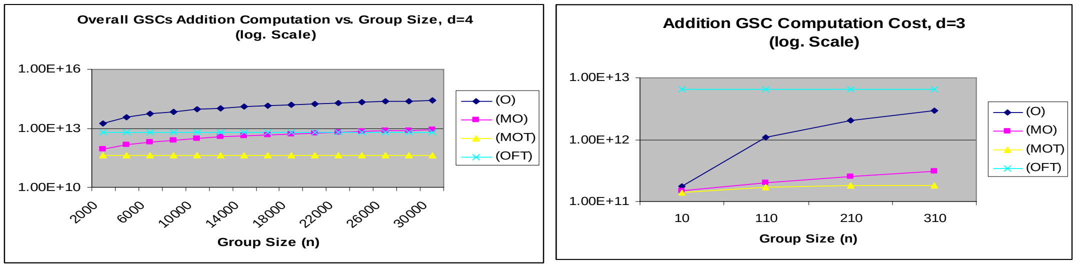


Figure 2.20. Member Addition Computation for (O), MO, MOT, OFT in terms of Group Size.(O) is by far the worst. MOT is the winner, while MO and OFT behave similarly. Similar is the graph for Member Deletion Computation when d=4.

Initial Leader Computation: MOT presents the best overall performance, followed by (O), (MO), and finally OFT. The major performance difference between the Octopus schemes and OFT is due to the fact that OFT is centralized with one group leader operating on the entire group, as opposed to 2^d leaders in Octopus schemes, each of which operates on a subset of the group. In addition, OFT uses RSA, a different cryptosystem for public key encryption. RSA is substantially more expensive than the DH scheme used by Octopus schemes. This is due to the expensive generation of RSA pairs of public/private keys. Furthermore, MOT is superior to the rest of the Octopus schemes, since TGDH that is used within the subgroup, is more efficient than the subgroup protocols used by (O) and MO.

Addition/Deletion Leader Computation: MOT is the clear winner, while MO and OFT behave similarly. (O) presents the worst performance, due to the inefficient manner by which the subgroup leader manipulates the partial subgroup keys.

Initial Communication: MO presents by far the worst performance. MOT along with (O), are the clear winners followed closely by OFT. The poor performance of MO is due to the high communication cost incurred when GDH.2 is applied within the subgroups.

Addition/Deletion Communication: MO and (O) present by far the worst performance. MOT reduces the resulting overhead to almost half from the perspective of OFT. This is a very significant result for a KA scheme, which is naturally more expensive w.r.t. re-keying communication cost, than a tree-based centralized scheme. On the other hand, re-keying due to membership changes occurs all too often in MANETs, hence reducing the resulting communication cost is of profound importance.

From our discussions, analysis, and results, it can be seen than MOT is superior to the rest of Octopus based schemes. The subgroup protocol used for (O), tolerates subgroup member failures, but becomes very expensive and inefficient upon the event of a subgroup leader failure, since the leader stores a large amount of information. The subgroup protocol used for MO is more sensitive to membership changes, which may affect the connectivity of the subgroup in such a way, that a whole new reconfiguration of subgroup members might be necessary. TGDH is a hybrid contributory protocol with an efficient (for re-keying) binary tree structure, and is hence a combination of the previous two diverse categories of protocols. The subgroup leader is basically used for communication regulation and after each round of execution it stores exactly the same amount of information as the rest of the members. In this manner, TGDH is not as sensitive as GDH.2 to member changes or failures, and not as sensitive as the centralized subgroup protocol of (O), to leader failures. Hence, TGDH provides a more efficient solution for the subgroups. Its beneficial characteristics are reflected to the whole Octopus scheme, by substantially boosting its performance, and improving its characteristics of flexibility and robustness even further.

2.7. Summary and Conclusions

In this chapter, we have started with a comprehensive investigation on a number of existing or novel KM protocols. We have studied the performance of these protocols associated only with the generation and distribution as a first performance attribute. Our main objective has been to design a KM framework that is secure, efficient, scalable, and robust in a MANET. Towards this end, we studied the two major families of protocols (contributory and centralized), we have discussed their limitations and assessed their feasibility to a distributed, resource constraint, infrastructure-less environment. We provided our own extensions to a number of representative schemes from the literature (i.e. Hypercube, ING, BD, etc.), in an attempt to conduct a thorough and spherical evaluation of the discussed protocols.

Our main contribution was the **design** of two **new** protocols (MO, MOT), and the **extension** and **adaptation** of the *original Octopus* **(O)** to a distributed, resource-limited environment. We have shown that MOT is efficient, scalable, secure, and robust. Our performance evaluation demonstrated its superiority over the rest of Octopus-based schemes and many other KA protocols. The following features of Octopus schemes have motivated our interest to extend them: *a*) their **hierarchical framework** through which they can tolerate network dynamics and be more scalable, *b*) a **subgroup leader** handling a relatively **small subgroup** is feasible in MANETs, *c*) **disruptions** are **handled locally** at first within the subgroup, and then reflected to the whole group via **very low cost re-keying over Hypercube**.

The benefits of the discussed schemes are accompanied with a number of drawbacks or omissions. First of all, for a fair and spherical analysis of the discussed protocols, we must consider the effect of a multi-hop routing network on KM operations. Moreover: (*a*) Hypercube requires excessive routing, as any protocol that connects entities deployed within a large network with no proximity assumptions. An optimization of the underlying routing may improve the performance of the scheme, (*b*) Hypercube is very expensive to start over upon failures - hence we need to look into efficient ways of overcoming failures of

Hypercube members, (*c*) GDH.2 member failures affect the protocols: their impact is less significant due to their localized nature, but still, we need to look into efficient ways of overcoming such failures in GDH.2, and (*d*) GDH.2 and Hypercube are not centralized, hence in order to accomplish higher performance and robustness, their communications schedule must be efficient and topologically driven, and be included in the overall overhead produced by each scheme. In the following chapters we will build upon the basic scheme introduced in this chapter, addressing these issues, and providing solutions to each.

2.8. Appendix: Bit Complexities of Cryptographic and Arithmetic Operations

I. Modular Multiplication (MM) and Modular Squaring (MS)

Modular Multiplication (MM) of two *K*-bit numbers: $(2K^2 + K)$ (Montgomery Method).

Modular Squaring (MS) of a *K*-bit number: $(1.5K^2 + K)$ (Montgomery Method).

In chapter 6 we conducted our analysis for the average case complexity, assuming that half of the bits of the binary representation of a number are 0. The results quoted here, follow the worst case scenario, and the bit complexities reported here are higher than those in chapter 6.

II. Modular Exponentiation (ME) Cost: C_E

Let the bit size of the exponent be l, this of the modulo be K, and this of the base be K. Let the processing cost of a ME be denoted as $C_E\,(l, K)$. A ME is performed by looking at the bit pattern of the exponent as successive powers of 2, and then successively squaring the argument and multiplying with modulus as necessary [14, 15]. The upper bound in the total number of MSs required is the number of bits (l) of the exponent. Then, the ME reduces to a number of MMs (as many as the number of bits equal to 1 in the binary representation of the exponent, with l as upper bound). Hence, a ME reduces to l MSs and l MMs.

Processing Cost of a ME: $C_E\,(l, K) = l\times(2\times K^2+K) + l\times(1.5K^2+K) = l\times(3.5\times K^2+2K)$.

In the case that the exponent and the modulo have the same size, then $C_E(K) \sim \mathrm{O}(K^3)$.

III. Algorithms for finding large primes

Theorem for Prime Numbers: By $P(n)$ we denote the number of primes < n, given by: $P(n) = \frac{n}{\log_e n}$. The density of primes D is thus: $D = \frac{dP(N)}{dn}$ = 1/l*ogn* - 1/log^2*n*.

We denote $log_e\, n = log\, n$. For a large n, the second term is small and can be omitted. This suggests that primes be found by **trial and error**: Select a large (odd) integer, test it for primality. If the test fails change and try again. With 512-bit integers expect success in $\frac{1}{2}\times 355=178$ attempts. The **average** number of **attempts** is: $E[T] = \frac{1}{2} \times \frac{\log_2 n}{\log_e 2} = 0.7 \times log_2\, n$.

Rabin and Miller algorithm *P* for Primality Test: A Primality Test provides an efficient probabilistic algorithm for determining if a given number is prime. It is based on the properties of strong pseudo-primes. The following algorithm has been proposed by Rabin and Miller, and has been implemented by Knuth: Given an odd n, let $n = 2^r \times s+1$ with s odd. Then choose a random a: $1 \leq a \leq (n\text{-}1)$. If a^s =1 mod n or $a^{\ 2js}$ = -1 mod n, where $0 \leq j \leq r\text{-}1$, then n passes the test. A prime will pass the test for all a. The test is very fast and requires no more than $(1+O(1))\ log_2\, n < 2 \times log_2\, n$ MMs. A number that passes the test is not necessarily prime. Monier and Rabin have shown that a composite number passes the test for at most 1/4 of the possible bases a.

IV. Calculation of Random Key Generation Cost: C_{rr}

A popular secure **pseudo-random number generator** is the **Blum-Blum-Shub (BBS)**:

1. Compute the complexity for choosing x relatively prime to n: pick odd $x<n$, with length(x) = length(n) = K. find relative prime after 0.7×length(n) attempts in average (prime number theorem). This is done only the 1st time BBS is used.

2. Calculate $x_j = x^2_{j-1}$ mod n.

3. Take b_j to be the least significant bit (**LSB**) of $x_{j.}$

4. GOTO_2: compute new x_j, take new b_j. Repeat as many times as the bit length of our key.

A way to **speed up** this slow operation is the following: after every multiplication extract the k LSBs of x_j. As long as $k \leq log_2 log_2 n$, the scheme is cryptographically secure. Step 2 requires an MS with cost ($1.5K^2+2K$). We repeat step 2 for (K/k) times in order to derive K randomly generated bits. k is fixed so that $k \leq log_2 K$. The total cost for the generation of one key is: $(K/(2 \times log2)) + (K/k) \times (1.5K^2+2K) < (K/k) \times 1.5K^2 + K$.

Hence, $C_{rr} = (K/k) \times 1.5K^2 + K = (1/k) \times 1.5 \times K^3 + K$, where $1 \leq k \leq log_2 K$.

V. Calculation of the Blinding Function C_g

C_g is the cost for the one-way function that blinds a key. It is suggested that it can be calculated with the use of the MD5 or the SHA method [41], [43]. We used MD5 to estimate the complexity of C_g. MD5 takes a message and turns to multiple to 512 bits. If the length of the key is K, then $T = \left\lceil \frac{K}{512} \right\rceil$ is the number of times we need to multiply 512 bits, to construct the padded message that will be used in the procedure. MD5 makes four passes over each 16-byte chunk of the message. Each pass has a slightly different method of mangling the message digest. Each stage computes a function based on the 512-bit message chunk and the message digest (128-bit quantity) to produce a new intermediate value for the message digest. At the end of the stage, each word of the mangled message digest is added to its pre-stage value to produce the post-stage value that becomes the pre-stage value for the next stage. The value of the message digest is the result of the output of the final block of the message. Every stage is repeated for all T 512-bits chunks, and in every stage a separate step is taken for each of the 16 words of the message. During every such step we have addition of 5 terms (including the previous blocks, the message digest and a constant). In two stages the message digest function has cost of 3×32 bits, and in the rest two the cost of the message digest function becomes 2×32 bits. The rest of the terms, as well as the output of the digest function is 32 bits. The left rotates imply complexity of 32 bits as well. The complexity for each message chunk (512-bit) for all the 16 blocks of 32 bits for all stages becomes: 1st stage: (4+3)×32+32, 2nd stage: (4+3)×32+32, 3d stage: (4+2)×32+32, 4th stage: (4+2) ×32+32. In total we have: 16×32×30 bits = 15360 bits. The complexity for all chunks of the message is: $T\times 15360$ bits = $\lceil \frac{K}{512} \rceil \times 15360 = 30K$. Hence, we roughly estimate: $C_g = 30K$.

VI. Random Primes Key Generation Cost C_r, Public Key Encryption Cost C_{PE}, and Public Key Decryption Cost C_{PD} (RSA PublicCryptosystem).

RSA Algorithm: Plaintext M is encrypted in blocks:

Encryption: $C = M^e \bmod n$ **Decryption:** $M = C^d \bmod n$

Public key = $<n, e>$ Private key = $<n, d>$

Requirements: There exist $<n, e, d>$ such that $M = M^{ed} \bmod n$ for all $M < n$. It is easy to calculate M^e, C^d for all $M<n$. However, it is computationally infeasible to find d given n, e.

Random Primes Generation Cost for RSA Public/Private Keys C_r:

RSA requires the generation of two prime numbers p, and q s.t. $p\times q = n$, where n will be used as the modulo in RSA. We use the Random Key Generation method to generate two such numbers. For a 512-bit key we get two 256-bit random numbers. Next, from each random number we generate a prime number. A sieve is used to eliminate obvious cases. Anything that survives the sieve is subjected to Fermat's test (if $x^{p-t} \bmod p<>1$, then p is not prime). Next, we multiply the two (alleged) primes p and q to produce the modulus n. We impose that approximately: $\log_2(p) = \log_2(q) = 0.5\log_2(n) = 0.5K$. As seen, the complexity for a random number generation (not necessarily prime) p is: $(1/k)\times 1.5(\log_2 p)^3+(\log_2 p)$, where $1\leq k\leq \log_2(\log_2 p)$ (1).

However, we want our random numbers p and q to be large primes as well. By method of trial and error we examine which p, q will pass the Primality Test (in average $\log_2(t)\times 0.7$ trials are required to find a large number t that fulfills all the "primality" requirements). Each number t is candidate to pass the Primality Test, which results in processing cost of approximately: $(1+O(1)) \times \log_2(t) < 2 \times \log_2(t)$ MMs with complexity: $2(\log_2(t))^2 + (\log_2(t))$. Hence, the overall complexity to derive prime t given any random number u is:

$(\log_2(t) \times 0.7) \times (2 \times \log_2(t)) \times (2(\log_2 (t))^2 + (\log_2(t))) = 2.8 \times (\log_2(t))^4 + 1.4 \times (\log_2(t))^3$ (2).

The complexity for the generation of two random numbers is given by (1). Then $0.7 \times \log_2(p)$ and $0.7 \times \log_2(q)$ trials are required in average to derive primes p, q, with complexity provided by (2). The complexity $C_r(K)$ for the prime random p, q is: $C_r(K)=0.7\ (1/k) \times 1.5(\log_2 p)^4+0.7 (\log_2 p)^2+2.8(\log_2 p)^4+1.4(\log_2 p)^3+0.7(1/k) \times 1.5(\log_2 q)^4+0.7(\log_2 q)^2+2.8\ (\log_2 q)^4+1.4(\log_2 q)^3 = 2 \times [0.7(1/k) \times 1.5(\frac{1}{2} \log_2 n)^4+0.7(\frac{1}{2} \log_2 n)^2 + 2.8\ (\frac{1}{2} \log_2 n)^4+1.4\ (\frac{1}{2} \log_2 n)^3]$

$C_r(K) < 0.5(\log_2 n)^4 + 0.35(\log_2 n)^3 + 0.35(\log_2 n)^2 = 0.5K^4 + 0.35K^3 + 0.35K^2$, $1 \leq k \leq \log_2(\frac{1}{2} K)$.

Public Key Encryption Cost C_{PE}:

For RSA public key encryption, we must ensure that the following equality holds: $e \times d = 1 \bmod (p\text{-}1) \times (q\text{-}1)$. If e is known, we can derive d with O(1) calculations. In many versions of RSA, e is assumed fixed with bit-size K_e. We wish to calculate the encryption of a message of bit-size K. For that a ME is required. So we obtain: $C_{PE} = 3.5 \times K_e \times K^2 + 2K_e \times K$.

Observation: From the equation: $e \times d = 1 \bmod (p\text{-}1) \times (q\text{-}1)$ we see that $K_e+K_d \geq log_2\, p+log_2\, q = K$. Thus, the selection of K_e affects the selection of K_d and vice versa. The most popular value for e is 3. Generally it is preferred d to be very large for the decryption, since the larger the d the more difficult it is for an attacker to break the algorithm. If we select e=3, we select a number d with size $K_d \approx K$. This is why, in the RSA, encryption is faster than decryption.

Public Key Decryption Cost $C_{PD:}$

For the decryption the same idea is adopted. We need to compute: $m= c^d \bmod n$. If we use *Shamir's assumption* for the unbalanced RSA, according to which $c^d \bmod n$ can be reduced to $r^{dl} \bmod p$ where $d1=d \bmod \phi(p)$ and $r = c \bmod p$, we achieve a speedup of length(p).

According to Shamir's Proposition we have: $n = p \times q$ but also $lgn=10 \times lgn'$ and $lgp=2 \times lgp'$, where the same equation that relates n' and p' holds as well: i.e. $n' = p' \times q'$. From these equations we can see that $lg\ n\ /\ lg\ p>5$, hence the use of mod p instead of mod n produces a decryption cost C_{PD} more than 25 times lower than the encryption cost C_{PE}. ($C_{PE} > 25 \times C_{PD}$).

A similar idea for the decryption used in the **Chinese Remainder Theorem**: we can speed decryption up to four times by computing: c^d mod p and c^d mod q instead. The **Chinese Remainder Theorem (CRT)** then allows us to compute c^d mod $p \times q = c^d$ mod n. The idea in the **CRT** is used for the decryption only, since p and q are known only to the member that decrypts the message. In the encryption, the member encrypting a message with the public key does not know p and q (unless it has created the public-secret pair). The resulting processing cost for the RSA decryption becomes: $C_{PD} = (3.5/25) \times K_d \times K^2 + (2/25) \times K_d \times K$.

Special case: If e=3, then $K_d = K$, and $C_{PD} = ((3.5/25) \times K^3 + (2/25) \times K^2) = O(K^3)$, $C_{PE} = O(K^2)$.

VII. Calculation of Symmetric Encryption and Decryption Costs C_{SE}, C_{SD} (DES)

Detailed description of the DES operation can be found among other documents in [41]. We calculate the number of computations by following the algorithm step by step: We perform permutations of: (*a*) the initial 64-bit input, (*b*) the "per round required keys" for each of the 16 DES rounds, and (*c*) the 64-bit output finally. The permutations (*a*), (*c*) have a specific structure and require a fixed number of operations (cost C_{PMT}.). To generate the "per round" keys we do an initial permutation of the 56 useful bits of key (cost C_{PMT}), and the output is divided into two 28-bit values. Hence, up to this point, 3 permutations of cost C_{PMT} each are needed. They are table driven, and the cost for each is the mapping of each generated block to a given row and column in the table, repeated as many times as the bit length of the output. Hence, C_{PMT} <64 bits. Then, we randomly permute 24 of those bits as the left half and 24 as the right half of the per round key. A random permutation of k bits has complexity about $k \times \log_2 k$. So, two permutations are executed that consume $24 \times \log_2 24$ bits each, per round. In each round, 48 bits are selected via a “table-driven permutation”, w.r.t. a table that consists of the previous randomly permuted 48 bits. For all 16 rounds, an additional number of $16\times C_{PMT}$ $<16\times 48$ bit operations is produced. In any round, the 64 bits are divided into two 32-bit halves: L_n, R_n. The output generated is 32-bit values L_{n+1} and R_{n+1}. Their concatenation is the 64-bit output of the round. Extra computations are needed only for $R_{n+1.}$ From value R_n and the “per round” key of 48-bits we produce an outcome F of 32 bits which is XORed with L_n The process requires about: (*a*) 48 bit operations (max.) to extend R_n from 32 bits to 48 bits, (*b*) 64 bit operations (max.) for mapping the outcome in the 64-bit boxes, and (*c*) 32 bit operations for the last XOR of F with L_n. The total processing for DES encryption becomes: $O(3\times 64+16\times 48+16\times 144)$. The same amount of computations is needed for decryption.

To encrypt data larger than 64 bits with a key length larger than 64 bits we break the plaintext into chunks of 64 bits and use DES to encrypt each block. If the key has more than 64 bits we break the encryption key into chunks as well and do the encryption of the message. So, we do DES encryption/decryption $\left\lceil \frac{K}{64} \right\rceil$ times. C_{SD} and C_{SE} are the same, linear to the key length.

$$C_{SE} = C_{SD} = \left\lceil \frac{K}{64} \right\rceil \times O(3\times 64+16\times 48\times 24+16\times 144) < \left\lceil \frac{K}{64} \right\rceil \times 3264 \ = 51\times K = C_{\mathrm{DES}}\times K.$$

In general, we can safely assume the following: $C_{\mathrm{SE}} = C_{\mathrm{SD}} = C_{\mathrm{DES}}\times K$, where $35 \le C_{\mathrm{DES}} \le 60$.

Chapter 3: Evaluating and Improving Key Agreement Protocols for Secure Group Communications subject to underlying routing

3.1 Introduction: Evaluating and Improving Key Agreement Protocols for Secure Group Communications subject to underlying routing.

The primary focus of the prior work was the analysis of the proposed KM schemes, based on the overhead resulting from the processing and exchange of KM-related data only. All these protocols have been implemented on a logical graph that makes it possible to evaluate their logical design and algorithmic flow in isolation from any other related function of the underlying network. That is also referred to as a logical abstraction, which is widely used in the literature to evaluate and compare KM protocols with the key generation algorithm as the only performance attribute. It is useful in order to isolate and investigate the strengths and flaws of different protocols w.r.t. conceptual design and metrics related to KM. However, although this method has its own merit of interest, it does not realistically reflect the actual overhead of a protocol when implemented on a multi-hop ad hoc network. In reality, the execution of a KM protocol involves a number of underlying auxiliary network functions (i.e. routing, clustering, leader election) that should be included in the computation of the metrics that reflect the overall overhead incurred in the network. For example, it is unrealistic to assume that members of the secure group can communicate with each other directly in such a network. The limited resources of wireless devices may significantly constrain the connectivity among nodes. Hence, it is expected that a path between two members may contain multiple relays which also forward KM related data. Their contribution should be also included in the computation of metrics related to the execution of a KM protocol.

In addition, each of the KM protocols considered relies upon the backbone network functions to a different extent or in a different way. It is obvious that our previous evaluation has not been totally fair or realistic. The consideration of network parameters that can accentuate the individual characteristics of the KM protocols as well as the differences in their performance will (*a*) provide more realistic comparison results and (*b*) shed more insight on the actual feasibility of each protocol in the resource limited environments studied.

For example, the hierarchical framework on top of which the three Octopus-based schemes are executed is supported by a clustering mechanism. Although the cost of adding and maintaining the clustering mechanism adds to the overall cost of each individual scheme, it does not alter the outcome of the comparative evaluation of the schemes. This is because all three schemes utilize clustering in the same manner. On the contrary, the leader election process and the communication regulation schedule within the subgroups of the three protocols differ. The different role of the leader in the subgroups potentially affects the leader election process required for each protocol. Similarly, communication regulation is not required for the subgroup in (O) and MOT, in contrast to MO, in which members interact among themselves for the subgroup key generation (KG). A simple schedule based arbitrarily on members' static *ID*s is likely to result in unnecessary routing. This is because such a schedule does not exploit topological proximity of members for the communication exchanges required towards the generation of the group key. As we are going to show in what follows, establishing a topologically aware schedule to execute a KA protocol, reduces the total number of relays invoked in most cases. Reducing the overall communication cost that results from all the participants in the KG, including the relays, is one of our major objectives. Previous work does not deal with basic characteristics or side-effects of wireless multi-hop ad hoc networks: (*a*) members' partial connectivity due to limited transmission range, (*b*) members' dynamic agreement on a global communication regulation schedule due to lack of central trusted entities, and (*c*) issues or robustness and fault-tolerance, since disruptions, failures, network partitions and merges occur all too often in such networks. As a result, the analysis of previously proposed protocols only gives a partial account on their real performance. In the following sections we propose methods towards capturing the actual behavior of the studied KA schemes, subject to underlying routing, in a wire-less multi-hop ad hoc network. Towards this direction, we first directly extend the proposed KA protocols including the underlying routing protocol blindly, that is without "topologically aware" features. This is the case when we arbitrarily assign member *ID*s. We then go beyond blind extensions of such protocols, by definition of a new topology driven communication schedule that optimizes or improves our own defined metrics of latency and bandwidth. The resulting protocols are significantly more efficient in terms of latency, actual communication cost or bandwidth (i.e. including the contribution of relay nodes as well) and computation cost. For this approach, we consider two cases for the group member graph:

Case 1: The group member graph represents a physical topology (each edge is physical link).

Case 2: The group member graph represents a logical topology (each edge is a logical link, bounded from the number of hops between two vertices provided by the routing. Intermediate hops are not necessarily group members)

In the current literature, the relevant work that deals partially with issues that arise when we combine KM with the underlying routing is limited. Amir et al. [14, 15], focus on robust KA to make GDH protocols fault-tolerant to asynchronous network events. However, their scheme was primarily designed for the Internet, and requires an underlying reliable group communication service and ordering of messages, so that preservation of virtual semantics is guaranteed. Hence, their contribution has limited scope and cannot be applied to a general ad-hoc network. Poovendran et al. [44], attempt to minimize the bandwidth for secure group communications with respect to energy expenditure. They utilize centralized tree key distribution schemes. The network topology is considered static, and there is no provision for adjusting the key tree structure to a dynamically changing network. The optimal solution of their formulation does not scale with group size. In [45, 46, 47], we extended the Octopus-based KA protocols to provide robust and efficient KM for group communications in MANETs. Again, the primary focus of this work was the analysis and performance evaluation of the proposed schemes, in isolation from network functions that interact with the protocols.

3.2. Direct Extension of KA protocols to a MANET, including routing, w.r.t. arbitrary fixed communications schedule

We model the connectivity among the parties with a connectivity graph G(V,E), where $E \in V \times V$, and an edge between any two nodes exists if and only if the two associated parties are within each other's radio range. We assume that the underlying routing is reliable and that messages are received in a timely manner. We do not deal with dynamic changes in the network such as link/node failures and mobility in this chapter. We assume that these are handled either by initiating the key establishment phase or by re-keying.

The most immediate approach of constructing group KA protocols over multi-hop ad-hoc networks is a **blind** one: underlying routing is integrated into KG without any topological optimizations. In this section we describe and evaluate the performance of this approach for a number of KA protocols discussed in the previous chapter: GDH.1, GDH.2, ING, BD, and Hypercube, which we now briefly recall as well for the sake of self-containment.

A. **GDH1:** This protocol assumes that all parties are connected according to a logical Hamiltonian path and consists of two stages: up-flow (collecting contributions from all members) and down-flow (allowing all members to compute the common key). In the up-flow stage, each member does one exponentiation and the up-flow message between M_i and M_{i+1} contains i intermediate values. After obtaining K_n, M_n initiates the down-flow stage. Each member M_i does i exponentiations: 1 to compute K_n and (i-1) to provide intermediate values to lower indexed members, by raising them to the power of its own exponent. The size of the down-flow message decreases on each link, a message between M_{i+1} and M_i includes i intermediate values.

B. **GDH2:** In order to reduce the total number of rounds, GDH.1 is slightly varied, so that: *a*) in the up-flow stage each member has to compose i intermediate values (each with i-1 exponents) and one cardinal value with i exponents; M_n is the first member to compute the key K_n and the last batch of intermediate values, *b*) in the down-flow stage M_n broadcasts the (n-1) intermediate values to all group members. Even in this protocol it is assumed that all parties are connected according to a logical Hamiltonian path; furthermore it is assumed that the last party on the path can reach all others using a broadcast channel.

C. **BD:** This scheme only requires 2 logical rounds and can be divided into three phases: (*1*) Member M_i generates random N_i and broadcasts $z_i = a^{N_i}$; (*2*) Every member M computes and broadcasts $X_i = ({}^{z_{i+1}}\!/\!{}_{z_{i-1}})^{N_i}$; (*3*) M_i can compute the key $K_n = z_{i-1}^{nN_i} X_i^{n-1} X_{i+1}^{n-2} ... X_{i-2} \bmod p$. The key defined by this scheme is different from the previous protocols, namely $K_n = a^{N_1 N_2 + N_2 N_3 + ... + N_n N_1}$. The protocol assumes that parties can simultaneously reach all other parties through broadcast channels.

D. **ING:** This protocol requires a synchronous start-up, requires that all parties are connected according to a logical ring, and completes in (n-1) rounds. In any given round, every participant raises the previously-received intermediate key value to the power of its own random exponent and forwards the result to the next party. After (n-1) rounds everyone computes the same key K_n.

E. **Hypercube:** This protocol assumes that the 2^d parties are connected with a d-dimensional cube and can be identified with the vectors of the d-dimensional vector space $GF(2)^d$. The protocol minimizes the number of simple rounds required for the group key establishment (2^d

parties agree upon a key with d simple rounds by performing DH key exchanges on the edges of the cube). After a vector basis $b_1,\ldots,b_d$ of $GF(2)^d$ is chosen, d rounds are run as follows:

1st round: Every member M_i generates a random number N_i and performs a DHKE with member M_k, where $i, k \in GF(2)^d$, for example $k=i \oplus 2^{j-1}$, using values N_i and N_k respectively.

j^{th} round: Every member M_i does a DHKE with member M_k, where $k=i \oplus 2^{j-1}$, where both members use the value generated in round i-1 as the secret value for the key establishment.

The performance of the above schemes with respect to the main metrics of interest, on logical networks without routing considerations is summarized in Table 3.1. Latency in our context is defined as the number of parallel protocol atomic steps, such as communication rounds or 1-hop transfers over any network physical graph G.

	Latency	Communication Cost	Exponentiations
ING	n-1	$n(n$-1)	n^2
BD	2	2n	n^2+n
GDH1	2(n-1)	$n(n$-1)	$(n^2+3n)/2$
GDH2	n	(n-1)(n/2+2)	$(n^2+3n)/2$
CUBE	log n	n log n	2n log n

Table 3.1. Performance of KA protocols on logical networks without routing integration.

The above protocols describe the KA algorithms and the resulting overhead originating only from the exchanges and computations among the “target” members, as indicated by the “logical implementation” without routing considerations. If we execute the latter protocols taking into account the underlying routing, then every message exchange among the intended group members may involve multiple relays (group members or not) that carry and propagate the intended KM material. The overhead invoked by these relays is not considered in the previous analysis of the discussed KA protocols. Furthermore, in certain cases, it is assumed that a member can directly reach the rest of the group members and broadcast to them in one round. Obviously, this assumption does not hold for a general framework. Moreover, using broadcast and flooding the network may be inefficient. The use of broadcast/ multicast brings about several issues: (*a*) underlying multicast should be optimized as well in order to improve the performance of the KG scheme, and such a cross-layer consideration is additionally complex, and (*b*) multicast may not be supported by all network nodes (heterogeneous).

If the communication schedule applied is not topologically aware, then the routes formed during the communication exchanges are random and the graph generated is actually not expected to resemble the original graph – logical w.r.t. KG. In fact it could look like anything between the original graph under the best case scenario, (i.e. a ring if ING is executed), i.e. optimizing the underlying routing, and a bipartite graph under the worst case scenario. This arbitrary factor that emerges when we merge the KG algorithm "blindly" with the underlying routing is what we will try to capture, model and measure or bound.

We now present in detail how we derive our analytical results from blending the KG algorithms of the protocols discussed above, with an arbitrary and therefore, topology-blind multi-path routing required when they are executed on any network graph G(V,E) with arbitrary communication schedule.

Notation1: Let the diameter of the physical graph G ($diam(G)$) be abbreviated as D (that is, the max number of hops between a pair of nodes in V), and the number of nodes as n. Let the number of hops in the path between two nodes N_i and N_{i+1} be denoted as: $R_{i,i+1}$ = (N_i, N_{i+1}). For our analysis, we assume that: $R_{i,i+1} \leq D \leq |V(G)|=n$.

Notation 2: Let the length of an element in the algebraic group used (over which the decision DH problem is assumed to be hard) be denoted as K (number of bits).

Notation 3: We denote the **combined communication** cost originating from the intended group members and from the relays involved as **overall communication cost** (**CCost**).We denote the **routing cost** (the number of hops visited by the underlying routing during the protocol execution) as **RCost**. We abbreviate the **latency** of the protocol execution as **Lt**.

We first discuss the computation of metric performances for all five KA protocols and then summarize them in a table. Here we refer to the worst case scenarios.

***A*. GDH.1:** We compute the efficiency metrics for this protocol by summing the metrics obtained for the up-flow and down-flow stages; since the two stages are symmetric, it is enough to describe the computation for one stage only, and then multiply both metrics by 2.

Up-Flow Stage: On a logical path, each member M_i composes i values to send to member M_{i+1}, for i=1,...,n-1. All relay members in between the route $R_{i,i+1}$ will carry the same message ($i \times K$ bits). Then, CCost due to step: $M_i \rightarrow M_{i+1}$ is equal to $R_{i,i+1} \times i \times K$ bits.

CCost (GDH.1) = $2\times \sum_{i=1}^{n-1} R_{i,i+1} \times i \times K \leq 2\times \sum_{i=1}^{n-1} D \times i \times K = D \times K \times n \times (n\text{-}1)$.

RCost (GDH.1) = $2\times \sum_{i=1}^{n-1} R_{i,i+1} \leq 2\times D \times (n\text{-}1)$.

Lt (GDH.1) = $2\times \sum_{i=1}^{n-1} R_{i,i+1} \leq 2\times D \times (n\text{-}1)$.

***B*. GDH.2:** As in the case of GDH.1, we compute the efficiency metrics for this protocol by summing the metrics obtained for the (not symmetric) up-flow and down-flow stages.

Up-Flow Stage: The efficiency metrics are identical to the up-flow stage of GDH.1.

Down-Flow Stage: Member M_n composes (n-1) values to broadcast to all members M_i.

Notation 1: We denote the broadcast (multicast) cost (or equivalently routing cost) in terms of number of hops required until the broadcast message reaches all group members as B_n.

Version_1: Assuming that in a realistic broadcast implementation each node broadcasts the same message only once to its neighbors, we could bound B_n as follows: (a) $B_n \leq |E(G)|$. Under the assumption of the multicast advantage and of omni-directional antennas, we bound B_n even further: (b) $B_n \leq |V(G)| \leq n$. Similarly, **CCost** is at most (a)$|E(G)| \times (n\text{-}1) \times K$, or ($b$) $n \times (n\text{-}1) \times K$ using a simple controlled flooding strategy, and **Lt** is at most D.

Version_2: Another implementation of the **broadcast** is obtained by (n-1) **simultaneous unicasts** from the sender to each member of a single message corresponding to each member i through the routing path $R_{n,i}$. **CCost** becomes: $\sum_{i=1}^{n-1} R_{n,i} \times (n-1) \times K \leq D \times K \times (n\text{-}1)^2$,

the associated **RCost** becomes: $\sum_{i=1}^{n-1} R_{n,i} \leq D \times (n\text{-}1)$, and **Lt** becomes: max $_j$ $\{R_{n,j}\} \leq D$.

The resulting **latency** is determined by the member M_i that will get the message sent by member M_n last, and takes time D under the worst case scenario.

CCost (GDH.2, V1) = $\sum_{i=1}^{n-1} R_{i,i+1} \times i \times K + B_n \times (n\text{-}1)$,

CCost (GDH.2,V1a) $\leq \frac{1}{2} n \times (n\text{-}1) \times D \times K + |E(G)| \times (n\text{-}1) \times K = (n\text{-}1) \times K \times (\frac{1}{2} n \times D + |E(G)|)$, or

CCost(GDH.2, V1b) $\leq \frac{1}{2} \times n \times (n\text{-}1) \times D \times K + n \times (n\text{-}1) \times K = n \times (n\text{-}1) \times K \times (\frac{1}{2} \times D + 1)$.

Similarly, **RCost** (GDH.2, V1a) = $\sum_{i=1}^{n-1} R_{i,i+1} + B_n \leq D \times (n\text{-}1) + |E(G)|$, or

RCost (GDH.2, V1b) = $\sum_{i=1}^{n-1} R_{i,i+1} + B_n \leq D \times (n\text{-}1) + n$.

Lt (GDH.2, V1) = $\sum_{i=1}^{n-1} R_{i,i+1} + D \leq D \times (n\text{-}1) + D = D \times n$.

CCost (GDH.2, V2) = $\sum_{i=1}^{n-1} R_{i,i+1} \times i \times K + \sum_{i=1}^{n-1} R_{n,i} \times (n-1) \times K \leq \sum_{i=1}^{n-1} D \times i \times K + D \times K \times (n\text{-}1)^2$

CCost (GDH.2, V2) $= D \times K \times (n\text{-}1) \times (\frac{1}{2} n + n\text{-}1) = (n\text{-}1) \times (\frac{3}{2} n\text{-}1) \times D \times K = \frac{1}{2}(3n^2 + 5n + 2) \times D \times K$.

RCost (GDH.2, V2) = $\sum_{i=1}^{n-1} R_{i,i+1} + \sum_{i=1}^{n-1} R_{n,i} \leq D \times (n\text{-}1) + D \times (n\text{-}1) = 2 \times D \times (n\text{-}1)$.

Lt (GDH.2, V2) = $\sum_{i=1}^{n-1} R_{i,i+1}$ + max $_j$ $\{R_{n,j}\} \leq D \times (n\text{-}1) + D = D \times n$.

***C.* BD:** The protocol can essentially be abstracted as the execution of two simultaneous broadcasts from each member to all others. We can analyze each broadcast from a user in each of the two broadcast phases as done for the **down-flow** stage of GDH.2.

Version_1: Controlled Flooding Strategy for Broadcast: In each phase, each member broadcasts a single message of size K to the remaining (n-1) members. In each phase, the **RCost** is still B_n, **CCost** becomes: $B_n \times (n\text{-}1) \times K$, and the resulting **Lt** is at most D.

CCost (BD, V1a) = $2\times|E(G)|\times(n\text{-}1)\times K$. **CCost** (BD, V1$b$) = $2\times n\times(n\text{-}1)\times K$.

RCost (BD, V1a) = $2\times|E(G)|$. **RCost** (BD, V1b) = $2\times|V(G)|$.

Lt (BD, V1a) = $2\times D$. **Lt** (BD, V1b) = $2\times D$.

Version 2: Using simultaneous unicasts, we obtain analytical expressions for the same costs:

CCost (BD, V2) = $2\times\sum_{i=1}^{n-1}\sum_{j=1, j\neq i}^{n} R_{i,j}\times K \leq 2\times n\times(n\text{-}1)\times D\times K$.

RCost (BD, V2) = $2\times\sum_{i=1}^{n-1}\sum_{j=1, j\neq i}^{n} R_{i,j} \leq 2\times n\times(n\text{-}1)\times D$.

RCost for member M_i: $2\sum_{j=1, j\neq i}^{n-1} R_{i,j} \leq 2\times(n\text{-}1)\times D$, **Lt** (BD, V2) = $2\times\max_{i,j}\{R_{i,j}\}\leq 2\times D$.

***D.* ING:** The ING protocol is executed on a logical ring where each member $M_{i \bmod n}$ processes the value received from member $M_{i\text{-}1 \bmod n}$ and communicates a new value to member $M_{i+1 \bmod n}$ for (n-1) times, following the route $R_{i,i+1}$ = (M_i, M_{i+1}). The overall communication cost per member i for the step $M_i \rightarrow M_{i+1}$ becomes $R_{i,i+1}\times(n\text{-}1)\times K$ bits. The following observation indicates how the overall Lt of the protocol is computed: In the worst case scenario, the size of the physical implementation of the ring can be $D\times(n\text{-}1)$. This also means that it takes time $D\times(n\text{-}1)$ for the contribution of member M_{i+1} to reach member M_i, or, equally, $D\times(n\text{-}1)$ is the required amount of time for any member to complete the (n-1) rounds. Hence, Lt for each member and equivalently for the whole ING becomes: $D\times(n\text{-}1)$.

We now summarize the three metrics of interest (*CCost*, *RCost*, *Lt*) for **ING**:

CCost (ING) = $\sum_{i=1}^{n} R_{i,i+1} \times (n-1) \times K \leq n \times (n\text{-}1) \times D \times K.$

RCost (ING) = $\sum_{i=1}^{n} R_{i,i+1} \times (n-1) \leq n \times (n\text{-}1) \times D,$

Lt (ING) = $\sum_{i=1}^{n} R_{i,i+1} \leq (n\text{-}1) \times max\{ R_{i,i+1} \} \leq (n\text{-}1) \times D.$

***E*. Hypercube (*d*-Cube):** This protocol takes d=$log_2 n$ rounds. During any round j, each member i performs a single DHKE with member $k = \varphi(i \oplus 2^{j-1})$. This schedule is defined by one among the candidate bases picked from the vector space $GF(2)^d$ that determine the communication schedule of members for the execution of the protocol. Members i and k exchange one message during round j, through the routing path $R_{i,k} = R_{k,i}$. latency is: $d \times D$.

We now summarize the three metrics of interest (*CCost*, *RCost*, *Lt*) for **Hypercube**:

CCost (HCube) = $\sum_{j=1}^{d} \sum_{i=1}^{2^d} R_{i,\varphi(i \oplus 2^{j-1})} \times K \leq n \times d \times D \times K.$

RCost (HCube) = $\sum_{j=1}^{d} \sum_{i=1}^{2^d} R_{i,\varphi(i \oplus 2^{j-1})} \leq n \times d \times D,$

Lt (HCube) = max $_i\{ \sum_{j=1}^{d} R_{i,\varphi(i \oplus 2^{j-1})} \} \leq d \times D.$

Summary: The prefix "*nt*" abbreviates the "non-topologically-oriented" extensions of the discussed KA schemes and the prefix "*wt*" (i.e. with topology) abbreviates the "topologically oriented" extensions of the same schemes.Table 3.2 summarizes the upper bounds (under the worst case scenario) on the overall communication cost, computation cost and latency of the five protocols discussed, when a non-topology oriented (*nt*) physical implementation is performed for the logical networks involved in them. By contrasting Table 3.1 and Table 3.2, we see that the efficiency of all the protocols decreases by at least a factor of *D*.

Remark: In this analysis, we have assumed that the group members already acquire temporary *id*s associated with a "*nt*" communication schedul. If the assignment of temporary *id*s to all the members, and the advertisement of a global transmission schedule, is done on the fly, then in Table 3.2, we should also add the associated latency and communication cost of the latter. If the assignment of such *id*s is done at random, then these *id*s could be hard-wired on nodes as well. Still, in the event of membership changes or failures, updates regarding the communication schedule should be announced to the corresponding nodes. In either situation, the overall overhead caused by the direct adaptation of the discussed KA protocols to multi-hop ad hop networks is further increased. We do not aim to further analyze this extra complexity of the "*nt*" extensions of the KA protocols. Our investigation of these adaptations has (*a*) helped towards a realistic estimation of the actual implementation of a number of KA protocols on the discussed networks, and (*b*) has pointed out the need to come up with far more efficient and "smart" extensions of these protocols for the same environments. Hence, our basic objective, in addition to blindly extending and analyzing the KA schemes, is to develop extensions that take advantage of the proximity of group members and result in considerably more efficient versions of the protocols. In the section that follows we show that the framework required for the "*wt*" schemes is simultaneously used for the distribution of the temporary *id*s to nodes and thus for the generation of the global transmission schedule. Hence, the additional complexity for distributing temporary *id*s is already included in the overhead required for generating the "*wt*" framework. If we show that the combined overhead, including the distribution of temporary *id*s and the advertisement of a global schedule associated with the "*wt*" approach, is lower than this associated with the "*nt*" approach computed so far, then we achieve even further reduction in the overhead of "*wt*" extensions. In that case, the value of our contribution would be even more significant.

	Latency	Bandwidth	Exp.
*nt*GDH1	$2(n-1)D$	$n(n-1)DK$	$(n^2+3n)/2$

*nt*GDH2	*nD*	$(\lvert E\rvert+\frac{1}{2}n(n\text{-}1))DK$ or $(n\text{-}1)(n/2+1))DK$	$(n^2+3n)/2$
*nt*BD	*2D*	$2n(n\text{-}1)DK$	n^2+n
*nt*ING	$(n\text{-}1)D$	$n(n\text{-}1)DK$	n^2
*nt*HCube	$D\log_2 n$	$n\log_2 nDK$	$2n\log_2 n$

Table 3.2. Performance of *nt* KA protocols over Mobile Ad Hoc Networks

nt	*Non – topological versions*
wt	*With topology consideration versions*
nf	*Non-auxiliary framework included*
wf	*With auxiliary framework inclusion*

Table 3.3. Abbreviations and symbols used in Chapter 3.

3.3. Implementation of KA over MANETs, including routing, w.r.t. a *wt* schedule

In the previous section, we re-evaluated those protocols executing them blindly on a multi-hop network (considering the underlying routing). The protocols were executed based merely on member *ID*s as designated by the KG algorithms. This "*nt*" approach may lead to excessive unnecessary routing, and consequently high communication cost. These results are very indicative of the actual overhead and rounds that will be incurred from these protocols when applied on a general ad-hoc network, even if the associated KG algorithm appears to be very efficient. In wireless multi-hop networks, bandwidth and power consumption are valuable resources, and nodes cannot afford to waste. Reducing the combined costs resulting from the routing and communication exchanges among nodes becomes essential if we want to apply the latter KA schemes on a resource-constrained MANET.

In this section, we try to improve the efficiency of each of these protocols. Towards this end, we are exploring the potential of optimizing their combined routing and communication costs by integrating into the design a topology oriented (*wt*) communication schedule. As we can see from our previous results, each protocol poses **two different optimization problems** as the **routing structure** of each protocol defines a **specific optimization function** for each of the two metrics of **latency** and **combined communication cost**. In summary, we will define and later focus on minimizing the following quantities or performance metrics each of which describes either the overall communication cost or latency that each corresponds to one or more of the discussed KA protocols (scaled down by *K*):

Overall Communication Cost (scaled by *K*):

$$CCost_1 = (n\text{-}1)\times \sum_{i=1}^{n} R_{i,i+1} \qquad (3.1).$$

$$CCost_2 = \sum_{i=1}^{n-1} 2\times i\times R_{i,i+1} \qquad (3.2).$$

$$CCost_3 = (n\text{-}1)B_n \qquad (3.3).$$

$$CCost_4 = \sum_{j=1}^{d} \sum_{i=1}^{2^d} R_{i,\varphi(i\oplus 2^{j-1})} \qquad (3.4).$$

Latency:

$$Lt_1 = \sum_{i=1}^{n} R_{i,i+1} \qquad (3.5).$$

$$Lt_2 = \text{max\_length}\ (B_n) \qquad (3.6).$$

$$Lt_3 = \max{}_i\{ \sum_{j=1}^{d} R_{i,\varphi(i\oplus 2^{j-1})} \} \qquad (3.7).$$

Given these quantities, we can develop **bounds** for the overall communication cost (*CCost*) and latency (*Lt*) of the five protocols considered as follows:

GDH.1: ***CCost*** from (3.2), and ***Lt*** from (3.5),

GDH.2: **CCost** from (3.2), (3.3) and ***Lt*** from (3.5), (3.6),

ING: ***CCost*** from (3.1), and ***Lt*** from (3.1),

BD: ***CCost*** from (3.3), and ***Lt*** from (3.6),

HCube: ***CCost*** from (3.4), and ***Lt*** from (3.7).

Finding approximations of optimal solutions to these quantities, or even improving those provided via the *nt* approach, results in more efficient metrics for the protocols. We introduce our *wt* approach in the next section.

3.3.1. Auxiliary Algorithms

We now present low-cost auxiliary protocols that will be used towards the generation of the core framework for running our *wt* extensions: *a* ***"root" election algorithm,*** *and a* ***"rooted" spanning tree generation algorithm***.

3.3.1.1. Root Election Algorithm

In Chapter 4, we will discuss a leader election scheme that takes into account a broad number of factors for the election (i.e. robustness, residual energy, processing power, available bandwidth, etc.), and receives input from all non-faulty legitimate group members. With this scheme, we select more than one leader: the currently active one, and a number of back-up ones to be sequentially activated once the current leader fails. The function of the leader is much broader than this of a mere starting point, since a leader is responsible for a higher number of operations that expand by far the role of a starting point. If we assume that the leader election process in an essential part of KM protocols, we can use either the active or one of the backup leaders as our starting point. We further assume that during network deployment, certain nodes are prior designated as active or backup leaders, and become the default leader during the initial operation of the secure group. As time progresses, the leader election process is initiated at regular intervals. Hence, at any time, the information about which node will serve as a starting point is propagated to all network nodes. The cost of the group leader election is not going to be included in the KA framework generation, since the purpose of the leader is much more general than just participating to the "starting point" process. The process will be analyzed in Chapter 4, however the cost for advertising the current and subsequent "starting points" to the rest of members requires latency $Lt = D$, and combined communication cost: $|E|\times K_1$ or $n\times K_1$ (multicast advantage), where $K_1<<K$.

3.3.1.2. Generation of a rooted Spanning Tree (ST)

To derive a good approximation for the minimum number of hops necessary to visit all group members, we use the well-known approach of simulating Hamiltonian paths and cycles with a full walk over a rooted spanning tree – denoted as ST - of the connected network graph G (V, E). A ST of G is a connected acyclic sub-graph T(V', E') of G, s.t. V=V' and $E' \subseteq E$, where $|E'| = |V|-1$. In a rooted ST, the tree edges are consistently directed w.r.t. a particular node (root). The purpose of our framework provides the flexibility to generate STs with potentially various attributes (i.e. low weight, where weight is typically: latency, loss rate, inverse of bandwidth, or node degree, diameter, edge count, etc.). We are interested in applying a very simple and lightweight algorithm for the generation of a ST, with extensions to handle dynamic changes: the link weights are associated with the distance (in number of hops) between the vertices (members) of each link. We distinguish two cases for the underlying physical graph and build the associated STs accordingly for each case:

Case (*a*): The graph of the secure group is physical. In this case, the weight of any link that connects two group members directly is set to 1. An arbitrary ST is sufficient for this ca.

Case (*b*): The graph of the secure group is logical, and a logical link may include non-group relays as well. We assume that a generic routing algorithm is prior run (i.e. Dijkstra) and informs each member of the minimum paths (w.r.t. number of hops) to reach a number of members in their proximity. This is a far more realistic case than (*a*) for our setting. Here, the group member graph is weighted, and use different methods to construct the required ST. We first construct a minimum spanning tree (MST) adapting *Prim*'s method to our specifications.

Framework Maintenance: Since topology changes can partition the network, we extend the notion of a ST to a spanning forest (SF), i.e. a (disjoint) collection of STs. The original ST may be separated into fragments (each of which is also a ST) and reconnect, depending on the dynamic changes, responding to multiple link/node failures and recoveries that can occur at arbitrary times. The border nodes of each fragment continually monitor their neighborhood for topological changes and notify the root. The framework maintenance is handled similarly to the proposal of Gallager, Humblet, Spira (GHS) [50], and Cheng et al. [51], where analytical results on the performance of their protocols are provided. Since we do not aim to maintain MSTs, in chapter 4 we propose a variation that significantly reduces the overhead. The impact of the partition and merges of the ST on the actual KA algorithms is studied in Chapter 4. Below, we give an overview of the following simple, low cost protocols to generate a ST for case (*a*). The corresponding algorithms for case (*b*) will be discussed next.

3.3.1.3. Overview of Rooted ST algorithm – physical secure group graph (case a).

For the initial generation of an arbitrary ST, given a central root, one candidate algorithm to apply is a variation of this proposed by Dolev et al. [52], that contains a Breadth First Search (BFS) ST construction for rooted systems. In the algorithm, every node maintains two variables: (1) a pointer to one of its incoming edges, and (2) an integer measuring the distance in hops to the root of the tree. The network nodes periodically exchange their distance value with each other. After reading the distance values of all neighbors, a node chooses the neighbor with minimum distance *dist* as its new parent. It then produces its own distance, which is *dist* +1. The root does not read the distance values of its neighbors and always sends a value of 0. More specifically, after reading all neighbor values for *k* times, the distance of a node is at least *k*+1. So, all direct neighbors of the root will select the root as their parent and update their distance to 1. This line of reasoning can be continued incrementally for all other distances from the root. Hence, after O(*D*) update cycles (latency) the entire tree will have stabilized. The overall communication cost produced is approximately: $O(D\times n)$ [52] because all members query their neighbors during every round. However, if the algorithm is modified so that members update their neighbors after their distance is stabilized, then the overall communication cost produced is reduced to: O(E).

In addition, we propose a more communication efficient approach towards producing a ST as quickly and simply as possible: each node x that becomes part of the ST sends a single "connect" message to each one of its neighbors (except for its ancestor in the tree). A neighbor node y accepts node x as its parent (and sends an ACK message to all 1-hop neighbors) if it receives the "connect" message for the first time from node x (i.e. it did not receive and accept a prior request from another node and it is not part of the ST already). It is obvious that this algorithm converges (as long as the graph remains connected) and does not generate cycles, since each node accepts only one "connect" message, that is, it obtains only one parent. The algorithm produces latency of approximately O(D) rounds. If we use of the "*multicast advantage*" property, we calculate the overall communication cost produced as:

$((n\text{-}1)[\text{ACKs}] + (n\text{-}1)[\text{connect}]) \times K_1 = 2 \times (n\text{-}1) \times K_1$, where $K_1 << K$.

3.3.2. *Wt* KA on a physical secure group member graph (case *a*).

We now use the auxiliary algorithms discussed to generate the required framework for the "*wt*" simulations of KA protocols subject to routing. We analyze the overall communication cost and latency by computing upper bounds on the seven quantities related to the optimization problems defined. The assignment of session *id*s is done after the generation of the basic framework and before the first execution of the KA over the given ST under the worst case scenario. Metrics ***CCost*** and ***Lt*** before the execution of the KA protocols become:

$CCost\,(aux) = (n+2n+2n) \times K_1 = 5 \times n \times K_1$, $Lt\,(aux) = 2D+2n$.

3.3.2.1. Solution to (3.1), (3.5): ST Full Walk

Using an MST to approximate the *Traveling Salesman Problem* (TSP) with *triangle inequality* provides a near-optimal tour of a complete undirected graph G, designating a routing path at most two times the optimal. The *triangle inequality* is a natural one, meaning that going by way of any intermediate stop cannot be less expensive (i.e. the cost function is the *Euclidean* distance). In our case, the network graph is such that allows the generation of a MST. Since the tree links are assigned with equal weights (1), any ST corresponds to an MST. A full walk of the MST traverses every edge exactly twice, resulting in a cost twice as much as the optimal cost. If the output of the *pre-order walk* is used as the schedule for the *Hamiltonian* tour in TSP, the total cost of the tour is at most $2\times R_{OPT}$. It is obvious that $R_{OPT}\geq n$ (link weights or hop distances are set to 1) for our graphs. We can simulate Hamiltonian paths and cycles, just by performing a full walk of the rooted ST. Any of the well-known tree visit walks traverses every edge exactly twice, resulting in a cost twice the number of tree members. We obtain: $\sum_{i=1}^{n-1} R_{i,i+1} \leq 2\times n$ in the simulation of Hamiltonian paths and rings, and (3.1), (3.5) are upper bounded as follows: $(3.1)\leq 2\times n\times(n\text{-}1)$ and $(3.5)\leq 2\times n$.

The following metrics can be now computed for protocols ING, GDH.1, GDH.2:

$CCost(ING) = CCost\ (aux) + (n-1)\times 2nK = 2n^2K + n(5K_1 - 2K)$, $Lt\ (ING) = Lt\ (aux) + 2n = 4n + 2D$,

$Lt\ (GDH.1) = Lt\ (aux) + 2\times 2\times n = 6n + 2D$,

$Lt\ (GDH.2\text{-}Stage1) = Lt\ (aux) + 2\times n = 4n + 2D$.

Since $2\times n \leq \sum_{i=1}^{n} R_{i,i+1} \leq n\times D$, we see that the *wt*-schedule reduces the combined routing and communication overhead, improving it by a factor of ${}^{2\times n}\!/\!_{n\times D} = {}^{2}\!/\!_{D}$ w.r.t. the *nt*-approach.

3.3.2.2. Solution to (3.2): Closest Point Heuristic

This expression is minimized if the larger the message a node is carrying (i.e. the larger the parameter i is), the fewer relays are involved (i.e. the smaller parameter $R_{i,i+1}$ is). Hence, the following inequality should hold for the minimization of (3.2): $R_{n-1,n} \leq R_{n-2,n-1} \leq .... \leq R_{1,2}$. The truth of the argument can be easily proven by assuming an arbitrary allocation of values in the associated *R*'s, and deriving the optimal value *Opt* through successive interchanges between *R*'s, each of which continually reduces the initial value until *Opt* is attained. Since the number of values a member communicates to its successor in the communication schedule is always incremental in GDH.1, the routing paths of successive members must be non-increasing w.r.t. hop count. In GDH.1 the last visited member M_n uses the established schedule backwards to forward the intermediate messages to the rest of members, until M_1 is reached. An approximation that accommodates this requirement and returns a tour with total cost not more than twice the cost of an optimal tour would be suitable for our purposes. A solution to the two goals discussed could be given by the ***closest point heuristic*** for building a TS tour to approximate (3.2). The closest point heuristic begins with a trivial cycle of an arbitrarily chosen vertex. At each step, a vertex *u* that is not on the cycle but whose distance to any vertex on the cycle (e.g. vertex *v*) is minimum is identified. We extend the cycle to include *u* by inserting *u* just after *v*. We repeat this process until all vertices are on the cycle. This heuristic returns a tour whose total cost is not more than twice this of an optimal tour.

If we fix the **backward schedule first**, so that the first edge selected in the cycle (minimum) is assigned to relay the maximum number of messages (*n*-1), the second edge is assigned to relay (*n*-2) messages, etc., we immediately satisfy the "non-incremental" requirement. We can obtain a ST by deleting any edge from a tour, and we already acquire the MST in our setting. Thus, the closest point heuristic has direct application to our problem. We use the **appropriate traversal method** to visit all vertices of the **ST** and establish the **GDH.1 backward schedule first**. Our aim is the following: the larger the message a member relays, the smallest the routing path it has to follow, in the average case. By examining the common traversal methods (*pre-order*, *in-order*, *post-order*), we select the *pre-order* tree walk. An intuitive reason for this is that a *pre-order* tree walk prints (visits) the root before the values in either sub-tree. So, it uses a rather greedy approach by adding the "best nodes" first, in a forward manner, the earliest possible in the generated backward schedule. On the contrary, the nature of the *in-order* and *post-order* walks is such that recursion prevents the immediate selection of the shortest paths in the tree between successive in the schedule nodes. Thus, longer paths (compared to the *pre-order* traversal) may end up connecting successive nodes, early in the generated schedule, when a better decision could have been taken. This is contrary to our second goal, according to which the shortest paths should be assigned to the nodes selected among the first for the backward schedule. Under this method, the **root** is the **last** member **visited**, whereas the **node** that **initiates** GDH.1 is the last node listed by the *pre-order* walk. By visiting the vertices indicated by the *pre-order* walk backwards, we obtain the forward GDH.1 schedule. Furthermore, if a node has already obtained information about the existence of a shorter (non-tree) path to the node requested, then it may use that path, otherwise the tree walk is executed in its normal mode.

We want to upper bound (3.2) by using a *wt* schedule that generates the backward GDH.1 path of the n nodes. The topology of the nodes determines the nature of the formed ST, and consequently the value of (3.2). Towards this end, we studied a few examples of particular ST and computed (3.2) over them. A few indicative cases considered are: the *single chain tree*, the *star tree of depth X*, the *fully balanced Z-ary tree*, etc. It will be shown that the value of (3.2) computed via pre-order traversal of any *ST* is upper bounded by n^2.

First of all, we start with computing (3.2) for two extreme pathological ST cases, i.e. the *single chain tree* and the *star tree*, and then we expand our computations to a *fully balanced binary tree*, a *fully balance Z-ary tree*, and an *arbitrary tree*:

Single Chain Tree: $\sum_{i=1}^{n-1} i = \frac{n\times(n-1)}{2}$.

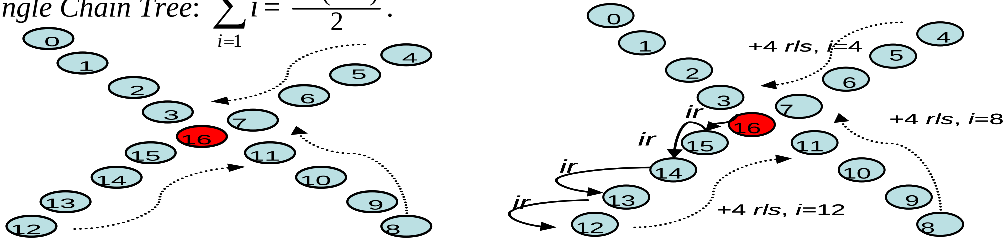


Figure 3.1. Star Tree with depth x=4, n=16.

Star Tree – depth1: $(n\text{-}1)+2\times\sum_{i=1}^{n-2} i=$ $(n\text{-}1)^2$, or $\sum_{i=1}^{n-1} i\,(ir$ data)+$\sum_{i=1}^{n-2} i$ (*rls* data) =

$\frac{n\times(n-1)}{2}+\frac{(n-1)\times(n-2)}{2}$ = $(n\text{-}1)^2$.

Star Tree – depth2: $(n\text{-}1)+(n\text{-}2)+(\sum_{i=2}^{n/2} 3(n-2i+1)+(n-2i)\,)+2\times 1=$

$(2n\text{-}1)+3(\frac{n(n-3)}{4})+(\frac{(n-2)(n-3)}{4})$ = $(n\text{-}1)(n\text{-}\frac{1}{2})$.

Star Tree – depthX: Obviously, the data communicated to the intended recipients is as designated by the original logical protocol itself (i.e. $\frac{n\times(n-1)}{2}$). Whenever the edge of a branch is reached (length x), the member associated with the most remote vertex of this branch relays its data through (x+1) members to the first unvisited vertex of an unvisited neighbor branch.

$$\sum_{j=0}^{\frac{n-1}{X}-1}(\sum_{i=1}^{x}(n-jx-i))+x(n-(j+1)x)=\frac{n\times(n-1)}{2}+x\frac{n}{2}\left(\frac{n-1}{x}-1\right)=(n-1)(n-\tfrac{x}{2})-\tfrac{x}{2}=n(n\text{-}1)\text{-}\frac{xn}{2}$$

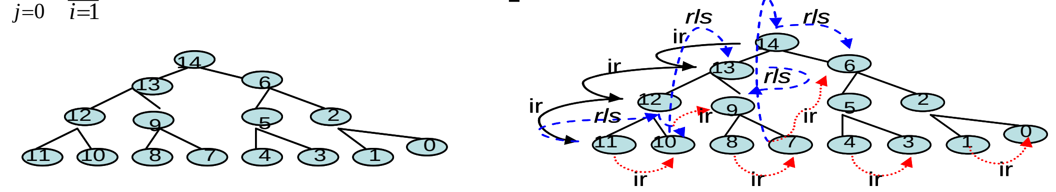


Figure 3.2. Fully balanced binary tree with 15 members. The black arrow denotes actual message communication between intended recipients (ir), the red arrow denotes virtual message communication between irs, the blue arrow simulates the communication designated by the red arrow, including a number of required relays (rls) between the end nodes (irs).

Fully Balanced Binary Tree, height $h=log_2\frac{n}{2}$. We combine the cases described above, following a similar reasoning to obtain the following expression: A+B+C. A is the data communicated to the intended recipients is as designated by the original logical protocol itself (i.e. $\frac{n\times(n-1)}{2}$). Every member has at least one relay including the intended destination. If every member needed only one relay to communicate its intended message, namely the intended recipient, then the physical execution of GDH.1 would coincide with the logical one. Then we would only need to compute value A. Values B or C become non-zero only if at least one member (differently placed in the tree for *B* and *C* respectively) has more than one relays. The value *A* is in effect the communication cost from the original logical execution of GDH.1. Value *B* deals with the data relayed by members associated with the leftmost leaf of a given branch. The message associated with such a member is relayed only twice. The first contribution is included in metric *A* and the second in metric *B*. We exploit the mirror symmetry of the fully balanced tree by summing together for metric *B* every two members that relay $f(i)$ and $(n-h-f(i))$ messages respectively. The function f maps branch i - to which the leftmost offspring belongs - to the number of messages produced by this offspring according to GDH.1. A branch is defined as a unique ramification of the tree starting from the root and ending at the parent of two leaf siblings. We note here that the enumeration of branches takes place at the one before last tree level, from right to left. In a fully balanced binary tree there exist (n+1)/4 such branches. *C* is the data relayed by members associated with the rightmost leaf of a given branch. We derive value *C* in a similar fashion. The basic difference is that the message associated with the rightmost member of a given branch is relayed more that twice, and the number of relays involved is not fixed but depend on the particular branch to which the rightmost member belongs. Following the same reasoning, we see that the mirror symmetry of the fully balanced tree can be explored in combination with the nature of the GDH.1 upward stage messages, to derive a simple, closed formula for *C*.

Lemma 1: In any fully balanced binary tree where the node *id*s are assigned from higher to lower via a pre-order tree visit the expression for function f can be determined as follows:

$g(i)$ = $f(i+1)$ - $f(i)$ = h_{cs}+1, where: $1 \le i < \frac{(n+1)}{4}$, and h_{cs} is the height of the common sub-tree that includes the branches: i, i+1.

Because of the symmetry of the fully balanced binary tree the following equalities hold:

$f(i+1)$ - $f(i) = f((\frac{(n+1)}{8})+1+i+1)$ - $f((\frac{(n+1)}{8})+1+i) = f((\frac{(n+1)}{4})+2-i)$ - $f((\frac{(n+1)}{4})-i)$

(3.8).

$f(i+1)$ - $f(i) = h_{cs}$+1, (3.9).

$f_l(1)$ = 1 (left offspring) and $f_r(1)$ = 0 (right offspring).

Proof: We now show by **induction** that (3.8), (3.9) are true for a fully balanced binary tree of any size if we fix the backward schedule first by means of a pre-order walk.

Initial Step: Given (3.8), (3.9) and the initial state, it is very simple to compute f or *g* at any point. For a binary tree with 7 nodes we obtain *g*(1)=3, and for a binary tree with 15 nodes we get: *g*(1)=3, *g*(2)=4, and *g*(3)=3. Indeed, in a fully balanced binary tree of 7 nodes we get: $f_l(1)$ = 1, $f_r(1)$ = 0, $f_l(2)$ =4, $f_r(2)$ =3 and (3.8), (3.9) hold. In a fully balanced binary tree of 15 nodes (Fig. 2) we get: $f_l(1)$ = 1, $f_r(1)$ = 0, $f_l(2)$ =4, $f_r(2)$ =3, $f_l(3)$ = 8, $f_r(3)$ = 7, $f_l(4)$ =11, $f_r(4)$ =10, and obviously (3.8), (3.9) still hold.

Intermediate Step: We now assume that (3.8), (3.9) hold for sub-tree *R*.

We will show that if we have computed function *f* or *g* at any point for a tree *R* of size (*n*-1) with member *id*s from (*n*-2) down to 0, assigned through a “backward pre-order” visit, then we can immediately apply these results to a fully balanced binary tree *T* of size (2*n*-1). Towards this end we construct tree *L* of size (*n*-1) by replicating *R*. Then the roots of these two sub-trees become the left and right offspring of the new root of tree *T*, and trees *L* and *R* become sub-trees of *T*. A pre-order walk visits recursively first the root of a tree, then the left and finally the right sub-tree. Since the assignment of *id*s is done backwards, the right sub-tree *R* of tree *T*, will be visited last, and its nodes will be assigned *id*s from (*n*-2) down to 0 via the pre-order visit. Hence, nodes in *R* that also belong to *T* are assigned the same *id*s as before and (3.8), (3.9) still hold for *R*, and thus for the right sub-tree of *T*. Since the two sub-trees are symmetric, (3.8), (3.9) hold for the left sub-tree of *T* by definition. So far function *g* has been computed for all branches of *T* except for these at points $\frac{(n+1)}{4}$ and $\frac{(n+1)}{4}+1$, each of which belong to a different sub-tree *L* or *R*, right in the middle of *T*. The last visited (by the backward pre-order) member of sub-tree *L* is the right offspring of branch 1 in *L* or the right offspring of branch ($\frac{(n+1)}{4}+1$) in *T*. This member is assigned an *id* with number (n-1) and its sibling on the left an *id* with number *n*. To determine $g(\frac{(n+1)}{4})$ we also need to compute the *id*s of members that belong to branch $\frac{(n+1)}{4}$ of *T* (or *R*). Since we know that the *id* of the root in *R* is (n-2) and also $h_R = h_T - 1 = log_2\,(\frac{(n+1)}{2})$, we easily compute the *id*s of the offspring of the leftmost branch ($\frac{(n+1)}{4}$) of *R*: $f_l(\frac{n+1}{4}) = (n-2)- h_R = n-1-h_T$, $f_r(\frac{n+1}{4}) = n-2-h_T$. For the right offspring we compute *g* at the missing point: $g(\frac{(n+1)}{4}) = (n-1)-(n-2-h_T) = h_T+1$.

We have just shown that (3.8), (3.9) hold for tree *T* as well, and this concludes the final step of our induction. Hence, our initial argument (*Lemma 1*) has been proven by induction.

It is easy to see that two fully binary sub-trees *L* and *R* of size (n-1) originate from the left and the right root offspring of the initial binary tree of size (2n-1). If we observe how the *id*s are assigned for any binary tree under our assumptions, we see that each of the two sub-trees *L* and *R* is already a fully balanced tree of size (n-1) that fulfills the required rules for the assigned *id*s. In fact, if we decrease the assigned *id*s in *L* by the number (n-1)/2 then we observe that sub-tree *L* coincides with sub-tree *R*. For example the tree in Figure 2 is divided to two sub-trees *L* (with leaf *id*s: 11, 10, 8, 7) and *R* (with leaf *id*s 4, 3, 1, 0). If we subtract the number (n-1)/2 = 7 from all *id*s of sub-tree *L*, it is easy to see that *L*=*R*. Then, we are provided with an additional argument to prove that (3.8) and (3.9) hold for the tree of size 15 as well. This is true indeed, because under the pre-order tree walk we recursively visit first the root of a given tree, then its left sub-tree and then its right sub-tree, starting from the root of each sub-tree. Hence, *id*s are assigned so that we can construct a “backwards pre-ordered” balanced tree *T* of size (2*n*-1) from a “backwards pre-ordered” balanced tree *L* of size (*n*-1): First we construct tree *R* by replicating tree *L*. Then we assign *id*s to *R* as follows: every member in *R* gets the *id* of the associated member in *L*, increased by the number (*n*-1)/2. Shifting the *id*s assigned in *L* by a fixed number obviously does not alter the results of (3.8), (3.9). Finally, the root in *T* is assigned the number (2*n*-1). This way, from a provably “backward pre-ordered” fully balanced tree of size *K*, we can construct “backward pre-ordered” trees of any size $B \geq 2K$. Equivalently any “backward pre-ordered” fully balanced binary tree can be recursively reduced to a “backward pre-ordered” tree of smaller size. The latter argument helps to compute simply and fast functions *f* and *g* for a fully balanced binary tree of any size. Below we abbreviate a number of these computations:

Tree Size: 7 g^T = (**3**)

Tree Size: 15 g^T =(3,**4**,3)

Tree Size: 31 g^T =(3,4,3,**5**,3,4,3)

Tree Size: 63 g^T =(3,4,3,5,3,4,3,**6**,3,4,3,5,3,4,3)

Tree Size: 127 g^T =(3,4,3,5,3,4,3,6,3,4,3,5,3,4,3,**7**,3,4,3,5,3,4,3,6,3,4,3,5,3,4,3), etc.

Notation: Given a tree member *u*, we denote the height of the sub-tree originating from member *u* as *h*(*u*). The height of the original fully balanced binary tree is denoted as *h*. Furthermore, let the parent *v* of node *u* be denoted as: *v* = *p*(*u*).

We are now ready to compute all three metrics *A*, *B*, *C*:

$$A = \sum_{i=1}^{n-1} i = \frac{n\times(n-1)}{2}.$$

$$B = \sum_{i=1}^{(n+1)/8} (n-h-f(i)) + (n-h-(n-h-f(i))) = \sum_{i=1}^{(n+1)/8} (n-h) == \tfrac{(n+1)}{8}(n-h).$$

The remaining contribution of the right-sibling leaves to the combined communication cost is computed by metric *C*. In fact, *C* computes the relay messages that originate from the right-sibling leaves that have not been included previously in *A*. The number of relays involved between a right-sibling leaf *u* and the next un-visited member *v* (or root of the next unvisited sub-tree) is: h_{rel} = (h(p(v))+1). Observe that the minimum number of such relays involved is three. However, one of these relays (the contribution of p(v) to v) is already computed in metric *A*. Hence, the hop-wise contribution of member *u* for metric *C* hop-wise is h(p(v)). The minimum number of relays originating from any right-sibling leaf is 2 for metric *C*.

Observation: A pre-order traversal visits tree nodes in such a way that the *id* difference between two successive left (or right) leaf siblings *u* and *v*, defined by function *g*, is determined by h_{rel} (u). Given that observation, the expression for the hop-wise contribution of member *u* for metric *C* can be equivalently expressed as: (h_{rel} -1) = (g(u) -1). Now we are ready to compute metric *C* as shown below:

$C = \frac{(n-1)}{2} h + \sum_{i=2}^{h-1} i \times \sum_{j=1}^{2^{h-i-1}} ((n-h-f(k_j)+f(k_j+1))$, where k_j are such that $h(k_j) = i$, and i is the hop-wise contribution of each member involved in *C*.

$$C = \frac{(n-1)}{2} h + (n\text{-}h) \sum_{i=2}^{h-1} 2^{h-i-1} \times i + \sum_{i=2}^{h-1} i \times \sum_{j=1}^{2^{h-i-1}} (f(k_j+1) - f(k_j))$$

$$C = \frac{(n-1)}{2} h + (n\text{-}h) \sum_{i=2}^{h-1} 2^{h-i-1} \times i + \sum_{i=2}^{h-1} i \times \sum_{j=1}^{2^{h-i-1}} g(k_j) \approx \frac{(n-1)}{2} h + (n\text{-}h) \times 2^{h-1} \sum_{i=2}^{h-1} i \times (\tfrac{1}{2})^i + 2^{h-1}.$$

Notation:

$\sum_{k=0}^{n} x^k$ is a geometric or exponential series and has the value: $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$ (3.10).

For |x| <1, we have the infinite decreasing geometric series: $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ (3.11).

Additional formulas can be obtained by integrating or differentiating formulas (3.10), (3.11). If we differentiate both sides of (3.11) and multiply by *x* we get:

$$\sum_{k=0}^{n} kx^k = \frac{x}{(1-x)^2} - \frac{x^n}{(1-x)^2} (x^2 + (n+1)x - (n+1)) \quad (3.12).$$

If we differentiate both sides of (3.12) and multiply by *x* we get: $\sum_{k=0,x<1}^{\infty} kx^k = \frac{x}{(1-x)^2}$ (3.13).

Similarly, if we differentiate both sites of (3.13) and multiply by *x* we get:

$$\sum_{k=0}^{n} k^2 x^k = \frac{x(1+x)}{(1-x)^3} + \frac{nx^{n+3}+(n^2-n-2)x^{n+2}-(2n^2+n-1)x^{n+1}+n(n+1)x^{n+1}}{(1-x)^3} \qquad (3.14).$$

Similarly, if we differentiate both sites of (3.14) and multiply by *x* we get:

$$\sum_{k=0,x<1}^{\infty} k^2 x^k = \frac{x(1+x)}{(1-x)^3} \qquad (3.15).$$

In order to upper bound metric *C* we use (3.10) and (3.12), by substituting *x* with $\frac{1}{2}$ and *n* with (*h*-1). For these values, (3.10), (3.12) can be replaced by (3.13), (3.15) respectively:

$$C \approx \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{n}{4} \times \left(\sum_{i=0}^{h-1} i \times (\tfrac{1}{2})^i - \tfrac{1}{2}\right) + 2^{h\text{-}1} \sum_{i=2}^{h-1} i \times 2^{-i} \times g(k_j)$$

$$C \approx \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{n}{4} \times \frac{3}{2} + 2^{h\text{-}1} \sum_{i=2}^{h-1} i \times 2^{-i} \times (i+1)$$

$$C \approx \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{n}{4} \times \frac{3}{2} + 2^{h-1} \sum_{i=1}^{h-1} (2^{-i} \times i - \tfrac{1}{2}) + 2^{h-1} \sum_{i=2}^{h-1} 2^{-i} \times i^2$$

$$C \approx \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{n}{4} \times \frac{3}{2} + \frac{n}{4} \times \left(\frac{\frac{1}{2}}{(1-\frac{1}{2})^2} - \frac{1}{2}\right) + \frac{n}{4}\left(\frac{\frac{1}{2}(1+\frac{1}{2})}{(1-\frac{1}{2})^3} - \frac{1}{2}\right)$$

$$C \approx \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{n}{4} \times \frac{3}{2} + \frac{n}{4} \times \frac{11}{2} = \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{3n}{8} + \frac{11n}{8}.$$

We now sum all metrics:

$$A+B+C = \frac{n \times (n-1)}{2} + \frac{(n+1)}{8}(n-h) + \frac{(n-1)}{2} h + (n\text{-}h) \times \frac{3n}{8} + \frac{11n}{8} = (n^2 + n(1+\frac{h}{4}) - \frac{5h}{8}) \quad (3.16).$$

Fully Balanced Z-ary Tree: The method for deriving the closed cost formula is similar to this of the binary tree. However, if a node has *z* children, only the last child relays its message more than two hops. The following expression holds for this type of tree: $n = \frac{z^{h+1}-1}{z-1}$.

Working in a similar fashion we obtain the expression: *A*+*B*+*C*, where *A* is:

$$A= \sum_{i=1}^{n-1} i = \frac{n \times (n-1)}{2} .$$

$$B= (\tfrac{z-1}{2}) \times \sum_{i=1}^{z^{h-1}} (n-h-f(i)) + (n-h-(n-h-f(i))) = (z-1) \sum_{i=1}^{z^{h-1}/2} (n-h) = \frac{z^{h-1}(z-1)}{2}(n-h)$$

$$B = \frac{1}{z^2}\left(z^{h+2} - z^{h+1}\right)\frac{(n-h)}{2} = \frac{1}{z^2}(z + z(z^{h+1}-1) - (z^{h+1}-1) - 1)\frac{(n-h)}{2}$$

$$B = \frac{1}{z^2}(z-1)(n(z-1)+1)\frac{(n-h)}{2} = \frac{z-1}{2z^2}(n^2(z-1) + n(1+h-hz) - h) .$$

$C = h \times \sum_{i=1}^{z-1} \frac{(n-1)}{z} \times i + \sum_{i=2}^{h-1} \frac{(z-1)}{2} \times i \times \sum_{j=1}^{z^{h-i}} ((n-h-f(k_j)) + f(k_j+1))$, where k_j are such that $h(k_j) = i$, and i is the hop-wise contribution of each member involved in *C*.

In this case the following holds: $g(k_j) = f(k_j+1) - f(k_j) = i + (z-1)$.

$$C = h \times \frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1) \times \sum_{i=2}^{h-1} z^{h-i} \times i + \sum_{i=2}^{h-1} \frac{(z-1)}{2} \times i \times \sum_{j=1}^{z^{h-i}} (f(k_j+1) - f(k_j))$$

$$C = h \times \frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1) \times z^h \sum_{i=2}^{h-1} z^{-i} \times i + \sum_{i=2}^{h-1} \frac{(z-1)}{2} \times i \times \sum_{j=1}^{z^{h-i}} g(k_j)$$

In order to upper bound metric *C* we want to use (3.12) and (3.15), by substituting *x* with $\frac{1}{z} \leq \frac{1}{3}$ and *n* with (*h*-1). However, for these values, (3.12) and (3.14) can be replaced by (3.13) and (3.15) respectively:

$$C < h \times \frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1) \times z^h (\sum_{i=0}^{\infty} z^{-i} \times i - \tfrac{1}{z}) + \frac{(z-1)}{2} \times \sum_{i=2}^{h-1} i \times z^{h-i} \times g(k_j)$$

$$C < h\times \frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1)\times z^{h}\times \left(\frac{\frac{1}{z}}{(1-\frac{1}{z})^2} - \frac{1}{z}\right) + \frac{(z-1)}{2} \times \sum_{i=2}^{h-1} i \times z^{h-i} \times (i+(z-1))$$

$$C<h\frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1)\times z^{h\text{-}1}\times \left(\frac{2z-1}{(z-1)^2}\right) + \frac{(z-1)}{2} \times z^{h}\times \sum_{i=2}^{h-1} i^2 \times z^{-i} + \frac{1}{2} \times z^{h}\times \sum_{i=2}^{h-1} i \times z^{-i}$$

$$C<h\frac{(n-1)}{z}\ \frac{z(z-1)}{2} + \frac{(n-h)}{2} (z\text{-}1)z^{h\text{-}1}\left(\frac{2z-1}{(z-1)^2}\right) + \frac{(z-1)}{2} z^{h}\left(\sum_{i=0}^{h-1} i^2 \times z^{-i} - \frac{1}{z}\right) + \frac{1}{2} z^{h}\left(\sum_{i=0}^{\infty} z^{-i} \times i - \frac{1}{z}\right)$$

$$C < h\times \frac{(n-1)}{z} \times \frac{z(z-1)}{2} + \frac{(n-h)}{2} \times (z\text{-}1)\times z^{h\text{-}1}\times \left(\frac{2z-1}{(z-1)^2}\right) + \frac{(z-1)}{2} \times z^{h}\times \frac{\frac{1}{z}(1+\frac{1}{z})}{(1-\frac{1}{z})^3} + \frac{1}{2}\ z^{h\text{-}1}\times \left(\frac{2z-1}{(z-1)^2}\right)$$

$$C < h\times(n\text{-}1)\times \frac{(z-1)}{2} + \frac{(n-h)}{2} \times z^{h\text{-}1}\times \frac{(2z-1)}{(z-1)} + \frac{1}{2} z^{h\text{-}1}\times \frac{1}{(z-1)^2} (z^3 + z^2 + 2z\text{-}1)$$

Given that ($n(z$-1)+1) = z^{h+1}, we get:

$$C < h\times(n\text{-}1)\times \frac{(z-1)}{2} + \frac{(n-h)}{2} \times\left(\frac{n(z-1)}{z^2} + \frac{1}{z^2}\right)\times \frac{(2z-1)}{(z-1)} + \frac{1}{2}\left(\frac{n(z-1)}{z^2} + \frac{1}{z^2}\right)\times \frac{1}{(z-1)^2} (z^3 + z^2 + 2z\text{-}1)$$

$$C<\frac{(2z-1)}{2z^2} n^2 + \left(\frac{h(z-1)}{2} - \frac{(2z-1)h}{2z^2} + \frac{1}{2(z-1)} \left(z+1+\frac{4}{z} - \frac{3}{z^2}\right)\right)n - \frac{(z-1)}{2} - \frac{(2z-1)h}{2z^2(z-1)} + \frac{z+1+\frac{2}{z}-\frac{1}{z^2}}{2(z-1)^2}\ .$$

Combining all three formulae for metrics *A*, *B*, *C*, computed above we obtain:

$$A{+}B{+}C \quad < \frac{n\times(n-1)}{2} + \frac{z-1}{2z^2}(n^2(z-1) + n(1+h-hz) - h) + \frac{(2z-1)}{2z^2} n^2 \quad + \quad \left(\frac{(z-1)}{2} h \quad - \frac{(2z-1)}{2z^2} h\right.$$

$$\left. + \frac{1}{2(z-1)} \left(z+1+\frac{4}{z} - \frac{3}{z^2}\right)\right)n - \frac{(z-1)}{2} - \frac{(2z-1)}{2z^2(z-1)} h + \frac{1}{2(z-1)^2} \left(z+1+\frac{2}{z} - \frac{1}{z^2}\right) \Rightarrow$$

$$A{+}B{+}C < n^2 + \left(\frac{(z-2)}{2} h + \frac{1}{2(z-1)} \left(3+\frac{4}{z} - \frac{4}{z^2}\right)\right)n - \frac{1}{2(z-1)} h - \frac{(z-1)}{2} + \frac{1}{2(z-1)^2} \left(z+1+\frac{2}{z} - \frac{1}{z^2}\right) \Rightarrow$$

$$A{+}B{+}C < n^2 + O\left(\frac{(z-2)}{2} h + \frac{2}{(z-1)}\right)n + O\left(1\text{-}\, z - \frac{1}{2(z-1)} h\right) \qquad (3.17).$$

Observation 1: The complexity of implementing GDH.1 via traversing any *Z*-ary balanced tree, **never exceeds the upper bound** of: $n^2 + O\left(\frac{(z-2)}{2} h + \frac{2}{(z-1)}\right)n + O\left(1\text{-}\, z - \frac{1}{2(z-1)} h\right)$, and its order is characterized by the factor: n^2 .

Observation 2: From a fully balanced *Z*-ary tree of *n* nodes, an arbitrary tree of *n* nodes can be constructed, by successively adding and removing leaves in the *Z*-ary tree at any level. For example, the transition from a *Z*-ary tree to a (*Z*+1)-ary tree of the same number of nodes *n*, does not alter the upper limit of the bandwidth complexity at any instance. In other words, the various instances of an arbitrary spanning tree of *n* nodes that can be produced, while transitioning from a balanced *Z*-ary tree to a (*Z*+1)-ary tree and vice versa, incur communication complexity upper limited by n^2 as well. This is the case, when the pre-order traversal method is used to visit all nodes of the arbitrary spanning tree.

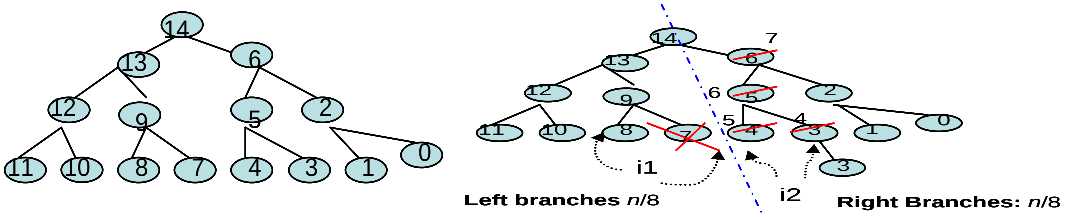


Figure 3.3. Arbitrary tree resulting from a fully balanced binary tree when one member is removed from branch i1, and a new one is added under branch i2.

As an example, we now compute the combined communication cost (or equivalently *A*+*B*+*C*) when GDH.1 is applied on an arbitrary tree of *n*=15 members that results from a fully balanced binary tree after:

(a) first removing a leaf from branch *i1* and adding a new member under one of the offspring of branch *i2*, or

(b) first adding a new member under one of the offspring of branch *i1* and then removing a leaf from branch *i2*.

Case (a):

Observation 1: If we traverse sequentially the branches of the tree, starting from branch 1, then we observe the following: all members that belong to branches *j*<*i1* maintain their previous *id*s, members that are leaves under branch *i1* as well as all members that belong to any branch between (*i1*+1) and *i2* have their *id*s shifted (increased) and equivalently the number of messages they need to relay increases by 1. Finally, all remaining members that are visited by the pre-order walk after the new member (placed as a leaf under any of the previous leaves associated with branch *i2*), maintain their previous *id*s.

Observation 2: Depending on whether we modify the left or right offspring of a given branch *i1* or *i2*, the margins for the shifted *id*s are differentiated. In Table 3.4 we show how members' previous *id*s are modified around those margins for all possible scenarios. By (+1) we denote the operation of increasing the numbering of previous members' *id*s by 1, and by (0) we denote the operation of decreasing the previously "increased by 1" members' *id*s so that their new *id*s now coincide with their previously assigned *id*s.

Scenarios:	*B*	*C*
SL1-: *i1*: left offspring	(*i1*+1) change (+1)	*I1* changes (+1)
SL2 -: *i1:* right offspring	(*i1*+1) change (+1)	(*i1*+1) changes (+1)
SR1-: *i2*: left offspring	*i2* changes (0)	*I2* changes (0)
SR2-: *i2*: right offspring	(*i2*+1) change (0)	*I2* changes (0)

Table 3.4. Detailed demonstration of how members' ids are modified (and consequently metrics B and C) after a left or right offspring is removed (branch i1) or added (branch i2).

Based on these two observations, we re-compute values *A*, *B*, and *C*, for the two cases distinguished below:

Case *a*.1: Assume that $|\, i2\text{-}\frac{n}{8} \,| \;\leq i1$. Since the contributions of nodes designated by the logical GDH.1 protocol are computed in *A*, it is obvious that *A* remains unchanged for both trees as long as they maintain the same number of members. However, values *B* and *C* do potentially change, since they include contributions from a varying subset of members. Based on *observation 1*, in order to exploit the remaining mirror symmetry regarding the tree leaves and the number of messages relayed by each, we separate the original common branch space in the following intervals: $[1, \frac{n}{4}$ - $i2]$, $[\frac{n}{4}$ - $i2{+}1, i1]$, $[i1{+}1, \frac{n}{8}]$. Any members that belong to any of these intervals share common properties with the corresponding members that belongs to the mirror interval (i.e. the mirror of $[1, \frac{n}{4}$ - $i2]$ is $[i2, \frac{n}{4}]$). These common properties are either the symmetry witnessed in previous cases, or a similar symmetry, with a slight fixed offset as we will see now. Then, we re-compute metric *B* for the random tree, for all scenarios ($SL\{1,2\}\times SR\{1,2\}$) denoted as B_r, by splitting the original summation in three smaller ones so that each corresponds to one of the aforementioned intervals. For metric *B* only two scenarios are distinguished, since *SL1=SL2=SL*: *SLSR1* and *SLSR2*.

B_r (*SLSR1*) $=B_1+B_2+B_3$ =

$$= \sum_{i=1}^{n/4-i2} (n-h-f(i))+f(i)) + \sum_{i=n/4-i2+1}^{i1} (n-h-f(i))+(f(i)+1) + \sum_{i=i1+1}^{n/8} (n-h-f(i)+1))+(f(i)+1) =$$

$$= \sum_{i=1}^{n/4-i2} (n-h) + \sum_{i=n/4-i2+1}^{i1} (n-h+1) + \sum_{i=i1+1}^{n/8} (n-h+2) = \frac{(n+1)}{8}(n-h) + (i1- \frac{n}{4} + i2+ 2\frac{n}{8} -2\, i1)$$

B_r (*SLSR1*) = $B + (i2\text{-}i1)$.

Similarly: B_r(*SLSR2*) = $B_1+B_2+B_3 = \sum_{i=1}^{n/4-i2-1} (n-h) + \sum_{i=n/4-i2}^{i1} (n-h+1) + \sum_{i=i1+1}^{n/8} (n-h+2) =$

B_r(*SLSR2*) = $\frac{(n+1)}{8}(n-h) + (i1\text{-} \frac{n}{4} + i2 +1)+2(\frac{n}{8} \text{-}\, i1) = B + (i2\text{-}i1+1)$.

For metric *C* only two scenarios are distinguished, since *SR1=SR2=SR*: *SL1SR* and *SL2SR*.

C_r (*SL1SR*) = $C_0 + C_1+C_2+C_3 = C_0 + C_{N0}$, where:

$$C_0 = \left\{ \begin{array}{cc} \frac{(n-1)}{2}h & i1, i2 < \frac{n}{4} \wedge i1, i2 > \frac{n}{4} \\ \frac{(n+1)}{2}h & i1 < \frac{n}{4} < i2 \\ 0 & i1 = \frac{n}{4} \end{array} \right\}$$

C_{N0} (*SL1SR*) = $\sum_{i=2}^{h-1} i \times \sum_{j=1}^{2^{h-i-1}} ((n-h-x(k_j))+y(k_j))$, where k_j are such that $h(k_j) = i$, and *i* is the hop-wise contribution of each member involved in *C*. Working in a similar fashion as before, and extending the results obtained for metric *C* for a fully binary balanced tree as shown for metric *B* we obtain:

$$C_{N0}(SL1SR)=(n\text{-}h)\times\frac{3n}{8}+\sum_{i=2}^{h_{n/4-i2}-1} i\times\sum_{j=1}^{2^{h-i-1}}(f(k_j+1)-f(k_j))+$$

$$\sum_{i=h_{n/4-i2}}^{h_{i1}} i\times\sum_{j=1}^{2^{h-i-1}}(f(k_j+1)-f(k_j)+1)+\sum_{h_{i1}}^{h-1} i\times\sum_{j=1}^{2^{h-i-1}}(f(k_j+1)-f(k_j)+2)=(n\text{-}h)\times\frac{3n}{8}+$$

$$\sum_{i=2}^{h_{n/4-i2}-1} i\times\sum_{j=1}^{2^{h-i-1}} g(k_j)+\sum_{i=h_{n/4-i2}}^{h_{i1}} i\times\sum_{j=1}^{2^{h-i-1}}(g(k_j)+1)\;+\sum_{h_{i1}}^{h-1} i\times\sum_{j=1}^{2^{h-i-1}}(g(k_j)+2)=>$$

$$C_{N0}(SL1SR)=(n\text{-}h)\times\frac{3n}{8}+\sum_{i=2}^{h-1} i\times\sum_{j=1}^{2^{h-i-1}} g(k_j)\;+\sum_{i=h_{n/4-i2}}^{h_{i1}} i\times\sum_{j=1}^{2^{h-i-1}} 1\;+\sum_{h_{i1}}^{h-1} i\times\sum_{j=1}^{2^{h-i-1}} 2\;=>$$

$$C_{N0}(SL1SR)=(n\text{-}h)\frac{3n}{8}+\frac{n}{4}\,\frac{11}{2}+\frac{n}{4}((\sum_{i=0}^{h_{i1}} i\times(\tfrac{1}{2})^i\text{-}(\sum_{i=0}^{h_{n/4-i2}} i\times(\tfrac{1}{2})^i)+\frac{n}{2}((\sum_{i=0}^{h-1} i\times(\tfrac{1}{2})^i\text{-}(\sum_{i=0}^{h_{i1+1}} i\times(\tfrac{1}{2})^i)$$

$$C_{N0}(SL1SR)\;=\;(n\text{-}h)\times\frac{3n}{8}\;+\;\frac{n}{4}\times\frac{11}{2}\;+\;\frac{n}{4}(\frac{1}{2^{h_{i1}}}(1+2h_{i1})\text{ - }\frac{1}{2^{h_{n/4-i2}}}(1+2h_{n/4-i2}))\;+$$

$$\frac{n}{2}(\frac{1}{2^{h-1}}(1+2(h-1))\text{ - }\frac{1}{2^{h_{i1+1}}}(1+2h_{i1+1}))\;=>$$

$$C_{N0}(SL1SR)=(n\text{-}h)\,\frac{3n}{8}+\frac{n}{4}\times\frac{11}{2}+\frac{n}{4}\{0,\frac{1}{2^{h_{i1}}}(1+2h_{i1})\}\text{-}\frac{n}{4}\,\frac{1}{2^{h_{n/4-i2}}}(1+2h_{n/4-i2}))+(2+4(h\text{-}1))$$

Taking into account that $2^{h_{i1+1}}=\left\{2^{h_{i1}}-1,2^{h_{i1}}+1\right\}$, we obtain the following upper bound for

$$C_{N0}(SL1SR)\text{: }(n\text{-}h)\frac{3n}{8}+\frac{n}{4}\,\frac{11}{2}+\frac{n}{4}\{0,\frac{1}{2^{h_{i1}}}(3+4h_{i1})\}\text{- }\frac{n}{4}\,\frac{1}{2^{h_{n/4-i2}}}(1+2h_{n/4-i2}))+(2+4(h\text{-}1))$$

$$C_{N0}(SL1SR)<(n\text{-}h)\times\frac{3n}{8}\;+\;\frac{n}{4}\times\frac{11}{2}\;+\mathrm{O}(\frac{n}{2^{h_{i1}}})+((2+4(h-1)).$$

Similar are the computations for $C_{N0}(SL2SR)$.

Finally we obtain for metric C_r $(SL1SR)$:

$$C_r\ (SL1SR) < \left\{\begin{array}{cc} \frac{(n-1)}{2}h & i1, i2 < \frac{n}{4} \wedge i1, i2 > \frac{n}{4} \\ \frac{(n+1)}{2}h & i1 < \frac{n}{4} < i2 \\ 0 & i1 = \frac{n}{4} \end{array}\right\} + (n\text{-}h)\frac{3n}{8} + \frac{n}{4}\frac{11}{2} + \mathrm{O}(\frac{n}{2^{h_{i1}}}) + (2+4(h\text{-}1))$$

$$C_r\ (SL1SR) < \left\{\begin{array}{cc} C + O(\frac{n}{2^{h_{i1}}}) + (2+4(h-1)) & i1, i2 < \frac{n}{4} \wedge i1, i2 > \frac{n}{4} \\ C + O(\frac{n}{2^{h_{i1}}}) + (2+5(h-1)) & i1 < \frac{n}{4} < i2 \\ C - (\frac{n-1}{2})h + O(\frac{n}{2^{h_{i1}}}) + (2+4(h-1)) & i1 = \frac{n}{4} \end{array}\right\}$$

Hence: $A+B_r\ (SLSR1)+C_r\ (SL1SR) <$

$$\left\{\begin{array}{cc} A+B+C+O(\frac{n}{2^{h_{i1}}}) + (i2-i1) + (2+4(h-1)) & i1, i2 < \frac{n}{4} \wedge i1, i2 > \frac{n}{4} \\ A+B+C+O(\frac{n}{2^{h_{i1}}}) + (i2-i1) + (2+5(h-1)) & i1 < \frac{n}{4} < i2 \\ A+B+C-(\frac{n-1}{2})h + O(\frac{n}{2^{h_{i1}}}) + (i2-i1) + (2+4(h-1)) & i1 = \frac{n}{4} \end{array}\right\}$$

It is obvious that in this case the combined communication and routing cost of the transition from a balanced binary tree to an arbitrary one, is affected only by a constant factor c.

Similar are the results for $C_{N0}(SL2SR)$

Case *a*.2: Assume that $|\ i2\text{-}\frac{n}{8}\ | > i1$.

The computations are conducted in a similar fashion as these for **Case *a*.1**, the results obtained are similar with these obtained for **Case *a*.1**, and the resulting complexity is practically the same. Therefore, we skip the analysis associated with Case *a*.2.

Case (b):

Observation 1: If we traverse sequentially the branches of the tree, starting from branch 1, then we observe the following: all members that belong to branches *j*<*i1* maintain their previous *id*s, members that are leaves under branch *i1* as well as all members that belong to any branch between (*i1*+1) and *i2* have their *id*s shifted (decreased) by 1, and equivalently the number of messages they need to relay decreases by 1. Finally, all remaining members that are visited by the pre-order walk after the member that is removed from branch *i2* (placed as a leaf under any of the previous leaves associated with branch *i2*), maintain their previous *id*s.

Observation 2: Depending on whether we modify the left or right offspring of a given branch *i1* or *i2*, the margins for the shifted *id*s are differentiated. In Table 5 we show how members' previous *id*s are modified around those margins in detail for all possible scenarios. By (-1) we denote the operation of decreasing the numbering of previous members' *id*s by 1, and by (0) we denote the operation of increasing the previously "decreased by 1" members' *id*s so that their new *id*s now coincide with their previously assigned *id*s.

Scenarios:	*B*	*C*
SL1-: *i1*: left offspring	(*i1*+1) changes (-1)	*i1* changes (-1)
SL2 -: *i1:* right offspring	(*i1*+1) changes (-1)	(*i1*+1) changes (-1)
SR1-: *i2*: left offspring	*i2* changes (0)	*i2* changes (0)
SR2-: *i2*: right offspring	(*i2*+1) changes (0)	*i2* changes (0)

Table 3.5. Detailed demonstration of how members' ids are modified (and consequently metrics B and C) after a left or right offspring is added (branch i1) or removed (branch i2).

Based on these observations, we re-compute *A*, *B*, *C*, for the cases distinguished below:

Case 1: Assume that $| i2 - \frac{n}{8} | \leq i1$. Since the contributions of nodes designated by the logical GDH.1 protocol are computed in *A*, it can be seen that *A* remains unchanged for both trees as long as they maintain the same number of members. Metrics *B* and *C* do change, since they include contributions from a varying subset of members. Based on *observation 1*, in order to exploit the remaining mirror symmetry regarding the tree leaves and the number of messages relayed by each, we separate the original common branch space in the following intervals: $[1, \frac{n}{4} - i2]$, $[\frac{n}{4} - i2+1, i1]$, $[i1+1, \frac{n}{8}]$. Any members that belong to any of these intervals share common properties with the corresponding members that belong to the mirror interval. These common properties are either the symmetry that we witnessed in previous cases, or a similar symmetry, with a slight fixed offset. We re-compute metric *B* for the random tree, for all scenarios ($SL\{1,2\}\times SR\{1,2\}$) denoted as B_r, by splitting the original summation in three smaller ones, each corresponding to one of the aforementioned intervals. For metric *B* only two scenarios are distinguished, since *SL1=SL2=SL*: *SLSR1* and *SLSR2*.

$$B_r\ (SLSR1) = B_1+B_2+B_3 = \sum_{i=1}^{n/4-i2}(n-h-f(i))+f(i))+\sum_{i=n/4-i2+1}^{i1}(n-h-f(i))+(f(i)-1)+$$

$$\sum_{i=i1+1}^{n/8}(n-h-f(i)-1))+(f(i)-1)=\sum_{i=1}^{n/4-i2}(n-h)\ +\sum_{i=n/4-i2+1}^{i1}(n-h-1)\ +\sum_{i=i1+1}^{n/8}(n-h-2)$$

$$B_r\ (SLSR1) = \frac{(n+1)}{8}(n-h)\text{ - }(i1\text{-}\frac{n}{4}+i2+2\frac{n}{8}\text{-}2\ i1) = B\text{ - }(i2\text{-}i1).$$

Similarly: $B_r(SLSR2) = B_1+B_2+B_3 = \sum_{i=1}^{n/4-i2-1}(n-h)+\sum_{i=n/4-i2}^{i1}(n-h-1)+\sum_{i=i1+1}^{n/8}(n-h-2)=$

$$B_r(SLSR2) = \frac{(n+1)}{8}(n-h)\text{ - }(i1\text{-}\frac{n}{4}+i2+1)+2(\frac{n}{8}\text{ - }i1) = B\text{ - }(i2\text{-}i1+1).$$

For metric *C* only two scenarios are distinguished, since *SR1=SR2=SR*: *SL1SR* and *SL2SR*.

$C_r\ (SL1SR) = C_0 + C_1+C_2+C_3 = C_0 + C_{N0}$, where:

$$C_0 = \left\{\begin{array}{cc} \frac{(n-1)}{2}h & i1,i2<\frac{n}{4}\wedge i1,i2>\frac{n}{4} \\ \frac{(n+1)}{2}h & i1<\frac{n}{4}<i2 \\ 0 & i1=\frac{n}{4} \end{array}\right\}$$

$C_{N0}\ (SL1SR) = \sum_{i=2}^{h-1} i\times\sum_{j=1}^{2^{h-i-1}}((n-h-x(k_j))+y(k_j))$, where k_j are such that $h(k_j) = i$, and *i* is the hop-wise contribution of each member involved in *C*. Working similarly, and extending the results obtained for metric *C* for a fully binary balanced tree as shown for metric *B* we obtain: $C_{N0}(SL1SR) < (n\text{-}h)\times\frac{3n}{8} + \frac{n}{4}\times\frac{11}{2}\text{ - O}(\frac{n}{2^{h_{i1}}})\text{ -}(2+4(h\text{-}1))$.

Finally we obtain for metric $A+B_r\ (SLSR1)+C_r\ (SL1SR) <$

$$\left\{\begin{array}{cc} A+B+C-O(\frac{n}{2^{h_{i1}}})-(i2-i1)-(2+4(h-1)) & i1,i2<\frac{n}{4}\wedge i1,i2>\frac{n}{4} \\ A+B+C-O(\frac{n}{2^{h_{i1}}})-(i2-i1)-(2+3(h-1)) & i1<\frac{n}{4}<i2 \\ A+B+C-(\frac{n-1}{2})h-O(\frac{n}{2^{h_{i1}}})-(i2-i1)-(2+4(h-1)) & i1=\frac{n}{4} \end{array}\right\}$$

In this case the combined communication and routing cost of the transition from a balanced binary tree to an arbitrary one, is affected (decreased) only by a constant factor *c*. Similar are the computations and results for $C_{N0}(SL2SR)$, hence we omit them here.

Corollary 1: Furthermore, if we extend the previous computations to a binary tree under any current state (not necessarily balanced), it can be easily seen that the transition from the existing to the next state, always causes a constant increase or decrease in the combined communication complexity (exactly as computed above).

Case *b*.2: Assume that $| i2-\frac{n}{8} | > i1$.

The computations are conducted in a similar fashion as these for **Case *b*.1**, the results obtained are similar with these obtained for Case *b*.1, and the resulting complexity is practically the same. Therefore, we skip the analysis associated with Case *b*.2.

Corollary 2: The transition from a fully balanced binary tree to an arbitrary binary tree with the same number of members is performed via an equal number of member removals or additions, in an arbitrary sequence. As already seen, the transition from the current state to the next always results in a constant increase or decrease in the combined communication complexity. Hence, the combined communication complexity for executing GDH.1 on the tree produced after the transition is not altered in this case. However, even when we have a transition with varying number of members, we see that we can predict the change in the overall complexity in average at least, unless the transition occurs under a prescribed sequence of member additions or removals. The computations for the case of the generalized *Z*-ary tree are conducted in a similar fashion, and the corresponding results are analogous.

In summary, for the case of interest, and for all cases where the member additions are not overwhelming compared to member removals, we can safely conjecture that the statement in observation 1 is true: the complexity of implementing GDH.1 through the traversal of any *Z*-

ary balanced tree is characterized by the factor n^2 and bounded by the expression: n^2+O($\frac{(z-2)}{2}h+\frac{2}{(z-1)}$)$n$+O(1- z - $\frac{1}{2(z-1)}h$).

3.3.2.3. Solution to (3.3), (3.6): Broadcast Tree

We apply a simple controlled flooding strategy based on a broadcast tree (BT) over the network nodes. An internal node that receives a message from its parent, forwards it to its immediate tree offspring only once. For each different source, the same ST is used, rooted at the given source each time. It is assumed the parent-children associations are modified on the fly. Now, we only need to perform broadcast over a rooted tree, which is quite simple as the root can broadcast to its children who can recursively broadcast into their sub-trees. With respect to GDH.2 protocol, (3.3) is further improved if each parent removes this part from the received message that corresponds to itself before forwarding the remaining message to its children. Under the worst case scenario the ST is degenerated to a chain and (3.3) becomes: $\frac{(n-1)n}{2}$. Computing (3.3), (3.6) on a BT with height $h(BT)$ we obtain: (3.3)< $\frac{(n-1)n}{2}$, (3.6) $\leq h(BT) \leq D$.

3.3.2.4. Solution to (3.3), (3.6): Simultaneous Unicasts

Another implementation of the logical broadcast is obtained by (n-1) simultaneous unicasts from the sender M_n to each member M_i of a single message through the routing path $R_{n,i}$. We obtain the following upper bounds for (3.3), (3.6): (3.3) = $\sum_{i=1}^{n-1} R_{n,i} \leq D\times(n\text{-}1)$, and (3.6) $\leq D$.

Given the analysis so far, the following metrics can be computed for GDH.1, GDH.2 and BD:

CCost(GDH.1) = *CCost(aux)* + $2\times(3.2)\times K$ < $5nK_1+2\times(n^2$+O($\frac{(z-2)}{2}h+\frac{2}{(z-1)}$)$n$+O(1- z - $\frac{1}{2(z-1)}h))\times K$ < $2\times(n^2$ +O($\frac{(z-2)}{2}h+\frac{2}{(z-1)}+2.5\frac{K_1}{K}$)$n$+O(1- z - $\frac{1}{2(z-1)}h))\times K$.

Lt (GDH.1) = Lt (aux) + (3.5) = $6n + 2D$.

$CCost\ (GDH.2) = CCost\ (aux) + (3.2) \times K + (3.3) \times K < 5nK_1 + (n^2 + \mathrm{O}(\frac{(z-2)}{2} h + \frac{2}{(z-1)})n + \mathrm{O}(1-$

$z - \frac{1}{2(z-1)} h)) \times K + n^2 \times K = (2n^2 + \mathrm{O}(\frac{(z-2)}{2} h + \frac{2}{(z-1)} + 5\frac{K_1}{K})n + \mathrm{O}(1 - z - \frac{1}{2(z-1)} h)) \times K.$

$Lt\ (GDH.2) = Lt\ (aux) + (3.5) + (3.6) = 4n + 3D$.

$CCost\ (BD) = CCost\ (aux) + 2 \times (3.3) \times K < 5nK_1 + 2 \times n \times \frac{(n-1)n}{2} \times K.$

$Lt\ (BD) = Lt\ (aux) + 2 \times (3.6) = 2n + 4D$.

3.3.2.5. Solution to (3.4), (3.7): Hypercube traversal over ST

We generate a rooted spanning tree *T* over all *n* group members and name them as designated by a *pre-order* tree traversal. As already discussed, during a full walk of the ST, each tree link is visited twice. In total, the full walk goes through 2*n* entities (some of the *n* members are traversed multiple times), and (2*n*-1) tree links. We now map every tree node visited consecutively during the full walk, to a new "entity" placed consecutively on a path. The *id*s are assigned incrementally to nodes in the path so that node with *id*: *j* precedes node with *id*: (*j*+1). These entities are in fact the *n* network nodes, some of which are replicated in the extended path, in the spaces designated by the full tree walk. All participating entities are physically one hop away from their predecessor and successor in the new path. In this path, a group member corresponds to possibly more than one entity and may be therefore assigned multiple *id*s. The execution of *wt*-Hypercube requires an additional round, since the number of participants is now double: 2*n* = $2(2^d)$ = 2^{d+1}. This algorithm provides a very simple method of computing a closed cost formula for the bandwidth and latency of Hypercube, for any physical deployment of nodes, for any arbitrary ST. The placement of nodes in the path automatically determines their peers for all subsequent (*d*+1) rounds. From this point, the protocol is executed as designated by the initial logical design. For example, during round *j*, member *i* does a 2-party DHKE with member $k = \varphi(i \oplus 2^{j-1})$. Members *i* and *k* exchange one message during round *j*, through the routing path $R_{i,k} = R_{k,i}$. However, the size of each such routing path can be now deterministically determined: the hop distance between two nodes is the lexicographical distance of their associated *id*s, namely $R_{i,k} = |k - i| = 2^{j-1}$, except for the case that both *id*s belong to the same node, and $R_{i,k} = 0$.

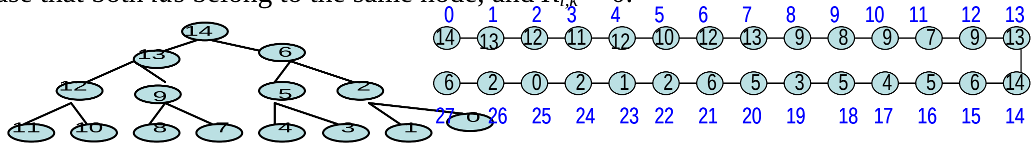


Figure 3.4. Replicating and renaming network nodes w.r.t. pre-order tree walk, for td-HCube.

The solution to (3.4) is: $\sum_{j=1}^{d+1} \sum_{i=1}^{2^{d+1}} R_{i,\varphi(i \oplus 2^{j-1})} < 2n \times \sum_{j=0}^{d} 2^{j} = 2n \times (2^{d+1}\text{-}1) = 2n \times (2n\text{-}1) = 4n^2\text{-}2n$.

Considering that certain routing distances whose end parts belong to the same node are zero, the resulting costs may be much lower than the expression computed above. Each message exchange involves a single packet.

In the proposed *wt*-Hypercube, in any given round, all message exchanges among peers involve the same routing distance. Hence, the same per round (and consequently total) latency corresponds to all entities. For example, in round *j*, the routing distance between any two peers is $2^{j\text{-}1}$. The solution to (3.7) becomes: $Lt = \sum_{i=0}^{d} 2^{i} = 2^{d+1} - 1 = 2n\text{-}1$.Given the solutions to (3.4), (3.7), we obtain the following metrics for *wt*-Hypercube:

$CCost\ (HCube) = CCost\ (aux) + (3.4) \times K < 5nK_1 + (4n^2\text{-}2n) \times K = 4n^2K + (5K_1\text{-}2K)n$.

$Lt\ (HCube) = Lt(aux) + (3.7) = 4n + 2D\text{-}1$.

The computations for *wt*-Hypercube increase, since the number of entities is now double:

$CComp(HCube) = 2n \log 2n = 2n\ (\log n + 1) = 2n(d+1)$.

Observing the corresponding metrics associated with *wt*-Hypercube, we see that we improve over the *nt* approach by a factor *x* varying with the topology of the network, where $x = \frac{d \times D}{n} \geq 1$. If the network is a single chain then the improvement is even greater and $x = O(d)$.

Summary: Tables 3.6, and 3.7, summarize the upper bounds on the bandwidth and latency costs of the five protocols, when the discussed topological improvements are performed. The metrics *CCost* and *CComp* are all scaled by *K*. *CCost* and *Lt* in these tables should be contrasted with those in Table 3.2; in most cases the efficiency of the protocols increases by a factor of *D* or *n*. The core framework has significant impact only to the resulting latency.

	Lt	CCost	Exp.
wt-*nf*-ING	$2n$	$2n^2 – 2n$	n^2
wt-*nf*-BD	$2D$	$2\times n\times \frac{(n-1)n}{2}$	n^2+n
wt-*nf*-GDH1	$4n$	$2\times(n^2 + \mathrm{O}(\frac{(z-2)}{2}h + \frac{2}{(z-1)})n + \mathrm{O}(1- z - \frac{1}{2(z-1)}h))$	$(n^2+3n)/2$
wt-*nf*-GDH2	$2n+D$	$(2n^2 + \mathrm{O}(\frac{(z-2)}{2}h + \frac{2}{(z-1)})n + \mathrm{O}(1- z - \frac{1}{2(z-1)}h))$	$(n^2+3n)/2$
wt-*nf*-CUBE	$2n-1$	$4n^2-2n$	$2n \log 2n$

Table 3.6. Performance of the *wt*-KA protocols over MANETs, w/o core framework (nf).

	Lt	CCost	Exp.
wt-*wf*-ING	$4n+2D$	$2n^2K+ n(5K_1 – 2K)$,	n^2
wt-*wf*-BD	$2n+4D$	$5n \frac{K_1}{K} +2\times n\times \frac{(n-1)n}{2}$	n^2+n
wt-*wf*-GDH1	$6n+2n$	$2\times(n^2 + \mathrm{O}(\frac{(z-2)}{2}h + \frac{2}{(z-1)} + 2.5\frac{K_1}{K})n + \mathrm{O}(1- z - \frac{1}{2(z-1)}h))$	$(n^2+3n)/2$
wt-*wf*-GDH2	$4n + 3D$	$(2n^2 + \mathrm{O}(\frac{(z-2)}{2}h + \frac{2}{(z-1)} + 5\frac{K_1}{K})n + \mathrm{O}(1- z - \frac{1}{2(z-1)}h))$	$(n^2+3n)/2$
wt-*wf*-CUBE	$4n+2D$ -1	$4n^2+ (5\frac{K_1}{K} -2)n$	$2n \log 2n$

Table 3.7. Performance of *wt* KA protocols over MANETs including core framework *wf.*

3.3.2.6 Graphical Illustration of Representative Results.

Below, we illustrate graphically the most significant results of our comparative analytical performance evaluation. We measure the *bandwidth* (CCost) incurred to the network from the execution of the discussed protocols while varying the *network size n*, in number of packets (or equivalently bits) and the associated *latency* in atomic steps. In our evaluation we use all versions of the protocols discussed: with our without topology considerations, i.e. *wt* or *nt* respectively, with of without the core framework, i.e. (*wf*) or (*nf*), respectively. We also vary the size of the network diameter *D* as follows: *D* = *an*, where 0.1<*a*<1.

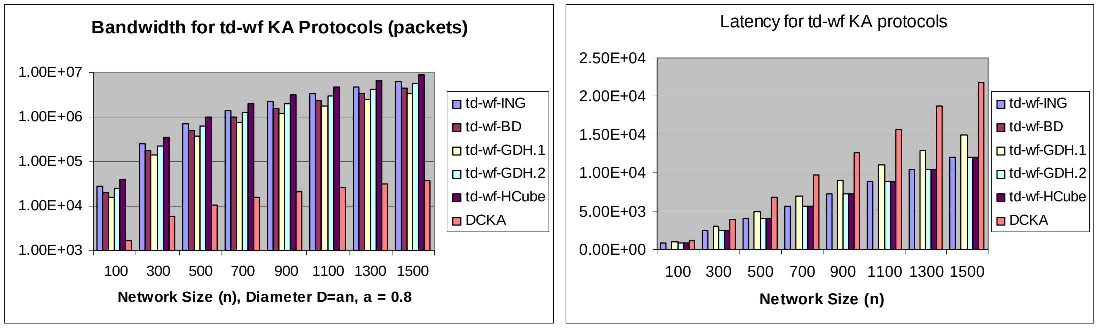


Figure 3.5. Bandwidth and Latency for the td-wf-versions, as the size of the network and network diameter grow (D=0.8n). GDH.1 has the best performance overall w.r.t. bandwidth, but the worse w.r.t. latency. The rest of protocols demonstrate comparable performance.

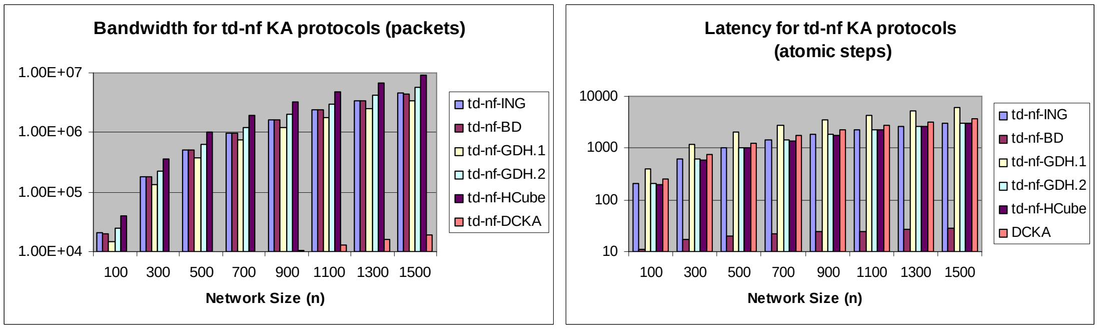


Figure 3.6. Bandwidth and Latency for the td-nf-versions, as the network size and diameter grow (D=0.8n). GDH.1 has the best performance in bandwidth, but the worst latency. The rest demonstrate comparable performance, except for BD, that has very low latency.

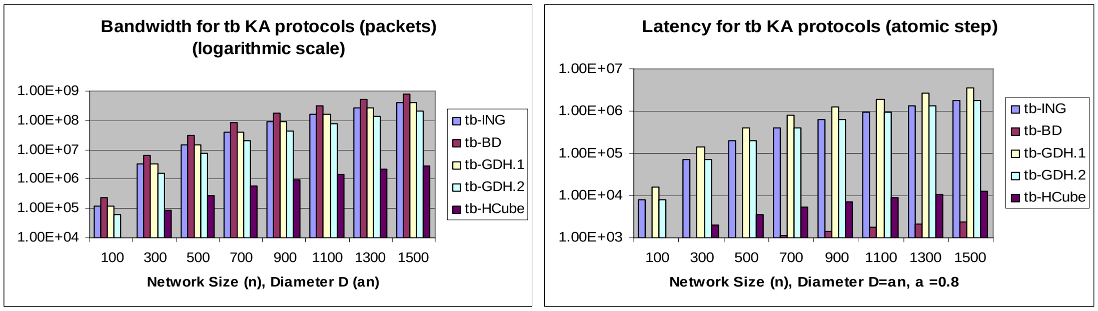


Figure 3.7. Bandwidth and Latency for the nt-versions, as the size of the network and network diameter grow (D = 0.8n). BD has the worst performance in bandwidth, but the best in latency. The nt-HCube outweighs the rest in their nt-version, whereas in its wt-version, it presents the worst performance, even though its wt-version isimproved from the nt one.

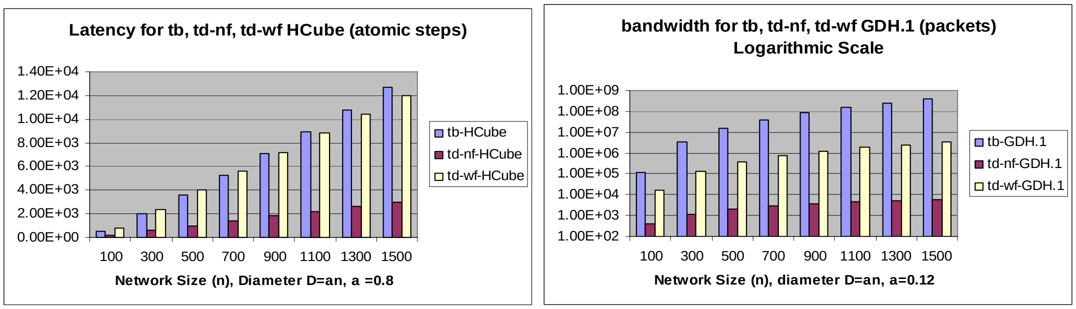


Figure 3.8. The first graph illustrates the latency for all HCube versions, as the network size increases. The nt-nf version presents the best performance overall, and the wt-version the worst. The second graph illustrates the bandwidth for all GDH.1 versions. Even in a small network diameter, the td-GDH.1 version prevails.

3.4. *Wt* KA on logical secure member graphs subject to underlying routing.

Group members now use the underlying routing protocol to gather information about other group members in their proximity. We assume that a path between group members may as well include non-member relays. We simply rely on the redundancy of the routing protocol to ensure that the exchanged messages are delivered in a timely manner. We assume a generic underlying routing protocol with the property that it always finds the minimum path (i.e. Dijkstra). It provides all nodes with the paths to at least the nearest subgroup members (with respect to the number of hops), and finds at least one path connecting two members. The upper limit in the number of hops between members considered immediate “logical neighbors” is dynamically set. The search diameter (TTL) is expanded until at least a pre-defined number of such neighbors are found or until the TTL threshold is reached. After the end of this process, every group member that is considered “connected to its group” acquires a number of virtual links to other members.

The weight of each such link is the **distance** between the two group members in terms of **number of hops (relays)**. We are interested in minimizing the combined routing and communication cost (*CCost*), resulting from the execution of the KA protocols discussed. Depending on the metric we want to optimize, we can adjust the link weight accordingly: it may include delays, traffic, robustness, mobility, geometric distances, etc. Having obtained the link weights, we will now generate a Minimum Spanning Tree (MST) over all group members. In section 3.2, the value of any link weight was always 1, and an MST would coincide with any ST. In this generalized case we need to find the MST of all MSTs originating from every single group member, if we want to apply an analogous method on the associated KA protocols. Obviously, utilizing a core framework that computes the MSTs from all group members is an over-kill. In what follows we describe a very efficient algorithm for extracting a “low cost” Hamiltonian path from a single MST. This algorithm applies to all the discussed KA protocols that require a ST to derive a *wt*, efficient communication schedule. We show that with the appropriate manipulation of the generated MST, we achieve even further improvement on the metrics of interest: *CCost*, *RCost*, and *Lt*.

Next, we introduce a **new algorithm** for optimizing the metrics associated with **ING, GDH.1, and GDH.2**, and in the following section we introduce a **new algorithm** for optimizing metrics associated with **Hypercube**. In both sections, we carry out extensive simulations over numerous network configurations to measure the overhead incurred to the network due to: *a*) the generation of the core framework and *b*) the execution of the discussed KA protocols with the new *wt* communications schedule. We describe our simulation set up and present the most characteristic results. We show that the new algorithms improve the performance of the new *wt*-versions of protocols even further.

3.4. *Wt* adaptations of GDH.1, GDH.2, ING, on logical graph, subject to routing

As in the previous section, we assume that a starting point is pre-agreed before deployment, and as the network operation progresses, the leader election process provides group members with new and auxiliary starting points. The knowledge of the starting point is propagated to the rest of group members via a simple broadcast tree. The required backbone MST will be rooted at the starting point. Below we provide a detailed description of the algorithm that generates the core framework and is used for the assignment of session *id*s to the members.

3.4.1 MST Generation

We generate a MST with starting point member *A* by applying a distributed version of *Prim's* algorithm. *Prim's* algorithm is based on a greedy strategy that is captured by the following "generic" algorithm which grows the MST one edge at a time. The algorithm manages a set of edges *H*, maintaining the following loop invariant: prior to each iteration, *H* is a subset of some MST. *Prim's* algorithm has the property that the edges in the set *H* always form a single tree. This strategy is greedy since the tree is augmented at each step with an edge that contributes the minimum amount possible to the tree's weight. In order to implement this algorithm as such however, all members must have global information on the link weights of all other members, so that they all see which member in *H* has the minimum link weight and allow the growing MST expand towards this direction. We adjust this algorithm to our distributed environment, by having each member that joins *H* report its candidate links to the root, and the root determines the next member *J* to join *H* by examining all unused candidate links of all members that currently belong to *H*. Then root *A* sends a *Join Flag* to member *J, J* joins *H* and so on and so forth. As it has been proven, the improved running time of the original *Prim's* algorithm is: $Lt1 = \mathrm{O}(E + V\log_2 V)$. Let the weighted path between any group member *J* and the root *A* be denoted as $R_{J,A}$. Also, let PK_S denote the bit size of the packet that carries a member's candidates' information up to the root, and K_S the bit size of the *Join Flag* from the root to the next member that joins *H*, where $K_S \ll K$. The combined communication cost and latency incurred for the adjustment of Prim's algorithm becomes:

$$CCost1(aux) = \sum_{j \in V} R_{j,A} \times (PK_S + K_S) \approx |V| \times (PK_S + K_S) \times \mathrm{avg}(R_{j,A}) \approx |V| \times (PK_S + K_S) \times \tfrac{1}{2} \max_j (R_{j,A})$$

$$Lt2 = 2 \times \sum_{j \in V} R_{j,A} \approx 2 \times \approx |V| \times \mathrm{avg}(R_{j,A}) \approx 2 \times |V| \times \tfrac{1}{2} \times \max_j (R_{j,A}) \approx |V| \times \max_j (R_{j,A}).$$

The running time of the adjusted algorithm in total becomes:

$$Lt = Lt1 + Lt2 = \mathrm{O}(E + V\log_2 V) + |V| \times \max_j (R_{j,A}) = \mathrm{O}(E + V(\log_2 V + \max_j (R_{j,A}))).$$

Still, we will measure these two metrics with our simulation experiments.

3.4.2 MST Manipulation

Until now, we have generated a MST starting from member *A*. A full walk on this MST will provide us with a Hamiltonian path with cost $C_{MST} < 2C_{OPT}$. We will investigate if we can do better than that for GDH.1, GDH.2, and ING. Thus, we look for heuristics that produce better results than the *closest point heuristic* or *the full walk on an MST with triangle inequality*. We start with the following observation: *if the generated MST was in fact a chain, then the desired Hamiltonian path would be directly provided and would result in the same cost as this of the MST*. Hence, the more the resulting MST resembles a single chain, the less the cost of the resulting Hamiltonian path (not tour) is expected to be. Based on this observation, we initiate the manipulation of the MST with the following **transformation**: During the formation of the MST the **two longest distinct paths** from all group members to the root are identified. The group member that marks the end of the longest path becomes now the new root of the transformed MST, and the associations between parents and offspring in the existing MST are sequentially altered to accommodate the transformed tree. This process, results in **unfolding the MST** to its longest path, or else in "**extracting**" the largest possible "path" from the MST (Figure 3.9 (*b*), (*c*)).

Next, each member that belongs to the new ST, recursively rearranges its offspring in the order of decreasing distances from their tree leaves (Figure 3.9 (*d*)). It is obvious that the backbone of the tree, which is the previously unfolded path, will be accessed last by a pre-order tree traversal. It can be directly seen that in the case we want to generate a Hamiltonian path from this tree, *all members that belong to the "unfolded" path will be visited only once*. No recursions occur on the unfolded path. Hence, the longer the unfolded path is (the less the resulting tree branches), the less the number of members that will be revisited is. Consequently, this modification results in the reduction in the routing overhead for the

Hamiltonian path formed. This is also the *intuitive idea* behind the use of this heuristic for protocols GDH.1, GDH.2, and ING.

In the case of GDH.1 and GDH.2, the benefit from having each member in the transformed MST recursively rearrange its offspring is even greater. For the upward stage of GDH.1-2, the MST is traversed as indicated by the Minimum Point Heuristic. Along the lines of a greedy strategy, we select to fix the backward GDH.1-2 schedule first. For that, we assign each of the *n* group members with a unique sequential *id* from $Z^{*} \cap [0, n\text{-}1]$. That is, we perform a pre-order visit of the MST by assigning session *id*s from the highest to the lowest one, as we traverse the tree. By fixing the backward schedule first via a pre-order MST traversal, we ensure that the members carrying the longest KM material are accommodated first. Hence, we still act along the lines of a greedy strategy. **Among siblings**, the following **invariant** is true: the higher the newly assigned *id* of a sibling (backward schedule), the fewer amount of hops (relays) a message originating from this sibling will go through until the destination is reached (successor or predecessor). The new *id* assigned to a member corresponds also to the number of KM elements the member must communicate to its successor or predecessor. Thus, among siblings, the longer the message, the less number of relays it involves. We stress again that we aim in improving the metrics of interest for the studied protocols by manipulating the MST with simple, lightweight, but effective heuristics, like the ones proposed. Below, we briefly summarize our algorithm. We briefly summarize our algorithm and analyze its overhead next:

1. Construct MST using distributed *Prim* (nodes report candidate links to root which determines the next link)
2. Transform MST by deploying (unfolding) its largest path, modify all parent-offspring associations properly
3. Recursively re-arrange all offspring visited in increasing distances from MST leaves (case of GDH.1-2)

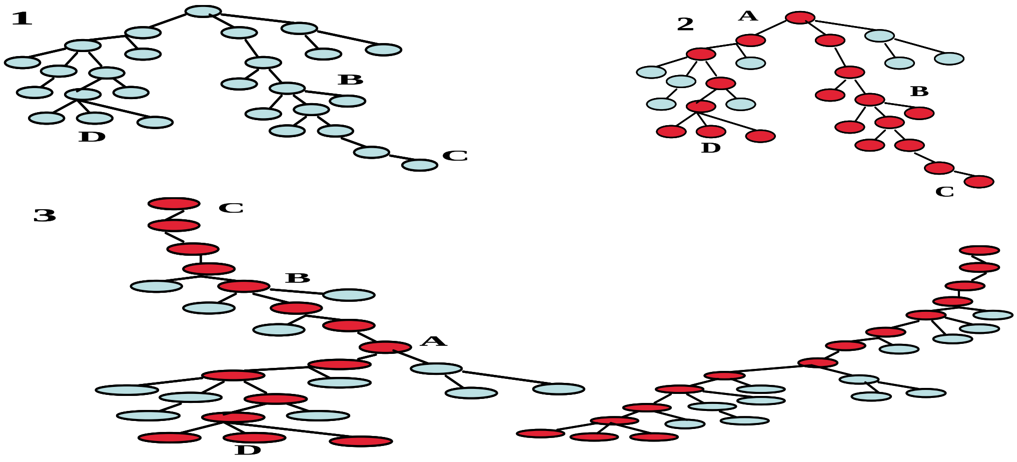


Figure 3.9. Manipulation of a MST: Transformation by unfolding it to the longest path and recursive re-ordering of each member's offspring. (a): Initial MST Prim, (b): Identification of the largest path to unfold, (c): MST unfolded to its largest path, (d): Re-arrange offspring from larger to smaller path.

3.4.3. Performance Analysis

Notation: Let $R_{max} = max_{i,j}(R_{j,\ i})$ be the longest virtual link between any two virtually connected members.

The two longest distinct paths are found along with the initial MST formation. The MST root stores data about the two members whose distances from the root are the longest, and follow two distinct paths to the root. Based on the next candidate that joins the evolving subset H, the root updates its related information accordingly. Thus, no extra communication overhead or latency is incurred to the network for this operation. After the two longest paths are determined, the root notifies the member that will serve as the new root, say member C. Starting now from C all members sequentially alter their parent-offspring association. This running time for this operation depends on the length of the new unfolded path (UP), i.e. $R_{UP} = R_{C,D}$, as illustrated in Fig. 3.9(c). The associated communication overhead and latency are:

$CCost2\ (aux) = Weight\ (MST) \times K_S = W(MST) \times K_S$, $\qquad Lt3 = R_{UP}$.

For the re-ordering process, each offspring recursively notifies its parent of its distance from the related tree leaf. For example, if member F collects the maximum distances of all offspring from the tree leaves, it picks the maximum among these values, adds its own link weight towards its parent, and sends this information to its own parent, who will collect similar information from all offspring, and so on and so forth. Each parent stores this information and applies a sorting algorithm to its offspring (i.e. QuickSort) to virtually reorder them for the coming pre-order traversal. The sorting calculations can be executed independently from the propagation of the maximum distance to the root. The overhead incurred from this operation is the same as before:

$CCost3\ (aux) = W(MST) \times K_S$, $\qquad$ $Lt4 = R_{UP}$.

Overall, the communication cost and latency of the core framework becomes approximately:

$CCost\ (aux) = CCost1(aux) + CCost2(aux) + CCost3(aux)$.

$$CCost(aux) = \frac{|V|}{2} max_j(R_{j,Rt}) \times (PK_S + K_S) + 2W(MST) \times K_S < \frac{1}{2}|V|^2 \times R_{max} \times (PK_S + K_S) + 2|V|R_{max} \times K_S$$

$$CCost\ (aux) < |V| \times R_{max} \times (\tfrac{1}{2}|V| \times (PK_S + K_S) + 2\ K_S).$$

$$Lt = Lt1 + Lt2 + Lt3 + Lt4.$$

$$Lt = \mathrm{O}(E + V(\log_2 V + max_j\ (R_{j,\,Rt}))) + 2 \times R_{UP} < \mathrm{O}(E + V(\log_2 V + |V| \times R_{max})) + 2 \times 2|V| \times R_{max}$$

$$Lt < \mathrm{O}(E + |V \log_2 V| + (|V|^2 + 4|V|) \times R_{max})).$$

3.4.4. Simulations

Simulations Set Up: We have conducted simulations to compare the routing cost of *wt*-(GDH.1, GDH.2, and ING) *vs.* their original versions on a logical secure member graph, subject to underlying routing. Our network graph represents a single cluster area where a single group is deployed. A number of nodes from this graph are randomly selected as group members. The group leader is randomly selected. At the end of the group “registration” period, the sponsor piggybacks the list of the legitimate members into the routing packets. We assume a generic Dijkstra routing protocol that finds the shortest paths between members. Through the underlying routing, each member obtains the routing path(s) to its closest neighbor(s). We dynamically determine the proximity with respect to the number of hops between two members. If no neighbors are found in the proximity, the search diameter (TTL) is gradually expanded until a pre-agreed number of members are found.

We further assume that while the backbone framework is being formed, the relative placement of members and hence the proximity lists do not change significantly. Such a change could result in a different “optimal” solution, and the one currently generated would become outdated and probably suboptimal. However, our algorithm is fairly fast. So, it is not too optimistic to assume that the topological changes that occur do not “offset” our solution much from the target. It is expected that the higher the mobility of nodes, the worse the performance of our algorithm is. Even though the *wt*-versions are more sensitive to mobility than the original, they still reduce significantly *CCost* and *L*t, even if the generated schedule is not optimal. On the other hand, the backbone framework can be periodically reconfigured, in order to capture all dynamic changes and reflect them to the *wt* protocol executions. For example, the auxiliary framework may be recalculated whenever the performance of a given protocol degrades to the median of the best execution (the first one after the auxiliary framework reconfiguration) and of the average execution of the original *nt* scheme.

For our evaluation, we generated various random graphs for a given input of the number of nodes *n* and the number of members *m*. For the same graph and the same input, we have varied the subgroup configuration, i.e., we have selected the *n* members in a random manner. For each graph of input <*n*, *m*> and for each subgroup configuration, we evaluated the three metrics of interest (*CCost*, *RCost*, *Lt*) of the *wt vs.* the *nt* versions, and averaged the results for all random graphs with the same inputs <*n*, *m*>. We have tested the following cluster-subgroup scenarios: **Cluster Size:** [100,…, 600], **Subgroup Size**: [8,…64].

Simulation Results: We illustrate indicative results produced by the discussed protocols and their *wt* versions, measured in terms of the total number of hops (relays and group members) required for the protocols to successfully terminate. The KM messaging is very heavy for the network nodes, so our aim is to reduce the overall number of bits (or packets) required and relieve as many nodes as possible from relaying large keying data. This is indeed the case with the new protocol versions: they achieve significant savings in terms of *RCost*, and consequently *CCost*. The graphs that follow reflect a number of important metrics that justify the value of our new *wt*-algorithms: (*a*) *CCost*, *RCost* of the *wt*-versions *vs*. the original under various scenarios of group and network size, and (*b*) *CCost*, *RCost* required for the generation of the auxiliary framework: *CCost*(*aux*), *CompCost*(*aux)*, under various scenarios of group and network size. The graphs in case (*a*) are all scaled by a factor *K*. *CCost* and *RCost* are significantly reduced in all scenarios captured by our simulations. In many cases the associated ratio becomes:

$$R_{COMM} = \frac{CCost(ING\_Opt,n,S)}{CCost(ING,n,S)} \approx \frac{1}{2}.$$

We illustrate the above with an indicative arithmetic example: for a subgroup of size 32, and network of size 200, the average relays produced (*RCost* or *CCost*) are 1215 for ING_Opt, and 2595 for ING, hence $R_{COMM} < \frac{1}{2}$.

Furthermore, we are able to verify that *CCost*(*aux*) and *CompCost*(*aux)* of the *wt*-versions match our analytical results. Indeed, let us for example recall the formula we derived for estimating *CCost* (*aux*). Assuming that the maximum number of neighbors for each member is 10-12 (verified from our simulations as well), and having set K_S = 8, we select $PK_S = 8\times 12 = 96$ (bit size of control packet×maximum number of neighbors) Also, we select R_{max} = 8, and obtain the following expression: *CCost* (*aux*) $< 8|V|\times(52|V|+16)$.

We evaluate this expression for a subgroup of 32 members and obtain: *CCost* (*aux*, *V*=32) < 430,080 bits. Indeed our results verify that this upper bound holds, since indicative values of *CCost* that we obtain for group size *n* and network size *S* are the following: *CCost* (*aux*, *n*=32, *S*=200) = 336977 bits, *CCost* (*aux*, *n*=32, *S*=300) = 395881 bits, *CCost* (*aux*, *n*=32, *S*=400) = 256436 bits. The same is the case with the rest of the group sizes we have included in our simulations. We should also observe that *CCost* (*aux*) increases as the network size grows, until the network reaches some threshold value. Then, the corresponding metric starts decreasing. The reasoning behind this behavior is the following: two members are considered “connected” until the hop distance between them (TTL) reaches a certain threshold. After this threshold is exceeded, the members are considered disconnected. As the network size increases, the density of subgroup members decreases, and naturally the “neighbors” of each member decrease, as expected. The less the neighbors of each node are, the lower *CCost*(*aux*) and *CompCost*(*aux*) quantities become. In addition, we also verify that the metrics associated with the auxiliary framework are kept reasonably low and add little to the overall overhead produced by the execution of the *wt*-versions, even under a growing secure group and/or network size. On the other hand, the control messages used for the generation of the framework have size $K_S << K$. Acceptable bit lengths for K are above 2048 bits so that a KM protocol can be considered computationally secure to-date. We illustrate the above with an indicative arithmetic example: We have found that *avg* (*CCost* (ING_Opt, n=32, S=300) = 3925, *avg* (*CCost* (ING, n=32, S = 300) = 5976, and *avg* (*CCost* (*aux*, ING, n = 32, S = 300) = 395881. It can be seen that the larger the constant K becomes, the bigger the difference in the overhead of ING_Opt *vs.* ING becomes. However, even if we assume that the bit size of K is as small as 1024 bits, we obtain the following results: *CCost* (ING_Opt + *aux*) = 3925×1024 + 395,881 = 4,019,200 + 395,881 = 4,415,081 bits, while *CCost* (ING) = 5976×1024 = 6,619,424. Obviously, the difference in the overall overhead (including the auxiliary overhead for the case of *wt*-ING) is considerable, even under this worst case scenario (small K, highest observed *CCost*(*aux*)). The difference in the overall overhead becomes even more impressive if GDH.1 is considered. We emphasize again that a new framework needs not be computed every time a *wt*-KA protocol is executed. We can re-compute the framework periodically. Hence, the impact of the framework OH on the overall cost of our algorithms is even lower in practice.

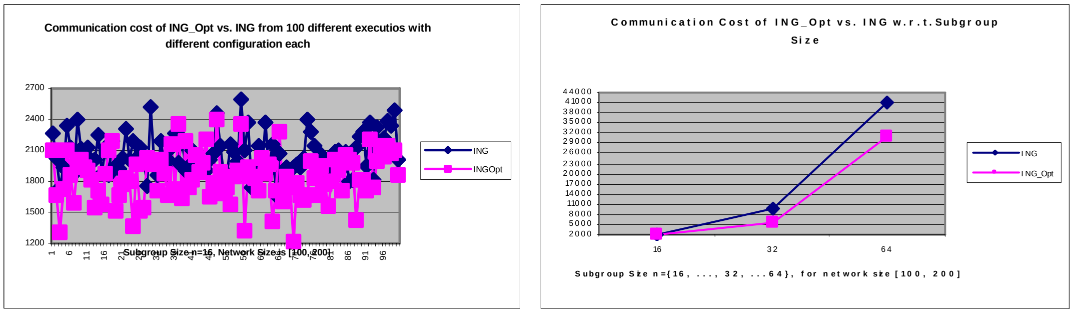


Figure 3.10. (a) CCost for ING_Opt vs. ING, for Subgroup size n = 16, and network size S = 100, for 100 different graph configurations. ING_Opt results in superior performance for the vast majority of different configurations, (b) CCost of ING_Opt vs. ING w.r.t. Subgroup size <16,…, 64>, in a network of size S in [100, 200]. Again, ING_Opt has superior performance.

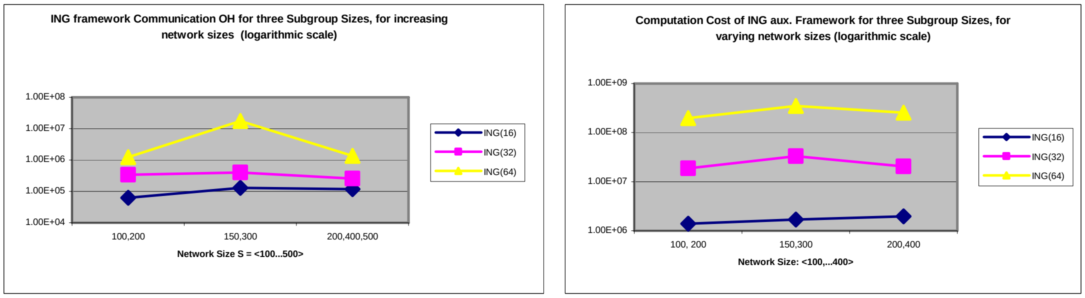


Figure 3.11. (a) CCost(aux) (bits) [3], and (b) CompCost(aux) (bits) for ING_Opt, for an increasing group size <16,…,64>, for three different scenarios of network size: S1=<100,200>, S2 = < 150, 400>, S3 = <200, 500>. The corresponding costs increase with the group size as expected. They also increase with the network size up to a certain threshold, for the same reasons discussed before.

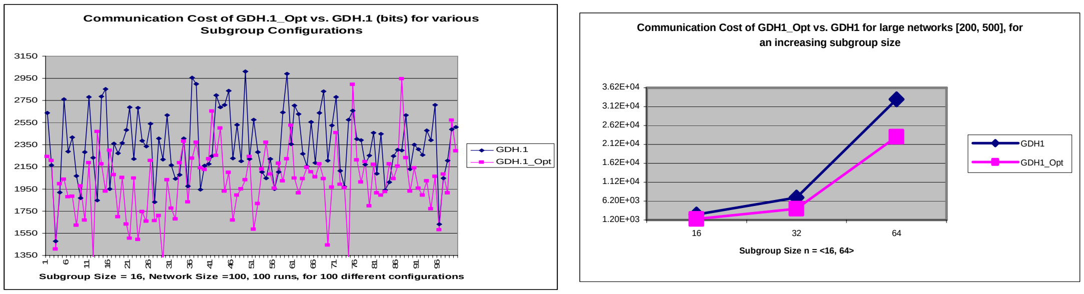


Figure 3.12. CCost of GDH.1_Opt vs. GDH.1 for: (a) 100 different group configurations on a network of 100 nodes (100 different runs) for group size of 16 members, (b) comparison w.r.t. number of members [16, 32, 64] in large network of [200,…, 500] nodes. The performance of GDH.1_Opt is significantly superior in the vast majority of cases, and this reflects on the average case as well. The ratio of improvement becomes: $0.53 < R_{COMM} < 0.70$.

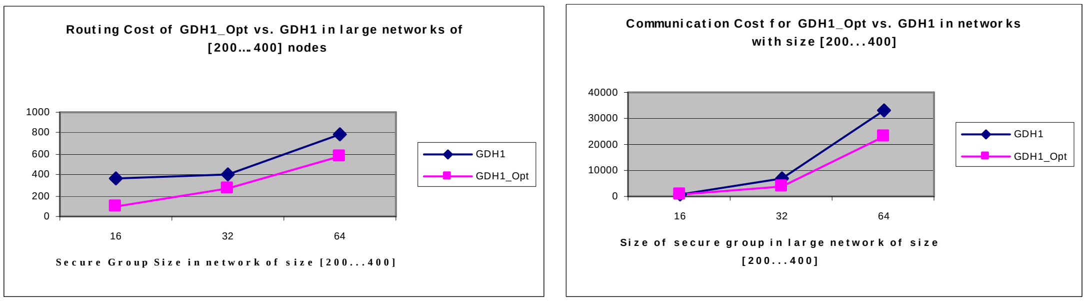


Figure 3.13. (a) RCost and (b) CCost of GDH.1_Opt vs. GDH.1 w.r.t. the number of group members [16, 32, 64] in a large network of [200,…, 400] nodes.

The behavior and performance of GDH.2 is similar to this of GDH.1, and we skip the related results in order to avoid unnecessary repetitions.

3.5. *Wt* adaptation of Hypercube on logical member graph, subject to routing

In this section, we introduce a new heuristic for generating an efficient, improved *wt*-Hypercube. We assume that a starting point is pre-agreed, and as the network operation progresses, the leader election process provides members with new and auxiliary starting points. The knowledge of the starting point is propagated to the rest of group members via a simple broadcast tree. We provide a detailed description of the algorithm that generates the core framework and is used for the assignment of session *id*s to the group members.

Our goal is to improve on the previous *wt*-approach. We will identify the 2^d parties on the d-dimensional space $GF(2)^d$ by selecting such a basis $b_1,\ldots, b_d \in GF(2)^d$ for the direction of communications per round, that results in optimizing the desired metrics of interest *CCost*, *RCost*, *Lt*. The previous method (MST manipulation) does not apply to this case because the group members communicate in pairs. If we used an MST, we would also need to define a deterministic method for extracting pairs out of an MST, which is not straight-forward. Furthermore, the selection of peers for the 1st Hypercube round and the selection of a unique basis in $GF(2)^d$, limits the degrees of freedom for the selection of peers for the subsequent rounds. So, similar solutions for the subsequent rounds may not apply in most cases. What is more, the initial selection of peers, even though optimal for the current round, may prove to be very unfortunate for the overhead incurred, after the end of all *d* rounds. Therefore, the use of MSTs as a core framework is highly unlikely to produce efficient Hypercube schedules. This solution is abandoned, and we are looking for more lightweight frameworks that can potentially improve the metrics of interest even further. It is clear how complex the problem is. The optimal solution can be found with the exhausting method of trial and error, but this is out of the question, and in particular for the environment of interest.

3.5.1. Overview.

The goal of our algorithm is to determine the most “efficient” pairing for the 1st round. In other words, the peers selected for the 1st Hypercube round must considerably improve the metrics of interest. Towards this direction, we use a “greedy” strategy under which each member makes the best matching selection possible (1st Pass) and then a “corrective operation” is initiated (2nd Pass) that goes around “dead-ends” and ensures that all members obtain a peer. We virtually place each member of a 1st round pair in one of the two columns: *left* or *right*. The selection of these peers is close to the optimal for the given round, as we will show next, irrespectively of its impact to the subsequent rounds. However, we keep the message exchanges between the peers of this round active in every subsequent round, by doing the following modification to the original Hypercube: only half of the initial *n* group members, those either in the left or in the right column, participate actively to all subsequent rounds, as designated by the original scheme. The members that belong to the non-active column become passive recipients in the subsequent rounds. Without loss of generality, we assume that the members in the *left* column are the *active* participants and those in the *right* column become the *passive* ones. The *active* members are assigned new sequential session *id*s $\in$ [0, $\frac{n}{2}$-1] after the 2nd Pass. Before the start of the following rounds, the active members decide on a unique basis, and execute Hypercube for the next (*d*-1) rounds. In our case, we select a basis that designates the new pairs at any given round j, where $0 \leq j \leq (d\text{-}2)$, according to the following formula: in round j, party i interacts with party $i \oplus 2^{j\text{-}1}$. In each such round, each active member communicates the newly calculated blinded value to its corresponding passive peer (member of right column). So, all group members participate indirectly to all rounds, since all receive the intended KM data, and process it as indicated by Hypercube. In the end, all group members obtain the same group key.

Given the previous observation, we can apply our algorithm on every round and not just the first one. Then, from the active parties of round j, only half remain active during round (j+1) and the other half become passive. Our heuristic greedy algorithm of “2 Passes” is applied once again to the currently active members, to determine the lowest weight pairings among them. So, instead of using a pre-agreed common basis for the direction of communications, we dynamically construct such a basis, based on the topological proximity of the active group members. However, this approach only works, if we de-activate half of the active participants of the current round, for all subsequent rounds. However, applying the heuristic as many times as the total number of Hypercube rounds is an over-kill. Also, the resulting protocol is quite different from Hypercube in nature, with features that may not be desirable for our network. An efficient and flexible solution is to apply the heuristic only for the first few R rounds. The threshold R can be dynamically set.

Given these considerations, we will conduct the following analysis by setting $R = 1$, in order to limit the overhead resulting from the backbone framework. We will show in the section of simulations that Hypercube performance improves substantially by this modification alone. Having defined our generic algorithm, it is trivial to change R and measure the improvement in the protocol. In this work, we find it sufficient to present only the results collected from $R = 1$, since: (*a*) the results are very satisfactory and the overhead of the auxiliary framework is low, (*b*) by increasing R the overhead due to the application of Hypercube is decreased, but the overhead of the backbone framework increases as well, and (*c*) by increasing *R* the nature of Hypercube is shifted from totally distributed to centralized.

1st Pass (Greedy Approach):

This process resembles the construction of an MST in that it follows a greedy strategy as well. The main difference is that only one member or one pair of members (that already belong to the structure being generated) is allowed to select the next candidate that will join the structure generated during the 1st Pass. Ideally, in the case that no corrective process is required (2nd Pass) the outcome of the 1st Pass is the following: a peer for every member is designated, with respect to the proximity among nodes. We denote the outcome of the 1st Pass as Set1. Towards the construction of Set1, a single token is circulated among members, and at each step, the token is passed to one member only. The details of the process follow:

Initially, the starting point is handed the token for the first time. The first time that a member is handed the token, will attempt to make the best selection available and find the closest in terms of number of hops peer. Assume that member *A* currently possesses the token. If there is such a peer *F* available, then members *A* and *F* will form a valid pair for the process, schematically represented with a horizontal line in the virtual sketch of Set1. If *A* is the starting point, there is always a peer matched to it, since all selections are available at this point, and also every member is “connected” to a number of members in a “connected” graph. If *A* is not the starting point, then there is a chance that its nearest neighbors have been reserved by other members, and *A* cannot find any peer available. We distinguish two cases:

Case (*a*): *A* finds peer *F* available: *A* enlists *F* as its peer and together form a horizontal line on the virtual sketch of Set1 (Fig. 3.14). Now both *A* and *F* merge their proximity lists (those that contain the members in their proximity) and arrange their elements in increasing order of hop distance. The exchange of merged lists has size $2D\times K_S$, where *D* is the upper bound of elements in a proximity list, and K_S is the bit size of any of the control messages. Member *A* still holds the token, and will give it to the first member from the merged and ordered proximity list that is still available (i.e. it is not already a part of Set1). Hence, *A* sends a *Join* control Flag to each member in the merged list sequentially, until it finds the first available member (say *H*), in which case it receives an *Accept* control Flag. Member *H* broadcasts this *Accept* control Flag and information about its new peer to all neighbors in its own proximity list. This way, neighbors know the status of *H* beforehand and do not need to query it when they become initiators. Members that are unavailable respond with a *Refuse* control Flag.

Then, member *A* provides the token to *H*. Member *H* becomes the initiator of a similar process, in order to find a peer of its own, and so on and so forth. If however, *H* does not find a peer, it gives back the token to its ancestor, member *A*. *H* becomes "grounded", and it is represented via a "vertical line" on the sketch of Set1 (Fig. 3.14). Member *A* continues scanning its merged list and holds the token until it finds another available member (say member *C*) that will produce a peer (say member *G*). Schematically, member *C* is placed at the end of a vertical link whose other end starts from the middle of the horizontal link formed by members *A* and *F*. Members *C* and *G* form a horizontal link. Any successor of the merged lists of *C* and *G* (grounded or paired) will be placed on a vertical link whose other end starts from the middle of the horizontal link formed by *C* and *G*. If the merged list is exhausted and member *A* has not found a successor to hand the token to, then it recursively gives the token to its predecessor. The same method is followed until all members are visited. As long as the members are connected as defined at the beginning, this method produces no deadlocks, i.e. all members are visited before the process ends. Assume that S_1 is the subset of members that have been visited during the 1st Pass, and S_2 is the subset of members that could not be visited during the 1st Pass. At least one member in S_2 must have a link to one or more members in S_1 otherwise the graph is not connected. Let this member be *J*. Now, all members in S_1 are part of either a horizontal line, or a vertical one. Clearly, no members that belong to a vertical line have a link to member *J*, otherwise such a member could form a pair with *J* on a horizontal line. The members that hold the token exhaust their merged lists until they find a successor. However, if at some point the process stalls, in the sense that no successors can be found, the token recursively goes up the whole structure (Set1), and at each step, all elements in the associated merged lists are scanned and the "neighbors" of all members that belong to Set1 will be examined. Then, the link to *J* will be eventually found, and *J* will join S_1 and will be also handed the token for the first time. Hence, all members will be eventually visited.

Case (*b*): Member *A* does not find any peer available: As stated in the previous case, *A* provides the token to its predecessor member, which follows the same process and recursively provides the token to its own predecessor under case (*b*) or to a successor under case (*a*). The predecessor has already found its peer, and consequently, it looks through its merged list to find the next potential element to provide the token to. The process ends when all members have been visited, and the token recursively goes back to the starting node.

2nd Pass (Corrective Operation):

The purpose of the 2nd pass is to "locally" re-arrange the grounded members and the pairs established during the 1st Pass so that all members obtain a peer at the end of this process. This time we access the structure (Set1) obtained after the end of the 1st Pass from down-up and we build the final Hypercube structure, denoted as Set2. We start with the horizontal links that have the following property: all members vertically attached to them (if any) are grounded. We denote the horizontal links with this property as *leaf* horizontal links. We start by re-arranging all associated members with the leaf horizontal links. Then, we go up one predecessor horizontal line at each step, until we reach the top, i.e. the horizontal link formed by the starting point. All paths starting from all leaf horizontal links eventually end up to the starting point. We denote the members that have been given the token and have also designated a peer for themselves during the 1st Pass as *initiators*. So, we start with leaf initiators and we go up one initiator at each step until we reach the starting point. The initiator is always paired and represented as a vertex on a horizontal link. During this pass however, we re-arrange the initiator, its peer, and all their vertical links, when they become ready, and after this step the initiator may end up either without peer or with a different peer.

Each member that lies at the end of a vertical link (initiator or grounded) uses two control flags: the *ActPass* Flag and the *Arranged* Flag. If for a member *ActPass* = 1, then the member is active at this point during the 2nd Pass process, and actively participates to this step of the algorithm that involves it. If *ActPass* = 0, then the member becomes inactive for all subsequent steps of the process. When *ActPass* is switched to 0 it remains 0 throughout the process. If *Arranged* = 1, then the status of this member is determined for the current step of the algorithm. The status of a member is determined by the value of *ActPass* flag. If *Arranged* =1 and *ActPass* = 1, then the member participates in the step that involves it actively, because no peer has been assigned to it. If *Arranged* =1 and *ActPass* = 0, then the member has been assigned a permanent peer, and consequently it is not active for the step that involves it and any subsequent step. In this case, the status of a member does not depend on the outcome of the execution or previous steps of the algorithm. If however *Arranged* = 0, then the status of the *ActPass* flag is unknown. Then, the member must wait for the outcome of the step that involves the initiator of the previous step, virtually placed below its own initiator. For example, initially, all grounded members participate in the process with the following values in their control flags: *ActPass* = 1, *Arranged* = 1. This is because the status of the grounded members is known – they have no peers assigned – and they will actively participate in the step that involves them. The matched peers of the initiators from the 1st Pass are also going to participate in the step that involves them, and they set their flags to the same values as these of the grounded members. However, the status of the initiators (the matched members that are represented as vertices on the left of the horizontal links) is unknown. Initially, they participate in the step that involves them with the following values in their control flags: *ActPass* = 1, *Arranged* = 0. After getting feedback from initiators that lie below them, they change the values of their control flags to: *ActPass* = {0, 1}, and *Arranged* = 1. We now give a detailed description of a single step of the 2nd Pass algorithm:

At any step of the 2nd Pass, every initiator monitors its own horizontal line until all vertices placed at the other end of the vertical links have set their *Arranged* flag to 1. Then, the status of all associated members (peer, grounded, initiators) is known and the process that will determine the status of the current initiator can now be activated. The initiator scans all these members to see how many have set the value of the control flag *ActPass* to 1. As discussed, those with *ActPass* = 0, have obtained permanent peers from the previous steps and do not participate to the process any longer. So, they can be ignored by the current initiator.

(*a*) If the number of members with *ActPass* =1 is *odd*, then the current initiator will obtain a permanent peer in this round, and will become inactive for the remaining steps. Therefore, the initiator can set its own control flags (previously set as follows: *ActPass* =1, *Arranged* = 0) to the following values: *ActPass* = 0, *Arranged* = 1.

(*b*) If the number of members with *ActPass* =1 is *even*, the current initiator will *not* obtain a permanent peer in this round, and will become active for the remaining steps. Hence, the initiator can set its own control flags (previously set as follows: *ActPass* =1, *Arranged* = 0) to the following values: *ActPass* = 1, *Arranged* = 1. In this case, the initiator gives priority to all the rest of members in its vicinity to obtain peers. This can be done, since the number of these members is even. The peer that was previously matched to this initiator will now be matched to one of the members that lie on the vertical links.

The current initiator, say *A*, determines the matching of its associated members according to the following method: first, it matches its ex-peer, say *B*, with its closest neighbor (in terms of number of hops) based on *B*'s proximity list. Then, *if* case (*a*) holds, it matches itself with the closest neighbor available, from its own or the merged ordered proximity list. It matches the vertices (members) on the vertical links, with the following simple algorithm:

Member *A* scans its own proximity list, and the first two elements available are always matched together. Then, the next two remaining elements are matched together, and so on and so forth. Then, member *A* scans the proximity list of member *B* and matches the first two available elements every time, until all or all except the last elements (if the number of elements is odd) are exhausted. Finally, the two remaining elements on each of the two proximity lists of *A* and *B*, (if there are any) are matched together. The reasoning behind this algorithm is the following: the two elements that are matched together are always those that are available and have the minimum hop distance from their associated horizontal link. This means that they both have the minimum hop distance either from the initiator or from its ex-peer (except for the last remaining pair). The idea is that these distances set up a threshold for the maximum distance between the newly matched members from the vertical links. This threshold is the minimum possible compared to a different matching.

Now all members associated with a given initiator (and the associated horizontal link) are matched. Since the initiator signals its known status (*ActPass*, *Arranged* = 1) to the upper level, the initiator associated with the upper level is ready to perform exactly the same steps and accommodate all members associated with it, following exactly the same approach. At the end, all members are accommodated if the number of members is even. This is so because the process for any given level is not executed unless all associated levels below this are accommodated (i.e. all members in them are paired).

3.5.2. Hypercube Structure Manipulation

After the 2nd Pass ends, all members are matched in pairs. This would suffice if we were strictly interested in improving the performance of the 1st Hypercube round only. However, we still want to improve the overall performance of the protocol. We have selected to do this with the approach introduced at the previous section for the GDH-based protocols in order to generate the final communication schedule. The Hypercube structure becomes equivalent to a

ST if we map every pair of members to one tree node. In fact, that is what we do in reality, since in all subsequent rounds after the 1st, only one peer of every pair participates, hence we want to generate a communications schedule only for the members that are active after the 1st round. The other half (inactive after the 2nd round) have been accommodated for the remaining rounds: they remain recipients of the BKs computed by the active senders, during all rounds. Only these active members participate in Hypercube structure manipulation and obtain new *ID*s. The resulting Hypercube structure does not only provide pairs for the 1st round, but connects all active members via a structure exactly similar to this of MST. We will use the previous methods that deal with the manipulation of an MST to provide a schedule that improves its performanc. The method is similar as before: we apply the MST manipulation to the Hypercube structure (Fig. 3.14). We unfold the structure to its longest path, recursively arrange the offspring of each node in this structure in decreasing order of hops, and perform a pre-order traversal of the transformed structure to assign the new *id*s to the active members. The improvement is analogous to this of the previous schemes. As for the subsequent rounds, there is still improvement compared to an arbitrary assignment of *id*s, but the higher the rounds, the more uncertain or degraded this improvement becomes. In reality, the resulting values for these metrics may be much lower, since members may find paths of lower distances than these designated by the transformed structure.

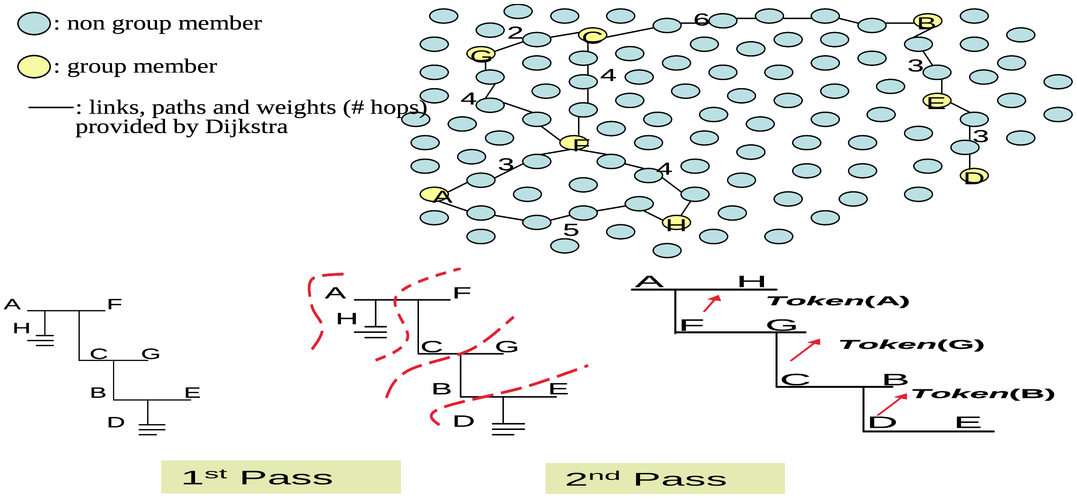


Figure 3.14. Example of the formation of a Hypercube structure from a network graph with arbitrary configuration of the group members: The 1st and 2nd pass are first executed on the network, and if the structure contains more than one path, MST manipulation is triggered.

3.5.3. Performance Analysis

1st Pass: All subgroup members except for the peers of the initiators become initiators themselves and attempt to find a peer or a successor initiator. Let the number of subgroup members be n. The peers of initiators are denoted as $x < \frac{1}{2}n$. The *Join* flag is sent only from the initiators to the members in their proximity lists that are scanned. Members that *accept* the request to *join* broadcast their decision and make it known to their proximity lists. By making use of the multicast advantage, the *Accept* control flag packet is transmitted once to their proximity, but goes through all the virtual links reported in their proximity lists. Initiators may scan more than one member, in order to find one to join (because they are using the merged proximity lists, and cannot directly check the status of certain members). Hence, we assume that in average, an initiator scans half of the members reported in its proximity list. Members that get scanned respond by “Refuse” until one that *accepts* is found. In summary, keeping the notation of the previous section, the overhead incurred because of the exchange of control flags and other information is computed as follows:

Join Flag: it is sent by (n-x) members and received at maximum by: $\frac{1}{2}(n\text{-}x)\, D$ members.

$$CCost\ (Join) < \sum_{i=1}^{n-x} \frac{D}{2}\, max_j R(i, j) \times K_S.$$

Refuse Flag: It is transmitted at maximum by $\frac{1}{2}(n\text{-}x)\, D$ members.

$$CCost\ (Refuse) < \sum_{i=1}^{(n-x)} (\tfrac{D}{2} - 1) \max_j R(i, j) \times K_S.$$

Accept Flag: It is broadcast by all members except for the starting point. An upper bound for *CCost* becomes: $CCost\ (Accept) < \sum_{i=1}^{n-1} D \max_j R(i, j) \times K_S.$

Exchange of Proximity Lists: The x pairs exchange their proximity lists and merge them. Also, let PK_S denote the bit size of the packet that carries a member's proximity list. The overhead incurred for this operation becomes:

$$CCost\ (List\ Exchange) = 2\times \sum_{i=1}^{x} \max_j R(i,j) \times PK_S,\ \ Latency{:}\ Lt < \sum_{i=1}^{x} \tfrac{D}{2} \max_j R(i,j)\,.$$

According to our previous definition, we select: $PK_S = D\times K_S$. Therefore, the overall *CCost* due to the 1st Pass operation becomes approximately:

$CCost\ (1^{st}\ Pass) = CCost\ (Join) + CCost\ (Refuse) + CCost\ (Accept) + CCost\ (List\ Exchanges)$.

$$CCost(1^{st}Pass) < [2\sum_{i=1}^{n-x} \tfrac{D}{2} max_j R(i,j) - \sum_{i=1}^{(n-x)} \max_j R(i,j) + \sum_{i=1}^{n-1} D \max_j R(i,j)\ ]\times K_S$$

$$+2\times \sum_{i=1}^{x} \max_j R(i,j) \times PK_S$$

$$CCost(1^{st}Pass) < [2\times \sum_{i=1}^{n-x} \tfrac{D}{2} max_j R(i,j) - \sum_{i=1}^{(n-x)} \max_j R(i,j) + \sum_{i=1}^{n-1} D \max_j R(i,j)$$

$$+2\times \sum_{i=1}^{x} D \max_j R(i,j)\ ]\times K_S < [D(n\text{-}x) - (n\text{-}x) + D(n\text{-}1) + 2xD\ ] \max_{j,i} R(i,j) \times K_S$$

$$CCost\ (1^{st}Pass) < [D\ (2n\text{+}x\text{-}1) + (x\text{-}n)] \max_{j,i} R(i,j) \times K_S\,.$$

2nd Pass: Whenever the status of all members locally associated with one paired initiator becomes known, the initiator matches all members that are available in its merged proximity list. It individually notifies each member about its new peer. The merged lists may contain at maximum $2D$ elements. However, some of the elements are duplicates, or in practice each individual list contains less than D elements. To simplify the case, if we assume that each proximity list contains the same number of active members, then there are $y = \frac{n}{2x} < 2D$ active elements in each. Hence, each initiator notifies $(y+1)$ members of their new peers. It is obvious that the status of an initiator becomes known, when all the members of all initiators that are “schematically” found below the given one, are accommodated. Therefore, the overall latency of the 2nd Pass is defined by the longest path of initiators, from the starting point to any leaf initiator. We can now compute the metrics related to the 2nd Pass algorithm:

$$CCost(2^{nd}Pass)<\sum_{i=1}^{x} y \max_j R(i,j)\, PK_S=\sum_{i=1}^{x}\frac{n}{2x}\max_j R(i,j)\, PK_S<\frac{n}{2}\times D\times \max_{j,i} R(i,j)\, K_S$$

$$Lt\ (2^{nd}\ Pass) = \max_Z \sum_{i=1}^{z}\frac{n}{2x}\max_j R(i,j)\text{, where } z \leq x.$$

MST Manipulation: In this part, only half of the members actively participate. Furthermore, the Hypercube structure replaces the MST in the original algorithm. The current overhead results from manipulating the structure similarly as the MST in the previous heuristic. We quote this overhead, as computed earlier, and we adjust it to this framework:

$CCost\ (HCube\_aux) = 2\times Wgt(HMST)\times K_S$,

$Lt\ (HCube\_aux) = 2\times Wgt(P_{\max})$, where P_{max} is the max. path w.r.t. virtual hops in the unfolded Hypercube structure that consists of $\frac{1}{2}n$ members.

We can now compute the overall overhead for the construction of this auxiliary scheme:

$CCost\ (aux) = CCost\ (1^{st}\ Pass) + CCost\ (2^{nd}\ Pass) + CCost\ (HCube\_aux)$.

$$CCost(aux)<[(D(2n+x\text{-}1)+(x\text{-}n))\max_{j,i} R(i,j)+\frac{n}{2}D\max_{j,i} R(i,j)+2\frac{1}{2}n\max_{i,j} R(i,j)\ ]K_S$$

$$CCost(aux)<(D(\frac{5}{2}n+x\text{-}1)+x)\max_{j,i} R(i,j)\, K_S<(3D+a)n\max_{j,i} R(i,j)\, K_S\text{, where } x=an,\ a<\frac{1}{2}.$$

$$CCost\ (aux) < (3D+1)\times n \times \max_{j,i} R(i,j)\times K_S.$$

$Lt\ (aux) = Lt\ (1^{st}\ Pass) + Lt\ (2^{nd}\ Pass) + Lt\ (HCube\_aux)$.

$$Lt\ (aux) = \sum_{i=1}^{x}\frac{D}{2}\max_j R(i,j)+\max_Z \sum_{i=1}^{z}\frac{n}{2x}\max_j R(i,j)+2\times Wgt(P_{\max})$$

$Lt\,(aux) < \frac{1}{2} xD \max_{i,j} R(i, j) + \frac{nz}{2x} \max_{i,j} R(i, j) = (\frac{1}{2} xD + \frac{nz}{2x}) \max_{i,j} R(i, j)$, where $z \le x$.

$Lt\,(aux) < n \times (\frac{D}{4y} + \frac{1}{2}) \times \max_{i,j} R(i, j) < \mathrm{O}(n \times c') \times \max_{i,j} R(i, j)$, where $c' = \mathrm{O}(1)$.

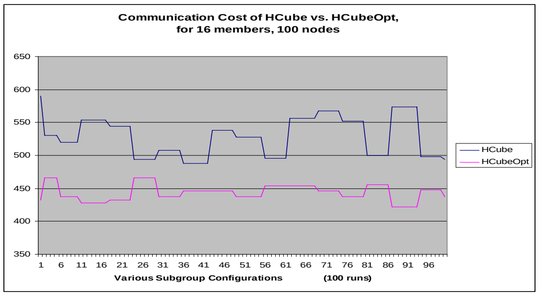
3.5.4. Simulations

Simulations Set-Up: The simulation set up is similar to the previous case, regarding the GDH-based protocols. We have tested the following cluster-subgroup scenarios: **Cluster Size:** [50…800], **Subgroup Size**: [8…64].

Simulation Results: Below we illustrate some indicative results on *CCost* and *RCos*t produced by the original Hypercube and its new *wt* version, measured the total number of hops required for the protocol to successfully terminate. The KM messaging is very heavy for the network nodes, so our aim is to reduce the overall number of bits (or packets) required and relieve as many nodes as possible from “relaying” large KM data. The following graphs reflect a number of important metrics that justify the value of our new *wt*-algorithm: (*a*) *CCost* of the *wt*-version *vs*. the existing one under various scenarios of secure group and network size, and (*b*) *CCost* (*aux*) and *CComp* (*aux*) required for the generation of the auxiliary framework, under various scenarios of secure group and network size. The graphs in case (*a*) are all scaled by a factor *K*. *CCost* and *RCost* are significantly reduced in all the scenarios captured by our simulations. Indeed, in most cases we observe that the improvement ratio becomes: $R_{COMM} = \frac{CCost(ING\_Opt,n,S)}{CCost(ING,n,S)} < \frac{1}{2}$. In fact: $0.32 < R_{COMM} < 0.7$. To illustrate these with an example, for a subgroup of size 16, and a network of size 200, the average relays produced are 561 for HCube_Opt, and 1604 for HCube, and $R_{COMM} = 0.35$.

We verify that the costs of the *wt* auxiliary framework of the *wt*-version match our analytical results. Indeed, we re-call the expression that estimates the *CCost* (*aux*) metric. We assume that the maximum number of neighbors of each member is around 10-12, and set $K_S = 8$.

Also, we select *max* ($R_{j,\ i}$) = 8, and D = 16. Then, we obtain: *CCost* (*aux*) < $3200\times|V\,|$. By computing this expression for a group of 32 members we obtain: *CCost* (*aux*, V = 32) < 102,400 bits. Indeed our simulations results verify that this upper bound holds, since some indicative values of *CCost* for group size *n* and network size *S* are the following: *CCost* (*aux*, *n* =32, *S*= 200) = 98106 bits, *CCost* (*aux*, *n* =32, *S* = 300) = 100725 bits , *CCost* (*aux*, *n* =32, *S* = 400) = 64136 bits. The same is the case with the rest of the group sizes we have included in our simulations. *CCost* (*aux*) increases as the network size grows, until some threshold value is reached. Then, the metric starts decreasing. This happens because two members are "connected" until the hop distance between them reaches a certain threshold. After the threshold is exceeded, the members become disconnected. As the network size increases, the density of group members decreases, and naturally the "neighbors" of each member decrease. The less the neighbors of each node are, the lower the overhead for the auxiliary framework is. We also verify that the metrics associated with the auxiliary framework are kept low and add little to the overall overhead produced by the execution of *wt*-HCube, even under a growing group and/or network size. On the other hand, the control messages used for the framework generation have size K_S<<K. We emphasize again that a new framework needs not be computed every time a *wt*-KA protocol is executed. The frequency of this event depends on the dynamics of networks and on our own specifications and requirements. For example, it may be recalculated whenever the performance of a protocol degrades to the median of the best execution (first) and of the average *nt*-execution.



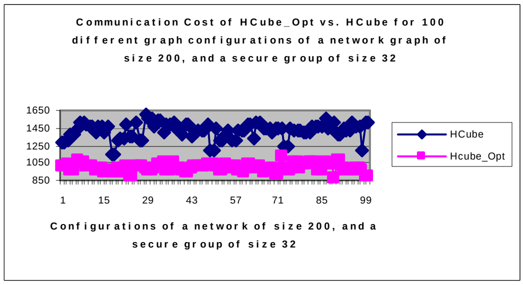


Figure 3.15. CCost of HCubeOpt vs. HCube for: (1) for 100 different group configurations on a network of 100 nodes (100 different runs) for a group of size 16, (2) for 100 different group configurations on a network of 300 nodes (100 different runs) for a group of size 32. HCube_Opt results in substantially superior performance for all the different configurations, in the two scenarios illustrated (better than this achieved with the previous wt- schemes).

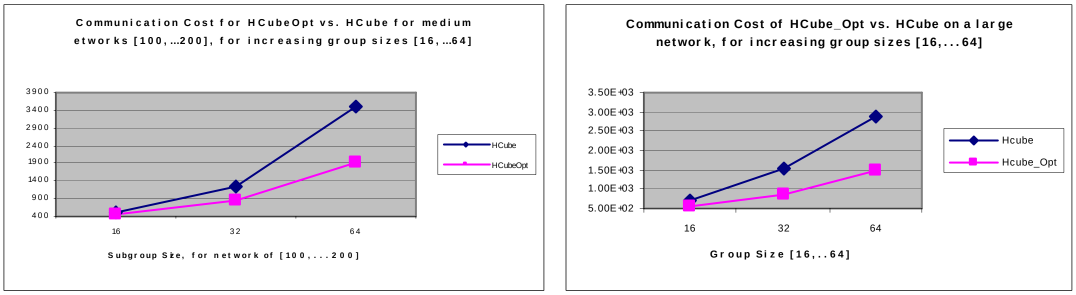


Figure 3.16. CCost of HCubeOpt vs. HCube w.r.t. the size of the group [16, 32, 64] in a network of (a): [100, 200] nodes, (b): [200, 500] nodes. HCubeOpt reduces the overhead in half. The performance difference grows even bigger as both the size of the group and network grow. So, HCube_Opt not only presents superior performance, but scales much better as well.

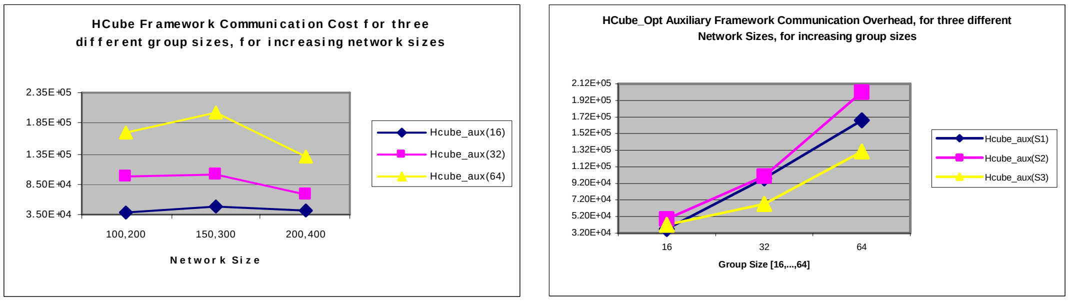


Figure 3.17. HCube_Opt CCost (aux) (bits): (a) three different group sizes for an increasing network size: [100, 500], and (b) three different scenarios of network size: S1=<100, 200>, S2 = <150, 400>, S3 = <200, 500>, for an increasing group size: [16, 32, 64]. The costs increase with the group size, as expected. They also increase with the network size and then at some threshold point they start decreasing (as discussed in the previous section).

3.6. Conclusions

In this section, we contributed towards the design and analysis of more sophisticated approximation - optimization methods of the communication, routing, and latency functions (metrics) of the following KA protocols: GDH.1, GDH.2, ING, BD, and Hypercube. We developed new algorithms (heuristics) for generating topology-oriented communication schedules, to be used by these protocols. The heuristics introduced achieve significantly better approximations of the metrics we want to optimize. Our comparisons of the new topology oriented and the original KA schemes are done via simulations. The new protocols achieve a dramatic overhead reduction. We believe that there is scope for even more efficient topology oriented approaches and we consider efficient simulations of KA protocols over multi-hop ad hoc networks as an interesting open problem.

3.7. *Wt* adaptation of TGDH on logical networks, subject to underlying routing.

TGDH is the protocol used within the subgroups of the most efficient and robust representative of the Octopus-based schemes, MOT. Similarly to the previous sections, we extend TGDH to a novel distributed and topology aware scheme: DS-TGDH. In particular, our aim is to modify TGDH to: *a*) be feasible in the most general resource-constrained flat MANET where no nodes with special capabilities exist, *b*) produce considerably lower overhead for the network nodes involved, *c*) handle disruptions (network dynamics and failures) with low cost and impact. We consider the underlying routing protocol in our design, and we apply a distributed TGDH version over a robust schedule, optimizing metrics of interest. We focus on the design and analysis of the “stealthy” TGDH and compare it with the original. Through our analysis and results we provide more realistic comparisons that more accurately advocate the pros and cons of each protocol over the environment of study.

The primary focus of prior work on the Octopus schemes was the analysis of the proposed schemes, with the KG as the only performance attribute. However, since each of the schemes considered relies upon the former functions more or less, or in a different way, our previous evaluation is not totally fair. The consideration of network parameters that can accentuate the individual characteristics of the protocols and the differences in their performance will provide more realistic results. For example, scheduling is not required for the subgroup in the centralized (O), in contrast to MO and MOT where members interact among themselves for the subgroup KG. The need to apply the subgroup schemes in MO and MOT (GDH.2 and TGDH) over a schedule that optimizes metrics of interest comes into the play. The network assumptions made in TGDH [53], (e.g. leader reaches all members via a single broadcast), are too simplistic to apply to a general MANET and mitigate the need to integrate the discussed functions in their design. In real multi-hop networks, a cross-layer consideration appears to be essential for a concrete framework. Previous results have designated MOT as the most efficient for the environment discussed. It would be worth exploring if MOT still prevails after a re-evaluation of the same schemes under more realistic assumptions.

Dividing such group into subgroups, each one corresponding to a fully connected graph of members, reachable by the subgroup leader via a single broadcast, would permit the execution of the original "centralized" TGDH exactly under the assumptions described in [53]. This approach is impractical for a large group: a high number of subgroups will be formed, very sensitive to even subtle mobility changes, and it is quite likely that they will contain very few members, even a single one. Even if such subgroups are in very close proximity they still cannot be merged. The result is a considerable waste in network resources, and an infeasible execution of Hypercube, where the crucial parameter *d* is likely to be high and unstable. Then, the resulting inter-cluster signaling overhead outweighs the benefits of Hypercube for the inter-cluster communication.

In the following sub-sections we present an adaptation of TGDH to meet the requirements of a general MANET. In particular, we modify TGDH so that: *a*) it is made distributed, *b*) it is executed under a schedule that optimizes our own defined routing and robustness metrics, under a topologically aware consideration, *c*) it tolerates failures and disruptions with low cost, *d*) it is far more efficient. We denote this novel scheme as **DS-TGDH** (*Distributed TGDH with Schedule*) and evaluate both protocols under the new assumptions.

3.7.1 Fault-tolerant Extension of TGDH

The pre-agreed, *ID*-based schedule of TGDH members is in fact the KG algorithm, and does not deal with communication regulation. That would be redundant since all the messaging goes through the sponsor. The BKs of each member at a given tree level is unicast to the sponsor who waits to collect all BKs of the same level and then combines them to a single broadcast to all the rest of members. This broadcast message signals the advancement in the tree level. Each member is now able to compute the designated secret value for the next level. It then blinds it and unicasts it to the sponsor. The same process is repeated for all levels upwards the tree until the root is reached. Hence, any two members communicate with each other via the sponsor. With this scheme, KA is executed in a centralized manner. TGDH may perform as claimed in [7] under the constraints of a limited network where any member is able to reach all group members with 1-hop and where the sponsor acquires extra bandwidth and power capabilities. Such a scenario does not reflect the general MANET case, so the resulting performance metrics are not realistic. The most significant drawbacks of TGDH as presented in [7] are the following:

(*a*) Any trusted member, with sufficient bandwidth and power capabilities should be ready to assume the duties of a leader. The most important *constraint* for MANETs is the existence of such node(s). What is more, members may not be able to reach the sponsor via a single transmission anyway. So, both directions may introduce excessive multi-hop routing. (*b*) Simultaneous transmissions of BKs at any tree level to the sponsor through multi-hop routing in a limited network area may increase the probability of collisions at the MAC layer, and thus the probability of re-transmissions, deteriorating even more the performance of the scheme. (*c*) Power is considered valuable resource in MANETs, so it is undesirable for members to serve frequently as relays in heavy KM messages. Also, the sponsor is burdened with heavy tasks that consume its residual energy all too fast. (*d*) The sponsor still remains a single point of failure communication-wise. If it fails during key establishment, the protocol is stalled until a new leader emerges, and a number of BKs must be updated.

TGDH has some **considerable advantages** as well: (*a*) it is simple, no sophisticated scheduling is required, (*b*) failure of a member has no impact on the protocol other than the update of a logarithmic number of BKs, (*c*) every member knows the BKs of all subgroup members and can proactively anticipate dynamic membership changes (evictions) that would result in the need to reconfigure the KG algorithm and restore the group key with minimal latency. Considering the pros and cons of TGDH, we present a more efficient, distributed version that uses a transmission schedule algorithm to mitigate the weak points of TGDH.

3.7.2 DS-TGDH Overview: Communications Schedule

3.7.2.1. Preparation Stage Overview

The sponsor initially collects through members' registration and link state information, all the required information about its subgroup members. We assume **a generic underlying routing protocol with the property that it always finds the minimum path** (i.e. Dijkstra). It provides all nodes with the paths to at least the nearest subgroup members (with respect to the number of hops), and finds at least one path connecting two members, as long as both are within the same cluster. The upper bound in the number of hops between members considered immediate "logical one-hop neighbors" is dynamically set. Routing provides members with information about the **robustness** of the paths to "neighbors". For example, if we know the motion parameters of two virtual neighbors (speed, direction, radio propagation range), and their coordinates, we can determine the duration of the time these two nodes will remain connected, denoted as Link Expiration Time (LET) [55]. The **routing path *r*** is characterized by the minimum among all LETs of its links, denoted as Route Expiration Time (*RET*). RET indicates the overall stability of a path: as soon as a single link on a path is disconnected, the entire path is invalidated, as argued in [55]. We as well choose *RET* as a component of our "path robustness metric", generated and communicated to members. We select the normalized product of the residual energies of all nodes included in the path: *EN* = $\frac{1}{|r|}\prod_{i \in r} E_{res}(i)$ as an additional metric to characterize path robustness. In this case we do not consider the minimum residual energy as a valid metric, because a node may participate to more than one routing paths. We rather choose to average the residual energies of nodes along the same routing path and we characterize the "**robustness**" of a routing path *r* by the value: $R_r = a \times EN + \beta \times RET$. The contribution of each parameter towards the computation of R_r can be fine tuned through a and β . Network nodes are equipped with GPS or other similar devices that allow the computation of their own position, and of distances among them as well. Member *x* maintains and updates a "proximity" list $\boldsymbol{L_x}$ with cardinality $|DMx|$, that includes all the "virtual neighbors", and the collected routing metrics for each.

3.7.2.2. Execution Stage Overview

A member *j* that belongs to the schedule tree *T*, selects one among the available routing paths at level (*l*-1) that leads to a member $k \in L_j$. Member *j* generates now the offspring <*j*, *k*> for level *l*. Members *j* and *k* are connected with a logical link, one of low cost in terms of routing, and are considered siblings at level *l*. They update all members in their proximity (lists $\boldsymbol{L_j}$, $\boldsymbol{L_k}$) of the new event (TreeFlag(*k*) is set to "*busy*" mode). Then, they individually proceed to generate the offspring for the next level from the lists L_j and L_k respectively. If another member *r* that did not receive the updates attempts to enlist member *j* or *k* in the tree they just "refuse" and *r* checks its remaining options. The virtual tree expands according to the following simple idea: *The more robust the members, the higher in the tree they will be placed*. A member *j* arranges L_j with respect to the metrics discussed and attempts to use these members in this order one by one as offspring (in successive levels), unless they are already "used" by other members. Whenever a leaf is reached (i.e. a path cannot be expanded anymore), all the information regarding the members that belong to the path traverses up the root. The root then can check if all members are included in the tree (optional). In fact, tree members need only know their immediate parent, grandparent (if any) and the parent's first sibling, in addition to their own siblings. Member *j* gives priority to this "unselected neighbor" *m* that satisfies at least one of the following rules in the order they are stated:

Rule_1: $m \in \{L_J \cap L_B /T\}$, where T is the current tree version, and B is the parent (or first sibling) of j. By this rule, we want to ensure that if j becomes faulty then B can take its place in the tree, and become a sibling of all the previous siblings of j, so that the impact of a failure is minimal (no need to prune the tree). To ensure that the selection of m at this stage ensures a relatively strong tree link also, we simultaneously impose that $R_r\,(j,m) > Th$, where Th is a threshold value. If there is more than one choice, the "shortest path" one is preferred.

Rule_2: $m \in \{L_j \ / \ T\}$and $r_{jm} \leq D$, s.t. $\forall\, x \in \{L_j \ / \ T\}$and $r_{jx} \leq D$: $R_r\ (j,\ m) > R_r\ (j,\ x)$. If rule_1 cannot apply, then rule#2 selects this member with the strongest link to j, among those with routing distance less than D hops.

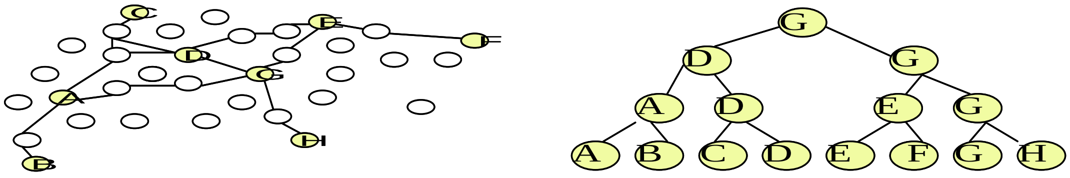


Figure 3.18. TGDH schedule formation from an arbitrary network graph.

These rules ensure two basic requirements for robustness and efficiency: *1)* Members with high risk of getting disconnected occupy the fewest possible internal nodes and are pushed towards the leaves. The impact of their "loss" is mitigated as much as possible when pruning the tree, *2)* Members placed high in the tree are more likely to satisfy rule1, since they have more neighbors available. Their failure would affect the schedule of a larger member subset, so it is important to remedy such failures with minimum extra cost and latency. At the other extreme, failure of a member associated only with a leaf has no impact to the tree. So, a *greedy* strategy for the offspring selection appears to be the best to pursue. Members, at all times, attempt to use their best available options, and keep pushing the "worst candidates" towards the leaves. Even if root A fails, the idea is that its nearest former neighbor, e.g. B, replaces A in the tree, and routing connects to B all nodes prior connected to A. It is very likely that these nodes remain relatively close to B as well. The other tree nodes remain unaffected. Our approach indirectly achieves: *a)* minimum delay schedule tree since members are enlisted in the tree in a *first come first served* manner, and *b)* producing a relatively balanced tree, since at any level, all members are free to expand.

3.7.2.3 Analysis and Evaluation of DS-TGDH

A. Evaluation of the Initial DS-TGDH Schedule

Member *j* does less than $2|L_j|$ comparisons of cost C_{CMP} = O(1) each. When *j* is handed the token, it updates the list L_j of the status change in its flag (*busy*). While a member *y* is being "tested" during the selection process by member *x*, its flag is set to "*lock*". Those members that contact *y* in the mean time, cannot enlist it as long as its flag remains to "*lock*" mode. After the selection stage ends, *x* unlocks *y*'s flag by setting it either to: *busy* or *free*. We obtain the DS-TGDH computation cost C_P, by summing all ramifications of the former algorithm under the worst case scenario: $C_P = L_j \times (3C_{CMP} \times K_S)$ (*rule1*) + $L_j \times (3C_{CMP} \times K_S)$ (*rule2*) + $(3C_{CMP} \times K_S)$ (*process_results*) = $(2L_j+1) \times (3C_{CMP} \times K_S)$

B. Impact of Dynamic Events on the DS - TGDH Schedule.

We want to guarantee a transmission schedule that can anticipate dynamic or membership changes as well. From the security point of view, it is shown in [53, 7, 10] how members' evictions and additions are handled in TGDH to preserve the basic fundamental security properties: *Forward*, *Backward*, *Group key secrecy*. Here, we want to show in addition how to resume TGDH schedule in the event of disruptions of any kind, with the minimum amount of overhead and latency, and ensure that the updated schedule is still functional and efficient.

B1. Eviction: The virtual tree may need reconfiguration after the removal of a member. The basic idea behind the eviction algorithm is the following:

Either the parent (*A*) or the first sibling (*X*) of the evicted member (*B*) substitute *B* in the tree path wherever it appears. Without loss of generality, let *B* be replaced by *A*. All former siblings of *B* must now become *A*'s siblings. For all members $x \in (L_B / L_A)$, routing finds the shortest paths to *A*, and they are all added to L_A. Compared to the rest of *B*'s former siblings, members *A* or *X* are either more robust or lie closer to *B*. It is thus quite likely that some of *B*'s former siblings: *a*) already belong to L_A or L_X, and the routing needs not generate extra paths and *b*) do not belong to lists L_A or L_X, but the paths to be formed are relatively short, since both ends lie in the proximity of the evicted member *B*. These statements are quite likely to hold if the network is relatively dense. The decision of which member will substitute member *B*, is assigned to the **last sibling** chosen by *B*, denoted as *L*. *L* is the least preferred member from the point of view of robustness and path length among the remaining available "neighbors" of *B* placed in the tree. All previous selections are considered more stable. The routing finds the shortest paths from *L* to both *X* and *A*. The shortest one, whose robustness is no worse than some threshold, designates which member will substitute *B*. The reason is that we want the substitute member to accommodate all affected members with low OH, ensuring as high robustness as possible. It is more likely that the best selection for the least robust sibling is also the best for the rest. We could also find the shortest paths to *X* and *A* for all siblings, and select the one that accommodates best the majority.

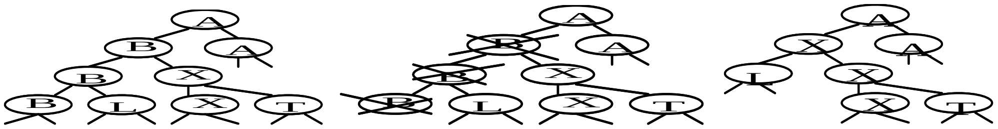


Figure 3.19. Schedule maintenance after B's failure, when parent A = P(B) "replaces" B.

B2. Addition: The routing finds the shortest paths from the new member *T* to members in the proximity, and the new proximity list L_T is created. Member *T* will be added as a leaf to this member in L_T to which it is connected via the minimum number of hops and fulfills a robustness criterion at the same time. A modification of *rule1* is used to select the best solutions. Initially, *T* looks for a member *B* so that both *B* and *Parent* $(B) \in rL_T$. This is necessary in order to make the tree robust to failures as discussed. If $R_r\ (T,B) > R$ and $R_r\ (T,Parent(B)) > R$, the solution is optimal, else if only $R_r(T,B)>R$, the solution is considered suboptimal. If none of the above is true, we apply a version of *rule2* over the members of L_T:

T is attached to this member J with which they share the minimum hop path $r_r(T, J)$, for which robustness is among the highest in L_T. Then, J expands one level and becomes the parent and the first sibling of T.

C. Analytical Evaluation of Dynamic Events on DS-TGDH.

Eviction: Let L be the last selected sibling of B. L does only two comparisons to determine which candidate will replace B. Let A be replacing B. Former B's siblings, h_B, that do not already acquire paths to A, use routing to obtain such paths and the associated metric values. However, $h_B<|L_B|$, since a subset of L_B has been prior reserved by other tree members. Also, any $j \notin L_A$ is likely to have been enlisted by B towards the end of its path. Let the parameter N_E denote the average number of members that computes such paths. It holds that $N_E< h_B/2$.

Addition: About DM_T shortest routing paths are discovered. The total computation cost C_P for member T is derived similarly to the corresponding eviction cost. Hence:

$C_P = L_T \times (4C_{CMP} \times K_S)(rule1) + L_T \times (2C_{CMP} \times K_S)(rule2) + (2C_{CMP} \times K_S)$ $(process\ Results)$

$C_P = (3L_T+1) \times (2C_{CMP} \times K_S)$ (bits).

	DS-TGDH Schedule Adjustment Costs
Deletion Communication	$N_E \times K_S$ (> h_B /2)
Deletion Computation	$2C_{CMP} \times K_S$
Add Communication	$DM_T \times K_S$ (<D paths)
Add Computation	$(3D+1) \times (2C_{COMP} \times K_S)$

Table 3.8. DS-TGDH Schedule Adjusted Costs for Dynamic Changes

3.7.2.4 DS-TGDH *vs.* TGDH Analytical Evaluation

TGDH: h rounds are required for the computation of the group key. By then, each member has communicated $h \times K$ bits to the sponsor, and has received a total of $h \times n \times K$ bits, from which the useful data is contained only in $h \times K$ bits. This means that each routing path from the sponsor to any member carries $(n\text{-}1) \times K$ bits of "redundant" KM data per round. The total expected number of BKs exchanged is: $E[BK_S] = (n \times h)$ [members] + $(n^2 \times h)$ [sponsor].

DS-TGDH: The number of pairs that interact in the initial case is reduced to half after each round. This is reflected in the way we have constructed the schedule tree. The total number of tree nodes is $2\times n$. Hence, we have $2\times n$ exchanges of key parts over the designated routing paths. Any member must get the BKs of the members of its co-path. The member that participates actively at any step to the KG knows all the required BKs up to this point. This is true if we consider how the schedule tree is built. If member *A* is active at step *i*, it has been active in all previous steps from the start of its path, and has collected and sent all the required BKs (1 per step). Let h_{UA} denote the number of times *A* appears in the tree ($\neq A$'s path to the root, h_A). During the upward phase, *A* uses h_{UA} paths and sends a BK over each. It does not send the BK obtained at one round right away to its descendants, but waits first to collect all the keys of the members in its co-path. So, the distribution of the required BKs follows a top-down approach. The number of BKs distributed from a parent *A* to offspring *B* is equal to the number of hops *B* is away from the root. So, parent *A* receives a message $(h_A - h_{UA})\times K$ bits long and sends h_{UA} BKs to its offpring: i.e. for offspring *j*, at level *i*, *A* sends $(h_A - i)$ BKs $(i \le h_{UA})$. If a member is active at step *i*, it has been active during all steps $j<i$.

$\boldsymbol{h_{UA}}$**:** number of times member *A* appears in the virtual tree.

$\boldsymbol{h_A}$**:** hop distnce of member *A* from the root in the actual tree.

$\boldsymbol{h_A - h_{UA}}$**:** hop distance of member *A* from the virtual tree root.

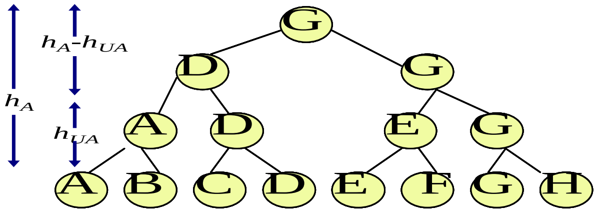


Figure 3.20. Virtual DS-TGDH and basic tree height parameters

Upward phase (UPWD): Member *A* collects all BKs required for the inactive members of its co-path before distributing any. It does (h_{UA}-1) exchanges of BKs, one with its peer for each round, until it becomes inactive.

Downward phase (DNWD): Member *A* uses each of the (h_{UA}-1) branches to send a BK to its corresponding offspring. For example, the BKs sent from *A* to *B* are: ($h_B - h_{UB}$). Member *A* receives from *D* a message (h_A - h_{UA})*K* bits long and sends h_{UA} BKs to its offpring: i.e. for offspring *j*, at level *i*, *A* sends (h_A-*i*) BKs ($i<h_{UA}$).

Analytical Results for DS-TGDH per member *A* and in total

A. Members Receive

Blinded Keys: BKs(*A*) = $(h_{UA}\text{-}1)_{\text{UPWD}} + (h_A - (h_{UA}\text{-}1))_{\text{DNWD}} = h_A$.

Routes Used: RU$_\mathbf{R}$ (*A*) = $(h_{UA}\text{-}1)_{\text{UPWD}} + 1_{\text{DNWD}} = h_{UA}$.

BKs (*Total*) = $\sum_{i=1}^{n} h_i$. **RU$_\mathbf{R}$ (*Total*) =** $\sum_{i=1}^{n} h_{UI}$.

Assuming h_{UA} = ½h_A in average, then: $E[RU_R]=E[\sum_{i=1}^{n} h_{UI}]=\sum_{i=1}^{n} E(h_{UI}) = ½\times\sum_{i=1}^{n} h_i$ (3.18).

B. Members Send

Blinded Keys: BKs(*A*,*level i*) = $(h_{UA}\text{-}1)_{\text{UPWD}} + (h_A - i)_{\text{DNWD}(i)} = h_{UA} + h_A\text{-}1\text{-}i$.

Routes Used: RU$_\mathbf{S}$(*A*)=$(h_{UA}\text{ -}1)_{\text{UPWD}}$ (1 BK/route)+$(h_{UA}\text{ -}1)_{\text{DNWD}}$ (h_A-½ h_{UA}) BKs/route in avg).

BKs (*Total*) = $(h_{UA}\text{-}1+\sum_{i=1}^{h_{UA}-1}(h_A - i)) = h_{UA}\times h_A - ½\times h_{UA}\times(h_{UA}\text{-}1) = h_{UA}\times(h_A \text{ - } ½ \times(h_{UA}\text{ -}1))$.

RU$_\mathbf{S}$ (*Total*, twice) = (h_{UA}-1).

Expected Total Sending Cost values for all Members:

***E*[*RU$_S$*]** $= E[\sum_{i=1}^{n} h_{UI}] = \sum_{i=1}^{n} E(h_{UI}) = ½\sum_{i=1}^{n} h_i$ (used twice) (3.19).

$\boldsymbol{E[BK_S]}$ = $E(\sum_{i=1}^{n} h_{UI}(h_I - \frac{1}{2}(h_{UI} - 1)))$=$E[\frac{3}{8}\sum_{i=1}^{n} h_I^2$]+$E[\frac{1}{4}\sum_{i=1}^{n} h_I$] $= \frac{3}{8}\sum_{i=1}^{n} E(h_I{}^2) + \frac{1}{4}\sum_{i=1}^{n} E(h_I)$

(3.20).

The schedule tree is not necessarily balanced. Any arbitrary schedule tree can be derived from a balanced one of the same size, since the following observation is true: to extend a path in the balanced tree from *h* to *h*+1 (2 nodes are added – property of schedule tree), another path must be abbreviated from *h* to *h*-1 (two nodes get eliminated). By induction, we see that for a path to be extended from height *h* to *h*+*k*, one or more other paths must be abbreviated by *k* hops totally. The average individual member participation in unbalanced trees remains the same for both phases. Hence: $\sum_{i=1}^{n} h_i = \sum_{i=1}^{n} h$ =$n \times h$ (3.21),

and the previous results of (3.18), (3.19), (3.20) can be revisited.

(3.22), (3.19), (3.21) => $E[RU_R]$= $\frac{1}{2}(n \times h)$, $E[RU_S] = \frac{1}{2}(n \times h)$ (3.22).

From (3.20), (3.21), with the use of probability theory we obtain:

$E[BK_S]$=$\frac{3}{8}\sum_{i=1}^{n} E(h_I{}^2)$ +¼($n \times h$)=$\frac{3}{8}\sum_{i=1}^{n}((E(h_I))^2 + \sigma_h^2)$ +¼($n \times h$), where σ_h^2 is the path variance (expected low).

Hence, $E[BK_S] = \frac{3}{8}(n \times h^2) + \frac{3}{8}(n \times \sigma_h^2)$ + ¼ ($n \times h$) (3.23).

Total Comm. OH	**TGDH**	**DS –TGDH**
BKs (per member)	h, $n^2 \times h$	(h-½× (½×h-1))×½ h
BKs Total	($n \times h$) + ($n^2 \times h$)	$\frac{3}{8} n \times (h^2 + \sigma_h^2)$+¼ $n \times h$
Routes Used or Discovered per member	Used: 1 (regular member), n (sponsor), Discovered: any	Used: 2× (h_{UA}-1), Discovered: 2×D
Total # of Routes Used or Discovered	Used: n, Discovered: >n	Used: $\frac{1}{2}(n \times h)$, Discovered: $2 \times n \times D$
Redundant Info	$h \times n \times (n$-1$) \times K$ bits (total)	No

Table 3.9. Summary of analytical results of TGDH vs. DS-TGDH.

3.7.2.5 Simulations Results for DS-TGDH and TGDH

i) Simulation Set-Up and Discussion: The simulation settings are similar to these for the previous heuristics regarding GDH-based and Hypercube schemes. We use two methods to select the subgroup leader: either randomly, or we pick the member with the largest "member" degree. At the end of the subgroup "registration" period, the sponsor piggybacks the list of the legitimate members into the routing packets. We have tested the following scenarios: **Cluster Size:** [100,…500], **Subgroup Size**: [2,…60].

ii) Simulation Results: Below we illustrate some indicative results on *RCost* produced by both DS-TGDH and TGDH, measured in terms of the total number of hops (relays) required for the protocol to successfully terminate. The following graphs reflect *RCost* produced from the KG in both schemes. The KM messaging is very heavy for the network nodes, so our aim is to reduce the overall number of bits (or packets) required and relieve as many nodes as possible from "relaying" large KM data. Also, we wish to dynamically distribute the KM tasks of a single-point of failure leader to all members. Indeed, DS-TGDH results in significant savings in terms of *RCost*, and in most cases the associated ratio is: $\frac{DS-TGDH(RC)}{TGDH(RC)}$<0.3. The total savings from the use of DS-TGDH are even more significant if we consider also the amount of redundant KM data, as calculated in our analysis, relayed by nodes during every round of TGDH. To illustrate the above with an example, for a cluster of size 300, and a subgroup of size 35, the averaged relays produced are 457 for DS-TGDH, 2987 for TGDH. This means that the overall redundant data communicated via the relays is: ${}^{2987}\!/_{2} \times 34 \times \lceil \log_2 34 \rceil \times 1024$ bits=311,986,170 bits. The reduction in *RCost* achieved by DS-TGDH is obvious, for any subgroup size, cluster size, and subgroup configuration. In many cases, the ratio of the reduction is greater than 3. DS-TGDH increase rate in *RCost* is much less compared to TGDH rate, subject to the cluster or subgroup size. Hence, the new protocol is not only more efficient and robust than the original, but scales much better as well.

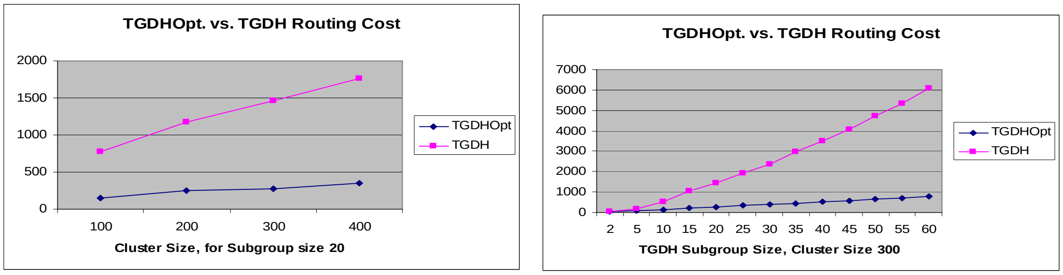


Figure 3.21. Total RCost for TGDH vs. DS-TGDH in terms of (1) cluster size, for subgroup size of 20 members, (2) subgroup size, for cluster size of 300 nodes.

3.7.2.6 Summary of the DS-TGDH Contribution

In this section we focused on the design of a decentralized, low cost fault-tolerant version of TGDH, denoted as DS-TGDH for a general MANET. DS-TGDH benefits from a combined consideration of the underlying routing and the existing logical TGDH KG algorithm to enhance the performance and behavior of the original ancestor. We presented a *wt* schedule on top of which DS-TGDH is executed, and we analytically evaluated the overhead of the schedule generation and execution, for the initial and the steady state. We explored the potential of DS-TGDH through different views, to provide accurate and spherical evaluation of both algorithms, and of their strong and weak assets. Through our analytical work and simulation results we show how we can achieve better performance with our scheme.

3.8. Overall Summary and Conclusions

In this chapter, we made a transition in modeling and analyzing a number of the most significant KA schemes with underlying routing consideration. This transition allows us to compute the actual overhead incurred to the multi-hop ad hoc network from the application of a given KA scheme. Through the combined consideration of the underlying routing and the original logical KG, we extended and improved the following KA protocols: GDH.1, GDH.2, ING, BD, Hypercube, and TGDH. We achieved this, by applying every discussed KA scheme on a topologically-aware (*wt*) communications schedule. Each protocol poses two different optimization problems as the routing structure of each defines a specific optimization function of the metrics of combined communication-routing cost and latency. We have seen that most of these functions correspond to *NP*-complete problems. Consequently, our main objective has been to find good approximation solutions that are also feasible in our discussed environments. Towards this end, we introduced a number of novel lightweight heuristics for generating communication-schedules that shift the performance of the discussed KA schemes towards the optimal, given the constraints imposed. Indeed, the resulting schemes present notably superior performance in terms of combined communication cost, routing cost, and latency compared to their ancestors. Next, in chapter 4, we are continuing on extending the new protocol versions, focusing on the impact of mobility on the *wt* KA protocols, and on improving on the robustness, fault-tolerance, and efficiency of the *wt-KA* protocols subject to the frequent dynamic changes in the network of study.

Chapter 4: Design of Improved, Resilient Key Agreement Schemes

4.1. Introduction: Towards Robust and Resilient Key Agreement Protocols

The inability of most contributory protocols to tolerate or recover from node failures, network partitions or merges, with low overhead is their major drawback in MANETs. For example, upon a failure at any point during group key establishment, the majority of contributory protocols start over from scratch. This is so because their event flow-chart (logical algorithm) does not anticipate such failures. The impact of such failures either on the communication schedule used and/or on the security properties of the KA protocols is quite significant and has not been studied to date. Our objective is to improve the robustness and resiliency of a number of KA schemes when they are run on the dynamic infrastructure-less environment of interest, so that they tolerate or recover from failures with the minimum possible extra overhead. In this section we are conducting a comprehensive investigation of the resiliency of the improved *wt*-versions of a number of contributory protocols (i.e. GDH.1, GDH.2, ING, BD, TGDH, Hypercube, etc.), as designed in Chapter 3. To improve the robustness and fault-tolerance of these protocols, we are further extending them, by introducing new algorithms and mechanisms (rules) in their event flow-chart, that ensure that all possible disruptions, at any phase of key establishment, are handled so that: *a*) the extra overhead and latency incurred to the network is maintained as low as possible, and *b*) the minimum amount of information that has been gathered up to the point of disruption is lost.

Method: We are methodically testing all potential failure scenarios on a given protocol at any point during execution. For each case of failure, we study its impact on: (*a*) the keying algorithm and messages, or equivalently the logical flow of the key generation algorithm, and (*b*) on the existing communications schedule, and consequently on the group connectivity. We want to understand how both (*a*) and (*b*) are affected from a “disruption”, and introduce (if possible) lightweight techniques to make the protocol capable of circumventing this “disruption” or resuming with very low extra overhead, from the perspective of both (*a*), (*b*).

Another very significant source of disruption, that makes sense mostly for centralized or hybrid schemes, is the failure of the group leader. On the other hand, different KM schemes, rely upon a different set of auxiliary network functions, (such as routing, communications schedule, leader and/or simple member recovery schemes, leader election, clustering, etc.), and in a different way. For example, in a contributory scheme, the impact of a leader failure (if there is one) is not differentiated in most cases than the impact of a simple member. On the contrary, in a centralized or hybrid scheme, the impact of a leader failure is very significant, as the whole protocol stalls, waiting for a new leader to undertake the execution, whereas the impact of a member’s failure, usually has not effect to the normal flow of the protocol, other than the operation of re-keying in order to preserve the fundamental security properties. This vast differentiation in the two categories of protocols results in different requirements for the design of leader election algorithms, and leader/member recovery schemes. Hence, it becomes obvious, that for a fair and realistic comparison of a set of KM protocols, we should consider in their design and evaluation, at least those network functions that are executed differently or incur different overhead for each scheme. The structure of this chapter is the following: in section 4.2, we introduce a novel version of Hypercube (R-Proactive). Dynamic events are handled proactively, and the novel scheme results in improving a number of important metrics compared to the original scheme. In section 4.3, we are extending all the discussed *wt*-versions of KA protocols to handle dynamic events with low impact in most cases. In section 4.4, we present and analytically evaluate a leader election algorithm, suitable not only for Octopus-schemes, but for many more centralized or contributory schemes.

4.2. Robust, Fault-Tolerant Hypercube (R-Proactive Hypercube)

In this work, we extend Hypercube to tolerate multiple member failures with low cost. We achieve this through the integration of Hypercube with a novel adaptively proactive algorithm. We assume that members have already been authenticated via some underlying mechanism and we focus on the design and analysis of a fault-tolerant Hypercube that can contribute to the robustness and efficiency of Octopus schemes. We evaluate and compare our algorithm with the existing approach [1, 2]. Through our analysis and simulations we demonstrate the superiority of the new algorithm in terms of robustness and efficiency. Hypercube constitutes the core of the Octopus schemes. The protocols applied to each subgroup are not fault-tolerant, but can be made to recover from failures with low overhead. The latter is also greatly attributed to the limited subgroup size and topological proximity of the subgroup members. This is not the case with the application of Hypercube, and that is where one of the main vulnerabilities of Octopus schemes lies. Hypercube does not exploit topological proximity of nodes. It ends up connecting any two subgroups, possibly through a considerable number of relays. A pre-agreed schedule is used based on some attributes of the leaders, like their "hashed" *IP*s, or *ID*s dynamically assigned beforehand. In particular, for the 2^d leaders to agree on a group key, *d* rounds of key exchanges are required. In every round, each leader selects a different peer to perform a key exchange with. Obviously, not all of them can be in relative proximity to each other anyway. The dynamic network characteristics result in frequent membership and mobility changes, with considerable extra overhead. The existing scheme does not anticipate disruptions. Upon failure(s) at any point during KA, Hypercube must start over: the underlying routing is invoked again, a significant amount of relays become involved in the group key exchange, and considerable delay (reflected in the total number of rounds required until the successful protocol termination) burdens the network. Extending Hypercube to tolerate failures with low overhead or latency becomes critical for its own feasibility and for the feasibility of Octopus protocols. In section 4.2.1 we describe the requirements of our framework, in 4.2.2 we introduce our *R*-Proactive algorithm, and in 4.2.3 we sketch a proof of how fault-tolerance is achieved. In 4.2.4 we present the analysis of the *R*-Proactive *vs. Basic* and the most indicative results of the comparative performance evaluation, and in 4.2.5 we present the simulations set-up and our results.

4.2.1. Requirements, Assumptions, and Fault-Tolerance for Hypercube protocol

Notation_1: Let $B(x) = a^x$ be the blinding of value x (exponentiation of x under base a) and $\varphi(x) = x \bmod n$. For simplicity, we often **replace** expression ***B*(*X*) with *BX***.

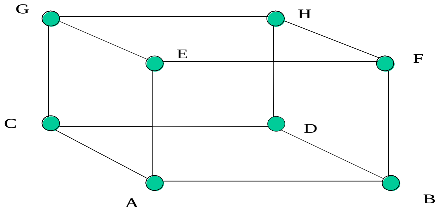


Figure 4.1. Hypercube Illustration with d=3.

We assume that the underlying routing is capable of establishing end to end paths, avoiding intermediate link failures. We do not consider cases where the network is partitioned. We **mainly focus on member disruptions and failures** during group key establishment. The impact of such failures either on the communication schedule or on the security properties of Hypercube is quite significant and has not been studied to date. Even a relatively low failure rate increases considerably the number of rounds it takes for the protocol to terminate. We wish to extend Hypercube to anticipate failures so that: *a*) the extra overhead and latency incurred in the network is maintained as low as possible, and *b*) the minimum amount of information that has been gathered up to the point of failure is lost. Also, we want to preserve the fundamental properties of *Forward*, *Backward* and *Group key Secrecy*. From this perspective, a member *k* that is no longer considered legitimate (misbehaving, evicted, faulty, disconnected) at any round should not be able to compute the final group key, even if it still receives all the future blinded values exchanged. To better illustrate this with an **example of**

eight nodes (Fig. 4.1), assume that member *B* is evicted, and should be excluded from the final group key $K = a^{a^{a^{ab} a^{cd}} a^{a^{ef} a^{gh}}}$. The elements: $a^{a}, a^{a^{cd}}, a^{a^{a^{ef} a^{gh}}}$ are exchanged in the clear among parties, and any member may get hold of them. It is then easy to see that *B* (contributed the initial secret value *b*) could compute *K* via successive exponentiations. We want to alter *K* in such a way that *B* is unable to reconstruct it, while most of the messages already exchanged are still useful (setting $K' = a^{a^{cd} a^{a^{ef} a^{gh}}}$ could be a solution - it excludes however member *A* as well). The extreme approach to exclude member *B* from the final group key is to start over Hypercube from scratch. A more improved version is to freeze Hypercube in round ***t*** during which the failure was observed, and restart it only among the "polluted parties", namely those that have so far interacted with *B* in any of the previous rounds. That way, a 2^{t}-cube within the 2^{d}-cube is restored first, and then the normal execution resumes at round ***t***. This reactive version, denoted as ***Basic***, still results in significant latency, particularly in cases where the node failure occurs near the final rounds.

4.2.2. R-Proactive Hypercube protocol

Hypercube has been mainly used within the Octopus framework in our work. Membership changes, failures and disruptions were always handled within the Octopus subgroups, and the changes were just securely propagated through Hypercube. It was assumed that under failures Hypercube would start from scratch, or a version of Basic could be applied. **R-Proactive** is a hybrid adaptively proactive approach to anticipating and handling member failures at any time during group key establishment. From the **security** perspective, our main concern is to maintain the property of Group Key Secrecy at the presence of disruptions. The properties of Forward/Backward Secrecy bound to the membership changes are ensured by Octopus that handles membership changes efficiently. Hence, we want to ensure that after a member is excluded from the protocol for any reason at any time during group key establishment, it is unable to compute the group key. In terms of **efficiency**, we will explore ways to maintain the security of our scheme, without starting over the protocol from the 1st round whenever failures occur. Limiting the total number of rounds required for successful termination at the presence of failures and reducing the associated costs are our main objectives.

4.2.2.1. Overview

R-Proactive consists of two stages: proactive (1st stage), and normal (2nd stage). The protocol still requires d rounds to terminate if no failures occur. The 1st stage extends until round R<d. The parameter R represents the level of "proactive-ness". It is dynamically adaptive and depends on the metrics we want to improve, and on the rate of failures observed: the higher the rate, the higher the value of *R* is set.

1st stage: Participants relay additional shares for the DHKEs that result in each member executing more than one DHKE with its peer per round, for the first ***R*** rounds. If one or multiple failures occur at this stage, Hypercube does not stall, and does not start over, but proceeds normally to the termination of this stage at round *R*. The multiple intermediate DH values computed by all members during this stage will be processed after round R. Members use the collected values to proactively compute multiple intermediate secret keys at this stage, each of which excludes one or more peers encountered during the previous rounds. Thus, if any of a member's peers fails, the member will select this key at round R from the available key pool it has computed, that excludes the faulty peers from the subgroup key computations so far. Peers that fail before round R have not effect on R-Proactive other than determining which intermediate keys from the key pool formed at round R will be used and by which members. For peers that fail after round R, the protocol resumes from round R instead of 1, after the designated members select the proper keys from the key pool, and use them for the subsequent rounds.

2nd stage: The algorithm switches to the original Hypercube. After members select the designated keys from the key pool, they use them for the subsequent rounds, and perform a single DHKE with their peers for the remaining rounds.

4.2.2.2. Detailed Description

We now describe in detail the R-Proactive algorithm for a general Hypercube of size d and show that it correctly terminates at the presence of failures. Because of the symmetry of members' participation to the algorithm, it suffices to focus on DHKEs and computations of members that belong to any *i*-cube for each round *i*. For the communications schedule, we assign each member with a unique id from Z^*, i.e. (*A*, *B, C, D, E,* …)= (0, 1, 2, 3, 4, ...). We select this base from the vector space so that member *j* communicates with peer $k = \varphi(j \oplus 2^{i-1})$ at round *i*. So, peer *A* communicates with peer *B* at round 1, with *C* at round 2, with *E* at round 3, with *I* at round 4, etc.

Notation_1: Party *A* that initiates Hypercube with secret value a is denoted as *A(a)*. In this analysis, the modular reduction of a secret generated value *x*, namely φ(x), prior to its blinding, *B*(φ(x)) is implicitly assumed, but not reflected to our equations, for ease of notation. We denote the operation of raising each base value in the vector $X_n^T = (x_1, x_2, .., x_n)$ to each exponent included in the vector $Y_m^T = (y_1, y_2, \ldots, y_m)$ as:

$$X_n \times^\wedge Y_m^T = \begin{bmatrix} x_1^{y_1} & x_1^{y_2} & \ldots & x_1^{y_m} \\ x_2^{y_1} & x_2^{y_2} & \ldots & x_2^{y_m} \\ \ldots & \ldots & \ldots & \ldots \\ x_n^{y_1} & x_n^{y_2} & \ldots & x_n^{y_m} \end{bmatrix}.$$

1. Stage 1 (Proactive)

Round 1: Parties execute one DHKE and store all values either generated or just received from their peer, in order to relay them all to the party they contact at the next round. At the end of this round parties *A* – *B*, and *C* – *D* store the following values:

A(*a*)<->*B*(*b*): $X^T_{AB} = (a^{ab}, a^a, a^b)$, $X^T_{1A} = (a^{ab}, a)$, $X^T_{1B} = (a^{ab}, b)$.

C(*c*)<->*D*(*d*): $X^T_{CD} = (a^{cd}, a^c, a^d)$, $X^T_{1C} = (a^{cd}, c)$, $X^T_{1D} = (a^{cd}, d)$.

Notation_2: We denote the global vector including all initial **blinded keys (BKs)** of all nodes generated at the beginning of round 1 as $BX^T_{INIT} = (a^a, a^b, a^c, a^d, a^e, a^f, a^g, a^h,\ldots)$. Let vector $BX^T_{INIT}(i, j)$ include those elements of BX^T_{INIT} that become available to member *j* at the end of round *i*. For example: $BX^T_{INIT}(0, A) = (a^a)$, $BX^T_{INIT}(1, A) = (a^a, a^b)$, $BX^T_{INIT}(2, A) = (a^a, a^b, a^c, a^d)$, etc. as we will see shortly.

Round 2: A party sends the following values to its peer: *i*) the **BKs** computed at the end of the previous round and *ii*) these computed and/or received at the beginning of round 1.

Message exchanges during round 2.

$C(a^{cd}, c) \text{ -> } A(a^{ab}, a)$: $XB^T_{CD} = (a^{a^{CD}}, a^c, a^d)$.

$A(a^{ab}, a) \text{ -> } C(a^{cd}, c)$: $XB^T_{AB} = (a^{a^{AB}}, a^a, a^b)$.

$D(a^{cd}, d) \text{ -> } B(a^{ab}, b)$: $XB^T_{CD} = (a^{a^{CD}}, a^c, a^d)$, etc.

Computations during round 2.

***A* does:** $X^T_{2A} = XB_{CD} \times\wedge\ X^T_{1A} = (a^{a^{CD}}, a^c, a^d) \times\wedge (a^{ab}, a) = (a^{a^{ab}a^{cd}}, a^{c\times a^{ab}}, a^{d\times a^{ab}}, a^{\alpha\times a^{cd}}, a^{ca}, a^{da})$

***C* does:** $X^T_{2C} = XB_{AB} \times\wedge\ X^T_{1C} = (a^{a^{AB}}, a^a, a^b) \times\wedge (a^{cd}, c) = (a^{a^{ab}a^{cd}}, a^{a\times a^{cd}}, a^{b\times a^{cd}}, a^{c\times a^{ab}}, a^{ca}, a^{bc})$

***B* does:** $X^T_{2B} = XB_{CD} \times\wedge\ X^T_{1B} = (a^{a^{CD}}, a^c, a^d) \times\wedge (a^{ab}, b) = (a^{a^{ab}a^{cd}}, a^{c\times a^{ab}}, a^{d\times a^{ab}}, a^{b\times a^{cd}}, a^{cb}, a^{db})$

Similar is the process for parties *D*, *E*, *F*, *G*, and *H*.

Notation_3: $K = a^{a^{ab}a^{cd}}$, $L = a^{a^{ef}a^{gh}}$, $K(A) = a^{b\times a^{cd}}$, $K(B) = a^{\alpha\times a^{cd}}$, $K(C) = a^{d\times a^{ab}}$,…., $L(E) = a^{f\times a^{gh}}$, $L(F) = a^{e\times a^{gh}}$, $K(BC) = a^{da}$, $K(AC) = a^{db}$,…, $L(AG) = a^{fh}$, etc.

Round 3: A similar process is performed as in round 2, with the exception that only a limited number of the newly generated DH secrets are stored in the resulting vector X^T_3. Each member *i* blinds the values contained in the stored X_{3i}^{T} to generate the vector XB_{3i}^{T} that will be communicated to the designated peer member: $i \oplus 2^{j\text{-}1}$, where *j* represents the current round number. We denote vector SX^{T} as one containing a subset of those keys that are comprised in vector X^{T}. Also, let $\tilde{K}_{i}^{T}$ denote the vector of values associated with the single value *K*, held by member *i*, and let $\tilde{L}_{j}^{T}$ denote the vector of values associated with the single value *L*, held by member *j* (*K*, *L,* as well as *K*(*AB*), *L*(*GH*) etc. computed during round 2).

Depending on the fault-tolerance level we want, we can control and limit the number of BKs a member is going to communicate to its peer during any given round. The maximum number of available BKs that can be sent from member *E* to *A* for example is provided below:

E->*A:* XB_{E}^{T} = $B(X_{2E}^{T} \cup X_{1E}^{T} \cup X_{0E}^{T})$ = ((*BL*, *BL*(*H*), *BL*(*G*), *BL*(*F*), *BL*(*FH*), *BL*(*FG*)) $\cup$ ($a^{a^{ef}}$, $a^{a^{gh}}$, a^{g}, a^{h}) $\cup$ (a^{e}, a^{f})).

If member *A* combines all these BKs with its own secrets from the previous rounds, i.e. $X_{2A}^{T} \cup X_{1A}^{T}$, - and all the rest of members act similarly - , the newly generated BKs provide Hypercube with the capability to go around any subset of simultaneous member failures instantly, without extra processing cost. The trade-off is a vastly growing number of keys, computed or received by each member during every round. However, we will show how members can tolerate any number of failures per round, proactively, using a relatively low number of keys per round:

Indicative Message Exchanges during round 3.

E ->***A****:* SXB_{E}^{T} = *SB* X_{2E}^{T} = (*BL*, *BL*(*H*), *BL*(*G*), *BL*(*F*), *BL*(*FH*), *BL*(*FG*), $a^{a^{ef}}$ = *BL(GH)*, $a^{a^{gh}}$ = *BL*(*EF*)).

A->***E****:* SXB_{A}^{T}= *SB* X_{2A}^{T}= (*BK*, *BK*(*D*), *BK*(*C*), *BK*(*B*), *BK*(*BD*), *BK*(*BC*), $a^{a^{ab}}$ = *BK(CD)*, $a^{a^{cd}}$ = *BK*(*AB*)).

***F*->*B*:** $SXB_F^T = SB\,X_{2F}^T =$ (BL, $BL(H)$, $BL(G)$, $BL(E)$, $BL(EH)$, $BL(EG)$, $a^{a^{ef}} = BL(GH)$, $a^{a^{gh}} = BL(EF)$), etc.

The two peers may communicate to each other the vector of initial BKs available to them so far. For example, *A* may also send to *E* vector $BX_{INIT}^T(2, A) = (a^a, a^b, a^c, a^d)$ and vice versa. If more than one failure occurs after round *R*, then for each additional failure, the rest of members need to obtain this one element from $BX_{INIT}^T(R,-)$ that *R*-Proactive designates. $BX_{INIT}^T(R,-)$ can be constructed **proactively,** by having each member *k* send its peer $BX_{INIT}^T(j-1,k)$ at every round *j*. This option burdens the communication exchanges per member per round *j* with 2^{j-1} additional keys. The benefit is that in the event of member failures after round *R*, all the other members already acquire the desired element from $BX_{INIT}^T(R,-)$, and process it right away. On the other hand, the second option is to distribute the designated element to members **on a need to know basis**, after a failure. As we will see, there are always non-faulty members that acquire the designated element.

Computations during round 3.

***A* executes:** $X_{3A}^T = S(SXB_E \times^\wedge SX_{2A}^T) = S\,[S(BL \times^\wedge \tilde{K}_A^T), S(B\,\tilde{L}_E^T \times^\wedge K), S(B\,\tilde{L}_E^T \times^\wedge \tilde{K}_A^T)]$.

From these vectors, member *A* only needs to include the following resulting keys in X_{3A}^T:

$X_{3A}^T =$ ($BL \times^\wedge K$, $BL \times^\wedge K(B)$, $BL \times^\wedge K(C)$, $BL \times^\wedge K(D)$, $BL(F) \times^\wedge K$, $BL(G) \times^\wedge K$, $BL(H) \times^\wedge K$, $a^{a^{ab}a^{ef}}$, $a^{a^{ab}a^{gh}}$).

Working similarly **we get for member *E*:** $X_{3E}^T =$($BK \times^\wedge L$,$BK \times^\wedge L(F)$, $BK \times^\wedge L(G)$, $BK \times^\wedge L(H)$, $BK(B) \times^\wedge L$, $BK(C) \times^\wedge L$, $BK(D) \times^\wedge L$, $a^{a^{ab}a^{ef}}$, $a^{a^{cd}a^{ef}}$).

The following equality holds for any party *J*: $BL \times^\wedge K(J) = BK(J) \times^\wedge L$ (4.1).

This can be instantly proven since $(a^L)^{K(J)} = (a^{K(J)})^L$.

Considering (4.1), we see that vectors X_{3A}^{T} and X_{3E}^{T} differ only in the last element. The first element is actually the DH key produced at the end of the 3^{d} round if the original, typical Hypercube is applied. If d=3, then all members agree upon this key at the end of the 3^{d} round, which contains equal contributions from all members. Observing the following six elements in both vectors, it can be seen that each one **excludes** the contribution of **only** one other member. The contributions (a, e) of members A, E, are naturally contained in all six elements (since if not, members A and E could always add their own secret contributions (a, e) themselves through one or more exponentiations to an element that contains neither of them). Members A and E interact at round 3 to compute secret keys that either contain the contributions of all parties so far, or exclude the contribution of one only party (other than themselves) per element (secret key).

Notation_4: Let Y = $BL\times^{\wedge}K$ = $a^{a^{a^{ab}a^{cd}}a^{a^{ef}a^{gh}}}$, and then let Y (B) = $BL\times^{\wedge}K(B)$ = $a^{a^{a\times a^{cd}}a^{a^{ef}a^{gh}}}$, etc. We combine the first seven elements of vectors X_{3A}^{T} and X_{3E}^{T} in the vector $\tilde{Y}_{A,E}^{T}$, so that element $\boldsymbol{t} \in \tilde{Y}_{A,E}^{T}$ => $\boldsymbol{t}$ = $Y(j)$, $j \in \{ \emptyset$, B, C, D, F, G, $H\}$. We use the following notation for the computed keys of the 3^{d} round:

$$X_{3A}^{T} = (\ \tilde{Y}_{A,E}^{T},\ a^{a^{ab}a^{ef}},\ a^{a^{ab}a^{gh}}),\quad X_{3E}^{T} = (\tilde{Y}_{A,E}^{T},\ a^{a^{ab}a^{ef}},\ a^{a^{cd}a^{ef}}).$$

If we choose to construct the vector that contains the parties' initial blinded secrets $BX_{INIT}^{T}(R,-)$ proactively, then the vectors X_{3A}^{T}, X_{3E}^{T} contain a part of $BX_{INIT}^{T}(R,-)$.

Hence: $\{ BX_{INIT}^{T}(3,A), BX_{INIT}^{T}(3,E) \} = BX_{INIT}^{T}(2,A) + BX_{INIT}^{T}(2,E)$

Hence, the newly computed vectors for members A, E at the end of the 3^{d} round become:

$$X_{3A}^{T} = (\tilde{Y}_{A,E}^{T}, BX_{INIT}^{T}(3,A), a^{a^{ab}a^{ef}}, a^{a^{ab}a^{gh}}), X_{3E}^{T} = (\tilde{Y}_{A,E}^{T}, BX_{INIT}^{T}(3,E), a^{a^{ab}a^{ef}}, a^{a^{cd}a^{ef}}).$$

By similarly processing all keys for the rest of members, at round j = 3, each party computes 2^{j}+1=9 keys as a result of 2^{j}+1 DHKEs with its peer.

Reactive Option: the last two elements in the vectors are somehow redundant. They are used to handle certain cases of simultaneous failures instantly, at no extra cost. The same can be achieved with vector $\tilde{Y}^{T}_{i,i\otimes 4}$ alone as well, with little extra processing.

Proactive Option: includes in the secret vector at the end of the round, 2^{j} = 8 elements, that correspond to a subset of the blinded initial members' contributions contained in BX^{T}_{INIT} .

Round 4 - Round *R*: The process is exactly similar to this of round 3 and can be omitted.

2. Stage 2 (Normal) - Rounds [R, d].

Members switch to original Hypercube, using the designated values from the subsets (vectors X^{T}_{R}) generated at round *R* as secret values for the DHKE during round *R*+1. Every party is involved in exactly one DHKE per round. If no member failure occurs while in stage 2, then all members agree on the same key at the end of round *d*. Upon failure(s), those members among the rest that have interacted directly or indirectly with the "faulty" member in any of the previous rounds, (i.e. "polluted"), must restart stage 2, by "going back" to round *R* and selecting (and processing) this element of X^{T}_{R} that correspond to the particular failure.

4.2.3. Fault-Tolerance with R-Proactive Hypercube protocol

1. **Member(s) Failure prior to Round R.**

Assume that member *B* fails during round *k*, while engaged in communication with party *T* = $B \oplus 2^{k-1}$. If *T* has received the blinded vector from *B*, it processes it exactly as indicated by *R*-Proactive, else it requests from party W=$B \oplus 2^{k-2}$ (the party that communicated with *B* at the previous round, i.e. if *T* = *F* then *W* = *D*) its own blinded vector. Upon receiving a blinded vector either from member *B* or *W*, member *T* processes it exactly as described in the *R*-Proactive. During the following rounds, all parties that would normally communicate with *B*, are now reconfigured to communicate with *T* (if *T* actually received the blinded vector from *B*, and *B* failed afterwards) or *W* (if *B* failed prior to sending its blinded vector to *T*, or if *T* fails as well) respectively, thereon. Should *T* and *W* also fail during the same round *k* or a subsequent one, the members that were supposed to contact them from this point and on, backtrack in rounds and are configured to communicate thereon with the party that was last in contact (most recent round) with any of these parties. This is the only effect that the failure of any member during stage 1 causes to the normal execution of *R*-Proactive, as we will show.

To illustrate this with an **example**, let *T* = *F*, and *W* = *D* referring to the 3-cube example of eight nodes (scheme 1). Assume that *B* fails prior to contacting *F*. Then, *F* will get SBX_{2D}^{T} instead of SBX_{2B}^{T} from *D*. Member *D* has contacted *B* during the previous round, and generates a BK with elements equal to those that *B* has generated. This BK passes on to member *F* during the current round. Hence, the only difference observed is the following: in the end, member *F* will compute vector X_{3F}^{T}, which differs from what it would compute otherwise (if *B* had not failed) only in that it includes $\tilde{Y}_{D,F}^{T}$ instead of $\tilde{Y}_{B,F}^{T}$. That means that member *F* will have paired with *D*, which is correct, and anticipated by the algorithm anyway. Also, it means that among all elements in vector $\tilde{Y}_{D,F}^{T}$ (held by member *F*) there exists one that excludes member *B*, (i.e. $L\times^{\wedge} K(B)$), but there is not an element excluding *D* in the case that *D* also fails at some later point. Similarly, all members that will be redirected to *D* instead of *B*, during the following rounds, will end up with legitimate (according to *R*-Proactive) final key vectors. Since at the final stage, no vector $\tilde{Y}_{B,F}^{T}$ is computed by any member, if *B* is the only "faulty" member, then member *B* is excluded instantly from the final key, if the rest of the members pick the element $L\times^{\wedge} K(B)$ as their final key. Similarly, the failure of any additional member(s) *i* prior to round *R*, is (are) also reflected to the computations, and in the end of round *R*, no vector of the form $\tilde{Y}_{i,j}^{T}$, for any of the remaining members *j* is created. Hence, all the remaining members contain an element in their vectors that excludes member *B*, an element that excludes member *i* (i.e. $L\times^{\wedge}K(i)$ or $L(i)\times^{\wedge}K$), and in general, an element excluding one failed member at a time. All the elements that correspond to the **non-failed** members are multiplied to construct the **final key**. This way, faulty members are excluded from the final key altogether with a single action, and each of the remaining members can instantly compute the desired product, as soon as round *R* is over.

Then, members compute: $K = (L\times^{\wedge}K(B))\times(L(i)\times^{\wedge}K)$.

2. **One or Multiple Failures during round R.**

Assume that *B* has failed prior to round *R*. The remaining members select key $L\times^{\wedge}K(B)$ from vector $\tilde{Y}^{T}$, as discussed. All members but *B* possess this key now (the failure of *B* has been accommodated so that no vector $\tilde{Y}^{T}_{B,I}$ exists to deprive party *I* from $L\times^{\wedge}K(B)$). Now, if member *G* fails as well, its contribution must also be excluded from the key that will be picked up after round *R*. Again, *G's* failure is accommodated during round *R* so that no vector $\tilde{Y}^{T}_{G,I}$ is computed by another party *I*. In that case, the key computed from all members except for *B* and *G* is: $K = (L\times^{\wedge}K(B))\times(K\times^{\wedge}L(G))$, as previously.

3. **Member Failures after Round R.**

Upon failure of any member(s), Hypercube is stalled at the current round *k*. The "polluted" members corresponding to round *R* pick the appropriate new key(s) from $\tilde{Y}^{T}$ and use it (them) instantly, or process it (them) via MEs and multiplications as we will see right next. Starting from round *R*+1, only the members that initially "polluted" the other half of the members at any given round (pollution propagation), in the sense that they communicated to those members values that included the contribution of the "faulty" member, need to update those parties with the new values (**Hypercube** is executed in **one direction** only), until round *k* is reached, and the original Hypercube is executed once again from round *k*. The same process is repeated upon subsequent member failures. Assume now that *C* fails. If the new key becomes $L\times^{\wedge} K(C)$, then member *G* that holds the vector $\tilde{Y}_{C,G}^{T}$ will not be able to compute the final group key, since the element $L\times^{\wedge} K(C)$ is not included in vector $\tilde{Y}_{C,G}^{T}$. In this case, the final key at round *R* is not computed instantly as in the previous cases, but members need to execute one extra ME to compute it. The final key at round *R* becomes: $K = a^{g(L\times^{\wedge} K(C))}$.

The BK a^g has been either already relayed to all members (proactive option), or it will be communicated to the members that request either from *G* or from members that have obtained this value from the first two rounds (reactive option). Any member may send value $a^{L\times^{\wedge} K(C)}$ to *G*. Upon failure of another member e.g. *D*, the key selected by all the rest is: $K = (a^{g(L\times^{\wedge} K(C))})\times(a^{h(L\times^{\wedge} K(D))})$.

In this section, we sketched how *R*-Proactive goes around all possible failure cases during execution. We have shown that the intermediate keys computed after each failure, always exclude the “faulty” members without the need to start the execution from scratch.

4.2.4. Analytical Evaluation of Basic *vs.* R-Proactive Hypercube

R-Proactive is a hybrid approach for handling failures at any round of execution. Considering that parties may not be directly connected, and may also be subject to abrupt dynamic changes, the impact of extra rounds in this framework is significant: the underlying routing protocol increases the total communication cost, excessive relay nodes may be required, and the overall performance degrades (i.e. QoS deteriorates due to more collisions at the MAC layer, the bandwidth usage becomes higher and the same is the case for the consumption of network resources). Reducing the extra routing cost and the total number of rounds becomes our most important priority, even at the expense of extra computation and storage cost, which must still be kept low. *R*-Proactive is designed with these requirements in mind, and is also very flexible since we can adjust the level of “proactive-ness”, by adaptively selecting the round *R* at which we switch from the *proactive* to the *normal* mode.

This “proactive-ness” translates to the **trade-off** between the number of the **total rounds** in the presence of failures (**latency**), and the extra **communication** or **computation costs** incurred to the network, due to the multiple keys that must be computed per round. *R*-Proactive can tolerate any number of failures, and this is a very useful feature for any framework upon which it is applied. For a group of size O(1000), typical values for *d* do not exceed 13. The value of *d* sets an upper limit to the value *R*.

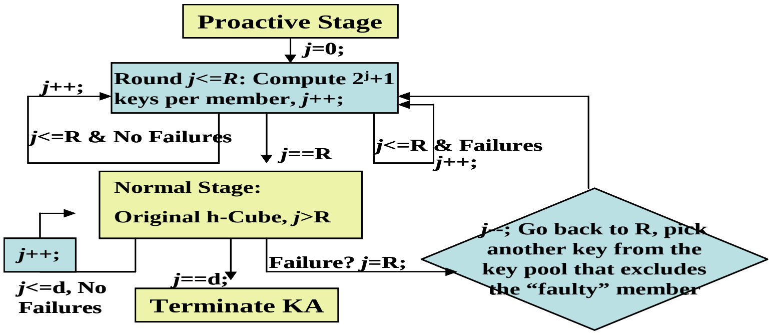


Figure 4.2. R-Proactive Hypercube Algorithm.

1) Analysis and Comparative Evaluation of Basic vs. R-Proactive

We now compute the analytical formulae of the metrics of interest for both schemes. We also conduct a comparative performance evaluation, assuming that up to ***M* failures** occur during one execution ($M<2^d$). We evaluate the schemes for various distributions of these *M* failures over the *d* rounds of the typical Hypercube execution (worst, best, average, and random case scenarios that correspond to particular distributions of *M* over the Hypercube rounds), and for multiple values of parameters *R* and *d*. The worst case scenario occurs if all *M* failures occur during the last round for both algorithms.

Notation_5*:* Let *K* be the number of bits per message, C_E be the processing cost (bits) of a modular exponentiation (ME). By “*RC*”, “*CM*”, and “*SC*”, we abbreviate the combined routing communication cost (RC), the computation cost (CM), and storage cost (SC).

A. Basic.

A failure before round *j* pollutes a 2^{j-1}-cube. It takes (*j*-1) rounds to update it. In each round, half of the parties are only senders and half only receive the updated values. All members perform 2 MEs per round (compute the current secret, and blind it for the subsequent round).

RC (no failures): $2^d \times d \times K$.

Round *j* failure: **RC**(*j*): $\sum_{i=1}^{J} 2^{J-1} \times K = 2^J \times K$, and **CM**(*j*): $2^{J+1} \times C_E$.

RC (with failures): $\sum_{i=1}^{d} (2^d - F(i)) \times K + \sum_{Failure=1}^{M(any\_round\_J)} 2^J \times K$, **SC** (per member): $d \times K$.

B. R-Proactive.

Stage 1: During this stage, each member blinds 2^{t-1} +2 shares from round *t*-1, computes 2^t+2 new DH keys, and transmits 2^t BKs, under the proactive option at round *t*. The total costs until round *R*, in the presence of failures become:

CM (*RProact1*):= $\sum_{i=1}^{R} (2^d - F(i)) \times (3 \times 2^{i-1} - 2) C_E$,

RC (*RProact1*): = $\sum_{i=1}^{R} (2^d - F(i)) \times (2^i - 1) K$, **SC** (max. per member, RProact1): = $2^{R+1} \times K$.

Here the parameter ***F***(*i*) represents the number of members that have failed up to round *i*. Failure of a member during this stage affects insignificantly the previous costs.

Stage 2: Upon failure at round *j*>*R*, the "polluted" members run *Basic* from *R*, selecting another key from their vector pool.

RC (*RProact2*): =: $\sum_{i=R+1}^{J} 2^{J-1} \times K = (2^J - 2^R) \times K$, **CM** (*RProact2*):= $((2^{J+1} - 2^{R+1}) + 2^R) \times C_E$.

R-Proactive succeeds in greatly reducing the total number of rounds of the Hypercube in the presence of failures, as well as the associated RC in the presence of failures. The **novelty** in our algorithm is that Hypercube never has to stall for a failure that occurs during the 1st stage, and as far as the 2nd stage, the number of rounds to remedy the failure is much less, since members only need to go back to round *R*. However, RC and CM are higher for *R*-Proactive, if the total number of transmitted bits is considered (the higher the proactive-ness level *R*, the higher the number of elements that must be exchanged to construct the framework of the 1st stage). However, if we implement Hypercube (*Basic*, *R*-Proactive) with ECDH, we achieve a substantial reduction in RC. Nevertheless, the amount of rounds *R*-Proactive takes to execute in the presence of failures is dramatically reduced compared to the original *Basic*.

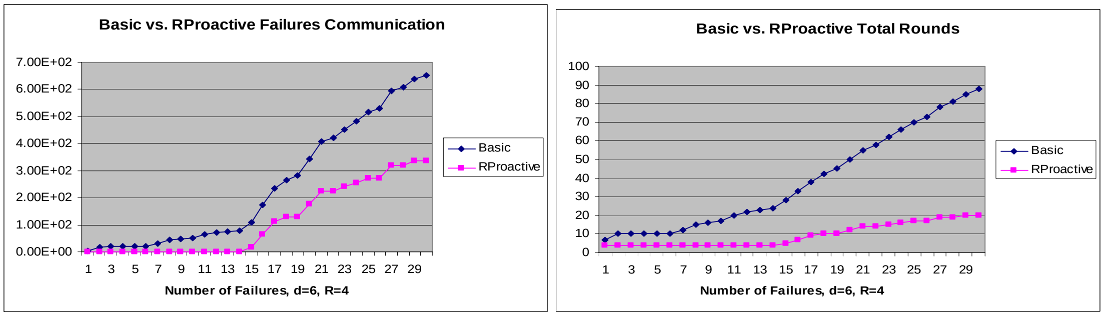


Figure 4.3. Total RC and Total # Rounds for 30 Failures: Basic vs. R-Proactive, d=6, R=4.

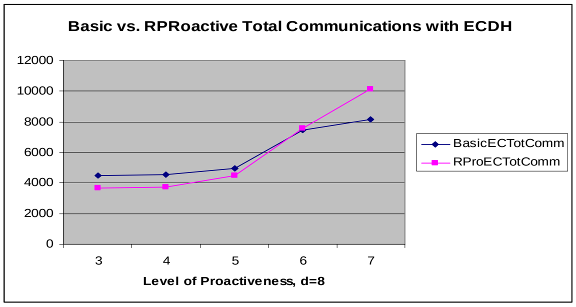


Figure 4.4. Total RC Basic vs. R-Proactive with ECDH, varying the Proactive-ness Level R.

2) Elliptic Curves Cryptography (ECC) Implementation

ECC is known to offer a security level comparable to that of other cryptographic systems of larger key sizes. If we substitute DHKEs with EC-DHKEs we achieve an obvious reduction in communication and storage costs (i.e. DH K = 1024 *vs.* ECDH m = 169 bit key sizes). We illustrate the benefit of ECs for *R*-Proactive with an arithmetic example. Consider an IP packet of 1500 bits, with a (head-tail) label of 128 bits, and available payload of 1372 bits. If we apply *R*-Proactive, setting R = 4, a member must send a maximum of 18 elements to its peer during round *R*. If DH is used, then only 1 element fits in the IP packet, and hence 18 packets will be sent from a node to its peer. If ECDH is used, then $\lfloor {}^{1372}\!/_{169} \rfloor$= 8 elements fit in one IP packet, so partners need to exchange only 3 packets. Hence, *R*-Proactive becomes more powerful with the use of ECs [55, 56].

4.2.5. Basic *vs.* R-Proactive Hypercube Simulation Results

1) Simulation Set-Up and Discussion:

We have conducted simulations to compare the combined routing/communication (RC) costs incurred to the network from the execution of *R*-Proactive *vs*. Basic Hypercube to gain a realistic view of the actual performance of both schemes over an ad-hoc multi-hop network. We use different graphs to generate the secure group and analyze the performance of the two algorithms. A number of nodes from this graph are randomly selected as subgroup leaders. We assume a generic Dijkstra routing protocol that finds the shortest paths between leaders. For our evaluation, we generated various random graphs of different sizes $S \in [100, 500]$ for each initial input of member number $n = 2^d$, and for the same graph we have varied parameter *d*: $2<d\leq 8$. For the same graph and input, we have varied the group configuration, i.e., we have selected the *n* members at random in every repetition. For *R*-Proactive we also varied parameter *R*, where $1<R \leq d$, for each graph and configuration, and for each *d*. Failures have been simulated with the use of a probability failure parameter *p*, where $0.01 \leq p \leq 0.04$ for each member that is alive during a round. The probability of member failures is uniformly distributed in terms of rounds. For a given input <*S*, *d*, *R*, *p*>, we ran both algorithms with and without failures, for 100 different group configurations and we averaged the results.

2) Simulation Results:

We illustrate in the following graphs some indicative results produced by *Basic* and *R*-Proactive. We can see that our simulations actually confirm our analysis for all cases.

Fig. 4.3.*a* shows a comparison in the RC overhead of the two schemes, for *d*=8, *R*=4, in the presence of failures, after the first group key establishment. So, the backbone framework for the schemes is not considered and we focus only on the steady state. Clearly, as the number of failures increases, R-Proactive presents an increasingly superior performance.

Fig. 4.3.*b* shows a comparison in the latency (total rounds) of the schemes for *d*=6, *R*=4, by varying the number of failures before the protocols successfully terminate and compute the current group key. R-Proactive presents an increasingly superior performance, and appears "impervious" to failures. This behavior compensates the relatively expensive framework required for R-Proactive (computed once).

Fig. 4.4 shows a comparison of the schemes in the total RC cost, when R-Proactive is implemented with ECs, for *d*=8, in the presence of failures. Both algorithms present comparable overall RC, so the use of ECs mitigates the expensive framework in R-Proactive.

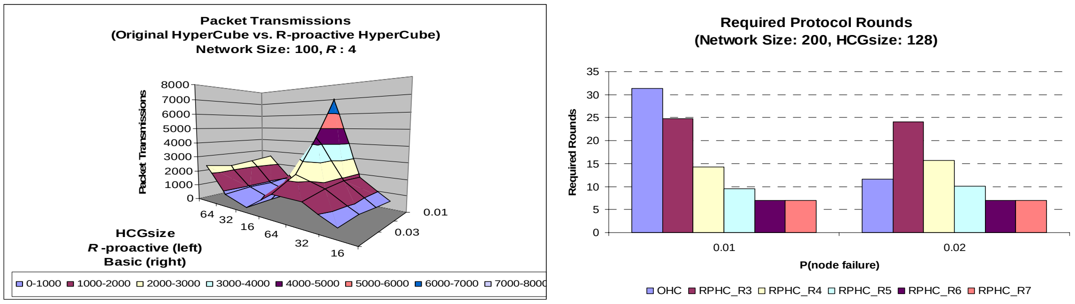


Figure 4.5. (a):Pckts Transmission OH for Basic vs. R-Proactive (R=4), p in [0.01, …, 0.04], (b): Rnds for Basic vs. R-Proactive (R in [3..7]) w/ failures, d=7, S=200, p=0.01, 0.02.

Fig. 4.5.*a* shows a comparison of Basic *vs.* R-Proactive for *R* = 4, in terms of the total number of packets sent under increasing failure rates [0.01,…,0.04] and group size [16,…,64]. When the failure rate increases, i.e. *p*>0.03, the total packet transmissions in *Basic* decreases. This happens because whenever the number of failed members is halved, we re-adjust the schedule of members in *Basic* and decrease the current number of rounds required for termination of the original scheme without failures by one. Still, *R*-Proactive is superior for almost all scenarios. However, in a real case scenario, it is not so simple to do the re-adjustment discussed in *Basic* for the members. They must be allocated new *ids* and coordinate to apply a new schedule, which is impractical and expensive.

Fig. 4.5.*b* shows a comparison of the schemes (for *R* in [3..7]), for a network of size *S*=200, and *d*=7, in terms of the total number of rounds under the presence of failures, for two failure rates *p* = 0.01, 0.02. *R*-Proactive achieves a significant reduction in the total number of rounds, particularly as *R* increases. For *p*=0.02, the number of rounds for *Basic* is quite low. This is also due to the round re-adjustment done. Even so, for *R*>4, *R-Proactive* achieves an even bigger reduction in the total round number.

4.2.6. Conclusions

This section focuses on the design of a Fault-Tolerant Hypercube protocol when it used as the core scheme to connect subgroup leaders for Octopus-based schemes in a MANET. We describe a hybrid proactive approach for extending Hypercube to tolerate failures suitable for the dynamic environment and for the application of interest. We presented comparisons of two different schemes: the original Hypercube with a slight variation: *Basic*, and our fault-tolerant adaptively proactive extension: *R*-Proactive. By exploring the features and evaluating the performance of the two algorithms, we show how we can achieve better performance with our scheme in the presence of failures. We also illustrate how the incorporation of ECC favors our design and improves its overall performance.

4.3. Algorithms for Handling Disruptions in KA protocols – Robustness

4.3.1 Introduction

Our method includes the formation of a spanning tree over the group members, at their initial state. We abbreviate the extended versions of the protocols as "*wt*" (*with topology*). Without loss of generality, our analysis in this section is conducted on such *wt*-versions, but it is valid for any other protocol version with structured communication schedules.

Since topology changes can partition the network, we extend the notion of a spanning tree (ST) to a **spanning forest** (SF), i.e. a (disjoint) collection of STs. A SF is a directed sub-graph in which all nodes are contained and there are no cycles, each node has at most one parent, and every two connected nodes belong to the same tree in the forest. The original ST may be separated into fragments (each of which is also a ST) and reconnect, depending on the topology changes, responding to multiple link/node failures and recoveries that can occur at arbitrary times. The border nodes of each fragment continually monitor their neighborhood for topological changes and notify the root. The maintenance of the framework is handled similarly to the proposals of Gallager, Humblet, Spira (GHS) [50], and Cheng [51], where analytical results on the performance of their protocols are provided as well. However, since we do not necessarily need to maintain minimum STs, we propose a number of variations to these algorithms that significantly reduce the resulting overhead. The impact of the partition and merging of the original ST on the actual group KA algorithms is currently studied.

We are now going to analyze each of the following group KA protocols: ING, GDH.1, GDH.2, BD, DS-TGDH, Hypercube. We will be using Fig. 4.6 to deploy all possible scenarios of disruption, study and improve the behavior and performance of the these protocols. We assume that the tree on the left is the ST or MST, formed over the network graph, to accommodate the *wt*-versions of the discussed group KA protocols. The new communications schedule of the *wt*-versions are designated by a full walk of the MST, and in the following figure, member *D* becomes the predecessor of member *C*. We then assume that at time T_0, the ST is partitioned in at least two fragments, say A1 and A2, after the ST got disconnected at one point at least, say at link (*C*, *D*). This could have happened because either members *C* and/or *D* became faulty, and/or the link (*C*, *D*) became faulty overall. Whenever this event occurs, we distinguish two cases: (*a*) this disruption occurs **after** member *D* has communicated the proper KM material as designated by the KG algorithm to its successor (member *C*), and (*b*) this disruption occurs **before** member *D* communicates the proper KM material to its successor (member *C*). The resulting fragments form a SF. The members of these fragments monitor the network at regular intervals and attempt to re-connect if possible with one or multiple other fragments, using techniques similar to these presented in GHS [50], and Cheng [51]. Then, assume that at time T_1, fragments A1 and A2 re-connect (merge) through the available link (*G*, *H*). We are now going to examine the impact of all these dynamic events to the logical design of the key generation algorithm of each protocol, as well as to the connectivity among parties. We set the time window *W* as the critical waiting period, during which the protocol stalls, waiting in case any of its fragments re-connect.

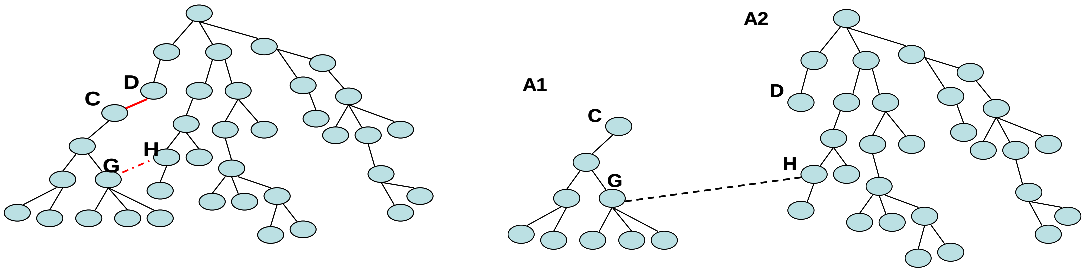


Figure 4.6. MST split in two (or multiple) fragments, i.e. A1 and A2 (link (C, D) becomes disconnected). A1, A2 attempt to re-connect through link (G, H).

Handling Breakages: Our method is differentiated from this proposed by Cheng [51] for handling breakages, in that we attempt to repair the breakage as close to the point of failure as possible. We achieve that by having the nodes sequentially probe the network, giving priority to these nodes close to the point of failure, through the use of a unique “repair token” circulated sequentially from one member to another until the two fragments get re-connected. In our example, assume that members *C* and *D* get disconnected. Now, member *C* belongs to fragment A1 and member *D* belongs to fragment A2. Member *C*, from its own perspective, after attempting to re-connect to member *D*, or to any relay that belongs to fragment A2, for a specified period of time (or maximum number of attempts), generates and passes a “repair token” to the first neighboring member close to the failure, assume member *B*. Member *B* gets the “repair” token and attempts to connect to member *D* or any other member that belongs to fragment A2 (or to any member outside fragment A1, in case of multiple fragments). Upon success, member *B* will become a gateway node connecting the two fragments, and the token will be destroyed. Upon failure, member *B* will pass the token to this neighbor that is closest to the point of failure and has not received the token so far. Assume that this member is its predecessor in the upward stage of GDH.1, say member *A*. The same process will be repeated, during a pre-defined waiting period t_{WAIT}. After the expiration of this window of time, we switch to the original method proposed by Cheng, according to which all members of one fragment attempt to connect to any member of the other fragment by polling the network at regular intervals.

Notation: Let: *n* be the number of group members, $d = \log_2 n$ be the number of Hypercube rounds, *m* be the number of merging fragments with equal number of members each: $s = (n/m)$, in a generalized example.

Below, we present the analysis of only a few of the discussed protocols, in order to avoid redundancies. We omit the analysis of GDH.1, GDH.2, and Hypercube.

4. 3. 1. *wt*-ING

Case 1: Fragments A1 and A2 remain disconnected for time > W

The *n* ING participants form a virtual Hamiltonian ring, and each link in this ring is traversed *n* times. Therefore, whatever the nature and timing of the disruption associated with the link (*D*, *C*) is, it is impossible for all participants to compute the original – intended – group key, even if member *C* received the designated contribution from member *D* before the disruption. The only exception is if the disruption occurred after the n^{th} (final) link traversal, which also coincides with the successful termination of the protocol. We recall that an atomic step in ING reduces to the communication of a distinct value over each link. If a disruption occurs over any link before the final step, then these members that still require elements that have not yet traversed the failed link (depending on their placement in the virtual ring), cannot compute the group key. However, all members may store the received values, and if the fragments merge later, and members *C* and/or *D* become “functional” again, then ING may resume from the point it got stalled.

After the waiting period has ended and no merges have occurred, fragments A1 and A2 execute ING separately from scratch. The nature of ING is such that the fragments cannot exploit the previously gathered contributions, except for their own generated initial secret values. This is because the contributions collected by any group member, at any point, contain at least one element that is not contained in at least one other member, and the final step of the algorithm is when all members gather all contributions, and are capable of forming the same group key. Considering this, A1 and A2 can only run ING separately but may use their initial secret contributions. Should A1 and A2 reconnect later through gateway link (*G*, *H*), there are three options for all group members to obtain a global group key:

Case1_1: They can resume the ING of the original MST from the point it was interrupted if all participants are currently non-faulty and have stored the corresponding values. The information that would be routed from the currently "broken link" (*D*, *C*) is now routed through gateway (*G*, *H*). The cost for establishing the original ING group key remains the computationally the same, but the communication cost is now higher, as members' contributions are not optimally routed anymore.

Case1_2: The remaining participants can start over the execution of a new ING, with new communication schedules that take into account the adjusted associations of parents-offspring, and gateway links. In particular, the smaller in size fragment(s) change their parent-offspring associations, setting the gateway member to a larger fragment as their root, i.e. member *G* becomes the root of fragment A1.

Case1_3: The most inexpensive and practical option is the execution of 2-party DH between the gateway members (i.e. *G*, *H*), using as secret keys those established after the execution of ING separately on each fragment. Then, the gateways of each fragment propagate the received BKs to their members, and all members obtain the same global group key. This solution is practical as the number of fragments that merge may be more than two and the merging can occur at different instances.

Case 2: Fragments A1 and A2 re-connect through gateway link (*G*, *H*) before time < W

We distinguish the corresponding three options as stated in Case 1.

Impact on Re-Keying from the merging Fragments.

ING does not lend itself to simple re-keying algorithms. Hence, GDH.2 can be used among the various gateways to establish a common group key and propagate the corresponding BKs to the members of each fragment. If re-keying is required after the merging, members run the "reduced ING" on the fragment that witnesses membership changes, and compute a new subgroup key for this fragment. Its gateway member becomes the initiator of the GDH.2 re-keying operation and sends the proper GDH.2 values to the gateways of the remaining fragments. Each gateway that collects the corresponding GDH.2 broadcast, computes the global group key, and sends to its members: (*a*) either one DH value only, sufficient for the members to compute the group key following the typical GDH algorithm (for efficiency and low overhead), or (*b*) the whole downward message (for robustness): if any other member becomes gateway of the corresponding fragment in the future, it should be able to initiate re-keying, by having stored all the required GDH.2 elements.

Remark: To preserve the property of forward secrecy, if *C* or *D* or any other members are found faulty at any round, a "reduced" version of ING 2 is run to exclude the faulty members from the subgroups of the corresponding fragments. The remaining fragments maintain their own subgroup key unless they include faulty members. Then, GDH.2 re-keying is executed among gateways. Each fragment is responsible for propagating the BKs obtained to its members, so that they can also compute the new group key that excludes the faulty members. We stress here that we have selected GDH.2 to perform re-keying, because it is the only protocol that lends itself to simple re-keying algorithms, initiated from **any single party**.

4. 3. 2. *wt* - BD

Case 1: Breakage: A1, A2 are re-connected through gateways *G*, *H* while time <*W*

A1 may re-adjust its parent-offspring associations before the 2nd round -if feasible, so that the whole tree becomes a broadcast tree again, with gateway *G* as the new root (controlled broadcast gets more efficient). However, A1 does not need to alter its parent-offspring associations. All traffic from A1 to A2 and vice versa can be routed through gateways *G*, *H*.

Case 2: Fragments A1, A2 get disconnected before round 1 for time > *W*

If the general root of the ST is now contained in A1, multiple fragments may be formed. Each fragment executes BD separately. If members *C* and/or *D* become faulty or evicted, then the remaining members involved in each fragment simply **will not include** the key parts originating from *C* or *D* in the computation of their combined group key (Chapter 2).

Case 3: Fragments A1, A2 get disconnected before round 2 for time > *W*

A1, A2 compute a separate BD key without restarting round 1. It suffices that members that belong to one fragment exclude from their computations the contributions (exponents) from members that belong to the other fragment(s), and initiate round 2 based on the exponents broadcast from members of their own fragment. Not all members need to be aware of the broken link. The root notifies only the **affected** ones: at this point each member *i* uses the parts sent from members (*i*-1), (*i*+1) to compute its own contribution (X_i) for the 2nd round.

Assume that the broken link (*b*1, *b*2) is such that member *i* is now included in fragment A1 (the root of which is member *b*2), and member (*i*+1) is included in fragment A2. Then the following steps are executed: the root of each fragment notifies its members of the boundaries of the subset of members from which they should expect to receive KM data. Of course, this step is optional, since the members of each fragment cannot receive data originating from members of other fragments anyway.

Fragment A1: member *b*2 is the starting member (root), and member *i* is the last member visited. Hence, member *b*2 now computes: X_{b2}= ($z_{(b2+1)}$/z_i) (instead of: ($z_{(b2+1)}$/ $z_{(b2-1)}$), and member *i* now computes: X_i = (z_{b2}/z_{i-1}).

Fragment A2: the root notifies member $b1$ of its new successor (i.e. member (i+1), if members are assigned *id*s via a pre-order tree traversal). Hence, member $b1$ computes X_{b1}= ($z_{(i+1)}/z_{(b1\text{-}1)}$), and member ($i$+1) computes: X_{i+1} = ($z_{(i+2)}/z_{(b1\text{-}1)}$).

The course of action is similar for different kinds of configurations after the breakage (i.e. when the initial root belongs to A1, and multiple other fragments are formed.

Case 4: Fragments A1, A2 get disconnected after round 2 for time > *W*

Case 4*a*: Members from all fragments (i.e. A1, A2) compute the same group key.

It is possible, if there is no strict requirement for forward secrecy, since all members have obtained the required KM material by then.

Case 4*b*: Members from different fragments (i.e. A1, A2) compute separate group keys.

If the requirement for forward secrecy is strict, A1 and A2 must compute a separate BD key. The members in each fragment do not need to re-compute everything from scratch. On the contrary, they use the KM material gathered during the previous 2 rounds. The course of action is similar to case 3, with the only difference that one member from each different fragment needs to change its contribution towards the 1st round, so that the group key of one fragment remains secret to the members of the other fragment. Then, each fragment runs a "reduced BD" version (during which only a small subset of the normal computations and communication exchanges need to be executed), and computes a separate group key.

BD Re-Keying:

BD can lend itself to efficient re-keying algorithms (Chapter 2), so if members *C* or *D* are found faulty at any time during the protocol execution the following steps are executed:

If *C* or *D* becomes faulty before round 2 then the rest of members simply **will not** include the KM material originating from *C* or *D* in the computation of the group key. The "*X* values" will be computed as in case 3, taking into account the removal of *C* or *D* as well.

If members *C* or *D* becomes faulty after round 2 then an arbitrary member (i.e. the root) will change its secret initial value and broadcast its new exponent to the related members. Then, a "reduced BD" will be executed by all members, to re-compute those among the "*X* values" that are modified, and then the new group key that excludes the faulty members.

<u>Impact on Re-Keying (Case 4) from merging Fragments.</u>

If the disconnected fragments merge to the original group and all merged members wish to establish a common group key, we summarize here all the distinguished cases:

Case 4*a*: All members already share a common group key.

Case 4*b*: We distinguish between three different options:

Case 4*b*_1: Members run "reduced BD": most of the already exchanged KM material will be preserved. Members that belong to different fragments exchange the exponents they already computed during the 1st round. At this point, *n* broadcast messages will be exchanged. Also, one member from each fragment computes a new exponent (to preserve forward secrecy). Then, the corresponding members re-compute the modified "*X* values", and broadcast them to all members so that they can compute a global group key (with fewer computations). Then, at this point, $4\times m$ broadcast messages are sent out, as shown in Chapter 2. Similarly, the computations performed per fragment are: *s* [MEs] + (*s*+2) [divisions].

Therefore, the **merging overhead** for the fragments becomes:

p_{4b_1} (*Group reduced BD*, *n*, *m*, *s*, *Comm*) = (n+ $4\times m$) broadcast messages generated,

p_{4b_1} (*Group reduced BD*, *n*, *m*, *s*,*Comp*)=$m\times$(*s*[MEs]+(*s*+2) [div/s])=*n*[MEs]+(*n* +2*m*)[div/s].

Now, the merged group has obtained a group key that follows the specifications of the original BD. Hence, a re-keying under 4*b*_1 incurs the overhead of the extended BD re-keying introduced in Chapter 2. The **re-keying** overhead under Case 4*b*_1 becomes:

p_{4b_1} (*Re-Keying (eviction) BD*, *n*, *m*, *s*, *Comm*) = 4 broadcast messages,

p_{4b_1} (*Re-Keying (eviction) BD*, *n*, *m*, *s*, *Comp*) = *n* [MEs] + (*n* +2)[div/s].

Case 4*b*_2: An expensive option is when merging members execute BD from scratch, which is highly undesirable. The operation of re-keying is executed exactly as discussed in Case 4*a*.

Case 4*b*_3: The merging fragments use their separate subgroup keys to run overlay GDH.2 and compute a global group key. As in the case of GDH.1 and ING protocols, it makes no sense to run an overlay BD, since the resulting expression for the final key cannot have been generated by a flat execution of BD. Again, similarly to the corresponding cases of the previously analyzed schemes, the gateway members exchange the proper BKs and propagate the received BKs to all members of their ex-fragment. A Re-keying under 4*b*_3 includes the following steps: the ex-fragment that witnesses the membership change will modify its subgroup key (re-keying), via the "reduced BD". Then, the gateway member of the given fragment will initiate the GDH.2 overlay re-keying: the shares of the broadcast message of the downward stage will be recalculated (w.r.t. the new subgroup key of the given fragment), and broadcast to the rest of group members. The **overhead** for **merging the fragments for the first time** (including the computation of subgroup keys) becomes**:**

p_{4b_3} (*subgroup regular BD, group overlay GDH.2, n, m s, Comm*) = $m\times 2s$ [total broadcasts for all fragments] + $\frac{1}{2}(m^2+3m\text{-}4)$ [broadcast for overlay execution of GDH.2] + $m\times n$ [or else a broadcast message of *m* elements distributed to all *n* members],

p_{4b_3} (*subgroup regular BD, group overlay GDH.2,n,m,s,Comm*) = $2n+\frac{1}{2}(m^2+5m\text{-}4)$ [bcast],

p_{4b_3} (*subgroup regular BD, group overlay GDH.2, n, m s, Comp*) = $m\times(s+1)\times s$ [total MEs for members in all fragments] + $\frac{1}{2}(m^2+3m\text{-}2)$ [computations for overlay execution of GDH.2] + *n* [computations by all members to compute the final group key],

p_{4b_3} (*subgroup reg. BD, group overlay GDH.2, n, m s, Comp*) = $n\times(2s+1)+\frac{1}{2}(m^2+3m\text{-}2)$.

The **re-keying** overhead within a given **fragment** becomes (reduced BD version - Chapter 2):

p_{4b_3} (*subgroup reduced BD, s, Comm*) = *4*.

p_{4b_3} (*subgroup reduced BD, s, Comp*) = *s* [MEs] + (*s*+2) [divisions].

The modified subgroup key triggers **overlay GDH.2 re-keying**, the overhead of which is:

p_{4b_3} (re-keying GDH.2, *n*, *m*, *Comm*) = $m \times n$ (or else a broadcast message of *m* elements distributed to all *n* members following a simple controlled flooding strategy), and

p_{4b_3} (re-keying GDH.2, *n*, *m*, *Comp*) = (*m*+*n*-1) [MEs].

In total, the **merging OH** at **steady state** for the **group** becomes:

p_{4b_3} (*Group merge BD-GDH.2, n, m, s Comm*) = (*m* + 4) broadcast messages generated,

p_{4b_3} (*Group merge BD-GDH.2, n, m, s, Comp*) = (*m*+ *n* + *s*-1) [MEs] + ($s \times (s$-1)) [div/s].

Summary: None of the two methods (4*b*_1, 4*b*_3) is clearly superior to the other in the sense that it improves on both metrics of interest. For example, merging the fragments under 4*b*_3 results in lower communication overhead, lower number of modular divisions, slightly higher number of MEs, lower overhead overall, compared to 4*b*_1. In this case however, the trade-off comes when the re-keying operation is required. Using 4*b*_1 over 4*b*_3 results in lower communication overhead, lower number of MEs, and higher number of modular divisions.

4. 3. 3. DS-TGDH or TGDH

Notation: Let $d = \log_2 n$ be the number of rounds for TGDH to successfully terminate.

Case 1: Breakage: A1, A2 are re-connected through gateways *G, H* while time <*W*

Without loss of generality, assume that the sponsor belongs to A2. All the traffic between A1 and A2 is routed through gateways *G* and *H*. Both the original TGDH and the improved DS-TGDH schemes are executed normally. The KG algorithm in both protocols is based on the design of two different logical trees from all the participating members. DS-TGDH reflects all topological changes and associations among members in the weight of the corresponding links. As discussed in Chapter 3, based on these weights, there are one (or more) logical trees that optimize certain metrics of interest. If the fragments of the group are connected through gateway members that alter the associations among members and their link weights, a new tree will be generated whenever the next “optimized key generation” is triggered.

Case 2: Fragments A1, A2 are disconnected before or during Round 1 for time > W

If the sponsor is contained in A1, multiple fragments are formed. In TGDH, each member must contact the sponsor during each of the *d* rounds to compute a group key. Therefore, each fragment executes (DS) TGDH separately and elects its own sponsor. If these fragments get partitioned as well, the same approach is recursively applied on each new network partition.

Case 3: Fragments A1, A2 are disconnected during Round 1< *j* < *d* for time > W

If the sponsor is contained in A1, multiple fragments may be formed. In TGDH, each member must contact the sponsor in each of the *d* rounds in order to be able to compute a group key. Therefore, each fragment executes (DS) TGDH separately and elects its own sponsor. If these fragments get partitioned as well, the same approach is recursively applied on each new partition. Hence, A1, A2 compute separately a (DS) TGDH subgroup key. Below, we give an overview on how a partition during any round is handled by each protocol so that: (*a*) each fragment resumes the execution from the current round during which the partition occurred and does not start from scratch, and (*b*) the KM material already exchanged is not found redundant within the fragments, but used towards the key establishment.

TGDH: If the given fragment does not have a sponsor, any member in it may become the sponsor for the remaining rounds. The existing or the new sponsor collects the *id*s of the members contained in its fragment and continues the process from round *j*, while members use the BKs collected up to round *j*. In TGDH, the communication exchanges between any two members in any round are all regulated by the sponsor. The sponsor replaces the contribution of the missing members by “dummy” values (BKs) or “zero flags” and the recipients of such “dummy” BKs or “zero flags” use their own previously computed BK unchanged for the subsequent round. Depending on the number of remaining members in a fragment, the overhead required to start over the execution may be lower than this incurred if we continue the execution from the current round, replacing the contribution of missing members with “dummy values. We use the following **rule** to determine which of the two overheads is lower: *if the number of remaining members in a fragment is x, and* $y = \log_2 x \leq (d\text{-}j)$*, then we choose to re-start TGDH for that fragment.* Also, the sponsor can optimize the original execution, by reducing the broadcast message of BKs in each of the remaining rounds to include the contributions of the existing members only in the fragment.

DS-TGDH: The remaining members in each fragment attempt to exchange their BKs with their originally scheduled peers for round *j*. If their partner is missing (belongs to another fragment or is faulty), they will use the BK computed before round *j* unchanged for the subsequent round (*j*+1), during which they are scheduled to communicate with another member. Also, these members replace their “missing” partner wherever they were supposed to appear in the virtual DS-TGDH for the subsequent exchanges in the remaining rounds. The same is the situation even if a pair of members is missing in a given round. Similarly, the nodes in the path upwards the root that are mapped to any of the missing peers, either get replaced by the last non-faulty member that communicated with any of the two peers, or are deleted, if those peers were leaves in the original DS-TGDH, respectively. The total number of rounds required for a subgroup key to be established in a given fragment may become lower, i.e. if there is only 1 member left from the list of the active members in round $k< d$. It is easy to see, that the upward stage of this algorithm always terminates, since the virtual tree always designates a tree root that computes the subgroup key of the fragment, even if this root is not the original one. The key point is that as long as the fragment remains connected, the substitutions suggested are always feasible, and hence a new DS-TGDH can be built, which is in fact a subset of the original tree, with different link weights that accommodate the fact that some members are missing. The downward stage of DS-TGDH on a fragment is completely determined by the upward stage, and is executed as defined by the upward. Having sketched the upward stage, the remaining members of the fragment know exactly the configuration of the new DS-TGDH tree. This tree may not be the optimal anymore for the current key establishment on the given fragment. However, it provides the best solution if we want to continue the execution without starting over.

Merging Fragments and Re-keying:

If the disconnected fragments merge to the original group and all merged members wish to establish a common group key, we summarize all the cases distinguished during our analysis.

Case 4_1_1 (TGDH): The fragments execute virtual TGDH. In this case, if only the sponsors of the fragments participate to a new TGDH using the already computed subgroup keys of their fragment as their initial shares, the resulting group key is equivalent to the group key that would be formed if the TGDH scheme was executed once over all the group members. If we analyze the virtual tree structure of TGDH, we see that any sub-tree of the original tree executes TGDH recursively and computes an intermediate subgroup key individually and independently. If we substitute any such sub-tree with a single entity that participates to TGDH with the intermediate subgroup key that corresponds to the sub-tree it substitutes, the resulting group key will be the same as this of the original execution. So, any TGDH group key that is constructed from fragments as we described, can be mapped to a TGDH group key constructed directly from single members. It suffices that both TGDH trees are logically reduced to the same configuration of their 1st level (tree leaves): the same member *id*s are associated with the same tree leaves, and the resulting pairs of members that are formed for the communications exchanges of the 1st level are equivalent in both trees. For the correct mapping, it suffices that attention is given to the number of members within each fragment, and to the number of fragments overall. During the deployment of the members contained in the fragments and after appending them to the members of the rest of fragments to generate the configuration for the 1st level, certain leaves must be associated with dummy or empty nodes so that the correct intermediate subgroup keys that correspond to fragments are formed.

Example: Let: *A*, *B*, *C*, *D*, and *E* be the group members, and let *A*, *B*, and *C* be the members contained in A1, and *D*, *E* be the members contained in A2. The intermediate subgroup keys for each fragment are: *K*(A1) = $a^{a^{ab}c}$, *K*(A2) = a^{de}, respectively. The group key that will be computed from the merging of the two fragments becomes: *K*(TGDH) = $a^{K(A1)*K(A2)}$. If we had a flat configuration we would be able to derive exactly the same group key if an "empty" value was placed between members *C* and *D* in the logical flat deployment of all members in the 1st tree level. Namely, the order of this configuration from left to right would be the following: *A, B, C,* $\varnothing$*, D, E*, leading to the following pairs of the 1st level: {[*A*, *B*], [*C*], [*D, E*], [$\varnothing$]}. This is exactly the deployment of the 1st level configuration of each TGDH sub-tree that corresponds to each fragment separately, if we append them all together.

The gateways of all merging fragments are responsible for the communication exchanges between the sponsors and the "super-sponsor" in the overlay TGDH, and also propagate the received BKs from the sponsors to all members in their fragments along with the information about the configuration of the TGDH overlay tree. This way, all members are able to generate the global group key, which has the structure of a typical TGDH key. The overhead associated with case 4_1_1 is computed by a flat execution of the original TGDH in two steps (TGDH within each fragment, and then merging of the fragments and broadcast of corresponding BKs to the remaining members in each fragment).

Hence, the **merging overhead** (including the computation of fragment subgroup keys) is:

$p_{4\_1\_1}$ (*Group fragment TGDH,n,m,s,Comm*):$m\times(2\times s)$ bcast or $\log_2 s$ messages per member] +[$(2\times m)$ bcast messages in the sponsors TGDH tree or $\log_2 m$ messages per sponsor]+[*m* bcast messages from all *m* sponsors to distribute the group key shares to their members],

$p_{4\_1\_1}$ (*Group fragment TGDH, n, m, s, Comm*): $(2n+3m)$ total messages,[$(\log_2 s+\log_2 m+1)$ messages per sponsor, $\log_2 s$ messages per member,

$p_{4\_1\_1}$ (*Group fragment TGDH, n, m, s, Comp, exp/s*) = [$m \times (4 \times s)$ in total within fragments, or $2\times \log_2 s$ per member]+[$(4\times m)$ in total in the sponsor TGDH, or $2\times\log_2 m$ per sponsor] + [n for the sponsors or members to compute shares of the final group key or the group key].

$p_{4\_1\_1}$ (*Group fragment TGDH, n, m, s, Comp, exp/s*): $(5\times n + 4\times m)$ in total, $(2\times\log_2 s + 2\times\log_2 m +1)$ per sponsor, $(2\times\log_2 s +1)$ per member.

The operation of **re-keying** in the **merged scheme** is handled as in the typical TGDH, and the overhead incurred is exactly the same. The **re-keying** overhead under 4_1_1 becomes:

$p_{4\_1\_1}$ (*Re-Keying (eviction) TGDH, n, m, s, Comm*): a broadcast of $h = \log_2 n$ modified BKs from the super-sponsor to all members,

$p_{4\_1\_1}$ (*Re-Keying (eviction) TGDH, n, m, s, Comp, exp/s*) = $2\times h$ [super-sponsor] + $2\times n$ [members] (a member does from 1 to (h-1) MEs based on the number of BKs that change for it, or 2 MEs in average).

Case 4_1_2 (DS-TGDH): The fragments execute DS-TGDH. The case is handled similarly to 4_1_1, as the logical algorithm is the same for both. DS-TGDH is just a decentralized version of TGDH. When the fragments merge, the application of DS-TGDH over the roots of the previous DS-TGDH fragments does not necessarily result in an optimized DS-TGDH. Still, the tree that is formed from the two-fold application of DS-TGDH is a DS-TGDH.

Hence, the **merging overhead** (including the computation of **fragment subgroup keys**) is:

$p_{4\_1\_2}$ (*Group fragment DS-TGDH, n, m, s, Comm*): $(2n+3m)$ total messages, [$(\log_2 s+\log_2 m+1)$ messages per root, $\log_2 s$ messages per member,

$p_{4\_1\_2}$ (*Group fragment TGDH, n, m, s, Comp, exp/s*): $(5\times n + 4\times m)$ in total, $(2\times\log_2 s + 2\times\log_2 m +1)$ per fragment root, $(2\times\log_2 s +1)$ per member.

The **re-keying** overhead under 4_1_2 is similar to this computed for 4_1_1.

Case 4_2: The gateway members of all merging fragment execute GDH.2 using the subgroup keys of their fragments as their initial secret shares. The gateways may propagate to the rest of members of their fragment either the whole broadcast message (robustness), or only the single value required for the generation of the group key (efficiency). Re-keying is executed as follows: only the fragment that witnesses the membership change initiates re-keying as in (DS) TGDH, and computes a new subgroup key. Then, the gateway of this fragment triggers re-keying via overlay GDH.2, using the newly computed subgroup key to re-calculate the KM material of GDH.2 downward stage. The **merging overhead** of members including the computation of the **fragment subgroup keys** becomes:

p_{4_2} (*subgroup regular TGDH*, *group overlay GDH.2, n, m s, Comm*) = $m \times 2s$ [total broadcasts for all fragments] + $\frac{1}{2}(m^2+3m\text{-}4)$ [broadcast for overlay execution of GDH.2] + $m \times n$ [or else a broadcast message of *m* elements distributed to all *n* members],

p_{4_2} (*subgroup regular TGDH,group overlay GDH.2,n,m,s,Comm*)=$2n+\frac{1}{2}(m^2+5m\text{-}4)$ [bcast],

p_{4_2} (*subgroup regular TGDH*, *group overlay GDH.2, n, m s, Comp*) = $m \times 4s$ [MEs all members all fragments]+$\frac{1}{2}(m^2+3m\text{-}2)$ [overlay GDH.2]+*n* [all members for final group key],

p_{4_2} (*subgroup regular TGDH*, *group overlay GDH.2, n, m s, Comp*) = $5 \times n + \frac{1}{2}(m^2+3m\text{-}2)$.

The **overhead** for the **re-keying** within a given **fragment** (size *s*) becomes:

p_{4_2} (*subgroup TGDH*, *s*, *Comm*) = $\log_2 s$,

p_{4_2} (*subgroup TGDH*, *s*, *Comp*) = $2 \times \log_2 s$ [sponsor] + $2 \times s$ [members] (a member does from 1 to ($\log_2 s$ -1) MEs depending on the number of BKs that change for it, or 2 MEs in avg.).

The modified subgroup key triggers **overlay GDH.2 re-keying**, the **overhead** of which is:

p_{4_2} (re-keying GDH.2, *n*, *m*, *Comm*) = $m \times n$ (or else a broadcast message of *m* elements distributed to all *n* members), and p_{4_2} (re-keying GDH.2, *n*, *m*, *Comp*) = (m+n-1) [MEs].

In total, the **re-keying overhead** for the **group** becomes:

p_{4_2} (*Group merge TGDH-GDH.2*, *n*, *m*, *s Comm*) = (m + $\log_2 s$) broadcast messages,

p_{4_2} (*Group merge TGDH-GDH.2*, *n*, *m*, *s*, *Comp*) = (m+ n + $2 \times s$-1) [MEs].

Faulty Members: If *C* or *D* are found "faulty" at any round (assume *j*) during key establishment then this information is propagated to the corresponding fragments. Then, (DS) TGDH stalls at round *j* and either the sponsor in the case of TGDH and/or only the members in contact with the "faulty" member up to round *j* in the case of DS-TGDH respectively, re-start the protocol from round 1 to round *j*, excluding the contribution of the "faulty" member to preserve forward secrecy (as in Basic). In either case, *j* new BKs will be generated and communicated to the members that are involved in this intermediate process (reduced (DS) TGDH). Then, (DS) TGDH will resume normally from round (*j*+1).

Summary: The two methods proposed (4_1_1, 4_2) are comparable to each other, and the superiority of one over the other depends on the values of the parameters *n*, *m*, and *s*. For example, if (*m*+2*s*) < *n*, then the method proposed in Case 4_2 is superior in terms of re-keying computation overhead to Case 4_1_1 by a factor of (*n*-*m*-2*s*).

4. 3. 4. Conclusions

We have methodically tested all potential failure scenarios each of the given protocols (*wt*-ING, *wt*-GDH.1, *wt*-GDH.2, *wt*-BD, (*wt*) TGDH, and (*wt*)-Hypercube) at any point during execution. For each case of failure, we studied its impact on: (*a*) the keying algorithm and messages, or equivalently the logical flow of the KG algorithm, and (*b*) on the existing communications schedule, and consequently on the group connectivity. We conducted a comprehensive investigation of the discussed schemes to understand how both (*a*) and (*b*) are affected from a "disruption". At the same time, we introduced lightweight techniques to make the protocols capable of circumventing certain "disruptions" or resuming with low extra overhead, from the perspective of (*a*) and (*b*). In addition, our analysis designates **how many** nodes and **which are affected from** certain kinds of failures.

4.4. Leader Election: Subgroup Leader Election for Octopus Based Protocols

Additional crucial parameters for the evaluation of KM protocols are those related to leaders (if any). The tasks undertaken by a leader, the impact of a leader’s failure to the protocol, the process of leader election, etc. determine significantly the nature, characteristics and performance of a KM scheme. This is even more so for hierarchical schemes, where groups are divided into subgroups, and each subgroup is provided with a subgroup leader, such as the Octopus-based protocols. The selection of the transmission range among subgroup leaders (inter-cluster) and the transmission range between an individual leader and its members (intra-cluster), is crucial for the operation of such protocols. For the purposes of our work we focus on the operation of Octopus protocols between two re-clustering instances. Although clustering is a side effect of the Octopus-based scheme, it constitutes a broad research direction of its own and is out of the scope of this work. Some more lightweight operations are incorporated into the Octopus-based schemes, to handle dynamic changes as “locally” as possible, and mitigate the need for re - clustering for the longest time possible. One solution is to have the leader election algorithm within each subgroup to elect **more than one potential leader**. These backup subgroup leaders can be in “hibernation” during the operation of the primary subgroup leader, or can be communicating information other than KM material about members, for a smoother hand-off and smaller delays in the subgroup. Finding **backup subgroup leaders** capable of covering the entire subgroup may not always be feasible, but it is highly desirable that they compare favorably to the other members in terms of the following characteristics: (*a*) robustness to dynamic changes, (*b*) high bandwidth, capacity, and residual energy, and (*c*) can cover a considerable portion of the subgroup, for a potentially long period. Octopus allows an isolated, localized handling of each network area, where different constraints prevail, resulting in a more efficient group key establishment, through the existence and maintenance of flexible subgroups.

4.4.1. Overview of a leader election algorithm for hierarchical protocols

We assume that initially, during the deployment of the network, a pre-agreed set of subgroup leaders, known to each other, undertake all the required tasks for the initialization of secure groups and subgroups, and consequently, for the establishment of secure group communication. These subgroup leaders advertise themselves to network nodes in their proximity (with fixed transmission level). Nodes that get one or multiple advertisement messages from subgroup leaders in the vicinity select and notify the leader they wish to join, based on certain criteria. For example, they may select this leader from which they receive a signal with the highest SNR. A subgroup leader at this point accepts potential group members and registers them. Each subgroup initiates the subgroup key establishment phase after the expiration of the advertisement period. In the mean time, the subgroup leader propagates the current subgroup identity *s_id* to its members, and through link state information, further in the network. This is essential for mobile nodes that wish to join the group but have not been reached by any of the advertisement messages yet. If the subgroup that accepts to register the newcomers has already established a subgroup key, but has not yet contributed its part to the second phase of Octopus, the subgroup simply re-keys, and all nodes obtain a new subgroup key that includes the newcomer. If a new node registers to a subgroup after this subgroup has contributed its part towards the 2nd phase of the Octopus, then this node is registered with the status of "guest", but is not allowed to get the current subgroup key. The new node will have to wait until after the current group key is established, and after another wait period during which the subgroup either probes for potential newcomers. After its expiration, the subgroup re-keys and triggers again the 2nd phase of Octopus that will generate a new group key (large scale re-keying) with lower cost.

Metrics and parameters for the subgroup leader election:

(*a*) The **residual energy** E_{res} , at the given instance when the measurement is taken.

(*b*) The maximum **transmission inter-cluster and intra-cluster ranges T_l and T_s**, at which the node operates when it becomes subgroup leader.

(*c*) The **node degree** of a candidate leader. It is actually related to T_s, since the number of nodes that can be reached from a given member depends on T_S.

(*d*) The combined power required from individual members to reach a given candidate subgroup leader (even if multi-hop routing is required). This metric reflects the total energy consumption within a subgroup from the point of view of the individual members. We denote this metric as **Combined Member Power (CMP)**, and we define: **CMP =** $\sum_{i=1}^{K} pow_i \times r_i$, where: K is the number of subgroup members, pow_i : the minimum power required so that a member either reaches the leader in 1-hop, or a relay member trying to get to the leader, and r_i is the number of times a member is requested to reach or serve as relay towards the leader.

(*e*) This metric reflects the number of messages required so that the leader is reached by all nodes. It is denoted as **Total Communication Hops (TCH)** and defined as **TCH** = $\sum_{i=1}^{K} r_i$.

Overview of the subgroup leader election algorithm.

We assume that the leader election process runs proactively, while a subgroup leader is currently active within the associated subgroup. The algorithm may be operating in parallel with the KG, synchronization or coordinating functions required within the subgroup. Subgroup members with E_{res} above **threshold** E_{res_THR} may contact the current leader and become **legitimate** candidates. The leader notifies all subgroup members of the identity of these legitimate candidate leaders. Then, the leader will trigger and initiate ***M*** measurements at arbitrary instances of its own choice, i.e. when it is not executing a heavy KM operation, or when it is idle, i.e. waiting for the expiration of t_{batch} period. The subgroup leader sends trigger messages to all the candidate leaders to initiate the measurements, and notifies all members that a measurement is taking place. Each member, after receiving the measurement notification messages, attempts to contact each of the candidate subgroup leaders using minimum relays, adjusting its transmission range if feasible (and reporting it),. Candidate subgroup leaders, wait for time t_{get} to collect this information from as many members as possible. After the expiration of this period, they calculate the metrics CMP and TCH. The next step is that each candidate leader, selects one or multiple intra-cluster transmission ranges, and for each selected transmission range, it measures the degree of subgroup nodes it is directly connected to. Each candidate leader, after computing pairs of transmission range and subgroup member degrees (T_s, $D_{\_Ts}$), sends all measurements to the current leader. The current leader feeds the measurements of each individual candidate into a **combined cost function** denoted as ***SL*(*R*)**, where ***R*** is the sequence number of the measurement in consideration, 1<*R*<*M*. Alternatively, the subgroup leader may store all these measurement values separately and consider only a subset, depending on which metrics are of interest to improve. The analytical formula of the combined cost function has the following structure:

$$R = \quad \alpha \times E_{res}(R) + \beta / CMP(R) + \gamma / TCH(R) + \delta \times {}^{D_{Ts}(R)}\!\big/_{T_S(R)},$$

where: $a, \beta, \gamma, \delta \geq 0$, are pre-selected and are fine-tuned to meet the special requirements of each subgroup. There measurements are triggered by the current subgroup leader at arbitrarily selected intervals. The larger the amount of time that the current subgroup leader remains active, the larger the amount of such measurements that can be taken, and the more accurate the selection of the successive leader becomes.

By taking a number of measurements at regular intervals, we incorporate indirectly in our costs, the robustness of the candidate leaders to dynamic changes, within the subgroup. For example, metrics such as: CMP, TCH, D_{Ts}, T_s are affected by mobility or membership changes, or failures, either of the candidate leader itself or of any of the members. Metrics CMP and TCH in particular consider indirectly the relative positions of subgroup nodes, and reflect these into practical measurements of energy consumption and communication cost in the subgroup overall. For example, this algorithm will capture a subgroup leader that is gradually moving away from most of the subgroup members, by producing CMP and TCH values that become larger and larger at every successive measurement. The current leader, collects and evaluates the proper values, and orders the candidate subgroup leaders in a list from best to worst candidate. It communicates the list to all candidates, so that they all become aware of the next best selections in case of one or multiple failures.

4.4.2. Analytical Cost of the Subgroup Leader Election Algorithm

Notation_1: Let *x* be the maximum number of candidate leaders, *K* be the number of subgroup members, *M* be the highest number of measurements conducted, *L* and *P* be the maximum number of times a member or a candidate leader adjust their transmission levels.

We have just described a proactive algorithm for leader election within the subgroups of Octopus protocols. We will now briefly evaluate the communication, computation, storage and energy cost of this algorithm for the subgroup members, current subgroup leader and candidate subgroup leaders. Initially, the current leader receives up to *K* messages from members requesting to be considered candidates, and selects *x* of them. The leader **broadcasts** to its subgroup **a message** triggering the measurement process, including the *ids* of the candidate leaders. Each member attempts to reach each of the *x* candidates, by adjusting its transmission power level if feasible, ***L*** times at maximum. If it still cannot reach the candidate it will use one or more relays. Thus, member *i* transmits: $\frac{L+1}{2} \times r_i$ messages in average, and the corresponding energy consumption becomes: $\frac{L+1}{2} \times r_i \times pow_i$.

Each candidate collects also the contribution of all members to the CMP and TCH measurements, processes them (adds them), and temporarily stores them (2*K*). The associated processing cost for each metric is O(*K*). Then, it measures its node degree, adjusting its own transmission level *P* times at maximum. Each measurement requires one broadcast message from the candidate leader. Members within transmission range send back an ACK reply.

Ideally, the candidate subgroup leader aims to capture all nodes within 1-hop, so it starts with a feasible transmission range that can cover as many subgroup members as possible. The communication cost for this measurement is approximately $P(1+K)$ at maximum. The overall processing cost is $P\times K$. Then, the candidate combines these measurements in one packet and communicates this packet to the current subgroup leader. Therefore, the subgroup leader receives x messages to store and process. It orders the candidate leaders either based on the value of a single metric, or on a combination of multiple metrics. Each metric value is combined with the associated values from the previous measurements. So, the leader must add x packets, and then run a sorting algorithm, e.g. merge-sort, quick-sort, etc. that takes about O($x$log$x$) processing time. Then, the processed list is broadcast to the x candidates. Next, we summarize the communication, computation, and storage overhead incurred to all subgroup members in order to execute this subgroup leader election algorithm.

Single Measurement

1. Current Subgroup Leader:

Communication Cost (# of messages): 1 + 1 = 2

Energy Consumption Cost: $2 \times pow(T_s)$

Computation Cost (# of packets): $c \times K + c \times x + c_2 \times x \times logx$, where $c, c_2 < 3$

Storage Cost (# of packets): $K + x$

2. Candidate Subgroup Leader *i*:

Communication Cost per Candidate *i* (# of messages): $1+ \frac{L+1}{2}\times r_i + P + 1$

Total Communication Cost (# of messages): $K + \frac{L+1}{2}\times\sum_{i=1}^{x} r_i + P \times K + x$

Energy Consumption Cost per Candidate *i*: $(2+P)\times pow_i + \frac{L+1}{2}\times r_i \times pow_i$, assume $r_i<K/4$

Total Energy Consumption Cost: $(2+P+\frac{L+1}{2}) \sum_{i=1}^{x} pow_i \times r_i$, assume $r_i < K/4$

Computation Cost (# of packets): $2\times c_1\times K + P \times K$, where $c_1 < 2$

Storage Cost (# of packets): $K + 2K + P = 3K + P$

3. <u>Simple Subgroup Member *i*:</u>

Communication Cost per Member *i* (# of messages): $1 + \frac{L+1}{2} \times r_i + P$, assume $r_i < K/4$

Total Communication Cost (# of messages): $K + \frac{L+1}{2} \times \sum_{i=1}^{K} r_i + P \times K$, assume $r_i < K/4$

Energy Consumption Cost per member *i*: $(1+P) \times pow_i + \frac{L+1}{2} \times r_i \times pow_i$, assume $r_i < K/4$

Total Energy Consumption Cost: $(1+P+\frac{L+1}{2}) \sum_{i=1}^{K} pow_i \times r_i$, assume $r_i < K/4$.

4.4.3. Feasibility of Leader Election Algorithm for Octopus-based Protocols

This algorithm is feasible for all hierarchical protocols, but it can be also feasible for small flat networks, or for large flat networks divided in regions: each region is a subgroup and has its own leader. This scheme is even more feasible for Octopus protocols for the following reason as well: (O), MO, and MOT, operate in 3 distinct steps. During the 1st step, all entities are active (communication-wise) and compute a subgroup key, during the 2nd step simple members are idle and subgroup leaders interact among themselves to compute the group key, and during the 3d step subgroup members become recipients of KM material sent by the leaders, and compute the group key. Since the measurements may be triggered at arbitrary instances, the 2nd step of initialization, or the 1st or 2nd step of re-keying, during which members are idle and are not executing heavy KM operations, can be exploited. All members (including candidates) can conduct the bulk of their measurements during these time frames. Then, they can piggyback the results of their measurements in the same packets that contain KM material. The recipients will be multiple: these designated by the KG algorithm, and these designated by the measurements process. This can be done during the 1st step of the subsequent key establishment, or re-keying within a given subgroup, when all members participate to the subgroup KG: their KM material will reach the recipient, as designated by the logical KG algorithm, possibly through a varying number of relays. This gives an extra advantage to the simultaneous application of the leader election algorithm during this step. In addition, if we make a slight modification to the leader election algorithm, so that all members directly report the results of their measurements to the current subgroup leader, then it is obvious that we reduce the extra communication cost even further. This happens at the expense of extra computations executed by the main subgroup leader of course. The subgroup leader may then use the 3^{d} step to distribute the processed list to its members, piggybacked in the packets that contain shares of the group key.

4.4.4. Integrate Logical Design and Auxiliary Network Functions

The existence of a robust leader is a very important issue, particularly for centralized and hierarchical KM protocols. In these protocols, the implications of leader failures are severe: in most cases, a new leader must be elected and the protocols must start over. In contributory schemes like GDH.1-2 on the other hand, the function of a leader is mostly coordinating. However, even if the duties of a leader are reduced, its failure may still be as devastating for the contributory scheme as well. For example, in GDH protocols, the leader duties, even though different than the traditional duties of a centralized leader, are assigned to the last member reached by the upward stage. In GDH.1, in particular, it is very important that the leader is as stable as possible. Although in GDH.2 any member is capable of initiating efficient re-keying, in GDH.1 only the leader may initiate member addition at least (and member eviction in a few cases, as GDH.1 does not lend itself to efficient re-keying). If the current GDH.1 leader has failed, the protocol must start over in order to include or remove a member, degrading the overall performance of the protocol.

In our analysis, we selected three protocols for the subgroups of the Octopus based schemes. From all these protocols, only GDH.2 may go around the event of a leader failure with lower cost and impact than the rest. A leader failure may be handled as a member failure indeed: GDH.2 goes around it with low cost re-keying, and with the election of any other member as a leader. In the other protocols, either the leader failure entirely stalls the execution (i.e. centralized (O), GDH.1), or a new leader election with strict specifications must be executed before the protocol resumes (i.e. TGDH). Hence, the benefits of the proposed scheme may be more significant for (O) and MOT, compared to MO. Given all that, if a very simple leader election scheme is used within MO instead, then MO takes the lead from the other protocols in terms of performance for the first time. This is so, unless the overhead from a potentially vast number of re-keyings followed by new leader elections, outweigh the performance improvement achieved. This would be the undesirable trade-off for not incorporating a more sophisticated leader election scheme. Of course, if the same leader election protocol is used in all three schemes, then the performance associations of the three protocols are not modified.

In addition, we have shown that the *wt* protocols are flexible to network partitions and merges. These protocols maintain most of their performance attributes even when applied to partitioned or merged groups. Even when they are not operating according to their full potential, lightweight techniques are introduced to ensure that the schemes adjust to abrupt changes and: (*a*) still improve on the most important metrics of interest compared to their original *nt*-ancestors, and (*b*) minimally deviate from the natural progression of the key generation algorithm. Of course, after a number of key establishments, the *wt*-protocols may be reconfigured to take into account the latest topological changes, as this would be the case anyway, even under less severe dynamic changes. We have contributed towards the above by extending the design of these *wt*-schemes with lightweight techniques and rules that can be used to make the schemes more robust and efficient to dynamic changes and disruptions. Although our basic motivation for this contribution was to extend our proposed Octopus solutions to deal successfully with network dynamics (complete Chapters 2, 3), the analysis conducted in this chapter has its own merit of interest as well.

Chapter 5: Two Level Hybrid Schemes for an Over-Layered MANET

5.1. Introduction

In this chapter, we are shifting our focus from totally flat to hierarchical MANETs with a specific configuration that resembles typical scenarios encountered in civilian cases or most often in the battlefield. Our physical environment is heterogeneous and consists of a number of wireless LANs often communicating with the aid of an overlay network. These overlay networks provide member with “special capabilities”, i.e. members with very high bandwidth, computational resources, storage capacity, etc. Furthermore, they provide robust and trusted nodes with sufficient capabilities to become group leaders. The network consists of heterogeneous nodes that result in producing links of variable qualities, uni/bi-directional asymmetric paths, larger bandwidth resources at higher tiers (satellites, planes), restricted at lower (cell-phones, laptops), different physical/communication mobility levels. At the low end mobility is more rapidly changing and higher degree of self-organization is observed. Nodes often need intermittent connectivity to reach others at the higher end.

Our objective in this chapter is to develop KM protocols for group communications for the discussed networks. Towards this end, we have designed a two-level hybrid KM scheme that models these environmental variations so that the first level of hierarchy represents nodes at the higher end, and the second level represents nodes at the lower end. The two-level hybrid scheme (2HH) exploits the diversity of the battlefield: it links key distribution to network topology, hierarchy, predicted or unpredicted member mobility, and routing. The 2HH has adaptability to the network limitations and topology. It may use a **combination of protocols in the two levels** to achieve improvement in one or more of the metrics of interest: i.e. to achieve substantial reduction in the communication or computation overhead, or both. Additionally, this scheme may simultaneously support a wide range of protocols of different categories. Our additional aim is that given the parameters of the network and a wide range of KM protocols, the most appropriate protocols to be applied to each of the two levels of this 2HH can be automatically selected. In the following sections, we describe the new model in detail, explore how combinations of a wide range of KM protocols affect the metrics of interest for the 2HH: (*a*) which combinations of protocols are best, (*b*) if and how the contribution (benefit) of the protocol combinations is affected by varying the network parameters, and (*c*) provide an analytical tool that automatically determines the best combinations for the 2HH, w.r.t. the metrics of interest, given the network parameters.

We will investigate the effect of all possible combinations of two diverse centralized KM protocols that make sense: the star-type Group Key Management Protocol (GKMP), and the tree-type Logical Key Hierarchy protocol (LKH). As part of our analysis, we will provide closed cost formulae of the resulting metrics of interest. After thoroughly exploring the contribution of these two protocols on 2HH, we extend our investigation on a number of different KM protocols, from all categories.

5.2. Two Level Hybrid Scheme (2HH) Model

The “secure” distribution of the group key is achieved with the use of some kind of public key infrastructure (PKI). For example, a group that executes DH-based KA protocols does not require such infrastructure, because the security of the group key is guaranteed by the inherent properties of the DH scheme.

5.2.1 PKI for Group Key Distribution (KD)

A PKI enables users of a basically unsecured public network to securely and privately exchange data through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority. A PKI provides for a digital certificate that can identify an individual or an organization and, when necessary, revoke the certificates. The PKI assumes the use of *public key* or *asymmetric cryptography*, which is the most common method for authenticating a message sender or encrypting a message. In public key cryptography, a public and private key are created simultaneously by a Certification Authority (CA) using the same algorithm. The private key is given only to the requesting party and the public key is made publicly available (as part of a digital certificate). A member uses its private key to decrypt text that has been encrypted with its public key by someone else. The most popular public key encryption/decryption mechanisms are: RSA (Rivest, Shamir, Adleman), Elliptic Curves, El Gamal.

The other mechanism for the exchange of messages among users is the private or symmetric key encryption/decryption method. The private key system is known as *symmetric cryptography* because we use one key to carry out the encryption and its inverse to decrypt the message. Methods for symmetric key exchange abound. Among the most popular ones are: DES (Data Encryption Standard) and its variations (i.e. 3-DES), IDEA, AES, etc. The symmetric encryption/decryption is a fast operation in comparison to the asymmetric one, but it is more vulnerable to attacks, and requires secure channels and many assumptions in order to be used in practice. The basic drawback of the public key encryption/decryption mechanisms is that they are very expensive and slow compared to the symmetric ones.

The mechanism that is used today for encryption/decryption is the best compromise of the two protocols. The exchange of messages among the group is performed with the symmetric key encryption/decryption method, which is fast and inexpensive. However, we distribute the private keys used in the symmetric protocol with the asymmetric method in order to make sure that these private keys are safe from attacks and as securely distributed as possible. In terms of the public key mechanism, we assume that every node is provided with its own private key and has published a public key that corresponds to its private. Every time a node is added or deleted from a group, we have to update the symmetric key used by the group in order to preserve forward or backward secrecy respectively. This update occurs by encrypting the new symmetric group key with the private key of each member, i.e. we use the public (asymmetric) method again. Also, the group leader in charge of the KD process, may generate a new secret key for every new member. The leader encrypts this secret key with the public key of the new member and sends this encrypted message to the member. The member decrypts the message containing its secret key with its private key (known only to itself) and obtains the secret key that is now known only to the leader and itself. The member will use the secret key to communicate privately with the leader from now on. In this manner, the public key method is used as a secure channel that communicates to the group members the secret keys that are going to be used by the symmetric key method. Therefore, centralized protocols that execute the task of KD for the multicast system need some PKI. For the purposes of this work, and for all the protocols integrated in the 2HH scheme that require a PKI, we select the **RSA public key encryption/decryption method**, and the **DES symmetric key encryption/decryption method** among others.

5.2.2 Parameters, Symbols, and Abbreviations

Abbreviations:

GSC: Group Security Controller

GSA: Group Security Agent

2HH: Two Level Hybrid Hierarchical Scheme

GKMP: Group Key Management Protocol

CBT: Core Based Tree Protocol

Parameters:

n: number of users in 2HH,

K: length (bit size) of an element for the asymmetric method in star-based schemes.

K': length (bit size) of an element for the asymmetric method in tree-based schemes.

d: number of descendants of a tree node, in tree-based KM protocols

h: height of the tree, in tree base KM protocols

p: probability of inserting or removing an entity other than the global leader of 2HH (or GSC)

Cryptographic Parameters:

C_{PE} /C_{PD}: Computation Cost per PKI Encryption/Decryption, C_{SE}/C_{SD}: Computation Cost per symmetric Encryption/Decryption, C_r: Computation Cost for a Pseudo-random Number Generator for primes (for RSA keys), C_{rr}: Computation Cost for a Pseudo-random Number Generator, C_E*:* Computation Cost for the operation of Modular Exponentiation (ME), C_g Computation Cost for the Hashing operation.

5.2.3. Two-Level Hierarchical Scheme (2HH)

The configuration of 2HH is illustrated in Figure 5.1. It consists of two levels, the upper level (first) and the lower level (second). Each level contains a different group or groups of entities, and the operation of KM is independent for each level. Hence, the KM of the whole 2HH scheme is done in a hierarchical manner as well. The characteristics of the network and of individual entities are different for each level. The bandwidth resources are larger at higher tiers (satellites, planes), and restricted at lower (cell-phones, laptops). The physical and communication mobility levels differ as well. At the low end mobility is more rapidly changing and higher degree of self-organization is observed. Nodes often need intermittent connectivity to reach others at the higher end.

First level: It consists of a single KM group whose members are the GSAs. The leader of this group becomes the GSC of the whole 2HH. If the KM protocol applied to the group is centralized, then the GSC securely distributes a group key to its members, the GSAs. The GSC is the most powerful entity in the system, followed by the GSAs. The GSAs are entities with enhanced capabilities compared to the simple members, they are considered to lie in between a GSC and a member, with respect to various characteristics: power, bandwidth, capacity, mobility profile, etc. The GSC usually delegates its duties to GSAs, who undertake the task of independently establishing group keys with a subset of simple members.

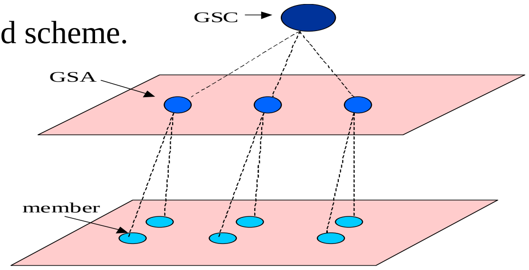
Second level: It consists of multiple groups. Each GSA (defined as member in the group of the first level) now becomes the leader of its own group of simple members. Hence, ideally, the second level consists of as many groups as the number of GSAs in 2HH. Each group may contain an arbitrary number of simple members. Therefore the following analogy obviously holds: *the GSAs are to the GSC what the simple members are to the GSA*. If we assume without loss of generality that the average size of the group handled by each GSA is n_2, and we further assume that the total number of GSAs that participate in 2HH is n_1, then, we compute the overall number n of entities contained in 2HH as follows: $n = 1+n_1\times n_2$.

We assume that all GSAs have a common mobility profile, partially characterized by the parameter p_1, which represents the probability of a GSA being added to or removed from the KM group of the 1st level. Similarly, we assume that all simple members have a common mobility profile, partially characterized by the parameter p_2, which represents the probability of a simple member being added to or removed from the KM group of the 2nd level.

Independency of Subgroups (Independent Update): Each subgroup in 2HH is an independent entity in the sense that the members of each different subgroup establish their own subgroup key, in addition to the global group key. Hence, the group of GSAs (1st level), and each of the groups of simple members (groups in the 2nd level) establish their own individual subgroup key. In fact, the global group key can be communicated to the entities in 2HH with the aid of these intermediate subgroup keys: it can be encrypted with these intermediate subgroup keys. Whenever a member is added to or evicted from a subgroup in either level, the subgroup key is updated, independently from the other subgroup keys. In that sense, the key updates are only limited within the various subgroups, and the process becomes more scalable to a growing size of the secure group.

Physical interpretation of parameter *p*: This mobility parameter maps the mobility characteristics of entities in 2HH to the rate of key updates required in a given time frame. For example, if the mobility characteristics of nodes are known or computable (i.e. speed, direction, mobility model, bandwidth, etc.), then the approximate time when a member will disconnect from its group can be predicted. This event is followed by an update in the group key of the evicted member, in order to maintain forward secrecy. **The frequency of the group key updates** is the important parameter for a KM protocol under the steady state. The relative mobility of members is among the major factors that determine the parameter *p*. However, other factors like resources' depletion, i.e. battery drainage, poor physical protection of members, etc. also play a role in determining accurate margins for parameter *p*. Hence, we can say that parameter *p* summarizes the dynamic events observed during a given time frame within a group that lead to re-keyings. Equivalently, the probability *p* can be interpreted as the group key update rate, or the frequency of re-keyings. The values that it might take depend on the network parameters which vary in different executions. In the end of this chapter, we describe the simulations conducted to capture the frequency of members being disconnected from their subgroups (i.e. partially parameter *p*), using three different mobility models. Our aim was to: (*a*) determine how the frequency of re-keyings is affected by different mobility profiles alone, and (*b*) provide realistic values for *p*, since it is used as a free parameter in our analysis of the different metrics (costs) of different combinations of protocols integrated in 2HH.

Definition: *Average Operating Cost* is the average computation cost for the addition or deletion of a GSA or member to the hybrid scheme.



Notation for Distributed Keying Elements

$K^1{}_{ij}$: GSA to member,

$K^2{}_i$: GSA to cluster,

K^3: GSC to members,

$K^4{}_i$: GSC to GSAs.

Figure 5.1. 2HH Model schematic illustration: First Level (upper): cluster between GSC and GSAs, Second Level (lower): GSAs to members.

Next, we are going to provide a detailed analysis of the two most significant protocols integrated into 2HH in various combinations: GKMP and CBT. The combinations that involve these two schemes will be thoroughly analyzed and will serve as the guiding example for all other combinations that involve different protocols.

Basic Assumption: Throughout 5.3, we assume that the probability for inserting/deleting a (new) GSA or a (new) member into/from the group is (1/2).

Discussion: In this chapter, we assume that there exists an underlying mechanism that ensures that all members that participate in key distribution are already authenticated. We further assume that the GSC acquires the capabilities required for playing the role of a CA. For every entity a designated pair of RSA public and private keys is assigned by a CA or TTP. The public key of this entity is published along with its identification. All public keys of the network entities are published at the GSC. The GSAs have access to the GSC at any time: the availability of the GSC for the GSAs is assumed in terms of services and bandwidth both for the up-link and for the down-link. Hence, the GSAs communicate with the GSC to ensure that a given member is authenticated and get its authentic public key. The GSC executes the following tasks: (*a*) monitors the network for membership changes, (*b*) authenticates the new members, (*c*) assigns RSA public and private keys, and (*d*) periodically refreshes the certificates of members. Members do not have direct access to the GSC through the up-link but communicate with their local GSAs. The GSC can access all members. If a member looses connectivity with its GSA, it is considered evicted from the particular GSA. In this case, the member either joins another GSA, or it becomes evicted from the whole network.

5.2.4. GKMP Overview (Group Key Management Protocol)

The GKMP scheme can be used as the organization scheme within each subgroup of a given GSA (2^{nd} level) or within the group of GSAs (1^{st} level) as well, in the 2HH model. It is a star-based centralized protocol. GKMP has a single group leader of group controller. Each member *i* stores a pair of keys: $GKP_i = \{SEK, KEK_i\}$. KEK_i is an abbreviation for "Key Encrypting Key" and denotes the individual secret key that is shared only between the group controller and member *i.*, with which SEK will be encrypted. SEK is an abbreviation for "Session Encryption Key" and represents the session key (group key) that will be shared among all group members for the purpose of the secure multicasting.

GKMP provides small storage (each member needs to store only two secret keys in addition to their RSA pair of public-private keys), at the expense of a large communication overhead for membership updates (evictions). Whenever a member is evicted from a secure group of size *n*, the current GSC computes a new group key (SEK) for the group. It encrypts the SEK (n-1) times with the KEK of each of the (n-1) remaining members, and sends the generated cyphertext to each member individually. Hence, the computation and communication costs required for this operation are linear to the size of the group, so GKMP is neither efficient nor scalable at steady state. The operation of a member's addition is more lightweight. The GSC generates a new SEK and encrypts it: (*a*) once with the previous SEK, acquired by all *n* previous members, and broadcasts the same single cyphertext to all *n* previous members, and (*b*) once with the KEK of the new member, and individually communicates the new cyphertext to the new member. Hence, the GSC now performs only two computations (encryptions), and communicates only two different cyphertexts to all members.

GKMP Analytical Evaluation

Initially the GSC generates the *n* secret keys (KEKs) it will send to the *n* group members via the asymmetric encryption method (RSA). Then, it generates the group session key (SEK) to for the multicast communication. It communicates SEK to each member individually via the symmetric encryption method (DES). SEK is encrypted with the KEK of each member. Below we discuss some representative metrics, the rest are computed in a similar manner.

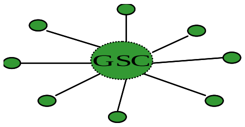


Figure 5.2. Schematic Illustration of GKMP configuration

Initial Computation: The GSC generates (*n*+1) new keys: the *n* KEKs and the SEK. The cost for the random key generation is given by the function C_r. Then, the GSC performs a public encryption (RSA) for each of the *n* KEKs, with cost C_{PE} for each. It then, unicasts the SEK to each member, symmetrically encrypted with its KEK. Thus, the GSC does *n* symmetric encryptions with cost C_{SE} for each. Hence, the cost is: $(n+1)C_r+nC_{PE}+nC_{SE}$. Each member first decrypts its KEK with cost C_{PD}. Then, it decrypts the SEK with cost C_{SD}.

Add Computation: When a new member is added, the GSC must change its SEK to preserve backward secrecy. First, it generates a new secret key for the new member, encrypts it with the members' RSA private key and sends it to the new member. Then it generates a new SEK. It broadcast it to all members, encrypted with the previous SEK, known to all members except for the new one. It also unicasts the new SEK to the new member symmetrically encrypted with the KEK of the new member. The total GSC cost is: $2C_r + C_{PE} + 2C_{SE}$. The new member asymmetrically decrypts the KEK and then, symmetrically decrypts the SEK. The rest of members need only decrypt the symmetrically encrypted (with the previous SEK) new SEK. Hence, the member cost becomes: C_{PD} (new member) $+C_{SD}$.

Add Communication: The GSC communicates a KEK to the new member and broadcasts the encrypted new SEK. It also unicasts SEK encrypted with the KEK of the new member to the new member. Thus, the cost becomes 3*K*.

Delete GSC Computation: When a new member is deleted from the group, the GSC changes the SEK to ensure forward secrecy. The GSC computes a new SEK, communicates the new SEK to each remaining member individually, symmetrically encrypted with the KEK of each member. Thus, the GSC also performs (*n*-1) symmetric encryptions, with cost C_{SE} for each, and the overall cost becomes: $C_r + (n-1)\ C_{SE}$.

5.2.5. Tree Based Key Management – Core Based Trees (CBT) Overview

CBT extends LKH. The *n* leaf nodes are the group members and the group key is associated with the tree root. Each member knows only all keys associated with the path extending from its leaf up to the root.

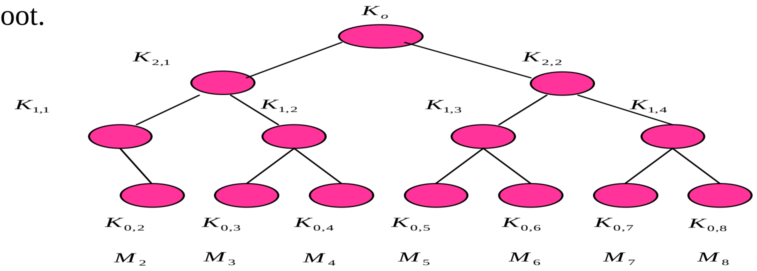


Figure 5.3. Illustration of CBT Configuration: member i is denoted as M_i, and its internal keys as $K_{j,\,i}$, where j is the tree level an internal key is placed on. Finally, K_0 is the group key.

A New Member joins the group: The key server (or GSC) assigns the new member to a leaf node. The new member receives all keys associated with the nodes on its path to the root. The received keys are independent from any previous keys in the tree. The GSC replaces all keys on the key path of the new member sequentially with fresh keys, starting from the leaf and going up the root. The GSC symmetrically encrypts each new successive key to be sent, with the new predecessor key just sent. The first key sent, associated with the tree leaf, is encrypted with the member's KEK. The GSC sends each of the new keys of the group members that use them on a need to know basis.

A Member leaves the group: All the keys known to the evicted member will be changed (so that forward secrecy is preserved). These keys are being replaced sequentially from the leaf up to the root. The GSC symmetrically encrypts each new successive key to be sent with the new predecessor key just sent. The first new key sent, associated with a given tree leaf, is symmetrically encrypted with the member's KEK. The tree configuration is very efficient because it only requires the update of $log_d\,(n)$ keys, assuming that the tree is balanced.

Evaluation of Cost for CBT

Initially, the GSC generates n secret keys to send to the n members of the group (associated each with a tree leaf) via the asymmetric method (RSA). Then, the GSC generates secret keys for all internal nodes in the tree. It symmetrically communicates to each member the keys associated with the nodes that belong to its path from the leaf up to the root. Thus, the GSC communicates approximately h secret keys to each member. The key associated with the tree root will be the session key for all multicast group communication. Below, we discuss some representative cost metrics, the rest are computed in a similar manner.

GSC Storage Cost: The GSC store all tree keys. The secret key associated with an internal node may be used by the GSC to encrypt a message for the d members (in a balanced d-tree) that contain this internal node in their paths from the leaf up to the root. The number of nodes in a balanced d-tree is: $d^0+d^1+d^2+\ldots+d^h = (d^{h+1}-1)/(d-1) = (dn-1)/(d-1)$.

Initial Computation: The GSC generates as many new keys as the number of nodes in the tree with cost C_r each. The total number of new keys generated are: $(dn-1)/(d-1)$. From these keys, the GSC asymmetrically encrypts the n keys that it is going to communicate to the members via RSA. The remaining keys, namely $(dn-1)/(d-1) - n = (n-1)/(d-1)$, are sent to the members symmetrically encrypted. The GSC symmetrically encrypts the key of each internal node with d different keys. Thus, it does $d \times (n-1)/(d-1)$ symmetric encryptions with cost C_{SE}

each. A member asymmetrically decrypts (with cost C_{PD}) its secret key. With this key it later symmetrically decrypts (with cost C_{SD} for each key) the h keys sent to it by the GSC.

Initial Communication: The GSC sends to the n members their n secret keys. Then, it sends $d\times(n\text{-}1)/(d\text{-}1)$ encrypted messages that will be symmetrically decrypted by the intended recipients. Hence, the cost becomes: $(d\times(n\text{-}1)/(d\text{-}1)+ n)K = ((dn\text{-}1)/(d\text{-}1) + (n\text{-}1))K$.

Delete Communication: The GSC, for each updated node, sends the new key symmetrically encrypted to $(d\text{-}1)$ children. It performs the same operation for all h updated nodes, thus the total cost becomes: $(d\text{-}1)\ h\ K$.

Parameters	GKMP	CBT
GSC Storage	$(n+1)\ K$	$(dn\text{-}1)\ K/\ (d\text{-}1)$
Member Storage	$2\ K$	$(h+1)\ K$
Init. GSC Computation	$(n+1)\ C_r\ +n\ C_{PE}\ +nC_{SE}$	$(dn\text{-}1)C_r\ /(d\text{-}1)+n\ C_{PE}+d(n\text{-}1)C_{SE}\ /(d\text{-}1)$
Init. member Comput.	$C_{PD} + C_{SD}$	$C_{PD}\ + h\ C_{SD}$
Initial Communication	$2\ n\ K$	$(\ n + d(n\text{-}1)/(d\text{-}1))\ K$
Add GSC computation	$2\ C_r + C_{PE} + 2\ C_{SE}$	$(h+1)\ C_r + C_{PE} + 2\ h\ C_{SE}$
Add member Comput.	$C_{PD} + C_{SD}$	$C_{PD} + h\ C_{SD}$
Add Communication	$3\ K$	$(2h + 1)\ K$
Delete GSC Comput.	$C_r + (n\text{-}1)\ C_{SE}$	$h\ C_r + (d\text{-}1)\ h\ C_{SE}$
Delete member Comput	C_{SD}	$h\ C_{SD}$
Delete communication	$(n\text{-}1)\ K$	$(d\ \text{-}1)\ h\ K$

Table 5.1. Communication and Computation Costs for GKMP and CBT protocols.

5.3. Hybrid Models Design and corresponding Cost Analysis

We integrate GKMP and CBT to 2HH, in six different combinations, and we calculate the metrics of interest for each such configuration.

5.3.1. First Scheme: Single unbalanced CBT over 2HH, Two Groups of Members

p_1: probability of GSA addition/deletion, p_2: probability of member addition/deletion.

$p1 < p2$ ($p1$, $p2$: motion frequencies).

n_1: # GSAs (clusters), n_2: # members in each cluster.

The resulting tree is unbalanced, with two distinct path lengths:

h_1 = - $\log_d p_1$ (GSA) and h_2 = $-\log_d p_2 < h_1$ (members).

The GSC distributes to GSAs and to members also the designated secret keys.

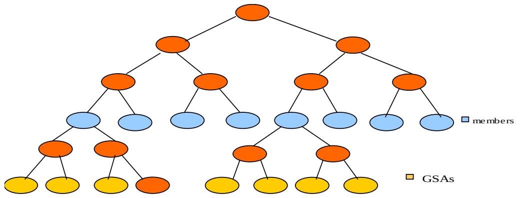


Figure 5.4. Single unbalanced CBT over 2HH, two groups of members (GSAs, members)

Storage Costs:

GSC:
$$\left\lceil \left(\frac{d^{h_2+1}-1}{d-1} + n_1 \frac{d}{d-1} \cdot \frac{d^{h_1-h_2}-1}{d^{h_1-h_2}} \right) K' \right\rceil = \left\lceil N_C \cdot K' \right\rceil$$

GSA: $\left\lceil (h_1+1) K' \right\rceil$ **Member:** $\left\lceil (h_2+1) K' \right\rceil$

Derivation of the expression N_c:

The first term of this expression is the storage cost for the tree of members only. Members only are associated with the leaves of a balanced sub-tree with height h_2. The members can apply the original CBT protocol, on such a balanced sub-tree, so the total number of nodes and thus the number of keys the GSC stores is: $(d^{h2+1}-1)/(d-1)$.

The GSAs are associated with tree leaves whose path to the root has size h_1. Equivalently, a sub-set of these members (selected at random) become the root to balanced sub-trees with height h_1-h_2. All non-empty leaves of these trees become associated with a GSA. This is how the original tree that comprises both members and GSAs is formed. As already discussed, the GSC stores the keys of all nodes of the tree.

The total number of nodes in the sub-tree of GSAs is: $(d \times d^{h1-h2} - 1)/(d-1)$. Each such sub-tree includes a maximum number of GSAs: d^{h1-h2}. However, given that the total number of GSAs is n_1, the minimum number of such GSA trees generated is: n_1/d^{h1-h2}. By adding the nodes belonging to the sub-trees of the two types (member sub-tree, GSA sub-trees), we take into account the members that are also roots of GSA sub-trees twice ($n_1/d^{h1\text{-}h2}$ such members in total) in our computations. Therefore, the total number of keys stored in each GSA tree if we subtract the tree root becomes: $(d\times d^{h1\text{-}h2}–1)/(d\text{-}1)\text{-}1= d\times(d^{h1\text{-}h2}\text{-}1)/(d\text{-}1)$. Thus, the total storage cost for the GSC, if we add everything together becomes: N_C $K’$ =

$$\left\lceil \left(\frac{d^{h_2+1}-1}{d-1} + n_1 \frac{d}{d-1} \cdot \frac{d^{h_1-h_2}-1}{d^{h_1-h_2}} \right) K' \right\rceil .$$

Initial Computation Cost:

GSC: $\left\lceil N_C \ C'_r + \left(n_1 + n_1 n_2\right) C'_{PE} + \left(N_C - 1\right) C'_{SE} \right\rceil$

GSA: $\left\lceil C'_{PD} + h_1 C'_{SD} \right\rceil$ **Member:** $\left\lceil C'_{PD} + h_2 C'_{SD} \right\rceil$

Derivation of the computation cost for GSC

Initially, N_C new keys will be generated by the GSC with cost C_r (random generator function) for each. The GSC first sends asymmetrically encrypted the secret keys of all the leaves (those associated with both members and GSAs: in total $n_1+n_1n_2$). Then, the GSC sequentially sends the secret keys of all the remaining internal nodes symmetrically encrypted with the member secret keys of members previously sent, with the same method we have already discussed. Here, the keys of the parent nodes of the members that are roots to GSA sub-trees must be communicated both to the members and to the members’ offspring as well. Therefore, the GSC symmetrically encrypts the key of such a node: (*a*) *d* times for its offspring that are members, and (*b*) $d\times z$ times, where *z* is the number of GSA sub-trees that exist below these members. As discussed, the number of symmetric encryptions performed by the GSC regarding the members only becomes:

$d\times(n_1\ n_2\ \text{-}1) / (d\text{-}1) = d\times(d^{h2}\text{-}1) / (d\text{-}1) = ((d^{h2+1}\text{-}1) / (d\text{-}1))\text{-}1$, (5.1).

For the GSA sub-trees, the GSC acts similarly. Although the key of the physical root of a GSA sub-tree (namely a member’s key) is neither encrypted nor communicated to the nodes of GSA sub-trees, the key of the parent of such a member plays the role of the physical root: it is encrypted and broadcast *d* times to the associated GSA sub-tree. In analogy to CBT, the number of symmetric encryptions done by the GSC for each GSA sub-tree becomes:

$d\times(n_1\text{-}1)/(d\text{-}1)$ or $d\times(d^{h1\text{-}h2}\text{-}1)/(d\text{-}1)$, (5.2).

We repeat here that the number of GSA sub-trees is: $n_1 / d^{h1\text{-}h2}$, (5.3).

The overall number of symmetric encryptions performed by the GSC is obtained from the expression: (5.1)+(5.2)×(5.3), and becomes: N_C-1.

Each GSA asymmetrically decrypts its private key sent by the GSC, and then symmetrically decrypts all the keys of nodes contained in its path to the root. The same is the case with each member. The difference is that the GSA decrypts a path of height h_1 whereas the member decrypts a path of height $h_{2.}$

Initial Communication Cost: $\left\lceil \left\{ N_C + n_1\left(n_2+1\right) - 1 \right\} K' \right\rceil$

The communication cost is the number of bits that have to be communicated from GSC to all the GSAs and members so that the proper keys are distributed. For the distribution of keys asymmetrically to all the leaves, the GSC uses: $(n_1 + n_1\ n_2)\ K$ bits. For the symmetric distribution of all the required keys to all the interior nodes the GSC uses: $(N_C\text{-}1)\,K$ bits.

Average Operating Cost for GSC:

$$\frac{1}{2}\{n_1 p_1((h_1+1)C'_r + C'_{PE} + 2h_1 C'_{SE}) + n_1(n_2-1)p_2((h_2+1)C'_r + C'_{PE} + 2h_2 C'_{SE})\} +$$

$$\frac{1}{2}\{n_1 p_1(h_1 C'_r + (d-1)h_1 C'_{SE}) + n_1(n_2-1)p_2(h_2 C'_r + (d-1)h_2 C'_{SE})\}$$

Both GSAs and members are viewed as “members” from the perspective of the GSC. In the case of a new GSA or member inserted into the group the GSC is responsible for creating and distributing the appropriate encrypted keys both for the case of a GSA addition (first term) and for the case of a member addition (second term). These two terms are derived exactly as we have analyzed in the CBT protocol for member addition. The third and fourth term represent the computation cost of a GSC for deleting a GSA and a member from the tree respectively. Similarly, the terms are derived exactly as we have analyzed in the CBT protocol for the case of member deletion. The total number of members that remain in 2HH after a member deletion is: $n_1(n_2\text{-}1)p_2$.

Average Operating Cost for GSAs: $n_1 p_1 (C'_{PD} + h_1 C'_{SD})$

A newly inserted GSA asymmetrically decrypts the private key sent by the GSC, and then symmetrically decrypts the private keys associated with its path to the root sent by the GSC as well. The cost for this operation is: $\frac{1}{2} n_1 p_1 (C'_{PD} + h_1 C'_{SD})$.When a GSA is deleted, the remaining GSAs perform the same actions as before, and the resulting cost for this operation becomes: $\frac{1}{2} n_1 p_1 (C'_{PD} + h_1 C'_{SD})$.When a member is added or deleted, no actions have to be taken by any other GSA, since the GSAs and members possess different group keys, and additionally, the GSC is responsible for executing the steady state related operations in this scheme.

Average Operating Cost for Members: $n_1 (n_2 - 1) p_2 (C'_{PD} + h_2 C'_{SD})$

Similar is the course of thought for deriving the formula for the addition or deletion of a member. Again, under this configuration, the addition or deletion of a GSA does not affect members. The GSC handles the GSAs and the members as two different kinds of members.

Average Communication Cost:

$$\frac{1}{2}\{n_1 p_1 (2h_1 + 1)K' + n_1 (n_2 - 1) p_2 (2h_2 + 1)K'\} + \frac{1}{2}\{n_1 p_1 (d-1) h_1 K' + n_1 (n_2 - 1) p_2 (d-1) h_2 K'\}$$

The first and the second terms are the communication cost for adding a new GSA or member to the tree respectively. The third and fourth terms are the communication cost for deleting a

GSA or member from the tree. Both GSAs and members are viewed by the GSC as two different kinds of members.

5.3.2. Second Scheme: GSC to GSAs: CBT, GSA to members (clusters): GKMP

The resulting tree of CBT is balanced with $h_1 = \log_d n_1$,

Storage Cost:

GSC: $\left\lceil \frac{(dn_1-1)}{d-1}K' \right\rceil$ **GSA:** $\left\lceil K+n_2K+\left(h_1+1\right)K' \right\rceil$ **Members:** $\left\lceil 3K \right\rceil$

The GSC stores the secret keys of all tree nodes. Any GSA stores the following keying material: (*a*) the keys associated with its path to the root (GSC), (*b*) the secret key used to communicates with the GSC individually, (*c*) the n_2 secret keys it communicates to all members in its cluster individually, and (*d*) the cluster session key it communicates to all members in its cluster. Members store their individual private keys, shared only with their leading GSA. These keys are distributed (and asymmetrically encrypted) by their leading GSA, and are used to symmetrically encrypt the session key. In addition, the members store the session shared with their leading GSA, and the session key shared with the GSC.

GSC Computational Cost: $\left\lceil \frac{(dn_1-1)}{d-1}C'_r+\frac{d\left(n_1-1\right)}{d-1}C'_{SE}+n_1C'_{PE} \right\rceil$

GSA Computational Cost: $\left\lceil C'_{PD}+h_1C'_{SD}+\left(n_2+1\right)C_r+n_2\left(C_{PE}+C_{SE}\right) \right\rceil$

Member Computational Cost: $\left\lceil C_{PD}+C_{SD} \right\rceil$

Overall Communication Cost: $\left\lceil \left(n_1+\frac{d\left(n_1-1\right)}{d-1}\right)K'+2\,n_1n_2K \right\rceil$

The GSC generates (dn_1-1)/(d-1) new keys via the random generator with cost C_r for each key. The GSC is responsible only for the distribution of keys to GSAs, and the term for the computational cost is directly derived from the CBT cost formulas. The GSA asymmetrically decrypts the secret key sent by the GSC and then decrypts the keys associated with its path up to the root symmetrically sent by the GSC as well. The GSA generates the secret keys of all its members (*n_2*), asymmetrically encrypts them and distributes them to the members, with associated costs indicated by the corresponding GKMP formulas. The member first asymmetrically decrypts its individual secret key, and then symmetrically decrypts the session key, sent by the GSA. By summing the total number of keys sent to all the GSAs from the GSC and the total number of keys sent to all members from their leader GSAs, we obtain the overall communication cost.

Average Operating Cost for GSC:

$$\frac{1}{2}\{n_1 p_1((h_1+1)C'_r + C'_{PE} + 2h_1 C'_{SE})\} + \frac{1}{2}\{n_1 p_1(h_1 C'_r + (d-1)h_1 C'_{SE})\}$$

The first term represents the cost incurred at the GSC for adding a new GSA to the tree of GSAs. In this scheme, the GSC is not involved with the addition and deletion of members. These operations are up to the GSA to handle. The cost formulae follow the CBT scheme. The second term represents the cost incurred at the GSC for deleting a GSA from the tree.

Average Operating Cost for GSA:

$$\frac{1}{2}\{n_1 p_1(C'_{PD} + h_1 C'_{SD}) + n_1(n_2 - 1)p_2(2C_r + C_{PE} + 2C_{SE})\} +$$
$$\frac{1}{2}\{n_1 p_1((n_2+1)C_r + n_2 C_{PE} + n_2 C_{SE}) + n_1(n_2 - 1)p_2(C_r + (n_2 - 1)C_{SE})\}$$

The first and the second term represent the cost incurred at the GSA for adding a new GSA to the CBT or a new member to the given GSA (exactly as indicated by the GKMP cost formulae) respectively. The third and fourth terms represent the cost incurred at the GSA for deleting another GSA (exactly as indicated by the CBT cost formulae) or a member from the given GSA (exactly as indicated by the GKMP cost formulae) respectively. When a GSA is to be deleted (third term) the GSC is responsible for removing it and for sending the proper updated keys to the corresponding nodes in the tree. However, another GSA must recruit the members of the evicted GSA. So, either there is a free GSA to which no cluster has been assigned yet, or a member of the cluster is randomly selected as GSA. The new GSA recruits all n_2 members and builds a new cluster by generating and distributing the proper keys as indicated in GKMP (third term).

Average Operating Cost for Members:

$$\frac{1}{2}\{ n_1 (n_2 -1) p_2 (C_{PD} + C_{SD})\} + \frac{1}{2}\{ n_1 p_1 (C_{PD} + C_{SD}) + n_1 (n_2 -1) p_2 C_{SD} \}$$

The addition term represents the cost incurred at any member for adding another member to the cluster (according to the GKMP formulae). The deletion term represents the cost incurred at any member for: (*a*) deleting a GSA from the tree, and (*b*) evicting a member from the same cluster. When a GSA is evicted then its members join in another cluster. The first part of the deletion term represents the cost for a member joining in another cluster if its GSA is deleted, an event with probability p_1. The last deletion term represents the computations done by the rest of members when a member is deleted from its group.

Average Communication Cost:

$$\frac{1}{2}\{n_1 p_1 (2h_1 +1) K' + n_1 (n_2 -1) p_2 (3K)\} + \frac{1}{2}\{n_1 p_1 ((d-1) h_1 K' + 2 n_2 K) + n_1 (n_2 -1) p_2 (n_2 -1) K)\}$$

The average communication cost sums up the total number of keys exchanged both in the case of a GSA or member addition and in the case of a GSA or member deletion respectively. The terms of the equation are derived following exactly the cost analysis of CBT and GKMP, and this of the operating costs for the current scheme.

5.3.3. Third Scheme: GSC to GSAs: CBT, GSA to members: CBT

$h_1 = \log_{d1} n_1$, $h_2 = \log_{d2} n_2$, each tree is balanced.

d_1 = fan out of upper tree, d_2 = fan out of lower tree

In this scheme both levels in 2HH follow independently the CBT protocol (do not interact). After this observation, it is easy to derive the cost formulae of the corresponding cost functions of the third scheme.

Storage Cost:

GSC: $\left\lceil \frac{(d_1 n_1 - 1)}{d_1 - 1} K' \right\rceil$ **GSA:** $\left\lceil (h_1 + 1) K' + \frac{(d_2 n_2 - 1)}{d_2 - 1} K' \right\rceil$ **Members:** $\left\lceil \left(h_2 + 1\right) K' \right\rceil$

Since both levels of hierarchy follow CBT independently, all costs formulae are directly derived from the CBT cost analysis. A GSA stores all keys of the tree it generates for it own cluster of members and also the keys associated with its path to the root in the tree of GSAs.

Computational Cost:

GSC: $\left\lceil \frac{(d_1 n_1 - 1)}{d_1 - 1} C'_r + \frac{d_1 \left(n_1 - 1\right)}{d_1 - 1} C'_{SE} + n_1 C'_{PE} \right\rceil$ **Members:** $\left\lceil C'_{PD} + h_2 C'_{SD} \right\rceil$

GSA: $\left\lceil C'_{PD} + h_1 C'_{SD} + \frac{(d_2 n_2 - 1)}{d_2 - 1} C'_r + \frac{d_2 \left(n_2 - 1\right)}{d_2 - 1} C'_{SE} + n_2 C'_{PE} \right\rceil$

A GSA decrypts the proper keys sent by the GSC and simultaneously generates and encrypts the keys that will be sent to its own cluster (tree) of members, exactly as indicated by the CBT cost formulae.

Communication Cost: $\left\lceil \left(n_1 + \frac{d_1 \left(n_1 - 1\right)}{d_1 - 1} \right) K' + n_1 \left[n_2 + \frac{d_2 \left(n_2 - 1\right)}{d_2 - 1} \right] K' \right\rceil$

Average Operating Cost for GSC:

$$\frac{1}{2}\{ n_1 p_1 ((h_1 + 1) C'_r + C'_{PE} + 2 h_1 C'_{SE})\} + \frac{1}{2}\{ n_1 p_1 (h_1 C'_r + (d_1 - 1) h_1 C'_{SE})\}$$

In this scheme, the GSC interacts only with the GSAs, as indicated by the CBT protocol. The first term is the operating cost incurred at the GSC for adding a new GSA and the second (deletion) term represents the operating cost incurred at the GSC for deleting a GSA from the tree, exactly as indicated by the CBT cost formulae.

Average Operating Cost for GSA:

$$\frac{1}{2}\{n_1 p_1 (C'_{PD} + h_1 C'_{SD}) + n_1 (n_2 - 1) p_2 ((h_2 + 1) C'_r + C'_{PE} + 2 h_2 C'_{SE})\}$$

$$\frac{1}{2}\{n_1 p_1 (\frac{d_2 n_2 - 1}{d_2 - 1} C'_r + n_2 C'_{PE} + \frac{d_2 (n_2 - 1)}{d_2 - 1} C'_{SE}) + n_1 (n_2 - 1) p_2 (h_2 C'_r + (d_2 - 1) h_2 C'_{SE})\}$$

The first term represents the cost incurred to the GSA due to: (*a*) the addition of a new GSA (as indicated by CBT), and (*b*) the addition of a member (as indicated by CBT). The second term represents the cost incurred to the GSA due to: (*a*) the deletion of a GSA, and (*b*) the deletion of a member. When a GSA is evicted then its members join in the cluster of another GSA. The first part of the deletion term represents the cost incurred at the GSA that recruits in its cluster all the members of the evicted GSA (an event with probability p_1).

Average Operating Cost for Member:

$$\frac{1}{2}\{n_1 (n_2 - 1) p_2 (C'_{PD} + h_2 C'_{SD})\} + \frac{1}{2}\{n_1 p_1 (C'_{PD} + h_2 C'_{SD}) + n_1 (n_2 - 1) p_2 h_2 C'_{SD})\}$$

The addition term (first) represents the cost incurred to members for the addition of another member to a cluster. The deletion term (second) represents the cost incurred to members when: (*a*) their GSA is deleted and they are assigned to a new GSA, and (*b*) a member is evicted from their cluster.

Average Communication Cost:

$$\frac{1}{2}\{n_1 p_1 (2h_1 + 1) K' + n_1 (n_2 - 1) p_2 (2h_2 + 1) K'\} + \frac{1}{2}\{n_1 p_1 ((d_1 - 1) h_1 K' + n_2 K' + d_2 \frac{n_2 - 1}{d - 1} K') + n_1 (n_2 - 1) p_2 (d_2 - 1) h_2 K'\}$$

The addition term (first) represents the keys required for: (*a*) the addition of a GSA to the CBT of GSAs, and (*b*) the addition of member to a cluster. The deletion term represents the keys required for: (*a*) the deletion of a GSA from the CBT of GSAs, (*b*) the transferring the member cluster of the evicted GSA to another GSA and generating and distributing new keys for them (as indicated by CBT), and (*c*) the deletion of a member from a given cluster.

5.3.4. Fourth Scheme: GSC to GSAs: Single CBT, Single Group of Members

p: same for all members.

Now, $h = -\log p$, $p = 1/(n_1 \times n_2)$, balanced tree:

Storage Cost:

GSC: $\left\lceil \frac{d^{h+1}-1}{d-1} \right\rceil$ **GSA:** $\left\lceil (h+1)K' \right\rceil$ **Members:** $\left\lceil (h+1)K' \right\rceil$

The GSC stores the keys of all nodes in the single balanced CBT tree. Any non-empty tree leaf is associated with either a GSA or a member. The GSAs and members store these keys associated with nodes in the tree that belong to their path towards the root.

Computational Cost:

GSC: $\left\lceil \frac{d^{h+1}}{d-1} C'_r + \left(n_1 + n_1 n_2\right) C'_{PE} + \left(\frac{d^{h+1}}{d-1} - 1\right) C'_{SE} \right\rceil$

GSA: $\left\lceil G'_{PD} + hC'_{SD} \right\rceil$ **Members:** $\left\lceil G'_{PD} + hC'_{SD} \right\rceil$

Communication Cost: $\left\lceil \left\{ \frac{d^{h+1}-1}{d-1} + n_1\left(n_2+1\right) - 1 \right\} K' \right\rceil$

All cost formulae directly follow the cost functions of CBT.

Average Operating Cost for GSC:

$$\frac{1}{2}\{n_1 p((h+1)C'_r + C'_{PE} + 2hC'_{SE}) + n_1(n_2-1)p((h+1)C'_r + C'_{PE} + 2hC'_{SE})\} +$$
$$\frac{1}{2}\{n_1 p(hC'_r + (d-1)hC'_{SE}) + n_1(n_2-1)p(hC'_r + (d-1)hC'_{SE})\}$$

The addition term (first) represents the cost incurred to the GSC due to: (*a*) the addition of a new GSA (as indicated by CBT), and (*b*) the addition of a member (as indicated by CBT). The deletion term (second) represents the cost incurred to the GSC due to: (*a*) the deletion of a GSA, and (*b*) the deletion of a member. The GSC handles views both GSAs and members exactly the same way, handles them as simple members, and places both at the leaves of a balanced tree with height *h*. Hence, the eviction of a GSA does not affect any members.

Average Operating Cost for GSA: $n_1 p(C'_{PD} + hC'_{SD})$

Since the GSAs are placed in the leaves of the trees with no hierarchy under them, they act similarly to the simple members. In the case of a GSA addition, the new GSA asymmetrically decrypts the private key sent by the GSC, and then symmetrically decrypts the private keys associated with its path to the root sent by the GSC as well. The cost for this operation is: $\frac{1}{2} n_1 p_1 (C'_{PD} + h_1 C'_{SD})$. The remaining GSAs decrypt any number z of secret keys, where $1 \leq z \leq h$. When a GSA is deleted, the remaining GSAs perform the same actions as before, and the resulting maximum cost incurred by a GSA for this operation becomes:

$\frac{1}{2} n_1 p_1 (C'_{PD} + h_1 C'_{SD})$.

Average Operating Cost for Member: $n_1 (n_2 - 1)\, p (C'_{PD} + hC'_{SD})$

The corresponding cost is derived similarly to the average operating cost for the GSAs.

Average Communication Cost:

$$\frac{1}{2}\{n_1 p(2h+1)K' + n_1(n_2-1)p(2h+1)K'\} + \frac{1}{2}\{n_1 p(d-1)hK' + n_1(n_2-1)p(d-1)hK'\}$$

5.3.5. Firth Scheme: GSC to GSAs: GKMP, GSA to members (cluster): CBT

$h_2 = \log_d n_2$ (balanced tree) $n_1\, p_1 + n_1\, (n_2 - 1)\, p_2 = 1$

Storage Cost:

GSC: $\lceil (n_1 + 1) K \rceil$ **GSA:** $\lceil 3K + K (d_2^{n_2} - 1 / d_2 - 1) \rceil$ **Members:** $\lceil (h+1) K' \rceil$

The storage cost formulae are derived in a straight-forward fashion, following the corresponding costs of GKMP for the group of GSAs, CBT for all member clusters, and merging both in the case of GSAs (use GKMP to store keying material regarding the group of GSAs, and use CBT to store keying material regarding its own cluster).

Computational Cost

GSC: $\lceil (n_1 + 1) C_r + n_1 (C_{PE} + C_{SE}) \rceil$ **Members:** $\lceil C_{PD}' + hC_{SD}' \rceil$

GSA: $\left\lceil (C_{PD}+C_{SD})+\frac{(dn_2-1)}{d-1}C_r{}'+n_2C'_{PE}+\frac{d(n_2-1)}{d-1}C_{SE}{}' \right\rceil$

The computational cost formulae are derived in a straight-forward fashion, following the corresponding costs of GKMP for the group of GSAs, CBT for all member clusters, and merging both in the case of GSAs (use GKMP to compute the keying material of the group of GSAs, and CBT to compute the keying material of its member cluster).

Communication Cost: $\left\lceil 2n_1K+(n_1n_2+n_1\frac{d(n_2-1)}{d-1})K' \right\rceil$

The communication cost is derived from the number of the keys sent initially from the GSC to GSAs (GKMP) and these sent from each GSA to their cluster members (CBT).

Average Operating Cost for GSC:

$$\left\lceil \frac{1}{2}\{n_1p_1(2C_r+C_{PE}+2C_{SE})\}+\frac{1}{2}\{n_1p_1(C_r+(n_1-1)C_{SE})\} \right\rceil$$

The addition term represents the cost incurred at the GSC for adding a new GSA to the tree of GSC. The deletion term is the cost of deleting a GSA from the tree. The costs are derived from the analysis of GKMP.

Average Operating Cost for GSA:

$$\frac{1}{2}\{n_1p_1(C_{PD}+C_{SD})+n_1(n_2-1)p_2((h+1)C'_r+C'_{PE}+2hC'_{SE})\}+$$

$$\frac{1}{2}\{n_1p_1(\frac{(dn_2-1)}{d-1}C'_r+n_2C'_{PE}+\frac{d(n_2-1)}{d-1}C'_{SE})+n_1(n_2-1)p_2(hC'_r+(d-1)hC'_{SE})\}$$

The addition term (first) represents the costs incurred to a GSA for adding another GSA (indicated by GKMP) or a member (indicated by CBT) to the scheme. The deletion terms (second) represent the cost for deleting a GSA or a member. The members that were left without GSA need to find a new leader, so the third term is the cost for connecting the new GSA to these members under a tree scheme.

Average Operating Cost for Member:

$$\frac{1}{2}\{n_1(n_2-1)p_2(C'_{PD}+h_2C'_{SD})\}+\frac{1}{2}\{n_1p_1(C'_{PD}+h_2C'_{SD})+n_1(n_2-1)p_2h_2\,2C'_{SD})\}$$

The addition term (first) represents the costs incurred at a member for adding a new member to the scheme (indicated by CBT). The deletion term (second) represent the cost incurred for members for deleting a GSA from a cluster and assigning all members of that cluster to another GSA, or deleting a member (indicated by CBT).

Average Communication Cost:

$$\frac{1}{2}\{n_1 p_1 3K + n_1 (n_2 - 1) p_2 (2h_2 + 1) K'\} + \frac{1}{2}\{(n_1 p_1 (n_1 - 1) K + n_2 K' + d \frac{n_2 - 1}{d - 1} K') + n_1 (n_2 - 1) p_2 (d - 1) h_2 K'\}$$

The addition term (first) represents the communication cost for: (*a*) adding a new GSA into the GKMP group, and (*b*) adding a member into the CBT group of members. The deletion term (second) represents the cost for: (*a*) deleting a GSA from the GKMP group of GSAs and also establishing a new CBT cluster between a new GSA and the members of the deleted GSA, and (*b*) deleting a member from CBT cluster of a GSA.

5.3.6. Sixth Scheme: GSC to GSAs: GKMP, GSA to members: GKMP

Storage Cost:

GSC: $\lceil (n_1 + 1) K \rceil$ **GSA:** $\lceil 3K + n_2 K \rceil$ **Members:** $\lceil 2K \rceil$

All formulae are straight-forward, following the corresponding costs of GKMP for the group of GSAs and for all member clusters, and merging both in the case of GSAs.

Computational Cost:

GSC: $\lceil (n_1 + 1) C_r + n_1 (C_{PE} + C_{SE}) \rceil$

GSA: $\lceil C_{PD} + C_{SD} + (n_2 + 1) C_r + n_2 (C_{PE} + C_{SE}) \rceil$ **Members:** $\lceil C_{PD} + C_{SD} \rceil$

Communication Cost: $\lceil 2n_1 K + 2n_1 n_2 K \rceil$

Average Operating Cost for GSC:

$$\frac{1}{2}\{n_1 p_1 (2C_r + C_{PE} + 2C_{SE})\} + \frac{1}{2}\{n_1 p_1 (C_r + (n_1 - 1) C_{SE})\}$$

The addition term (first) represents the cost for adding a new GSA to the group of GSAs (as designated by GKMP). The deletion term (second) represents the cost for deleting a GSA from the GKMP group of GSAs.

Average Operating Cost for GSA:

$$\frac{1}{2}\{n_1 p_1 (C_{PD} + C_{SD}) + n_1 (n_2 - 1) p_2 (2C_r + C_{PE} + 2C_{SE})\} +$$
$$\frac{1}{2}\{n_1 p_1 ((n_2 + 1)C_r + n_2 (C_{PE} + C_{SE})) + n_1 (n_2 - 1) p_2 (C_r + (n_2 - 1)C_{SE})\}$$

The addition term (first) represents the cost for: (*a*) the addition of a GSA to the GKMP group of GSAs, and (*b*) the addition of a member to the GKMP cluster of its GSA leader. The deletion term (second) represents the cost for: (*a*) the deletion of a GSA from the GKMP group of GSAs, the assignment of the members of the deleted GSA to a new GSA, and the formation of a new secure cluster between them (indicated by GKMP), and (*b*) the deletion of a member for the GKMP cluster of a GSA.

Average Operating Cost for Member:

$$\frac{1}{2}\{n_1 (n_2 - 1) p_2 (C_{PD} + C_{SD})\} + \frac{1}{2}\{n_1 p_1 (C_{PD} + C_{SD}) + n_1 (n_2 - 1) p_2 C_{SD}\}$$

Average Communication Cost:

$$\frac{1}{2}\{n_1 p_1 3K + n_1 (n_2 - 1) p_2 3K\} + \frac{1}{2}\{(n_1 p_1 (n_1 - 1)K + n_1 p_1 2 n_2 K) + n_1 (n_2 - 1) p_2 (n_2 - 1)K\}$$

5.3.7. Other Schemes

Until now, we have considered only two popular KM protocols, GKMP and CBT, and we have merged them together in various combinations in 2HH. GKMP provides lower storage cost but higher communication overhead than CBT. CBT, on the other hand, achieves substantially lower communication overhead, particularly whenever a component is deleted, at the expense of storage and computational cost. The aim of our analytical work is to model and analytically evaluate these combinations, to find out which improves which metrics of interest, and which exploits the advantages of the two individual protocols most efficiently. Obviously, we can incorporate many more KM protocols to the 2HH scheme. The most important factors that determine the best such combination are the individual performance of each of the applied KM protocols, the number of entities in the two levels of 2HH (n_1, n_2), and their mobility frequencies (p_1, p_2). Also, some protocols may scale better than others. In 2HH, the GSC is assumed to be robust and trusted. Thus, for the upper level, centralized schemes are encouraged. This is not always the case with the GSAs however. Depending on the network specifications, centralized or contributory schemes may be used for the lower level. The most robust protocols tolerate higher mobility rates in the network without breaking down as often as the rest of protocols do. Another issue is how the overall performance of the hybrid scheme is affected when two or more protocols overlap. For example, when two different protocols are applied to the two levels, the GSA computation and communication costs are calculated from the overlap of these two different protocols. The resulting cost formulae may not be predictable any more, and may significantly modify, namely either accentuate or amplify, certain characteristics of the individual protocols applied. Hence, to minimize these costs we must consider not only how the protocols individually operate but also how they interact when combined (i.e. in the case of GSA).

Example: one protocol may present high computation cost for the GSC and low for members, whereas another protocol may present low computation cost for the GSC and high for members. If we combine these two protocols so that the second one is applied to the 1st level and the first one to the 2nd level, we end up with higher cost for the GSA, in comparison to what we would have achieved had we combined them the opposite way. However, the second combination would result in higher cost for the GSC. As for the total communication cost (it depends on all three components: GSC, GSAs and simple members), we must conduct a performance evaluation to decide on the best combination given the parameter values.

By taking all these issues into account we will be able to “extract” these combinations that are best for the 2HH scheme, in the sense that they result in the optimization or substantial improvement of one or more of the metrics of interest. The most efficient protocols are OFT, TGDH, CBT, MOT, ELK, GKMP. The existence of a GSC ensures that centralized schemes can be applicable for the 1st level of the 2HH scheme at least. These protocols have all been modeled and analytically evaluated in Chapter 2. We stress here again the following facts:

(*a*) For the non DH-based centralized protocols (i.e. CBT, GKMP, OFT, ELK), we maintain the same RSA-based PKI model we developed in Chapters 2 and 5, and use the expressions previously calculated for the following RSA-cryptographic functions: C_r, C_{PE}, C_{SE}, C_{PD}, C_{SD}.

(*b*) The DH-based protocols integrated in 2HH (i.e. GDH.2, TGDH, MOT) have better performance than RSA-based. In addition, the random key generator required for keys to be used either in symmetric encryption or for the DHKEs (C_{rr} is the corresponding cost function) is much cheaper than this used for the RSA PKI (C_r is the corresponding cost function). This is so because the RSA random number generator must compute on the side two extra large primes of given length. Further details can be found in the appendix of Chapter 2.

We integrated all the discussed protocols in various combinations in 2HH, modeled them, analyzed them and derived the corresponding cost formulae, similarly to the first six schemes we have demonstrated. Since the method is exactly the same, we skip the detailed analytical evaluation of the remaining combinations. Rather, we directly proceed to illustrate the results of the comparative performance evaluation, including most of the combinations used:

1. Single Key Tree (CBT), two Member Groups

2. GSC-GSAs CBT, clusters GKMP

3. GSC-GSAs CBT, clusters CBT

4. Single CBT, single Members Group

5. GSC-GSAs GKMP, clusters CBT

6. GSC-GSAs GKMP, clusters GKMP

7. GSC-GSAs OFT, cluster {OFT, ELK, CBT, GDH.2, MOT, TGDH, GKMP}

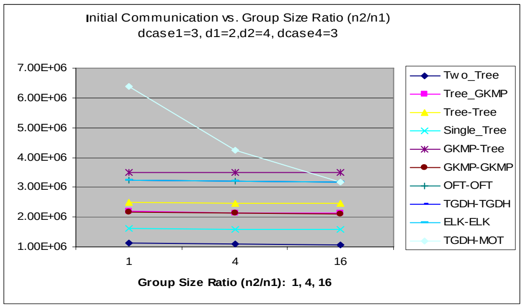
8. GSC-GSAs TGDH, cluster {OFT, ELK, CBT, GDH.2, MOT, TGDH, GKMP}

9. GSC-GSAs CBT, cluster {OFT, ELK, GDH.2, MOT, TGDH}

5.3.8. Comparative Performance Evaluation and Graphic Results

In what follows, we are graphically illustrating the most significant results of our analytical evaluations of the discussed combinations. We measure the computation and communication metrics for all entities (GSC, GSAs, members), for initial and steady state, in terms of a growing group size rate (n_2 / n_1), or in terms of a growing mobility rate (p_2 /p_1). Different combinations in 2HH prevail for different metrics, and for different group size and mobility rates. We integrated into 2HH a large variety of feasible combinations. In addition, a large portion of the sampling space has been examined. Given these two facts, we claim that the following holds: given the specifications of our network and of 2HH, i.e. group size (n_1, n_2) in both levels, or group size ratio (n_2 / n_1), mobility in both levels (p_1, p_2), or mobility ratio (p_2 / p_1), our results directly and automatically provide the best combinations for one or multiple metrics of interest, for one or multiple specifications fixed (varying degree of freedom). We illustrate the initial and steady state costs of the 2HH scheme, when varying either the group size ratio or the mobility ratio respectively, while keeping the remaining parameters constant (which take different fixed values at a time, so that the sampling space is more thoroughly tested). We note here that the trees constructed by the following protocols: OFT, ELK, TGDH, and MOT, are binary by default (d = 2).

Initial Costs



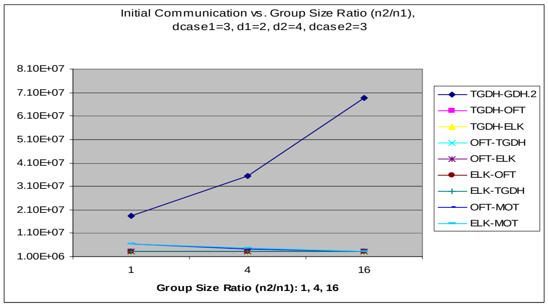


Figure 5.5. Initial Communication Cost vs. Group Size Ratio (n_2/n_1): 1, 4, 16. Single CBT, Two CBTs and ELK-TGDH reduce OH the most. Most protocols do not seem to be very sensitive to a growing Group Size Ratio, except for TGDH-MOT and TGDH-GDH.2 (that also present the worst performance). The mobility ratio is constant; the following values have been used to evaluate the OH in terms of a variable group size ratio: 2, 5, 10, 15, 20, 25.

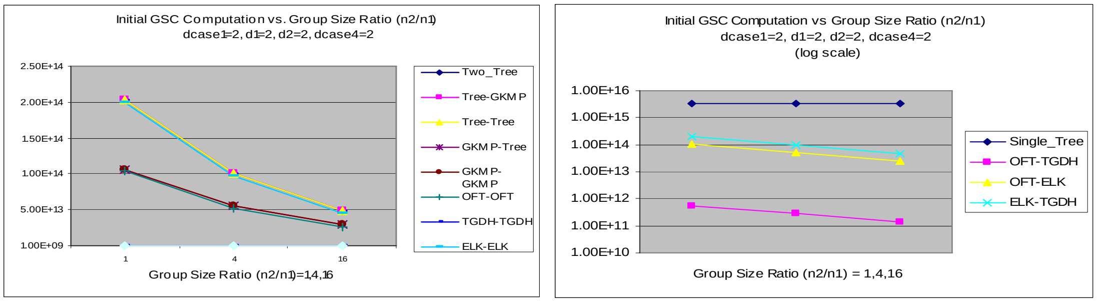


Figure 5.6. Initial GSC Computation Cost vs. Group Size Ratio (n_2/n_1): 1, 4, 16, for varying number in the tree offspring d (dcase1 (Two_Tree(CBT)) = 2, d1 = 2, d2 = 2 (Tree(CBT)-GKMP, Tree(CBT)-Tree(CBT), GKMP-Tree(CBT)), dcase4 = 2 (Single_Tree(CBT))). The combinations that reduce OH the most are: TGDH-ELK, TGDH-OFT followed by OFT-OFT, ELK-ELK, GKMP-GKMP. Single CBT again presents the worst performance and is not sensitive to Group Size Ratio. The mobility ratio is constant; the following constant values have been used to evaluate the OH in terms of a variable group size ratio: 2, 5, 10, 15, 20, 25.

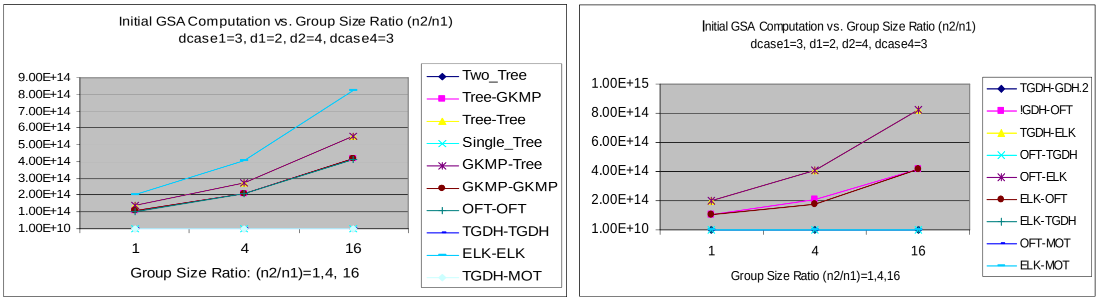


Figure 5.7. Initial GSA Computation Cost vs. Group Size Ratio (n_2 / n_1): 1, 4, 16, for varying number in the tree offspring d (dcase1 = 3, d1 = 2, d2 = 4, dcase4 = 3). The combinations that reduce OH the most are: TGDH-MOT, OFT-MOT, ELK-MOT followed by GKMP-GKMP, OFT-OFT. Combinations that use ELK in the 2^{nd} level result in the worst performance. The mobility ratio is kept constant; the following constant values have been used to evaluate the OH in terms of a variable group size ratio: 2, 5, 10, 15, 20, 25.

Operating Costs

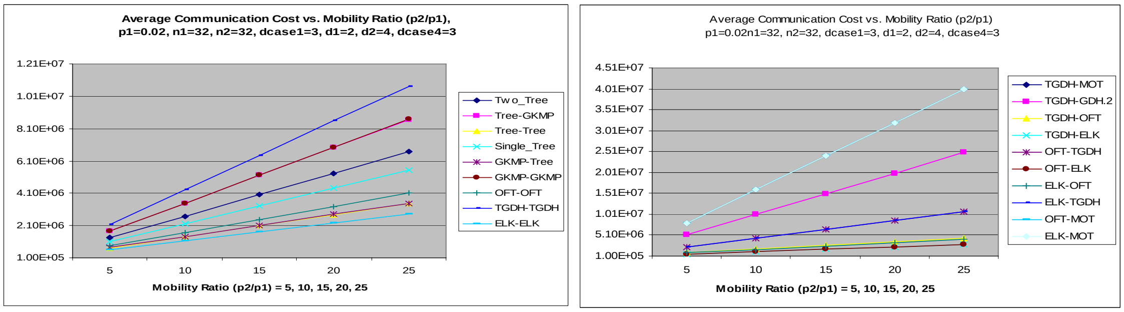


Figure 5.8. Average Communication Cost vs. Mobility Ratio (p_2 /p_1): 2, 5, 10, 15, 20, 25. The s that reduce the corresponding OH the most are: those that include OFT and/or ELK in either level, followed by GKMP-GKMP, and CBT-GKMP. Combinations that use TGDH in any level, Tree (CBT)-GKMP, Tree(CBT) –Tree(CBT) result in the worst performance. It appears that all costs are sensitive to the varying mobility ratios. The Group Size Ratio is kept constant; the following constant values were used to evaluate the OH in terms of a variable Mobility Ratio: (1, 4, 16), and n_1=(32, 16, 8), n_2 =(32, 64, 128).

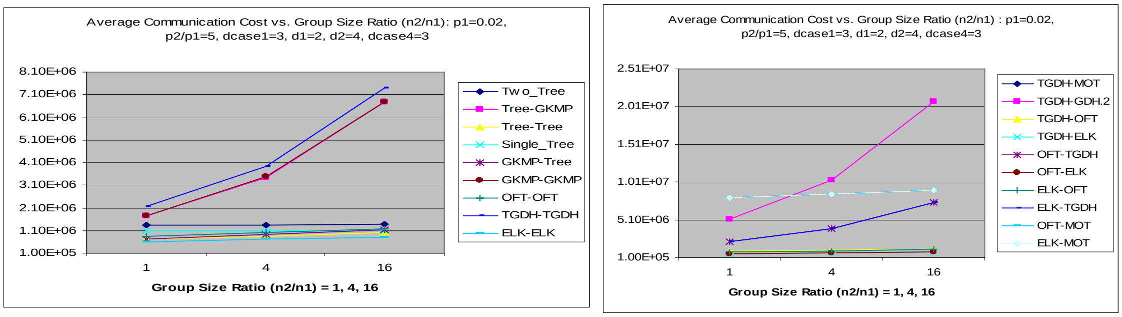


Figure 5.9. Average Communication Cost vs. Group Size Ratio (n_2 / n_1): 1, 4, 16, for varying number in the tree offspring d (dcase1 = 2, d1 = 3, d2 = 4, dcase4 = 3). The combinations that reduce OH the most are: those that use OFT or ELK in the 2nd level of 2HH, followed by GKMP-GKMP, and Tree (CBT)-GKMP. Combinations that use TGDH in either level, Tree (CBT)-GKMP, Tree (CBT)-Tree (CBT) result in the worst performance. The costs of almost all combinations are sensitive to a varying Group Size Ratio. The mobility ratio is constant; the following constant values were used to evaluate the OH in terms of a variable group size ratio: 2, 5, 10, 15, 20, 25. In these graphs in particular, we used p_1 = 0.02, and (p_2 / p_1) = 5.

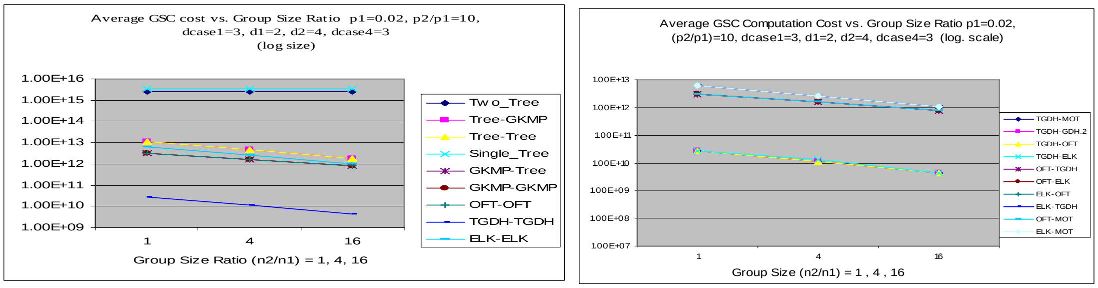


Figure 5.10. Average GSC Operating Cost vs. Group Size Ratio (n_2 / n_1): 1, 4, 16, for varying number in the tree offspring d (dcase1 =3, d1 = 2, d2 = 4, dcase4 = 3). The combinations that reduce the corresponding OH the most are: the use of TGDH in any level followed by OFT-OFT, ELK-ELK, GKMP-GKMP, and Tree (CBT) -GKMP. Single-Tree and TwoTree result in the worst performance. Most costs appear to be sensitive to the Group Size Ratio. The mobility ratio is kept constant; the following constant values have been used to evaluate the OH in terms of a variable group size ratio: 2, 5, 10, 15, 20, 25. In these graphs in particular, we used p_1 = 0.02, and (p_2 / p_1) = 10.

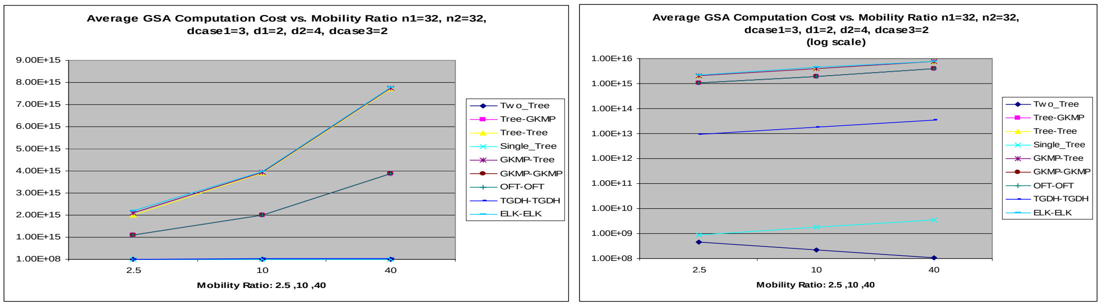


Figure 5.11. Average GSA Operating Cost vs. Mobility Ratio (p_2 /p_1): 2, 5, 10, 15, 20, 25, for varying number in the tree offspring d (dcase1 = 3, d1 = 2, d2 = 4, dcase4 = 2). The combinations that reduce the corresponding OH the most are: Single_Tree (CBT), Two_Tree (CBT), followed by combinations that use TGDH in any level. Combinations that use ELK in any level of 2HH result in the worst performance. Almost all the discussed costs are sensitive to a varying mobility ratio. The Group Size Ratio is kept constant; the following constant values were used to evaluate the OH in terms of a variable Mobility Ratio: (1, 4, 16), and n_1 = (32, 16, 8), n_2 = (32, 64, 128). In these particular graphs we have used the following values regarding the most significant metrics: n_1 =32, n_2 = 32.

5.3.9. Discussion - Conclusions

The 2HH demonstrates adaptability to the network topology and the various network constraints. A combination of protocols may be used to improve one or more metrics of interest. Given the network parameters, such as: users mobility frequencies, user characteristics (processing capabilities, bandwidth, battery life), overall number of users, and number of users of certain categories, we can now automatically select the most suitable key distribution protocols for both its levels and combine them to a very efficient hybrid scheme.

For our evaluation, we set the basic parameters to take values as indicated below:

$n_2/n_1 = t$: (1, 4, 16) and n_1=(32, 16, 8), n_2 =(32, 64, 128), $n_1 * n_2$ = 1024.

$p_2/p_1 = y$: (5, 10, 15, 20, 25) or {(2.5, 10, 40), and p_1 = (0.08, 0.04, 0.02), p_2 = (0.2, 0.4, 0.8)}.

The Operating Costs *are very sensitive* to ratios *t* and *y*. Furthermore, all metric functions are sensitive to the length of keying elements (*K*) and consequently to the following cryptographic functions: C_{PE} /C_{PD}, C_{SE} /C_{SD}, C_r, and C_{rr}. Naturally, metrics *are sensitive* to the parameters: p_1, p_2,, n_1, and n_2 alone. Certain metrics are *sensitive* to the tree configurations, namely to the selection in the number of descendants of internal nodes: d_1, and d_2 (where CBT is used according to the original definition, namely: CBT-CBT, GKMP-CBT, CBT-CKMP), $dcase_1$ (Single_Tree (CBT)), $dcase_4$ (Two_Tree(CBT)), respectively. The overall performance of 2HH is much better when no contributory protocols are applied (which is expected). However, certain combinations of non-contributory with contributory schemes reduce the performance more than certain cases that combine two non-contributory protocols do. Different combinations of protocols may prevail for different costs depending on the partial performance of each protocol regarding the particular cost. Among undeniable winners for the vast majority of cases are combinations that use OFT and/or TGDH protocols with other efficient protocols (regarding a given metric).

Example: The first six schemes combine *CBT* and *GKMP* protocols in different ways:

For all costs related to 2HH, except for the Average Communication Cost, GKMP-GKMP and Tree (CBT)-GKMP are the prevailing schemes. For the Average Communication Costs however clearly Tree (CBT)-Tree (CBT) and GKMP-Tree (CBT) are the schemes that result in the best performance. For GSC Initial and Operative Costs, *Single_Tree (CBT)* presents the worst performance (since it is overloaded with the tasks of the GSAs as well), but presents much better performance along with *Two_Tree (CBT)* with respect to the Communication Costs (the communication between the GSC and the GSAs is spared).

5.4. The effect of Mobility on the Performance of the 2HH scheme

In this section, we are investigating the effects of mobility on the behavior and performance of the discussed KM schemes, and mainly on the 2HH scheme. We want to better understand the requirements imposed by various mobility models. These requirements will then be met with the improved design of KM. Our main objectives are the following: (*a*) the design of an analytical and simulations tool that operates as follows: Given the network and mobility specifications and parameters, and given the specific metrics we want to improve, it “automatically” designates the most efficient combination(s) of KM protocols to be integrated to 2HH, so that the desired overhead is significantly reduced, (*b*) we want to measure exact variations in the re-keying (mainly) overhead due to mobility of individual protocols, or combinations of protocols integrated into 2HH, (*c*) we want to improve on our analytical results gathered, and make them more accurate, by substituting the arbitrarily attributed probabilities of members’ evictions and additions with real measurements, and (*d*) the mobility model, speed of group members, transmission range, and other mobility parameters, can be another input to the theoretical and simulation tool that take as input various network parameters (i.e. number of members at each level, mobility profile of nodes, transmission range, bandwidth, etc.) and determines which combinations of protocols are most suitable for the two levels of hierarchy.

The approach we follow is to **reflect various mobility patterns into our analytical cost models and then design KM schemes that offer the best trade-offs with respect to these requirements**. In our simulations, we consider the most crucial mobility and network parameters for all network nodes (speed, mobility model followed by group members, transmission range of group members, network and cluster sizes). We want to determine how the probability of members being evicted from their cluster varies, with varying the mobility and network parameters. We are then able to test the effects of mobility on practical metrics: e.g. number of communication exchanges, number of broadcasts, number of rounds, amount of storage and computation, running time, etc., required for re-keying, or initialization.

We are dealing with mobile nodes that move ad-hoc, but in certain applications there is the tendency for nodes to move in groups, presenting obvious similarities in their mobility patterns. Given this observation, we have considered the following mobility models for our analysis and simulations, details of which can be found in [57]: (*a*) the **Random Way Point Mobility Model (RWPM),** and **the Smooth Random Way Point Mobility Model (SRWP),** for nodes that move completely ad-hoc, (*b*) the **Reference Point Group Mobility Model (RPGM)** or the **Reference Velocity Group Mobility Model (RVGM)**, for nodes that appear to be moving in groups (similar mobility pattern). RPGM describes the group membership by its physical displacement from the group reference center. In RVGM, the similarity in members' movements is considered as the most important characteristic of a mobility group.

5.4.1. Overview and Analysis

Notation: First, we define a number of probabilities and parameters as follows:

S: the number of clusters (subgroups) of a broad KM group.

J: the probability for any member to become evicted from its own subgroup.

m: the rate of new incoming members into the group (from the outside world).

λ: the probability that an arbitrary member already evicted from its own subgroup joins another subgroup.

Method: We model a generic 2HH over various networks of different size and configurations as follows: our network consists of clusters of equal size that represent the subgroups formed at the 1st or 2nd level of the hierarchy of 2HH. We fix a number of network and mobility parameters (i.e. members' transmission ranges, members' average speed, etc.). The mobility of all group members is characterized with the same mobility model, every time the simulation starts over. We use three different mobility models to test the performance and behavior of our schemes. The duration of each simulation run is approximately 500 seconds. During this period, we measure and then average the frequency of members being disconnected from their subgroup.

The probability *J* can then be defined as follows: $J = \frac{\#evicted\_members\_per\_cluster}{\#members\_per\_cluster}$. This probability in our simulations is totally defined by the mobility patterns of members, and depends on the mobility model used, the members' velocity, transmission ranges, etc. However, in reality, evictions of members occur for reasons unrelated to mobility as well. For instance, for security reasons, whenever suspicious behavior is observed, or members have expired credentials, they are evicted from their subgroup. In addition, members are subject to battery drainage, or may become faulty at any time. Hence, the "total eviction" of a member partially depends on unprecedented factors that are orthogonal to its mobility. This probability is not reflected in our current simulations. Our simulation set up is such that all members that are evicted from their subgroup eventually join in another subgroup. So, in our settings no member becomes totally evicted from the network. This is so, because we assume that whenever a member reaches the borders of our grid shaped topology, it bounces back in the opposite direction, and gets back into the group. Of course, for the duration of the simulation, an evicted member may not actually join another subgroup even though it moves within the borders of our topology. It can happen that it moves in areas that are unreachable from any member of any subgroup. However, because of the fact that the simulation is run for a short period we do not handle this case as total eviction from the group. Rather, we assume that eventually, the member joins an arbitrary subgroup. In reality however, there exists a probability that an evicted member from any subgroup does not join any other subgroup. This probability partially depends on members' mobility and can be captured with simulations, but also depends on other factors, i.e. security, protocol policies, waiting period, etc. that may be different for each case, and introduce another parameter (w.r.t. security, policy), that may also constitute an input for 2HH. In this work, we have denoted this probability with (1-λ), and we view this as a free parameter that can be provided as input to 2HH along with the network and protocol specifications. Similar is the case with the rate *m* of new members that join the secure group. This rate does not involve the mobility pattern and profiles of members and cannot be realistically captured in simulations. Again, we introduce this probability as a free parameter. This parameter should be provided as input to 2HH along with the network and protocol specifications.

Considering all the above, our analytical models are formulated by integrating the discussed (free or measured) probabilities and rates into the previously "arbitrarily fixed" probabilities of evicted or added members into the clusters of the secure group. Hence, the probability of a member being evicted from its subgroup is *J*, while the overall probability of an arbitrary member being evicted from its subgroup and totally disconnected from the secure group becomes: $J\times(1\text{-}\lambda)$), and the overall probability of an arbitrary evicted member to join a given cluster becomes: $J\times\lambda\times\frac{1}{(S-1)}$ of a member joining a new cluster. Similarly, the probability of a member's addition to a given cluster becomes: ($m\times\frac{1}{S}+J\times\lambda\times\frac{1}{(S-1)}$). As we have explained, the parameters and/or rates: S, m, and λ, are orthogonal to mobility. The purpose of our simulations is to capture the probability J of a member being evicted from its subgroup, as an effect of the mobility of members. As we can see, the measurable parameter J, along with parameters: λ, m, and S, that are introduced as input to 2HH, fully determine membership changes during initial or steady state. Hence, by measuring the mobility-dependent parameter J, for varying mobility specifications, we gain insight as to how mobility affects the performance of KM schemes, because we can determine the frequency of disruptions (in the case of initialization) and/or membership changes at steady state. Since, the KM handles these events either by starting over the KM protocol (in the case of a contributory scheme), or by invoking the re-keying operation (steady state), we can accurately determine the resulting overall extra communication, computation, and storage overhead produced by the participants. These costs are easily calculated: we only need to substitute in the analytical evaluations provided in section 5.3, the existing entity addition/ eviction probabilities with the newly defined probabilities (these of inserting or deleting a GSA or a simple member), that were previously arbitrarily set to ½ p_i.

In summary, our contribution in this section is to expand the 2HH model so that it includes mobility parameters in its input, and accurately reflect all these to the analytical formulae that will determine the best set of combinations to be used for the existing network configuration. Towards this end, we conducted simulations to measure the only parameter - J - that reflects the mobility profiles of members. We obtain an analytical and simulation tool that offers the best trade-offs and improved KM design w.r.t. our requirements.

Below, we summarize the basic results (p_{EV}, p_{ADD}) that can be integrated in all analytical formulae derived throughout section 5.3:

- Avg. probability of a member being evicted from its subgroup: p_{EV} =J.

- Avg. probability of a member becoming disconnected from the group: $p_{OUT} = J \times (1-\lambda))$.

- Avg. probability of any evicted member to join another cluster: $p_{MOVE} = J\times \lambda \times \frac{1}{(S-1)}$.

- Avg. probability of a member's addition to any cluster: $p_{ADD}= (m\times \frac{1}{S} + J\times \lambda \times \frac{1}{(S-1)})$.

5.4.2 Simulation Set Up

We conducted simulations over various network configurations (various network graphs) with varying network size (500, 800, 1000 nodes), consisting of a varying number of clusters (16, 32, 64, 128). We measured the impact of a number of different mobility models on the connectivity of entities to their cluster. We assume that the clusters represent subgroups of a group KM protocol (or subgroups of the 1st and 2nd level in 2HH).

We measured the number of members that disconnect from their subgroups during a given window of time, and we average the results. These results were averaged over a period of 500 seconds of simulation time, for different network configurations and parameters. We report the average frequency of member evictions in the whole network (i.e. from all subgroups) per second. In our simulations, we used RWPM, RPGM, and SRWPM mobility models, and we ran them with three different member speeds: 5m/s, 10m/s, 20m/s. We assume that the transmission range of all members, denoted as T_x, is 70m, selected at random. For our setting we use a square grid topology, and we assume that members that reach the borders "bounce back" into the network. Our results are very indicative of how different mobility models affect the performance of KM, in a given network. Changing the network configuration and even a few of the network specifications (i.e. T_X) result in different values of the discussed probabilities, and probably in different outcome of the comparative evaluation of the mobility models used in our simulations. Hence, we cannot generalize the results of our simulation for all inputs, and "hard-wire" the measurements of the probabilities in our analytical evaluations. Another option could be to provide direct formulae that compute these probabilities analytically; in our previous analytical results the notion of mobility is restricted in arbitrarily selected probabilities of nodes joining or leaving particular groups. Refining and extending the existing analytical models to better reflect how mobility affects the various key KM schemes is a second option. To this end, we started by basing the guidelines of our analytical investigation on the work of McDonald et al. [58], where a novel framework for dynamically organizing mobile nodes in wireless ad hoc networks into clusters is presented. Also, Bettstetter et al. [59], and Kurose et al. [60], introduced a simple analytical mobility model for MANETs based on Poisson distribution. However, in this work they are inevitably making some very simplistic assumptions. In addition, since it is generally accepted that nodes within a MANET cannot be moving completely ad hoc (on the contrary, most nodes have an intended destination towards which they are heading), a more complex distribution than Poisson should be considered for modeling the movements of nodes in MANETs. Although their analysis is useful, and their results could be used in our existing analytical schemes to reflect the effect of mobility, even for verification purposes, we believe that our simulations yield more accurate results: we are not making as many simplifying assumptions and in addition, we are testing multiple mobility models of different nature (ad-hoc and group models). Hence, using simulations, we also compare the effect of different mobility models on the performance of KM protocols, which cannot be achieved with the existing analytical work. McDonald considers the random assumption as optimal as well, and uses initially the Poisson distribution, but provides useful insight on: how mobility affects clusters, the probability that a node is clustered, how to compute cluster survival time, and other useful results. These results could be integrated in our existing analytical formulae, for estimating the probabilities produced, or for verification purposes. Extending the work of McDonald, Bettstetter, and Towsley, to include more complex mobility models for MANETs is a very broad and challenging problem of its own. To our knowledge, there is not a single study that provides a unified, spherical, analytical approach.

Although the simulation results cannot be generalized for all networks, all we need to do every time we are provided with different input, is to run the same simulations for the parameters provided, measure the probability *J* and process the results as shown in our example, incorporate the value of *J* in the probabilities formulated in 5.1, and finally, directly substitute these probabilities in the corresponding analytical formulae derived in 5.3. This methodology can be transparent to a simple user. The user provides the necessary parameter input and specifications to a “black box” that executes all the discussed simulations, analytical evaluations, and determines the most efficient set of combinations for 2HH.

5.4.3 Simulation Results

In Table 5.2, we illustrate some of the indicative results of the average number of member evictions due to mobility, from all network members, per second, per mobility model, per speed, in networks of varying size (500, 800, 1000 nodes). From these values, parameter *J* can be directly computed. For example, for RPGM, for a members speed of 20m/s, and for cluster size of 64 members, in a network of 500 nodes, the average number of evictions per second obtained is: 5.143. We obtain *J* by normalizing this value, hence: *J* = (5.143/500) = 0.010286. As seen from this set of simulations, RPGM results in the lowest number of member evictions. This is expected, since this model captures the behavior of members that move as a group and not ad-hoc, and hence the probability of members getting disconnected from their cluster is reduced. The difference in the values of *J* for different parameters is reflected in the extra overhead required for re-keyings.

Mobility Model: RWPM model **Network Size:** 500 nodes

Speed **Clusters**	**5m/s**	**10m/s**	**20m/s**
16	3.362	5.252	5.274
32	4.132	5.77	7.774
64	3.84	6.396	7.154
128	3.342	6.178	10.174

Mobility Model: RPGM model **Network Size:** 500 nodes

Speed Clusters	5m/s	10m/s	20m/s
16	1.31	1.682	2.676
32	1.66	2.018	4.006
64	1.92	2.734	5.143
128	2.216	3.918	6.664

Mobility Model: SRWPM (smooth random way point) **Network Size:** 500 nodes

Speed Clusters	5m/s	10m/s	20m/s
16	3.712	7.37	14.826
32	4.14	6.72	15.908
64	3.48	8.69	18.116
128	4.368	8.928	15.978

Table 5.2. Member Eviction Rates for 3 different mobility models and speeds, varying the cluster and network size.

The values illustrated in Table 5.2 are useful towards a realistic comparison of the initialization overhead required for the successful key establishment in either a contributory or a centralized protocol. We will use a simple arithmetic example, for our point to come across, and gain insight on the effect of mobility on the various KM protocols. Assume that each cluster in a network of 500 nodes has size of 32 nodes, and the mobility patterns of members can be best simulated by the RWPM model, and the average speed of each member is approximately 10m/s. Let the average T_X of each node be 70m. We assume that each cluster (GSAs to members) executes one of the following protocols: (*a*) CBT, (*b*) GKMP, (*c*) GDH.2. We want to examine the impact of members' disruptions due to mobility on any of these protocols for the case of initialization. The impact of such disruptions during steady state, are straightforward to compute from our analytical formulae of section 3. From table 3, we obtain the average rate of member evictions for our parameters: 5.77 members/s. The average number of clusters formed in this network is: $\left\lceil \frac{500}{32} \right\rceil$ = 16. Therefore, the average rate of member evictions per cluster is: 5.77/16 = 0.36 members/s.

Case (*a*): In GKMP, the GSA communicates to each member in its cluster two consecutive messages, hence it must send 64 packets for the successful initialization, denoted as T_{BCAST}(64). After the expiration of this period, the task of the GSA is over. It is assumed that the GSA has executed the necessary cryptographic computations beforehand. If during this period one or more member disconnect, the task of the GSA is not disrupted. The key establishment terminates successfully. The GSA engages in a re-keying operation to ensure that the property of forward secrecy is preserved. No matter how many members disconnect during the T_{BCAST}(64) period, the GSA will execute the operation of re-keying only once. Thus, in the worst case scenario, the extra overhead incurred to the GSA and members from membership changes during initialization, is either zero (no changes) or this required for a single “member eviction”. This overhead is independent of the rate of member evictions per cluster, and of the T_{BCAST}(64) period.

Case (*b*): Similar is the case with the centralized CBT scheme. The only difference is that in case re-keying is required after initialization, the overhead incurred to the GSA and simple members is different from this of GKMP. Again, this overhead is independent of the rate of member evictions per cluster as well as of the T_{BCAST} (64) period.

Case (*c*): In GDH.2 the case is very different: the failure of a member results in the re-initialization of the key establishment, as this event affects both the content of KM messages, and the sequence of exchanges. The execution that involves a cluster of 32 members requires at least 33 rounds. In each round, a member receives the proper data from its predecessor, processes it accordingly, and sends the processed message to its successor. The running time of each round can be broken down to the amount of time required for processing the KM data (T_{PROC}), and for the propagation of the processed message (T_{PROP}). We simplify the overall amount of time for the key establishment as: $T_{INIT} = 33(T_{PROC}\ (avg) + T_{PROP})$. Even if extra relay members are involved for the communication of a message to its intended recipient the following holds: $T_{PROC} >> T_{PROP}$, and we can ignore the factor T_{PROP}, since T_{PROC} may take even seconds in the case that the computational and storage capabilities of members are limited. For example, setting T_{PROC} *(avg)* = 1s, we obtain in an ideal case: T_{INIT} = 33s. Even under this optimistic scenario, the average member evictions during T_{INIT} are: 33×0.36 members = 11.88 members. That means that it is very likely that GDH.2 starts over many times, more than these indicated by the number of evicted members, since the longer it takes for the protocol to terminate, the probability for further evictions increases. This result alone, points out how inefficient contributory protocols are for highly mobile environments, and how bad they scale. One way out is to proceed to key establishment despite the evicted members, and execute a single re-keying afterwards, as is the case in centralized schemes. This option however is rarely feasible, since in most cases, the protocol stalls after members' failure. If we know how many times GDH.2 must start over it is simple to compute the amount of extra communication, computation, and storage overhead required.

Chapter 6: Elliptic Curves Cryptography (ECC) and its Impact on the Key Management Protocols Studied

6.1 Introduction

Since the invention of public key cryptography, numerous public key cryptosystems have been proposed. Each of these systems relies on a difficult mathematical problem for its security. None has been proven to be intractable. In practice, they are believed to remain intractable with the current computational technology (or computationally infeasible). The longer it takes to derive the key with the best known algorithm for a problem, the more secure a public-key cryptosystem based on that problem is. Today, three types of public keys systems, classified according to the mathematical problem on which they are based, are generally considered both secure and efficient:

1. The integer factorization systems (i.e. RSA is the best known example)
2. The discrete logarithm problem (DLP) systems (i.e. Digital Signature Algorithm (DSA), DHKE, El Gamal)
3. The elliptic curve discrete logarithm problem (ECDLP) systems (EC cryptosystems).

ECC was proposed independently by N.Koblitz [61] and V.Miller [63]. Since then, a vast amount of research has been conducted on its secure and efficient implementation. In recent years, ECC has received increased commercial acceptance as evidenced by its inclusion in standards by accredited standards organizations such as ANSI, ISO, NIST, etc. Before implementing an ECC system, several choices have to be made. These choices include selection of EC domain parameters (underlying finite field, field representation), and algorithms for field arithmetic, EC arithmetic, and protocol arithmetic. The selections can be influenced by security considerations, application platform, constraints of the particular computing environment (e.g., processing speed, code size, memory size (RAM), gate count, clock time, power consumption), and ccommunications environment (e.g., bandwidth, response time). It is difficult to decide on a single "best" set of choices. ECs can be used in public key cryptography to generate pairs of Public- Secret keys. They do not introduce new cryptographic algorithms: rather existing ones can be implemented with ECs:

1. **Key Agreement**: EC- Diffie-Hellman
2. **DH-Signature:** EC-DSA
3. **Public Encryption:** EC-El Gamal

EC groups were proposed in 1985 as a substitute for the multiplicative groups modulo *p* that are used in the classical DLP. All protocols that are based on the classical DLP can be translated to the EC setting. More precisely, we provide the following definitions for DLP, and ECDLP below:

Classical DLP version: Given elements *a*, *y* in a group *G* of order *n*, we cannot compute in a feasible amount of time an element *x* so that $y = a^x \bmod n$.

EC version of DLP (ECDLP): Let *A*, *B* be known points on an EC *E* so that $B = k \times A$, for integer *k*. We cannot compute integer *k* in a feasible amount of time.

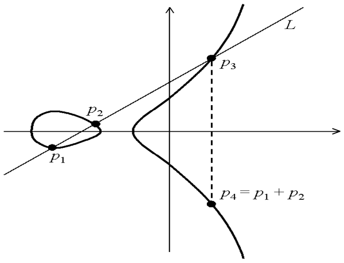


Figure 6.1. Illustration of an EC and the operation of point addition.

6.1.1 Basic Advantages

Although there are known sub-exponential, super-polynomial algorithms for breaking RSA and DH based on modular arithmetic, sub-exponential algorithms that could break ECC have not been found so far. So, there exist no good attacks on DL-ECC contrary to integers. Hence, ECC implementation is believed to be secure with much smaller key sizes: for example, a 4096-bit key size for RSA can be replaced by a 314-bit key size for EC. In general, for the same level of security, EC-based systems can be implemented with much smaller parameters, leading to significant performance advantages. Such improvements are particularly important in the wireless arena where computing power, bandwidth consumption, hardware implementations, memory, and battery life of devices are more constrained. ECs are inherently efficient in terms of computation, and they have the potential to induce substantial overhead savings when applied to our protocols, substituting RSA or DH.

All cryptographic parameters used for the evaluation of our KM protocols (i.e. C_E, C_{PE}, C_{SE}, etc.) depend on key lengths. So, ECC may improve the performance of KM protocols, by reducing the value of these parameters. In addition, shorter key lengths facilitate the routing of the encrypted data. The IP packets have a size limit usually lower than 4Kbits. It is not desirable to break the KEK to packets. That makes the actual key vulnerable (insecure), and the corresponding operation more inefficient. A substantial lower key size, as provided by the use of ECs, totally eliminates this problem.

6.1.2. Performance Discussion

The RSA algorithm only uses one type of calculation - exponentiation. The nature of the exponent determines the speed of the operation. For signing and decryption, the exponent (private key) is large, so the calculation is quite slow. For verification and encryption, however, the exponent can be quite small, i.e. 3. Therefore, any analysis of RSA speed consists of slow times for signing and decrypting and much faster speeds for verification and encryption. In El Gamal systems completely different mathematical operations are used, so signing and decryption times do not equal each other, nor do those of verifying and encrypting: signing is faster than verification, decryption is faster than encryption. The difference between public and private key operation times are trivial compared to this difference in RSA. Even though EC public key operations are a bit slower than those in RSA, the sum of the public and private key times in ECC is much smaller than in RSA. For example, at the 163-bit ECC/1024-bit RSA security level, an EC exponentiation for ECs over arbitrary prime fields is roughly 5 to 15 times as fast as an RSA private key operation, depending on the platform and optimizations. At the 256-bit ECC/3072-bit RSA security level the ratio is increased to between 20 and 60. In [63], [64], and [65], sample timings for RSA and ECC operations on different platforms are provided, resulting from extensive experimentations. Below, we summarize the most significant results of the comparison among the three types of public key schemes.

RSA *vs.* El-Gamal *vs.* ECC:

- **Key Generation:** Most Expensive for RSA
- **Signature Generation (RSA *vs.* ECDSA):** RSA Slower (4-8 times)
- **Signature Verification (RSA *vs.* ECDSA):** RSA Faster (50 times),
- **Public Key Encryption (RSA *vs.* EC-ElGamal):** RSA faster
- **Public Key Decryption (RSA *vs.* EC-ElGamal):** RSA slower
- **ECC:** Smaller Signatures/Certificates, Less Resources

ECC Drawbacks: ECC is slower for public key operations, such as signature verification and public key encryption. Furthermore, DH KA is simpler from the perspective of fine tuning domain and network parameters. In addition, the dominant certificate authority on the internet at the present time, VeriSign, supports RSA.

In summary, each public scheme discussed has certain advantages in comparison to the rest. What the most appropriate scheme to use is depends on the network and protocol parameters, on the specific application or protocol and their associated parameters, and on our own requirements. Still, in the growing wireless industry, the numerous advantages of ECC over the rest of public schemes have made it an attractive security alternative.

6.2. Method and Objectives

In this chapter we investigate the efficiency of the implementation of KM protocols, and in particular Octopus-based schemes, under ECC. The protocols that we examine are based on RSA or DH public key systems, but provide the flexibility for the use of other cryptographic schemes. Our aim is to exploit further all these protocols, gain an insight on how all the metrics of interest are affected, and examine if further improvement in certain metrics can be achieved by the substitution of RSA or DH with ECC. Our findings can also be applied to the Hybrid Hierarchical scheme we proposed in the previous chapter.

In particular, we substitute all DH-based operations with ECDH operations in the three Octopus-based schemes, and we substitute the RSA operations with EC-El Gamal in the centralized OFT scheme, used for the initial comparative evaluation of Octopus schemes. We analyze the complexity of the EC-based schemes and derive analytical formulae for the overall communication, computation, storage costs of the EC-versions of (O), MOT, and OFT. Then, we conduct a comparative evaluation of the EC-equivalents of these schemes *vs.* the original. Our analytical evaluation achieves the following: given the KM protocol, operation to be executed (initial key establishment, re-keying) and the required parameters regarding the original cryptosystem and its EC-equivalent, we can now easily estimate the approximate gain or loss in any of the metrics of interest. As an auxiliary step, prior to the discussed analysis, we worked to derive approximate estimates of the bit-complexity of all cryptographic parameters involved in the analysis of our KM protocols with ECs. Although results on parameters such as latency, energy costs, etc., associated with all KM operations exists for various platforms, through extensive experimentations, we are interested in a more generic analysis, disassociated as much as possible from given platforms. This can be achieved if we focus on the bit-complexity of the EC-based cryptographic parameters.

6.3. Evaluation of EC-based cryptographic parameters (overview, evaluation)

6.3.1. Overview of EC definitions and properties

ECC requires the use of two types of mathematics: EC point arithmetic and underlying finite field arithmetic. An EC is a set of points specified by two variables that are elements over a field F_q. A field is a set of elements with two custom defined arithmetic operations, usually addition and multiplication. Most of the computation for ECC takes place at the finite field level. The most common choices for the underlying finite field are:

- F_{2^m} , also known as *characteristic two* or *even* (containing 2^m elements, where m>1),
- F_p , also known as *integers modulo p, odd,* or *odd prime* (containing p elements).

In software environments, the use of F_{2^m} offers significant performance advantages over F_p . The revised standard FIPS 186-2, revised by NIST, has 10 recommended finite fields, 5 prime and 5 binary. For each of the binary fields one random and one *Koblitz* EC (defined as: $y^2 + xy = x^3 + ax^2 + 1$, a = {0, 1}) was selected. In the analysis that follows, we randomly selected the *Koblitz* curve over $F_{2^{163}}$. We choose a polynomial basis representation for the field elements: each element can be viewed as a polynomial whose coefficients are either 0 or 1. In polynomial multiplication, a product polynomial of degree greater than 162 may be produced. It is reduced by the **irreducible polynomial** or **reduction polynomial** $p(x) = x^{163} + x^7 + x^6 + x^3 + 1$, as specified in the revised standard.

When implementing an EC cryptosystem, one is required to compute: $Q = P \times a$, where points Q, $P \in E$, and $a \in Z^*$. This operation is called ***scalar multiplication* (*SM*)**. Let the cost of the *SM* operation be denoted as C_{SM}. The operation of *scalar multiplication* (*SM*) in EC-based systems is equivalent to the *modular exponentiation* (*ME*) operation in DH-based or RSA-based systems. Therefore, in order to evaluate the cost (bit-complexity) of the EC-based schemes, we need to evaluate the parameter C_{SM} first. The operation of *SM* (i.e. $Q=P\times a$) is computed by a sequence of ***Point Doublings*** **(*PD*)** and ***Point Additions*** **(*PA*)** of the EC point (i.e. P). Let $a = (a_m\, a_{m-1} \ldots a_0)$ be the binary representation of the multiplier a where the most significant bit of a is 1. Q can be computed by the following algorithm, known as "r*epeated double and add*", proposed in [66]:

$Q \leftarrow P$, For $i = (m\text{-}1)$ to 0 $\{Q \leftarrow Q+Q;$ if $(a_i = =1)$ $Q \leftarrow Q+P;$ $\}$, Return (Q).

For a randomly selected multiplier a with binary representation one needs m doubling steps and an average of (m/2) adding steps. Hence, $C_{SM} \approx (m/2)\, C_{PA} + m\, C_{PD}$.

Other binary representations may be used as well. One among them is the signed digit representation. A particularly useful representation is the non-adjacent form (NAF) which has the property that no two consecutive coefficients a_i are non-zero. NAF(a) has the fewest non-zero coefficients of any signed digit representation of a. The "*repeated double and add*" algorithm under a NAF representation of the multiplier a can be efficiently computed using a slight modification of the previous algorithm [67]. The average density of non-zero coefficients among all NAFs of length m is approximately m/3. It follows that the expected running time of the algorithm proposed in [69] is $\mathrm{T}(m) = (m/3)A+ mD$. It is suggested in [67] that the running time of the previous algorithm can be further improved if using a window method which processes w digits of a at a time. The running time of this algorithm becomes: $\mathrm{T}(m) = (m+1)D +(2^{w-2}-1) + m/(w+1))A$. For m=163, the running time is minimized when w = 4, and $\mathrm{T}(m) = (m+1)D +(3+m/5)A$. Even further improvements are proposed in [67], at the expense of other costs however, i.e. storage (when values are pre-computed), and under specifically adjusted software. These improvements efficient as they are, add significantly to the complexity of implementation. For our analysis, we use the two simplest methods for computing C_{SM} as computed.

The next step is to compute the bit complexities for the *PA* and *PD* operations. Both *PA* and *PD* reduce to a series of polynomial operations. In the following figure, quoted from [67], the algorithms for the discussed operations on the EC: *E*, are illustrated.

Adding two distinct point on an EC (*PA*)	Doubling a point on an EC (*PD*)
INPUT: EC points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ OUTPUT: $R = P + Q = (x_3, y_3)$ Compute $\theta = \frac{y_2+y_1}{x_2+x_1}$ Compute $x_3 = \theta^2 + \theta + x_1 + x_2 + a$ Compute $y_3 = \theta(x_1 + x_3) + x_1 + x_3 + y_1$ Return (x_3, y_3)	INPUT: EC points $P = (x, y)$ OUTPUT: $R = P + P = (x_3, y_3)$ Compute $\theta = x + \frac{y}{x}$ Compute $x_3 = \theta^2 + \theta + a$ Compute $y_3 = x^2 + (\theta + 1)x_3$ Return (x_3, y_3)

Figure 6.2: Illustration of the Point Addition and Point Doubling operations on EC E:

Both *PA* and *PD* translate to a number of basic finite field operations over the irreducible polynomial *f*(*x*) as defined earlier: additions, multiplications, squarings and inversions. For the particular category of EC we are interested in, these finite field operations are straight-forward to compute and are summarized in Table 6.1.

	Additions	Multiplications	Squarings	Divisions
Point Addition	9	2	1	1
Point Doubling	4	2	2	1

Table 6.1. Finite Field Operations for Point Addition and Point Doubling on the EC: E.

The following step is to compute the bit-complexities of each of the discussed finite field operations. A considerable amount of research is dedicated to studying and improving the performance of finite field operations for all categories of ECs. Usually, the evaluation of the methods discussed is tightly bound to a specific platform, software and hardware, dedicated to a specific problem. The comparative performance evaluation is usually done with respect to the resulting timings (latency), energy costs, hardware and software performance (i.e. number of cycles for each arithmetic unit, software instructions), etc. We are interested however in the most generic analysis, disassociated as much as possible from specific platforms, with special software or hardware design. Such an analysis can be achieved by computing the bit-complexities of the finite field operations. Towards this end, we have performed an extensive study of a number of efficient known algorithms for finite field operations. Among the studied algorithms, we distinguished those that: (*a*) do not address special cases or have very special constraints, and (*b*) are not too hard to implement under the majority of platforms, software or hardware equipment. We completed some of these algorithms by deriving their bit-complexity or running time whenever it was not provided. Then, we proceeded to further filtering these algorithms, in order to keep only the most efficient and simple ones. We repeat that our main objectives are to compute the cryptographic parameters for the operations of *PA* and *PD* (i.e. C_{PA}, C_{PD}), and consequently for the operation of scalar multiplication (*SM*) (i.e. C_{SM}). With our generic analysis the number of bits translates to the number of packets, which translates to the Communication, Computation, and Energy Costs. This connection is linear only when assuming that the power level is stable (e.g. max. transmission power level), and the data rate is stable as well. However, this is highly unlikely, since the power level depends on platform, hardware, software implementation, network, transmission level, clock cycles etc. So, we cannot derive the bit complexity of these cryptographic parameters from the energy costs alone.

6.3.2. Finite Fields Operations Complexity Computation

In this section, we are deriving the bit-complexities for the following field element arithmetic operations modulo the irreducible polynomial *f*(*x*): addition, multiplication, squaring, inversion, modular reduction. Below, we are quoting the algorithms we have selected, and we are completing them by deriving the bit-complexities for each. Then, we will incorporate the results into the EC cryptographic parameters.

Notation: Let $t = \lceil \frac{m}{32} \rceil$, and let $s = 32t - m$. In software, *a*(*x*) is stored in an array of *t* 32-bit words: A = (A[*t*-1],… A[1], A[0]), where the leftmost s bits of A[*t*-1] are unused. Addition of field elements is performed bitwise, thus requiring only *t* word operations. If C =

(C[n],…,C[1],C[0]) is a vector, then C{j} denotes the truncated vector (C[n],…,C[j+1],C[j]). Let the symbol *BC* denote the bit complexity for the algorithms presented.

Modular Reduction: It is based on the observation that $x^i = x^{i-m}\, r(x) \pmod{f(x)}$, where $f(x) = x^m + r(x)$, and $i \geq m$. Let $c(x)$ be a binary polynomial of degree at most $2m$-2. The following algorithm reduces $c(x)$ modulo $f(x)$ one bit at a time, starting with the leftmost bit.

Modular Reduction (one bit at a time)

```
INPUT: binary polynomial $c(x)$ of degree at most $2m-2$
OUTPUT: $c(x) \bmod f(x)$
        Precomputation: Compute $u_k(x) = x^k r(x)$, $0 \leq k \leq 31$.
        For $i$ from $(2m-2)$ downto $m$ do
         If $c_i = 1$ then: let $j = \left\lfloor \frac{(i-m)}{32} \right\rfloor$ and $k = (i-m) - 32j$. Add $u_k(x)$ to C{$j$}.
        Return ((C[$t$-1],…, C[1], C[0])).
```

Table 6.2. Algorithm for Modular Reduction w.r.t. irreducible polynomial f(x) over F_{2^m} .

Complexity of Modular Reduction: In average c_i =1 half of the times (i.e. $\frac{m}{2}$), hence step 3 is executed for $\frac{m}{2}$ times. Under the worst case scenario, let c_i =1, for $\frac{3m}{2} \geq i \geq m$, activating the longest additions (i.e. C{$\frac{m}{2}$},…, C{0}), corresponding to additions starting from $\frac{3m}{2}$ bits to $2m$ bits, successively increased by 1.

Then, $BC\ (MR) < \sum_{\frac{3}{2}m}^{2m} 1 = \sum_{1}^{2m} 1 - \sum_{1}^{\frac{3}{2}m} 1 = m(2m+1) - \frac{3m}{4}\left(\frac{3m}{2}+1\right) = \frac{7m^2}{8} + \frac{m}{4}$ (worst case).

Similarly, $BC\ (MR) > \sum_{m}^{\frac{3}{2}m} 1 = \sum_{1}^{\frac{3}{2}m} 1 - \sum_{1}^{m} 1 = \frac{3m}{4}\left(\frac{3m}{2}+1\right) - \frac{m}{2}(m+1) = \frac{5m^2}{8} + \frac{m}{4}$ (best case).

Also, $BC\ (MR) = \frac{3m^2}{4} + \frac{m}{4}$ (average case).

Squaring: Squaring is a linear operation in $F_{2^{163}}$: if $a(x) = \sum_{i=0}^{m-1} a_i x^i$, then $a(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. The binary representation of $a(x)^2$ is obtained by inserting a 0 between consecutive bits of the binary representation of $a(x)$. Then, we compute $a(x)^2 \pmod{f(x)}$.

Complexity of Squaring: The difference with the previous case is that $c_i = 0$ for $\frac{3m}{4}$ times, and step 3 is executed $\frac{m}{4}$ times. The complexity is computed similarly as for the operation of modular reduction, taking into account the extra cases under which $c_i = 0$.

Therefore, $BC\ (SQ) < \frac{3m}{2} + (\frac{3m}{2}+2) + (\frac{3m}{2}+4) + \ldots. + (2m\text{-}2) + 2m$ =>

$$BC\ (SQ) < \frac{3m}{2}\ \frac{m}{4} + 2\sum_{1}^{\frac{1}{4}m} 1 = \frac{3m^2}{8} + \frac{m}{4}\ (\frac{m}{4}+1) = \frac{7m^2}{16} + \frac{m}{4}\ \text{(worst case scenario)}$$

Similarly, $BC\ (SQ) > m + (m+2) + (m+4) + \ldots. + (\frac{3m}{2}\text{-}2) + \frac{3m}{2}$ =>

$$BC\ (SQ) > m\frac{m}{4} + 2\sum_{1}^{\frac{1}{4}m} 1 = \frac{m^2}{4} + \frac{m}{4}\ (\frac{m}{4}+1) = \frac{5m^2}{16} + \frac{m}{4}\ \text{(best case scenario).}$$

Also, $BC\ (SQ) = \frac{3m^2}{8} + \frac{m}{4}$ (average case scenario).

Addition: For field elements this is trivial: the two blocks of bits are combined with the bitwise *XOR* operation.

Complexity of Addition mod *f*(*x*): The complexity of the pure addition is *m*. Then the intermediate result undergoes modular reduction, using the previously discussed algorithm. The resulting polynomial here is of degree (*m*+1) at maximum. Hence Step 3 will be applied only once at maximum, and an addition using *m* bits will be executed: $BC(A) = 2m$.

Multiplication: Among all methods proposed in [67], we select the fastest one, which is also among the simplest, in the sense that the modular reduction is performed after the polynomials multiplication. Let $a(x)$, $b(x)$, be the two polynomials to be multiplied, of degree at most (*m*-1) each. This method is based on the observation that if $b(x)x^k$ has been computed for some *k*, then $b(x)x^{k+j}$ can be easily obtained by appending *j* zero bits (or *j* zero words if we are computing $b(x)x^{32\ k+j}$) to the right of the vector representation of $b(x)x^k$..

Right to Left method for Polynomial Multiplication
INPUT: binary polynomials $a(x)$ and $b(x)$ of degree at most 2*m*-2

```
OUTPUT: $c(x) = a(x)b(x)$
            $C \leftarrow 0$.
            For $k$ from 0 to 31 do
for $j$ from 0 to ($t$-1) do:  if the $k$th bit of A[$j$] is 1 then add $B$ to C{$j$}
            if $k \neq 31$ then $B \leftarrow Bx$.
            Return ((C[$t$-1],…, C[1], C[0])).
```

Figure 6.4. Algorithm for Polynomial Multiplication.

Complexity of Polynomial Multiplication: In average a_i =1 half of the times (i.e. $\frac{m}{2}$), and thus the iterative step is executed for $\frac{m}{2}$ times. Under the worst case scenario, let a_i =1, for $\frac{m}{2} \leq i \leq m$, activating the longest additions (i.e. C{0} downto {$\frac{m}{2}$}), corresponding to additions starting from 2m downto $\frac{3m}{2}$ bits, successively decreased by 1.

$$BC\ (Mult) < \sum_{\frac{3}{2}m}^{2m} 1 = \sum_{1}^{2m} 1 - \sum_{1}^{\frac{3}{2}m} 1 = m(2m+1) – \frac{3m}{4}\ (\frac{3m}{2} +1) = \frac{7m^2}{8} + \frac{m}{4} \text{(worst case scenario).}$$

$$BC\ (Mult) > \sum_{m}^{\frac{3}{2}m} 1 = \sum_{1}^{\frac{3}{2}m} 1 - \sum_{1}^{m} 1 = \frac{3m}{4}\ (\frac{3m}{2} +1) - \frac{m}{2}(m+1) = \frac{5m^2}{8} + \frac{m}{4} \text{(best case scenario).}$$

Also, $BC\ (Mult) = \frac{3m^2}{4} + \frac{m}{4}$ (average case scenario).

Complexity of Polynomial Multiplication (mod *f*(*x*)): The polynomial $c(x)$ undergoes modular reduction according to the corresponding modular reduction algorithm. We now combine the bit complexity derived for the polynomial multiplication with this of modular reduction to derive the overall complexity of the process. The distribution of zeros and ones in the intermediate polynomial $c(x)$ does not necessarily follow this of polynomial $a(x)$, for any of the three scenarios (best, worst, and average). Therefore, we combine the complexities derived for all three cases (scenarios) of modular reduction in each individual case of polynomial multiplication. This results in the computation of lower and upper bounds for each individual scenario of modular polynomial multiplication as follows:

$BC\ (Mult\_worst)+BC(MR\_best)<BC(M\_Mult\_worst)<BC\ (Mult\_worst)+BC\ (MR\_worst)$ =>

$(\frac{7m^2}{8}+\frac{m}{4}) + (\frac{5m^2}{8}+\frac{m}{4}) < BC(M\_Mult\_worst) < (\frac{7m^2}{8}+\frac{m}{4}) + (\frac{7m^2}{8}+\frac{m}{4})$ =>

$\frac{3m^2}{2}+\frac{m}{2} < BC\ (M\_Mult\_worst) < \frac{7m^2}{4}+\frac{m}{2}$.

$BC\ (Mult\_avg) + BC\ (MR\_best) < BC\ (M\_Mult\_avg) < BC\ (Mult\_avg) + BC\ (MR\_worst)$ =>

$(\frac{3m^2}{4}+\frac{m}{4}) + (\frac{5m^2}{8}+\frac{m}{4}) < BC\ (M\_Mult\_avg) < (\frac{3m^2}{4}+\frac{m}{4}) + (\frac{7m^2}{8}+\frac{m}{4})$ =>

$\frac{11m^2}{8}+\frac{m}{2} < BC\ (M\_Mult\_avg) < \frac{13m^2}{8}+\frac{m}{2}$.

$BC\ (Mult\_best)+BC\ (MR\_best) < BC\ (M\_Mult\_best) < BC\ (Mult\_best) + BC\ (MR\_worst)$ =>

$(\frac{5m^2}{8}+\frac{m}{4}) + (\frac{5m^2}{8}+\frac{m}{4}) < BC\ (M\_Mult\_best) < (\frac{5m^2}{8}+\frac{m}{4}) + (\frac{7m^2}{8}+\frac{m}{4})$ =>

$\frac{5m^2}{4}+\frac{m}{2} < BC\ (M\_Mult\_best) < \frac{3m^2}{2}+\frac{m}{2}$.

Inversion: The fastest algorithm proposed in [67] is a variant of the Extended Euclidean Algorithm (EEA) for polynomials. The algorithm computes the inverse of a non-zero field element $a \in F_{2^m}$. The algorithm maintains the invariants $ba+df = u$, and $ca+cf = v$ for some d and e which are not explicitly computed. At each iteration, if $\deg(u) \geq \deg(v)$, then a partial division of u by v is performed by subtracting $x^j v$ from u, where $j = \deg(u) - \deg(v)$. In this way, the degree of u is decreased by at least 1, and on average by 2. Subtracting $x^j c$ from b preserves the invariants. The algorithm terminates when $\deg(u) = 0$, in which case $u = 1$ and $ba + df = 1$; hence $b = a^{-1} \bmod f(x)$.

```
Extended Euclidean Algorithm for inversion in $F_{2^m}$

INPUT: $a \in F_{2^m}$, $a \neq 0$
OUTPUT: $a^{-1}$ mod $f(x)$
        $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f.$
        While deg($u$) $\neq$ 0 do
            $j \leftarrow \deg(u) - \deg(v)$.
            If $j<0$ then: $u \leftrightarrow v, b \leftrightarrow c, j \leftarrow -j.$
            $u \leftarrow u + x^j v, b \leftarrow b + x^j c.$
        Return ($b$).
```

Figure 6.5. Algorithm for Inversion (mod f(x)), based on the Extended Euclidean Algorithm.

Complexity of Inversion (mod $f(x)$): The selected algorithm executes simultaneously the inversion and the modular reduction. The iterative step is invoked m times, since the degree of the variable u that is initiated to the polynomial a with degree m, is reduced by at least 1 in every iteration. The bit-complexity induced by this step is approximately $2m$ (two additions involving polynomials of maximum degree m). Therefore, $BC\ (M\_INV) = 2m^2$.

In Table 6.3, we summarize the complexities of the computed arithmetic operations. As a next step, we will incorporate these results in the EC cryptographic functions: C_{SM}, C_{PA}, C_{PD}.

Arithmetic Operations over F_{2^m}	Bit-Complexity
Modular Reduction	$[(\frac{5m^2}{8}+\frac{m}{4}), (\frac{7m^2}{8}+\frac{m}{4})]$
Addition (mod $f(x)$)	$2m$
Multiplication (mod $f(x)$)	$[(\frac{5m^2}{4}+\frac{m}{2}), (\frac{7m^2}{4}+\frac{m}{2})]$
Squaring (mod $f(x)$)	$[(\frac{5m^2}{16}+\frac{m}{4}), (\frac{7m^2}{16}+\frac{m}{4})]$
Inversion (mod $f(x)$)	$2m^2$

Table 6.3. Bit-Complexities of arithmetic operations (mod f(x)) over F_{2^m} .

6.3.3. Evaluation of EC-based cryptographic parameters

Combining the computations quoted in Table 6.1 and Table 6.3, we can now compute the cost of the cryptographic operations associated with ECs, i.e. C_{PA}, C_{PD}, C_{SM}. The complexities of the arithmetic operations are derived based on a binary representation of the polynomials, under this study cases. A NAF representation is expected to contribute cryptographic parameters of lower complexity. We have selected the binary representation as the algorithms described and their bit-complexities have been derived based on this representation of the polynomials involved. The costs of these parameters are computed below:

Point Addition: 9A + 2Mult + 1Sq + 1 Inv =>

$$18m + 2(\frac{5m^2}{4}+\frac{m}{2}) + (\frac{5m^2}{16}+\frac{m}{4}) + 2m^2 < C_{PA} < 18m + 2(\frac{7m^2}{4}+\frac{m}{2}) + (\frac{7m^2}{16}+\frac{m}{4}) + 2m^2 \ =>$$

$$\frac{77m^2}{16} + 19.25m < C_{PA} < \frac{95m^2}{16} + 19.25m \quad => \ C_{PA} \in [(\frac{77m^2}{16} + 19.25m), (\frac{95m^2}{16} + 19.25m)].$$

Point Doubling: 4A + 2Mult + 2Sq + 1 Inv =>

$8m + 2(\frac{5m^2}{4}+\frac{m}{2})+(\frac{5m^2}{8}+\frac{m}{2})+2m^2 < C_{PD} < 8m+2(\frac{7m^2}{4}+\frac{m}{2})+(\frac{7m^2}{8}+\frac{m}{2})+2m^2 \Rightarrow \frac{41m^2}{8}+$

$9.5m < C_{PA} < \frac{51m^2}{8}+9.5m \quad \Rightarrow \quad C_{PD} \in [(\frac{41m^2}{8}+9.5m), (\frac{51m^2}{8}+9.5m)].$

Scalar Multiplication (*BF*): = $\frac{m}{2}PA + m\,PD \Rightarrow$ **$C_{SM} = 0.5mC_{PA}+mC_{PD} \Rightarrow$**

$0.5m(\frac{77m^2}{16}+19.25m)+m[(\frac{41m^2}{8}+9.5m)<C_{SM}<0.5m(\frac{95m^2}{16}+19.25m) + m(\frac{51m^2}{8}+9.5m) \Rightarrow$

$7.5m^3+(19+\frac{1}{8})m^2<C_{SM}<9.5m^3+(19+\frac{1}{8})m^2 \Rightarrow C_{SM} \in [(7.5m^3+(19+\frac{1}{8})m^2), (9.5m^3+(19+\frac{1}{8})m^2)]$

Operation	Maximum Bit-Complexity
Scalar Multiplication	$9.5m^3 + 19.125m^2$
Point Addition	$6m^2 + 19.25m$
Point Doubling	$6.5m^2 + 9.5m$

Table 6.4. Upper Bounds for Bit-Complexities of cryptographic operations over F_{2^m} .

6.3.4. EC-DH to substitute DHKE in DH-based schemes

Now that we have computed the basic EC cryptographic parameters, we are ready to analyze the EC-DH public key scheme. ECDH is exactly similar to DH, with the only difference that a scalable multiplication (SM) is executed in the place of a modular exponentiation (ME) in the case of DH. In the following figure both algorithms of ECDH and DH are contrasted, so that their similarity becomes clear. The EC version of a DH-based KM protocol is simply produced by substituting the operation of DHKE with EC-DHKE wherever it occurs in the original DH-based protocol, or equivalently, substituting the MEs with SMs. The EC-based KM protocol versions have better communication costs as their keys are smaller.

6.3.5. EC El Gamal to substitute RSA operations in RSA-based schemes

In this section, we provide and analyze the operations that correspond to the EC-versions of KM protocols that are based on the RSA protocol for their public operations (i.e. OFT, etc.). In essence, the RSA public scheme is substituted with the El Gamal scheme. El Gamal is also used for public encryption and public decryption, as illustrated in the Table 6.5., and it is DH-based. Hence, in order to derive EC-El Gamal from the original El Gamal, all DH operations in El Gamal, or MEs, are directly substituted with EC-DHKEs or SMs, as discussed before.

Choose random a [2,m-1]	Choose random b [2,m-1]
Compute A_T=$\boldsymbol{P}$ x $\boldsymbol{a}$	Compute B_T=$\boldsymbol{P}$ x $\boldsymbol{b}$
Send $A_{T,}$, Receive B_T	Send $B_{T,}$, Receive A_T
Choose random k [2,m-1]	
Compute pair (C_1, C_2) = ($\boldsymbol{P}$x$\boldsymbol{k}$, $\boldsymbol{P_m}$+ $\boldsymbol{k}$x$\boldsymbol{B_T}$)	
Send (C_1, C_2)	Receive (C_1, C_2)
	Compute $\boldsymbol{P_m}$= C_2- **b**x$\mathbf{C_1}$

Table 6.5. Sequence of operations required for public encryption (column 1) and public decryption (column 2) in EC El Gamal, quoted from [70].

El Gamal C_{PE} Operations:	**El Gamal C_{PD} Operations**:
3 Comm/tion mssgs, 1 Plaintext Embedding $\boldsymbol{C_{PlEC}}$, 2 Rand. Number Gener/tion $\boldsymbol{C_{rr}}$, 3 Scalar Mult/tions $\boldsymbol{C_{SM}}$, 1 Points Additions $\boldsymbol{C_{PA}}$.	1 Comm/tion Mssg, 1 Rand. Number Gen/tion $\boldsymbol{C_{rr}}$, 2 Scalar Mult/tions (C_{SM}), 1 Points Additions $\boldsymbol{C_{PA}}$, 1 Plaintext Recovery $\boldsymbol{C_{ECPl}}$

Table 6.6. EC-El Gamal communication and computation operations for public encryption and decryption over F_{2^m} . Two extra operations are required: Plaintext Embedding with cost C_{PlEC}, (for the encryption), and Plaintext Recovery with cost C_{ECPl} (for the decryption).

Plaintext Embedding and Plaintext Recovery

To encrypt a plaintext *m* with ECs we need to embed *m* to a point P_m on the given EC. For that, we need a method that takes an arbitrary text and embeds it on an EC, so that it always gives a bijection between the points of the EC and the plaintext block. The method of Plaintext Recovery takes a point P_m of an EC and produces the original plaintext *m*. Before we illustrate the algorithm, we present some auxiliary notation and lemmas.

Notation: Let y = sqrt(z) denote the square root of an integer z, and p be a prime number.

Lemma 1: For p>2, every element $a \in Z_p^*$ has either 0 or 2 distinct square roots in Z_p^*.

Proof: Let x be a square root of a. Then $–x$ is also a square root of a. In addition, x and $–x$ are distinct mod p, so a has at least 2 square roots. We look to see if there are more. Assume that y is another square root of a. Then $x^2 = y^2 \Rightarrow x^2 – y^2 = 0 \Rightarrow (x-y)(x+y) = 0$. This has the 2 solutions $y = \pm x$. This lemma gives a count of how many quadratic residues there are in Z_p^*. Since every square maps to 2 distinct elements, exactly half of the elements of Z_p^* must be squares (i.e., there are $(p-1)/2$ squares). Hence, an element is square with probability (1/2).

Also, we use the following simple method [41] for finding square roots (mod p) more easily:

Square Roots (mod p): If we select p s.t. $p = 3 \pmod 4$ prime, there is a way to find the solution more easily. Let $y \in Z^*$, and $x = y^{(p+1)/4} \pmod p$. If y has a square root mod p, then sqrt(y) mod $p = \pm x$. If not, then sqrt($-y$) $= \pm x$.

Proof: According to Fermat's Theorem: $y^{p-1} = 1 \pmod p$. So, $x^4 = y^{p+1} = y^2 y^{p-1} = y^2 \pmod p$. This implies that $(x^2 – y)(x^2 + y) = 0 \pmod p \Rightarrow x^2 = \pm y \pmod p$. Exactly one of y or $–y$ is a square root (mod p), since if both are, i.e. $y = a^2 \pmod p$ and $–y = b^2 \pmod p$, then $-1 = (a/b)^2 \pmod p$, which means that 1 is a square (mod p) which is impossible when $p = 3 \pmod 4$. So, if y has a square root then $y = x^2$ and sqrt(y) $= \pm x$, else $–y$ has a square root and $–y = x^2$.

Complexity: This of a modular exponentiation of size p.

Representing Plaintext in ECs: There is no deterministic method to map a plaintext m to a point in a given EC, only fast probabilistic methods that fail with small probability $p_f = (1/2)^k$, where k is sufficiently large, so that p_f is acceptable. One of these methods is known as *Koblitz*. Below, we provide an overview and derive the corresponding bit-complexity.

Koblitz Plaintext Embedding: Assume that K is a large integer so that p_f is acceptable, and assume that $(m+1)K < p$. We want to map the plaintext m to a point in the x co-ordinate of an EC by setting $x = mK+j$, $0 \leq j \leq K$. As discussed, this happens with probability (1/2) only.

So, we will adjoin a few bits at the end of m and adjust them until we get a number x such that x^3+ax+b (mod p)) is a square.

Step 1, …*K*: For j = 0, 1, …, K-1, we compute: $x = mK+j$, z = x^3+ax+b (mod p). Then, x corresponds to a point of the following EC: E: $y^2=x^3+ax+b$ (mod p), only if z is a square and we try to calculate sqrt(z). If p =3 (mod 4), then we use the method in [41] quoted here too. If sqrt(z) = y, we take P_m = (x, y), otherwise we increment j by one and try again with the new x. We repeat this step until we find a square root or until j=K. If j=K, then we fail. Since x^3+ax+b (mod p) is a square approximately half of the time, we have $(1-(1/2)^K)$ probability of success. An indicative value for the variable K is usually 30. We assume that K repetitions at maximum are required before we obtain a square root.

Koblitz Plaintext Recovery: To recover text m from point P_m = (x, y), we set: $m = \left\lfloor \frac{x}{K} \right\rfloor$.

Bit Complexity for Koblitz Method: We assume that in practice we repeat for R =10 the discussed steps with the following corresponding complexity:

1) **Substitute *x* in *E*:** 1 ME for x^3, 1 modular multiplication for (ax), and 2 additions for z, resulting in overall complexity of: $((6\times 4\times m^2 + 4\times m) + 2\times m^2 + 2\times 2m) = 26m^2+8m$.

2) **Compute Square Root** with [41]: 1 ME with l = (m/4), 1 modular squaring, resulting in total complexity of: $(6\times(m/4)\times m^2+(m/4)\times m+(1/2)\times m^2+(1/4)\times m)= 1.5m^3+0.75m^2 + 0.25m$.

While the cost for the Plaintext Recovery via the Koblitz method is negligible, the cost for the Plaintext Embedding becomes: $C_{PlEM} = R\times(1.5m^3 + 27m^2 + 9m) = 15m^3 + 270m^2 + 90m$.

EC El-Gamal related Operations	Bit Complexity
Random Number Generator (BBS) - $C_{rr}(m)$	$4m^3/\log_2 m + 0.7m$
Plaintext Embedding (Koblitz) - C_{PlEC} (m)	$15m^3 + 270m^2 + 90m$
Plaintext Recovery (Kobitz) - C_{ECPl} (m)	-------
Public Encryption - C_{PE} (m)	$43.5m^3 + 333m^2 + 109.25m$
Public Decryption - C_{PD} (m)	$19m^3 + 44.25m^2 + 19.25m$

Table 6.7. Bit-complexities associated with the EC-El Gamal scheme. The abbreviations "*PE*" and "*PD*" correspond to: *Public Encryption* and *Public Decryption*.

6.4. Comparative Evaluation of the original *vs.* EC-versions of Octopus and OFT

After finding approximate values for all cryptographic operations involved in the KM protocols discussed, we now derive the analytical formulae for all corresponding schemes. We provide the results of our comparative performance evaluation of the original and the EC-versions of the Octopus-based and of OFT schemes. We illustrate the graphical results of this evaluation and discuss the most significant ones. The bit complexity of the operation of SM is generally higher than this of a ME with the same input. However, due to the fact that the key sizes of the EC-versions are much lower, the resulting bit-complexities (OH) of a SM in the EC-versions of Octopus schemes are much lower than these of a ME in the original Octopus schemes. Every ME is replaced by a SM in the EC-based schemes. Hence, the computation costs for the EC-based Octopus schemes are significantly lower. As far as OFT, the case is different for the computation costs. EC-*PE* (EC El-Gamal) is more expensive than RSA-*PE*, even when the input is smaller (shorter keys for ECs). It turns out that the corresponding costs of the entities (group leader) that do *PE*s are higher for the EC version of OFT. This is not the case with EC El-Gamal *PD* OH which is lower than the corresponding RSA *PD* OH. In this case, entities that perform EC *PD* (i.e. simple members) incur lower OH.

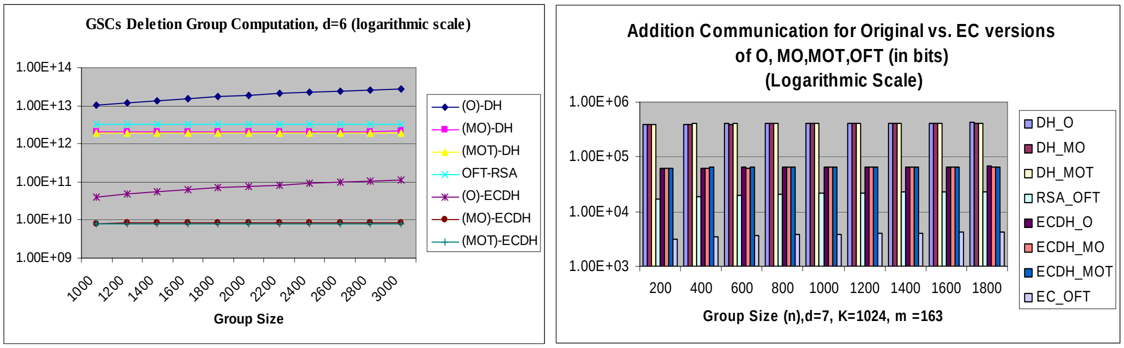


Figure 6.9. (a) Deletion Communication Cost for the subgroups for d=6 and (b) Addition Communication Cost for d=7, original vs. EC versions of Octopus and OFT (logarithmic scale). In (a), the EC-versions of Octopus schemes reduce the resulting OH significantly, exactly for the same reasons analyzed for Figure 6.1 (b). In (b) the EC-versions prevail significantly. The associations among all protocols of the same type remain unchanged. OFT prevails as far as the original schemes followed by MOT, and then EC-OFT globally prevails for both the original and EC-based versions, followed by EC-MOT.

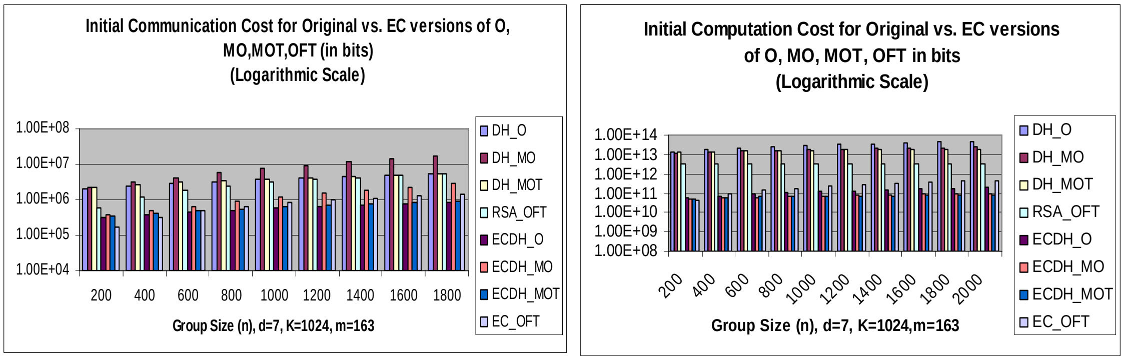


Figure 6.10. (a) Initial Communication Cost, and (b) Initial Computation Cost for all Subgroup Leaders for original vs. EC versions of Octopus schemes, d = 7, and OFT (logarithmic scale). In both cases the EC-versions of the schemes prevail significantly. The associations among all protocols of the same type remain unchanged (as computed in Chapter 2). In (a), (O) and MOT prevail in the original and EC versions. In (b) EC-MOT prevails for the EC-versions, exactly for the same reasons analyzed for Figure 6.1 (b).

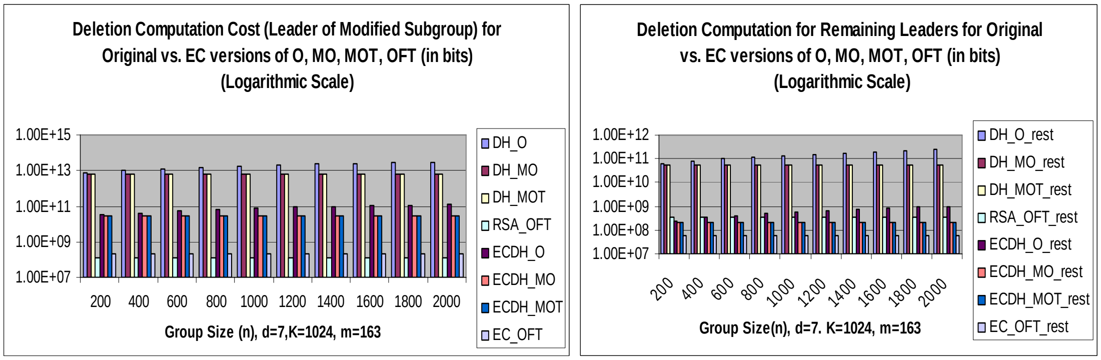


Figure 6.11. (a) Deletion Computation Cost for the Leader of the Modified Subgroup, and (b) Deletion Computation Cost for the Leaders of the Remaining Subgroups for the original vs. EC versions of Octopus and OFT schemes (logarithmic scale). In both (a) and (b) the EC-versions of Octopus schemes produce significantly lower OH and EC-MOT prevails. In the case of OFT however, in (a) the original OFT results in a better performance that the EC-OFT. This is because the EC-El Gamal PE is more expensive than its RSA equivalent. In (b), members perform PDs which are cheaper for the case of EC compared to RSA.

6.5. Conclusions and Future Directions

In this chapter, we computed the bit complexities of parameters required for the cost formulae of the EC-versions of DH-based or RSA-based KM protocols. We incorporated these parameters in the cost formulae of the KM operations of the discussed protocols and conducted a comparative performance evaluation. Indeed, ECs improve significantly the performance of DH-based schemes for all cases, and RSA ones for most cases.

For a complete, spherical investigation of the advantages of EC-based KM protocols, a very significant research direction is to extend these performance evaluations to more KM protocols with respect to additional metrics as well (i.e. generic latency, etc.). Towards this end, it would be of great value to implement the EC-equivalent Octopus-based protocols and include them in our existing Octopus-based implementation test-bed. This way, the overall latency and battery consumption can be measured experimentally and compare the corresponding metrics in the KM protocols with and without ECs.

Bibliography

[1] K. Becker, U. Wille, "Communication Complexity of Group Key Distribution," *Proc.5th ACM Conference on Computer & Communications Security*, pp. 1-6, San Francisco, CA, November 1998.

[2] N.Asokan, P. Ginzboorg, "Key-Agreement in Ad-Hoc Networks," *Computer Communications*, Vol. 23, No. 18, pp. 1627-1637, 2000.

[3] M. Steiner, G. Tsudik, M. Waidner, "Diffie-Hellman Key Distribution Extended to Groups," *3rd ACM Conference on Computer & Communication Security*, pp. 31-37 ACM Press, 1996.

[4] W.Diffie, M.Hellman,"New directions in cryptography", *IEEE Trans. on Information Theory*, 22(1976), 644-654.

[5] M. Hietalahti. "Key Establishment in Ad-Hoc Networks," *M.S. Thesis, Helsinki University of Technology, Dept. of Computer Science and Engineering*, May 2001.

[6] A.Perrig, "Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication," *Int'l Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99)*, pp. 192-202, July 1999.

[7] Y. Kim, A. Perrig, G. Tsudik, "Simple and Fault Tolerant Key Agreement for Dynamic Collaborative Groups," *Proc. 7th ACM Conference on Computer and Communication Security (CCS 2000)*, pp. 235-244.

[8] I. Ingemarsson, D.Tang, C.Wong. "A Conference Key Distribution System", *IEEE Trans. on Information Theory*, 28(5): 714-720, Sept. 1982

[9] M.Burmester, Y.Desmedt. "A Secure and Efficient Conference Key Distribution System", *Advances in Cryptology–EUROCRYPT'94, Lecture Notes in Computer Science.* Springer – Verlag, Berlin, Germany.

[10] D. McGrew, A.T. Sherman, “Key-Establishment in Large Dynamic Groups Using One-Way Function Trees,” *IEEE Trans. On Software Engineering*, Vol 29, No. 5, pp. 444-458, 2003.

[11] H. Harney, E.Harder, “Logical Tree Hierarchy Protocol,” *Internet Draft*, draft-harney-sparta-lkhp-sec-00.txt, Internet Eng. Task Force, March ‘99.

[12] H.Harney, C.Muckenhirn, “Group Key Management Protocol (GKMP) Specification/Architecture,” *RFC 2093, 2094,* Internet Engineering Task Force, July’97.

[13] A. Perrig, D.Song, D. Trygar, “ELK, a New Protocol for Efficient Large-Group Distribution,” *Proc. 2001 IEEE Symposium on Security and Privacy*, pp. 247-262, Oakland, CA, May 2001.

[14] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, G.Tsudik, “Exploring Robustness in Group Key Agreement”, *Proc. of the 21th IEEE Int’l Conference on Distr. Computing Systems*, pp. 399-408, Phoenix, AZ, April 16-19, 2001.

[15] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik, “Secure Group Comm/tion Using Robust Contributory Key Agreement”, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, no. 5, pp. 468-480, May ‘04.

[16] L.Zhou, Z.Haas, “Securing Adhoc Networks,” *IEEE Network Magazine*, vol. 13, no.6, pp. 24-30, Nov/Dec 1999.

[17] J.Kong, P.Zerfos, H.Luo, S.Lu, L.Zhang, “Providing Robust and Ubiquitous Security Support for Wireless Ad-Hoc Networks,” *Proc. 2001 IEEE Int’l Conf. on Network Protocols (ICNP 2001)*, pp. 251-260.

[18] S.Capkun, L.Buttyan, J.Hubaux, “Self-Organized Public Key Management for MANET,” *IEEE Trans. on Mobile Computing*, Vol. 2, No. 1, pp. 52-64, Jan-Mar. 2003.

[19] S.Yi, R.Kravets, “Key Management for Heterogeneous Ad hoc Wireless Networks,” University of Illinois, Urbana-Champaign, CS dept., TR#UIUCDCS-R-2001-2241, UILU-ENG-2001-1748, July 02.

[20] L.Eschenauer, V.Gligor., “A Key Management Scheme for Distributed Sensor Networks,” *Proc. 9th ACM Conference on Computer and Communication Security (CCS’02)*, pp. 41-47, Nov, 2002.

[21] H.Chan, A.Perrig, D.Song, “Random Key Predistribution Schemes for Sensor Networks,” *Proc. 2003 IEEE Symposium on Security and Privacy*, pp. 197-213, May ‘03.

[22] L.Lazos, R. Poovendran, “Energy-aware Secure Multicast Comm/tion in Ad-hoc networks Using Geographic Location Information”, *IEEE Int’l Conf. of Acoustic Speech Signal Processing (ICASSP’03),* pp. 201-204, Hong Kong, China, April, 2003.

[23] S.Zhu, S.Setia, S.Xu, S.Jajodia, “GKMPAN: An Efficient Group Re-keying Scheme for Secure Multicast in Ad-hoc Networks”, *IEEE Computer Society, MobiQuitous 2004,* pp. 45-51.

[24] C.Boyd, “On key agreement and conference key agreement”, *Procs of the Information Security and Privacy: Australasian Conference. Lecture Notes in Computer Science*, vol. 1270. Springer-Verlag, New York, 294-302, 1997.

[25] O.Rodeh, K.Birman, D.Dolev, “Optimized Group Re-key for Group Communication Systems”. *In Network and Distributed System Security*, San Diego, CA, Feb. 2000.

[26] L.Dondeti, S.Mukherjee, A.Samal, “A Distributed Group Key Management Scheme for Secure Many to Many Communication”, *TR PINTL-TR-207-99*, Dept. of Computer Science, University of Maryland.

[27] R.Barua, “Extending Joux’s Protocol to Multi-party Key Agreement”, Proc. of the 4th Int’l Conf on Cryptology in India (INDOCRYPT 2003), *Eds S.Mitra T.Johansson*, LNCS 2904, Springer-Verlag, 03.

[28] J. Katz, M.Yung, “ Scalable Protocols for Authenticated Key Exchange“, *Advances in Cryptology - EUROCRYPT’03, Springer-Verlag*, LNCS Vol 2729, pp. 110-125, Santa Barbara, USA.

[29] M.Li, R.Poovendran, C.Berenstein, "Optimization of Key Storage for Secure Communications", *Procs of the 35th Annual Conference on Information Sciences and Systems (CISS),* John Hopkings, Baltimore, Maryland, March 2001.

[30] M.Waldvogel, G.Caronni, D.Sun, N.Weiler, B.Plattner, "The VersaKey framework: Versatile group key management", *IEEE Journal of selected areas in Communications, special issue on Middleware* 17, 9, August 1999, pp.1614-1631.

[31] R.Canetti, J.Garay, G.Itkis, D.Micciancio, M.Naor, B.Pinkas, "Multicast Security: A Taxonomy and some Efficient Constructions*", Procs of the IEEE INFOCOM '99*, Vol.2., pp. 708-716, New York, N.Y, March 1999.

[32] S.Rafaeli, L.Mathy, D. Hutchinson, "EHBT: An Efficient Protocol for Group Key Management", *Procs of the 3rd Int'l Workshop on Network Group Communications, London, U.K., Nov. 2001.* Lecture Notes in Computer Science, vol.2233. Springer-Verlag, New York, pp.159-171.

[33] S.Rafaeli, D. Hutchinson, "Hydra: a Decentralized Group Key Management", *Procs of the 11th IEEE Int'l WETICE: Enterprise Security Workshop, A.Jacobs*, June 2002, Pittsburg, PA. IEEE Computer Society Press, Los Alamitos, CA.

[34] S. Mittra, "Iolus: a Framework for Secure Multicasting", *Procs of the ACM SIGCOMM, vol. 27, 4*, Sept. 1997, New York, N.Y., pp. 277-288.

[35] L.Dondeti, S.Mukherjee, A. Samal, "Scalable Secure One to Many Group Communication using Dual Encryption", *Computer and Communications. 23, 17,* Nov. 1999, pp. 1681-1701.

[36] S.Setia, S.Koussih, S.Jajodia, "Kronos: A Scalable Group Re-keying Approach for Secure Multicast", Procs of the IEEE symposium on Security and Privacy, May 2000, Oakland, CA. IEEE Computer Society Press, Los Alamitos, CA.

[37] B.DeCleene, L.Dondeti, S.Griffin, T. Hardjono, D.Kiwior, J.Kurose, D.Towsley, S.Vasudevan, C. Zhang, “Secure Group Communications for Wireless Networks”, Procs of the IEEE MILCOM 2001, June 2001.

[38] M.Striki, J.Baras "Efficient Scalable Key Agreement Protocols for Secure Multicast Communication in MANETs", *Collaborative Technologies Alliance (CTA) Symposium, College Park MD, May 2003*.

[39] M.Striki, J.Baras, G.DiCrescenzo, “Modeling Key Agreement Protocols in Multi-Hop Ad Hoc Networks”, *Proc. 2006 Int’l Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 39-45, Vancouver, CA, July 2006

[40] T.Cormen, C.Leiserson, R.Rivest. Introduction to Algorithms, *MIT Press, McGraw Hill Book Company*, March 1990

[41] W.Trappe, L.Washington, Introduction to Cryptography with Coding Theory, 2002, Prentice Hall.

[42] C.Kaufman, R.Perlman, M.Speciner, Network Security: Private Communication in a Public World, Prentice Hall, 2002

[43] R.Rivest, “The MD5 Message Digest Algorithm”, RFC 1321, 1992.

[44] L.Lazos, R.Poovendran. “Cross-Layer Design for Energy Efficient Secure Multicast Communications in Ad Hoc Networks”. *Procs of the ICC 2004 Conference, Paris, France, June 2004.*

[45] M.Striki, J.Baras. “Efficient Scalable Key Agreement Protocols for Secure Multicast Comm/tion in MANETs”. *CSHCN TR 2002*.

[46] M.Striki, J.Baras. “Key Distribution Protocols for Secure Multicast Communication Survivable in MANETs”. *Procs of the IEEE MILCOM 2003*, Boston MA, October 2003.

[47] M.Striki, J.Baras “Fault-Tolerant Hypercube extension for Efficient, Robust Communications in MANETs with Octopus Schemes”, *HyNET TR- 2005-108*.

[48] M. Harchol, T. Leighton, D. Lewin. Resource discovering in distributed networks. *Procs 15th, ACM Symp., on Principles of Distributed Computing*, May 1999, 229-237

[49] S. Kutten, D. Peleg. Deterministic Distributed Resource Discovery. *19th annual ACM SIGOPS symp. on Principles of Distributed Computing*, Portland, OR, 16-19 July 2000.

[50] R. Gallager, P. Humblet, P. Spira. "A Distributed Algorithm for Minimum-weight Spanning Trees", *ACM Transactions on Programming Languages and Systems* (TOPLAS), 5(1): 66-77, January 1983

[51] C. Cheng, I. Cimet, S. Kumar. "A Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology", *In symp. Procs' on Communication architectures and protocols*, 330-337, ACM Press, 1988

[52] S. Dolev, A. Israeli, S. Moran. Self-stabilization of Dynamic Systems assuming only Read/Write Atomicity. *Distributed Computing*, 7:3-16,1993.

[53] M.Striki, J.Baras, K.Manousakis, "A Robust Distributed TGDH-based Scheme for Secure Group Communications in MANETs", *Procs of ICC'06 Conference, Instanbul, Turkey, June 11-15, 2006*

[54] S.J.Lee,W.Su,M.Gerla,"Wireless Ad hoc Multicast Routing with Mobility Prediction", *Mobile Networks and Applications 6, pp. 351-360, 2001, Kluwer Academic Publishers.*

[55] A. Hodjat, I.Verbauwhede, "The Energy Cost of Secrets in Ad-Hoc Networks", *IEEE Circuits and Systems (CAS) workshop on Wireless Communications and Networking*, Pasadena, CA, 2002.

[56] A.M.Fisiran, R.B.Lee, "Workload Characterization of Elliptic curve Cryptography and other Network Security Algorithms for Constrained Environments", *IEEE Int'l Workshop on Workload Characterization*, WWC-5, pp. 127-137, Nov. 2002.

[57] T.Camp, J.Boleng, V.Davies, “A survey of Mobility Models for ad-hoc Network Research”. *In Mobile Ad-Hoc Networking - Research, Trends, and Applications*, vol. 2(5), of Wiley J. Wirel, *Communications and Mobile Computing*, pp. 283-505, 2002

[58] B. McDonald, T.F.Znati, “A Mobility-Based Framework for Adaptive Clustering in Wireless Ad Hoc Networks”, *IEEE Journal of Selected Areas in Communication,* vol. 17, No 8, August 1999.

[59] C.Zhang, B.DeCleene, J.Kurose, D.Towsley, “Comparison of Inter-Area Re-keying Algorithms for Secure Wireless Group Communications”, *Performance Evaluation,* vol. 49, Issue 1-4, pp. 1-20, September 2002.

[60] C.Bettstetter, G.Resta, P.Santi, “The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks”, *IEEE Transactions on Mobile Computing,* vol.2, No. 3, pp. 257-269, Jul-Sep. 2003.

[61] N.Koblitz, “Elliptic Curve Cryptosystems”, Mathematics of Computation, 48 (1987), pp.203-209.

[62] V.Miller, “Uses of Elliptic Curves in Cryptography”, Advances in Cryptology-Crypto’85, LNCS 218, 1986, pp. 417-426.

[63] V.Gupta, S.Gupta, S.Chang,”Performance Analysis of Elliptic Curve Cryptography for SSL”, *ACM Workshop on Wireless Security, Mobicom 2002, Atlanta, GA, Sept. 2002.*

[64] M.Brown,”Software Implementation of the NIST Elliptic Curves over prime fields,” *D.Naccache Ed.,Topics in Cryptology – CT-RSA 2001, LNCS, vol. 2020, Springer-verlag, 2000,* pp.250-265.

[65] RSA Performance on ARM, Palm: http://www.digisec.se/mcrypt_ performance .htm

[66] A. Menezes, M.Qu, S.Vanstone. “Elliptic Curve Systems”, *Proposed IEEE P1383 Standard, pp.1-42, April 1995.*

[67] D.Hankerson, J.L.Hernandez, A.Menezes, “Software Implementation of Elliptic Curve Cryptography over Binary Fields”, *Cryptographic Hardware and Embedded Systems – CHES 2000.*

[68] R.Schroeppel, H. Ormal, S. O’Malley, “Fast Key Exchange with Elliptic Curve Systems”, *Advances in Cryptology, CRYPTO’95, D. CopperSmith, Ed., vol. 963 of Lecture Notes in Computer Science, pp. 43-56.*

[69] E.Kaltofen, V.Shoup, “Fast Polynomial Factorization over High Algebraic Extensions of Finite Fields”, *Procs. of the 1997 Int’l Symposium on Symbolic and Algebraic Computation, ACM Press, 1987, pp. 44-48.*

[70] T.El Gamal, “A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithm,” *IEEE Transactions on Info Theory, 31: 469-472*, 1985.