ABSTRACT

Title of Document:	DESIGNING ROBUST COLLABORATIVE SERVICES IN DISTRIBUTED WIRELESS NETWORKS.
	Anuja Anilkumar Sonalker, Ph.D, 2007
Directed By:	Dr. John S. Baras, Electrical & Computer Engineering Department

Wireless Sensor Networks (WSNs) are a popular class of distributed collaborative networks finding suitability from medical to military applications. However, their vulnerability to capture, their "open" wireless interfaces, limited battery life, all result in potential vulnerabilities. WSN-based services inherit these vulnerabilities. We focus on tactical environments where sensor nodes play complex roles in data sensing, aggregation and decision making. Services in such environments demand a high level of reliability and robustness.

The first problem we studied is robust target localization. Location information is important for surveillance, monitoring, secure routing, intrusion detection, on-demand services etc. Target localization means tracing the path of moving entities through some known surveillance area. In a tactical environment, an adversary can often capture nodes and supply incorrect surveillance data to the system. In this thesis we create a target localization protocol that is robust against large amounts of such falsified data. Location estimates are generated by a Bayesian

maximum-likelihood estimator. In order to achieve improved results with respect to fraudulent data attacks, we introduce various protection mechanisms. Further, our novel approach of employing watchdog nodes improves our ability to detect anomalies reducing the impact of an adversarial attack and limiting the amount of falsified data that gets accepted into the system. By concealing and altering the location where data is aggregated, we restrict the adversary to making probabilistic "guess" attacks at best, and increase robustness further. By formulating the problem of robust node localization under adversarial settings and casting it as a multivariate optimization problem, we solve for the system design parameters that correspond to the optimal solution. Together this results in a highly robust protocol design.

In order for any collaboration to succeed, collaborating entities must have the same relative sense of time. This ensures that any measurements, surveillance data, mission commands, etc will be processed in the same epoch they are intended to serve. In most cases, data disseminated in a WSN is transient in nature, and applies for a short period of time. New data routinely replaces old data. It is imperative that data be placed in its correct time context; therefore, as a secondary problem, we studied time synchronization in WSNs. We designed a single hop time synchronization protocol, and then extended it to cover multi-hop scenarios. Our use of hash chains, a simple cryptographic mechanism, enabled the creation of a lightweight protocol that is resilient to various attacks. We also identified certain attack cases that our protocol is not robust against, and indicated possible means for securing against these attacks. We also showed that our protocol is efficient in computation and storage requirements.

DESIGNING ROBUST COLLABORATIVE SERVICES FOR DISTRIBUTED WIRELESS SENSOR NETWORKS.

By

Anuja Anilkumar Sonalker

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
2007

Dedication

To my parents, Anilkumar and Anuprita, my husband Manoj, and my family, the pillars of support in my life.

Acknowledgements

This work would not have been possible without the support and guidance of my advisor Dr John S. Baras, and without the inspiration and critical thinking provided by Dr. David Safford, IBM Thomas J. Watson Research Labs. I am deeply indebted to Dr George Moustakides whose brief sojourn at the University of Maryland College Park was very timely for my research. I would like to thank Dr Virgil D. Gligor and Dr. Jonathan Agre for their valuable inputs. I am extremely grateful to Erik Metalla for his guidance, wisdom and knowledge he so generously shared. I would also like to extend many thanks to my committee for their valuable comments.

I am forever indebted to my parents for their constant encouragement, unconditional love, and all their sacrifices that have made this day possible for me. They taught me to dream big, and follow through. I thank my parents for always encouraging my inquisitiveness, especially my Dad for all his help with those crazy "experiments" I did as a six year old. I am grateful to my husband, for his unwavering commitment to my PhD and who never complained even once during the difficult times. I owe a lot to him. I am thankful to God for the sister who ensured that I set the bar high, and a brother who thinks his sister strongest. They are the reason I want to be a better person. I will forever be indebted to my other set of parents, who welcomed me into their family with open arms, and let many a family occasion take backseat to my PhD. They are truly a godsend. I am grateful to my brother-in-law for his sacrifices and loving care that ensured us that everything would always be okay back home. I will always be indebted to my *Dada mama*, who helped me take the first successful step in my career, as a result of which I am here today. Last but definitely not the least, I am ever so grateful to *Anil Kaka* for his selfless help and *Pandit Kaka Kaku* for being there for me. I owe my success to you all.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Distributed and *Collaborative systems* have become pervasive in many environments today due to their modularity, scalability, redundancy, fault tolerance, ease of repair without loss of functionality, etc. Some examples of distributed systems that have become all-pervasive are peer-to-peer file sharing networks, shared server clusters like Storage Area Networks (SANs), coalition networks, mobile agents and Wireless Sensor Networks (WSNs). Usually, in these systems, there exists some form of coalition to share data and resources, or to make decisions collaboratively. While these systems are immensely popular, their flexibility, collaborative and dynamic nature has opened up many security problems. For example, WSNs are susceptible to eavesdropping, jamming, insertion as well as masquerading attacks. A distributed network that is collaborative in nature (shares data or resources between components of the network) is particularly susceptible to the individual components being compromised or the communication between individual components becoming unreliable. An intelligent adversary can disrupt communication between various components that together provide a service. In order to provide truly reliable functionality and dependability, these systems must be protected from malicious attacks, and their security becomes a very important issue for successful and secure deployment of distributed collaborative networks. Distributed collaborative networks usually run one or more *collaborative services*.

Collaborative services entail the use of shared resources, create and rely on joint infrastructures, are involved in taking global measurements to provide a global view, or to make joint decisions based on available information. In a distributed system, collaborative services are targets for malicious attackers who wish to foil the global measurement or the decision making process. A WSN is an example of a highly collaborative system that exhibits properties like using shared resources, forming joint infrastructures, taking global measurements, and sometimes making joint decisions.

WSNs have gained tremendous popularity due to their fast and efficient deployment and self-organization in a wide variety of scenarios where a fixed networking infrastructure is not possible. They can be viewed as a completely distributed system with collaborating entities. The primary goal of a WSN is to provide collaborative services in a distributed (decentralized) manner, for example, sensing, monitoring, information aggregation, data communication and routing. However, compared to other distributed networks they have additional constraints. They are subject to power consumption restrictions (due to limited battery life), have limited communication bandwidth, limited and unsecured storage which is subject to capture, lower computation ability, and openness associated with wireless interfaces. This nature of a WSN makes it vulnerable to protocol attacks like capture, eavesdropping, fabrication, service disruption, etc. Furthermore, there are various points in the network where an adversary can insert bogus data, alter data, or capture nodes and use them to send fictitious data resulting in a substantially different outcome. It is interesting to see how one can build secure collaborative services for such vulnerable environments that can withstand highly malicious behavior, tolerate false data and at the same time are easy to setup and configure in remote locations. In this thesis, we design two robust collaborative services for WSNs that are lightweight in terms of the computation and communication involved, provide the desired service in a robust manner by tolerating a substantially large amount of false misleading data. These two services are Robust Target Localization and Robust Time Synchronization. Our approach is built on the following principles:

- Practical assumptions,
- Light weight algorithms to provide desired service
- Improved robustness of the service by leveraging *intelligence* from the existing network to keep adversarial behavior in check,
- Use of cryptography to protect confidentiality of data, and message authenticity as necessary, to protect communication and improve resilience of the protocol.

1.1 Problem Introduction

Our work is primarily focused on military and tactical environments where sensor nodes play complex roles in data sensing as well as aggregation in a reliable and robust manner. The applications WSNs are being used for in such a tactical environment demand a high level of reliability and robustness.

The first robust service we would like to build is a Target Localization Service, which is essentially a location tracking service. Location information is important for various critical and non-critical services like mitigating Sybil[1] attacks, secure routing, sensing and tracking, surveillance, monitoring, intrusion detection, value-added services and on-demand services to name a few. Lately, various government programs like LOCO [53],

[1] A Sybil attack is one where a single physical entity assumes multiple identities. Each identity is used to siphon shared resources resulting in the Sybil node receiving a disproportionate share of resources. A Sybil can use its disproportionate resources to launch other attacks on the network. Cloning or replication involves assigning the same identity to multiple physical nodes, often after capture. We follow the approach of [46] and consider cloning attacks to be orthogonal to Sybil attacks.

APWN [9] and WAND [84] have shown great interest in robust WSN node localization and tracking in tactical environments. If the WSN provides other add-on services that depend on location information for disambiguation or implicit authentication, then the reliability and security of the localization and tracking service is vital to the success of the rest of the network. If target position is being estimated in a distributed manner, i.e., using multiple sensor nodes, then these nodes require loose single-hop synchronization among themselves. This is not impossible to achieve in practice and ensures that measurements taken during the same epoch will be collected and processed in the same iteration.

Target Localization or *tracking* deals with tracing the path of (usually moving) entities through some surveillance area where tracking devices may be deployed. The salient difference between tracking and most *localization* schemes in the broad sense is that in the localization schemes, nodes compute their own location in the field using various schemes thereby *localizing* themselves, whereas in location tracking schemes the surrounding nodes compute the location of the target using various schemes thereby *tracking* the target. Smart applications that use such topological and real-time tracking information are: traceback schemes [52], disaster relief, on-demand services, patient monitoring, surveillance, tactical applications, traffic monitoring, military and homeland security applications like military vehicle, detecting self healing land mines, monitoring, intrusion detection and intrusion prevention, etc to name a few. The lack of robust tracking schemes that hold up well in adversarial settings has motivated us to pursue this problem.

As a secondary problem, we examine time synchronization as a collaborative service that we would like to secure against tactical adversarial behavior. In a distributed collaborating environment, time is a critically important element. In order for any collaboration to succeed, all collaborating entities must have the same relative sense of time. This ensures that any measurements, surveillance data, mission commands, etc will be processed in the same epoch they are intended to serve. In most cases, data disseminated in a WSN is transient in nature, and applies only for a short period of time. New data routinely replaces old data. Therefore, it is imperative that data always be placed in its correct time context. A protocol for a distributed system maybe highly secure with the provably strongest cryptosystems one can bring to bear. Instead of attacking the cryptosystem, an intelligent adversary can simply desynchronize the collaborating entities or change timestamps associated with messages to cause the application (and the system) to function erroneously or even breakdown. Therefore, time is a critical element that must be protected, especially in a distributed network. In the second part of this thesis, we show that our proposed secure and resilient time synchronization algorithm can ensure a well bounded real-time maximum synchronization error within the network even in the face of various attacks.

1.2 Our Contributions

In the first part of this thesis, we formulate and solve the problem of robust target localization, and in the second part we address the problem of robust time synchronization. To achieve robust target localization:

a) We examined the necessary and sufficient security requirements for target localization and its participants to communicate securely (since the protocol is distributed) and applied appropriate protection mechanisms to the various components. (Section 3.3)

b) At the heart of the protocol is a particle filtering algorithm, which is a Bayesian maximum likelihood multi-step estimator. The particle filtering algorithm accepts samples (inputs) from various nodes surrounding a target, and attaches probabilistic weights to each of them. These weights are approximations to the relative posterior probabilities of the sample measurement representing the target and sum up to 1. The next step involves resampling the measurements to replacing older degrading measurements with newer updated measurements thereby improving upon the earlier estimate and creating a trajectory, tracing the path of the target through the deployment. By nature, particle filters are complex, expensive and have a certain degree of error associated with the measurements. They, however, have excellent tracking capabilities as they generate new estimates incrementally over older ones. By making the particle filtering algorithm distributed, the complexity and operational cost to the network is distributed across multiple nodes. Some nodes perform sensing and data relay operations while others perform the actual aggregation (estimation). In order to achieve improved results (with respect to fraudulent data) attacks, we applied data integrity and privacy protection mechanisms at the sensing nodes to enable secure and reliable communication. Additionally, the privacy mechanism shields the measurement data from an adversary who can now, in its best attempt, only probabilistically guess and insert malicious data *if* it possesses an authentic key. Message authenticity ensures that a message cannot be altered in transit in an undetectable manner. Additionally, we establish a general bound on the validity of measurements with loose time synchronization, whereby replay attacks are mitigated. Therefore, we are able to reduce adversarial impact significantly.

c) Our novel approach of employing watchdog nodes that provide sanity checks in terms of distance bounds, frequency of message input and anomalous behavior both in the presence and absence of activity, enables detection of certain inconsistencies and elimination of anomalous data and behavior at the aggregator. (Section 2.3.4) This further reduces the impact of an adversarial attack, and narrows the amount of falsified data that gets accepted into the aggregator.

d) At the aggregator side where data is fused to provide a meaningful interpretation (target estimate), aggregator failure or compromise can result in a point of failure. We increase the robustness of the protocol to single point of failures by shifting the aggregation function from one leader node to another in real time as the target moves through the sensing field. At most, this results in a temporary failure if an aggregator malfunctions. Target estimation resumes as soon as the target moves into the vicinity of the next leader node. Furthermore, without any additional overhead, moving the aggregator function across the network improves the resilience of the protocol to powerful attacks like adaptive node capture. Earlier, if an adversary had to capture a majority of nodes in a neighborhood to cause the outcome to degrade, it now has the extremely hard and impractical task of first guessing the next leader who will bear the aggregation function, and then compromise a majority of nodes in that neighborhood *within* the short amount of time that the aggregation function is resident on that leader. This provides a substantial amount of resilience to the protocol.

e) We formulated the problem of robust node localization under adversarial settings and cast it as a multivariate optimization problem allowing us to solve for the system design parameters that correspond to the optimal solution. (Section 3.2) Our novel use of the Simultaneous Perturbation Stochastic Approximation (SPSA) technique to cast adversarial behavior as perturbation resulted in solving the multivariate optimization problem with only 2 measurements of the objective function per iteration (irrespective of the dimensions of the optimization problem). This resulted in a significantly lightweight solution compared to regular particle filtering that is also real-time efficient and facilitates online target location estimation. (Section 3.2)

f) For the problem cast above, we have also shown how the solution is δ-robust (see section 3.2 for definition) under maximum undetectable contamination of the input, data and loss of a bounded number of honest, functional players to an adversary.

g) We derived a lower bound on the number of particles that must be active in the particle filter in order to ensure that the solution is always δ-robust.

h) We examined the dependencies associated with this solution and their effects on the outcome. (Section 3.4)

i) Finally, our decision to design the network as a heterogeneous WSN (using nodes with varying capabilities) helps achieve lower hardware cost and extends mission life.

As a secondary problem, we examined and designed a robust time synchronization service. We primarily designed a single hop time synchronization protocol to provide this service and then extended the same to cover multi-hop scenarios.

a) We addressed the problem of robust time synchronization and identify the various properties that are necessary to assure the same.

b) We designed a robust single hop time synchronization protocol using a simple cryptographic mechanism called hash chains. Using this mechanism we have been able to create a light weight protocol that provides resilience to various attacks like replay, redirection, etc. (Section 4.3)

c) We showed that our protocol is robust against various adversarial attacks. (Section 5.1) We also identified certain attack cases that our protocol is not robust against, and indicated possible means for securing against these attacks.

d) We also showed that our protocol is efficient in computation and storage requirements for wireless sensor networks.

Being closely coupled, these two services (Robust Target Localization and Robust Time Synchronization) together form a secure foundation for many WSN applications like geographic routing, pervasive computing, monitoring, surveillance, etc

1.3 Thesis Organization

The organization of this thesis is as follows. In Chapter 2, we first provide an overview of existing work done in the area of sensor network localization (Section 2.1). In Section

2.2.1 we present the system model and in Section 2.2.2., we formally define the capabilities of the adversary and the performance bounds of an intelligent adversary. We then give an outline of our Robust Target Localization Protocol in (Section 2.2.3) and the protocol specification in Section 2.2.4. Finally, in section 2.3 we elaborate upon the novel features of the proposed protocol. Chapter 3 deals with the analysis of our protocol's security and robustness under the influence of tactical adversaries. We formulate the attack model of the adversary, and in separate sections analyze the security, and robustness of the protocol under attack. We also list the various dependencies associated with the protocol and what their influence on the protocol outcome is, if any. Chapters 4 and 5 deal with Robust Time Synchronization. In chapter 4, we first describe the current body of work in the area of both time synchronization as well as secure time synchronization (Section 4.1.1 and 4.1.2). We then enumerate the properties of a Robust Time Synchronization Protocol that are essential to a distributed collaborative network (Section 4.2). We describe the various components of our scheme in Section 4.3 and our proposed scheme in Section 4.4. Specifically, we formulate and discuss our adversary model in Section 4.4.1 and the protocol specification for both single and multi-hop synchronization in 4.4.3. In Chapter 5, we analyze the security of our protocol to show that it satisfies the properties specified in Section 4.2 sufficiently (Section 5.1). Finally, we conclude in Chapter 6 with a summary of our results and a glimpse of our proposed future directions.

Chapter 2 Robust Target Localization

In this chapter we first provide an overview of existing work done in the area of sensor network localization. We then present the system model and formally define the capabilities of the adversary. We then give an outline of our Robust Target Localization Protocol and the environment of operation. Next, we provide the protocol specification. Finally, we elaborate upon the novel features of the proposed protocol.

2.1 Current Research in Secure and Robust Localization

There are three main branches of localization namely, node localization, target localization and location service. Most contemporary research has been focused on node localization, with most researchers having proposed a number of location determining schemes for sensor networks in non-adversarial settings [8][34][64][65] [66][70]. Recently, few researchers have provided unique solutions for node localization in adversarial settings [21][50][51][63][71][73]. Though these techniques solve a multitude of problems, some of them use self-positioned verifiers, pre-shared secret keys, some perform only verification requiring the claimant to initiate, and some others rely on simplified assumptions that do not hold in practice. Since we are interested in a highly tactical deployment environment these schemes are unsuitable for our environment. Furthermore, some of these schemes rely on an inherent assumption that the self-positioned verifiers cannot become malicious or be compromised. Since we are expecting to deal with Byzantine behavior, practically speaking, every node is susceptible and we cannot rely on such schemes.

Location determination schemes can be broadly classified into range dependent and range-independent schemes. The former schemes rely on time, angle, received signal strength, power measurements or measurement of quantities that are a direct measure of the distance traveled by the signal. Range independent schemes do not utilize such techniques. For example, using wireless beacon messages, one hop connectivity information, etc. Another useful classification is centralized computation vs. de-centralized computation of location, depending on where and how the location computation process takes place. For example, some nodes hand off their position estimates to a central node to compute target location while others each compute location themselves after gathering required information from their neighbors and the environment. Yet another useful classification is infrastructure-based and infrastructure-less schemes. The former are based on GPS and other external unchanging infrastructures, while the latter are independent of these. Our scheme falls under the range-independent, de-centralized and infrastructure-less schemes. Range dependent schemes based on Time of Arrival (TOA)[72][10], Time Difference of Arrival (TDOA) [4][64], Angle of Arrival (AOA)[22], and Received Signal Strength Indicator (RSSI)[66] to name a few, are meant for non-adversarial scenarios and are easily susceptible to failure in the presence of adversaries. Similarly, range-independent schemes like [62] are also susceptible to various attacks like Sybil attacks, wormhole attacks etc. Two secure localization protocols proposed by researchers recently viz., SeRLoc [50] and Secure Positioning [71] were also analyzed. These protocols have been created for adversarial scenarios and are secure against many attacks. However, they have heavy dependencies on trusted locators or verifiers, directional antennas (expensive hardware), GPS based static infrastructure, and have computational and storage overheads. Particularly, few drawbacks of SeRLoc include dependency on GPS-based locators, hardware requirement of special spatial/sectored antennas, high power transmission requirement for the locators, pre-deployment knowledge and pre-loaded cryptographic quantities (keys, hash tables). Moreover, they assume locators are trustworthy and cannot be compromised by an adversary, DoS attacks are not considered since they are MAC level attacks, jamming is not considered since it can be easily eliminated by Spread Spectrum and coding techniques, locator communication range *R* must be known apriori by sensors, and the scheme has a high computational overhead as sensor nodes perform heavy computation to determine location based on beacon information. Further, this solution trades computational expense for resolution in that the centre of gravity (CoG) is computed using a grid system to improve computational expense due to which resolution is diminished. To further refine position, grid resolution must be increased, causing increased computation and processing time. From a security perspective, the use of a shared symmetric key only prevents external adversarial attacks but is still prone to insider and node compromise attacks. To their credit SeRLoc is, however, robust and accurate in the presence of Sybil, select wormhole and various other attacks compared to most other solutions in this area. Rope [51], a successor to SeRLoc, which provides all the benefits associated with SeRLoc's sectored antennas, as well as some new properties like distance bounding fares well, but still carries most deficiencies associated with SeRLoc like expensive hardware requirements, high computational cost, etc. Secure Positioning [71], another secure localization solution is based on trilateration using static infrastructure. Here, verifiers of the boundary triangle are part of the infrastructure and are assumed to be trustworthy and never compromised. This protocol is vulnerable to a wormhole attack. Further, it uses least median square method to dampen error due to contamination of distance estimates. This method, as a result, suffers from high degradation even at one-third contamination.

In conclusion, most secure and non-secure protocols are based on assumptions that are sometimes impractical, and at other times too rigid to facilitate truly ubiquitous and mobile applications. We are therefore motivated to build a secure target localization protocol that does not have a fixed infrastructure, can localize moving targets, is light-weight and efficient in computation as well as communication, and is robust and secure to the desired degree against false data in highly adversarial scenarios.

2.2 Our Approach to Robust Target Localization

2.2.1 System Model & Assumptions

2.2.1.1 Sensor Model

Our design incorporates heterogeneous capability devices. We mainly have two types of sensor nodes, type *A* and *B*. Type *A* sensor nodes are long range, low power, high battery life high end sensor nodes that are used for data aggregation, transmitting long distances, and memory intensive operations. They are also known as *aggregators* or *leader* nodes. Computation costs, memory usage and storage are normally not a concern to these types of nodes. These nodes are typically capture resistant or very hard to capture. Type *B* nodes have a substantially short range of operation, and have lesser resources. These are mainly used only as 1-hop sensing and relay elements, and can be captured by a reasonably strong adversary.

As with any captured sensor node, all data, keys (if any) and resources of the captured sensor node are available to the attacker. All nodes obey protocols unless they malfunction or are malicious. All sensor node antenna types are known and calibrated. Sensor nodes need not always be stationary, but in our work, we assume stationary nodes to reduce uncertainties in the final outcome. These uncertainties can be modeled if the motion model of the sensors is defined.

2.2.1.2 Trust Model

Since all sensor nodes operate in an ad-hoc manner, no sensor node directly trusts another node. Type *A* sensor nodes, which compute the aggregated estimate of the position of the target, are assumed to function as per the algorithm unless malfunctioning, compromised or turned malicious. In other words, we trust all computations of good nodes, while communication between any nodes need not be trusted or reliable. All good Type *B* nodes obey protocol unless they are malfunctioning, compromised or turn malicious. The locations of the nodes in a neighborhood (cluster) are relatively known to the leader of the neighborhood. We will show later on, that these location values need not be trusted by an aggregator as such discrepancies can be identified by neighborhood watchdogs.

2.2.1.3 Assumptions

This scheme relies on loose time synchronization within the sensing cell as well as between adjacent leader nodes. Since all synchronization events are single-hop, this is a very realistic assumption. Over time, internal clocks of different sensor nodes may drift apart, so resynchronization after some time maybe required. Most time synchronization schemes incorporate resynchronization techniques. In practice, this assumption is not hard to achieve and has been demonstrated successfully with very good results in similar distributed network architectures.

The particle filtering algorithm which is used by type A nodes to compute an aggregate estimate of a target's location cannot be altered in any way, shape or form. To an adversary it appears as a black box. If an aggregator node is compromised, an adversary can only supply malicious or malformed data to the particle filtering algorithm to influence the output. It cannot cause the algorithm to behave in a manner inconsistent with its nature. We assume all nodes in our setup to be stationary.

2.2.1.4 Target State Model

Since we do not stress on any particular technology to determine target measurements, we need a model describing the relationship between the states, velocity and other parameters of the moving object. This helps to relate incoming measurements to the target location based on known values like the previously known position of the target, its motion model (foot, car, military tank, etc), and new incoming beliefs about the targets location. Typically measurements about a target can be signal strength measurements signifying range, ultra sound delay measurements, directional measurements, x-y coordinates and velocity information, etc. In our model, we use the x-y states and velocity information which is trivial to gather. Usually, for a particular deployment, one knows what it is tracking. For example, a foot soldier, or a battle tank will have a distinguishably different travel velocities.

We represent the state transition model of the target as follows:

$x_k = \Phi x_{k-1} + \Psi u_k,\ k = 1,....,K$ describes the sensor node at time *k*, where $x_k = \left[X_k, V_{x_k}, Y_k, V_{y_k}\right]^T$ is the state space representation of the properties that uniquely determine x_k with a period of observation θ seconds, and $u_k = \left[u_{x_k}, u_{y_k}\right]^T$ is the incorporated sensor noise model. Specifically, X_k, Y_k are the co-ordinates of the node, and $V_{x_k} = \dot{X}_k,\ V_{y_k} = \dot{Y}_k$ are the *x* and *y* velocity components. u_{x_k}, u_{y_k} are the *x* and *y* noise components.

$\Phi = \begin{pmatrix} 1 & \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \theta \\ 0 & 0 & 0 & 1 \end{pmatrix}$, and $\Psi = \begin{pmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{pmatrix}$ are the coefficient matrices. The values of Ψ have been empirically determined for the given deployment. These values can alternatively be determined as a function of target velocity, environment characteristics like attenuation, etc. For a contiguous measurement tracking system like ours θ=1.

2.2.2 Adversary Model

An adversary can compromise any type *A* or *B* sensor node though by design, it is harder for an adversary to compromise a type *A* than a type *B*. The adversary does not have to adhere to protocol. An adversary may be internal i.e., it may be part of the sensing network, or external, i.e., it may be an outsider to the sensing network. Characteristics of our adversary include:

Adversary Motivation: To completely disrupt the secure position estimating process, motion tracking process or, to throw the estimate way off track.

Access: The adversary has access to the wireless sensor network in a way that it can eavesdrop on communication occurring within its range, has access to schedules and secrets (if any) maintained on a compromised node. An adversary has knowledge of the particle filtering algorithm employed as well as all parameters stored on a compromised node.

Skills and Resources: The adversary may be as skilled as the Type *A* sensor nodes deployed. An insider adversary has access to all network resources as well as its own additional resources. If an adversary captures a node, its capabilities then include full control over resources, data, and secrets of the captured node. In other words, if captured, a node can be used as a collaborating adversarial node.

Tactics: The adversary can be active/passive, can insert, modify, replay, redirect messages and assume identities of other nodes (masquerade). It, however, cannot fabricate messages that decrypt correctly on behalf of a node that it has not compromised. In other words, an adversary cannot circumvent the cryptosystem. Though deliberately jamming communication (partially) is possible in any wireless network, we believe mitigating this attack is outside our current scope. Existing work using Spread Spectrum [69] or other coding techniques [78] is known to mitigate jamming attacks. Also, the presence of watchdogs also is a deterrent to selective jamming. We are, however, interested in those tactics of the adversary that result in malicious or malformed input being accepted by the system, and proving the robust working of our proposed algorithm in the presence of such malicious data.

We will show later how an adversary can use these tactics to launch various attacks on our proposed protocol.

2.2.3 Protocol Description

We begin with a setup of randomly distributed type *A* and type *B* sensor nodes as shown in Figure 1. Sensing cells and leaders have been established using leader election algorithms. (We discuss this feature in detail in Section 2.3.1) Each sensing cell has a single type *A* leader. Leaders of adjacent sensing cells can communicate with each other.

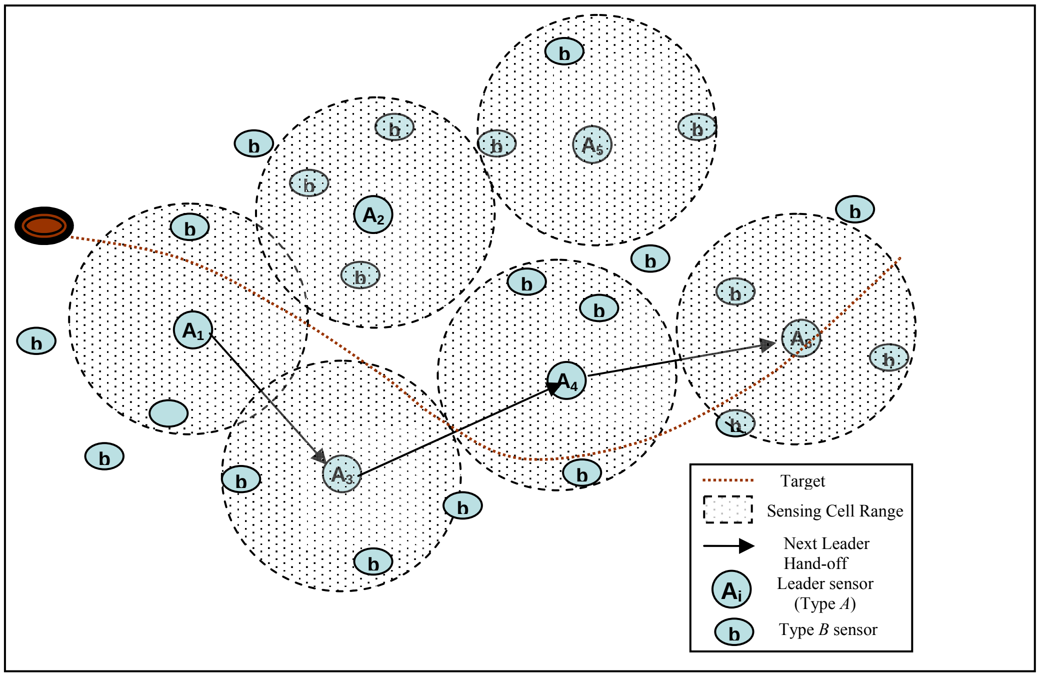


Figure 1: A typical tracking scenario with Moving leader hand-off.

When a target comes into sensing range of one of the sensing cells, two things happen: the leader of the sensing cell broadcasts an *alert* signal to all neighboring leaders and, a small subset of the type *B* sensor nodes take measurements x_k^i at time *k*, where *i* is the *i*th sensor and send it to the type *A* leader of their cell. (If the sensor scheme additionally employs energy conservation schemes like sleep scheduling [58][26], then the leader sends an awake signal to its sensing cell members one step ahead of time. This is a knee-jerk action, i.e., when a leader node receives an alert signal from its neighboring leader node, it automatically issues a *wake-up* signal to its sensing cell members.) The leader node has the apriori *belief* state from previous measurements till time $k-1$ i.e., $p\left(x_k \mid z_{1:k-1}\right)$ which is an estimate of the previous position of the target (at time $k-1$).

Using the new measurements received from the type *B* nodes, and the prior belief state $p\left(x_k \mid z_{1:k-1}\right)$, the leader computes the maximum likelihood of the target's location. This estimate is computed using a particle filter and Sequential Monte Carlo (SMC) approximation, where the measurements x_k^i are the particle filter inputs. Based on this estimate, the leader chooses the next leader for data aggregation and sends its estimate to the next leader. This estimate now becomes the prior belief state of the next leader. The process repeats until the target leaves the sensing field. This repetition gives us a contiguous estimate of the target's path i.e., its trajectory. For example, in Figure 1 above, the target (shown by a red oval) enters the sensing field near A_1's cell. As soon as the target enters A_1's sensing cell range, A_1 sends an alert signal to all neighboring leaders (A_2 and A_3 in our case). At the same time the type B sensor nodes in A_1's cell take and send measurements pertaining to the observed target to A_1. As soon as A_2 and A_3 receive an alert signal, they alert/awake their type *B* sensor nodes and these cells are ready for measurement. Since A_1 is the first node in the sensing field to compute this target's location, it does not posses an apriori belief state. Therefore, it randomly draws the initial apriori belief state from the sample space. A_1 now computes an estimate for the target's position using measurements it received from the type B nodes in its cluster and the initial apriori estimate. A_1 computes the target's position for each time window that measurements come in. It makes the estimate available to the neighbor in the estimated direction, namely to A_3 in our example. As the target moves into the cell of A_3's leadership, A_3 repeats the process above. This process continues with handoffs of the prior estimate to subsequent leaders ($A_1 \rightarrow A_3 \rightarrow A_4 \rightarrow A_6$ in our case) until the target moves out of range of the sensing field.

2.2.4 Protocol Specification

2.2.4.1 Notations and Definitions:

1. We define all participating principals based on the two types of sensor nodes *A* and *B* as set $A, B \in P$ where *P* is the set of all sensor nodes deployed in the field belonging to the same organization. Further, Let *X* be the target traversing the deployment field.
2. We denote a sensing cell by index *i* and the leader of that sensing cell as A_i. Similarly, all type B sensor nodes in a sensing cell *i* are denoted as B_{ij} where i denotes the cell affiliation and j denotes the individual type B sensor node.
3. $A_i \rightarrow B_{ij} : M$, denotes a message $\mathcal{M}$ sent from principal $\mathcal{A}_i$ to principal $\mathcal{B}_{ij}$, in sensing cell *i* and reads "Leader node $\mathcal{A}$ sent message *M* to $\mathcal{B}_j$,
4. *n* is the number of leader cells $\mathcal{A}$ and m_i is the number of type *B* sensor nodes in the *i*th sensing cell. *m_i's* can be different for different cells, but for simplicity we assume all cells to have the same number of type *B* nodes.
5. * is used as the short hand for all in the nodes related to the position that the symbol appears in. For example, B_{i*} stands for all type *B* nodes in sensing cell *i*. Similarly, A_* stands for all leader nodes A_i for *i*=1 to n.
6. $\mathcal{PK}_{\mathcal{A}i}$ is the public key of A_i
7. TTP is the trusted third party that generates verifiable ID-binding key pairs for the leader nodes, PK_i and SK_i are the public and private keys generated by the

TTP's key generator PKGen. sig_{TTP} is the signature of the TTP. Every node recognizes this signature.

8. <p.f.i>$_j$ is the particle filter input from each reporting node j. For simplicity, we omit the notation *i* when we are talking of a single cell. An apriori estimate from a leader node adjacent to A_i is denoted by <p.e>$_{i-1}$.

9. Γ is the SMC particle filtering function that returns the final output after performing the three internal operations (Initialization, importance and resampling), ε^2, ξ and $\xi^2_{\max}$ are the minimum location estimation error in a benign environment, location estimation error observed and maximum tolerable estimation error respectively. They are further elaborated upon and quantified in the next section.

10. θ is the time interval of observations considered in this round (measurement window).

11. A simple Verifiable ID-binding Key generation is used by a Trusted Third Party to generate public-private key pairs for all Type A nodes. Type B nodes are installed with the public keys of those type A nodes whose range they will fall approximately within during deployment. Type A nodes are only installed with their respective secret keys, and the public keys of other type A nodes.

    Key Generation Method:

$$TTP : PKGen(ID_i) \rightarrow PK_i, SK_i$$
$$TTP \rightarrow i : \{i, (PK_i) sig_{TTP}\} sig_{TTP},$$

2.2.4.2 Specification

Phase 1: Aggregation and Position Estimation Phase at A_i

For every type B sensor that senses an active target in cell *i*:

(1) $B_{i*} \rightarrow A_i : \mathbb{M} : \left(< p.f.i >_j, t_k, B_{ij}, A_i, \{< p.f.i >_j, t_k, B_{ij}, A_i\} SK_{B_{ij}} \right) PK_{A_i}$

(2) $A_i : \forall j \;\; check < p.f.i >_j$ such that $\varepsilon^2 \leq \xi \leq \xi^2_{\max}$ AND $\mathfrak{V}(\mathbb{M})$ *is true*

$$\text{If } \textit{true}, \; A_i : \prod_{j=1}^{m} < p.f.i >_j = \langle \text{Target\_Location} \,|\, \theta \rangle_i$$

(3) Process repeats from (1) for next time interval θ

Phase 2: Hand off Phase at A

(1) $A_i \rightarrow * : \left\{ A_i, t_k, \langle \text{Target\_Location} \,|\, \theta \rangle_i \right\} SK_{A_i}$

(2) A_{i+1}: If $\left\{ A_i, t_k, \langle \text{Target\_Location} \,|\, \theta \rangle_i \right\} SK_{A_i}$ is *true*,

$$< p.e >_{(i+1)-1} = \langle \text{Target\_Location} \,|\, \theta \rangle_i$$

Verification Function $\mathfrak{V}(\mathbb{M})$

Input: Message $\mathbb{M}$

If $\left(v \in [\hat{p} - v_l, \hat{p} + v_u] \,\&\, \text{TS}(\mathbb{M}) \,\&\, (\text{sig\_val}) \,\&\, N2N\_flag = 0 \right)$

Output (1); (Message verification PASS)

Else Output (0); (Message verification FAIL)

Verification function $\mathfrak{V}(\mathbb{M})$ is used by aggregators to verify whether the frequency of incoming messages is within allowable limits, a message is fresh, whether it has a valid signature, etc. It outputs *true* or *false* based on the result of verification process. The following is verified using this function:

a. The frequency of input is bounded within $[\hat{p}-\nu_l, \hat{p}+\nu_u]$ where $\hat{p}$ is the baseline frequency, and ν_l and ν_u are the allowable lower and upper deviations in frequency.
b. Timestamp is fresh
c. Signature verification (sender, recipient, integrity of <p.f.i>)
d. No anomaly reported by neighboring nodes in cell. (For details see lemma 1 and 2 in Section 2.3.4)

2.3 *Salient Features of our Robust Target Localization Scheme*

The following features of our protocol help make it robust against falsified data, secure against various attacks, scalable, achieve consensus regarding measurements, and detect inconsistencies in neighborhoods.

2.3.1 Hierarchical Capability-based Heterogeneous network

This design feature helps improve mission life, promotes optimal power management and makes for a cost effective design.

In any deployed sensor network, power and bandwidth are of prime concern. Inefficient algorithms and inefficient allocation of roles to participating entities can lead to exhaustion, starvation and early termination of the life of a deployed network. Processing power, capabilities and life of a sensor node are directly related to its cost. In order to be cost-effective, we need to have an intelligent mix of the use of relatively inexpensive, less sophisticated *workhorse* type sensor nodes and the more expensive mini-computer type sensor nodes. We, therefore, segregated tasks in the deployed sensor network on the basis of function and invest in hardware accordingly. We chose to deploy a heterogeneous network comprising of two types of sensor nodes, sensor type *A* and type *B* as described in [25].

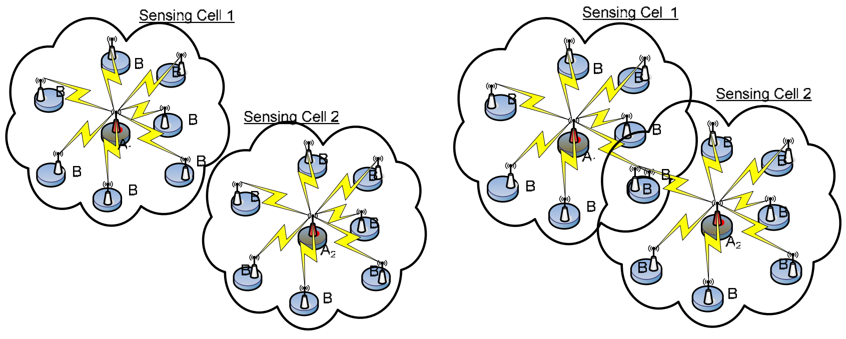


Figure 2: A Typical Sensing Cell in two configurations

Type *A* sensor nodes are long range, low power, high battery life sensor nodes that are used for data aggregation, capable of transmitting long distances, and performing memory intensive operations. Computation costs, memory usage and storage are normally not a concern to these types of nodes. Type *B* nodes have a substantially shorter range of operation, and have lesser resources. These nodes are mainly used only as 1-hop sensing and relay elements. Their task is to simply sense and transmit the information locally over relatively short distances. Here, we introduce the notion of a sensing cell (Figure 2) which is the region of administration of a single leader. The entire sensing field can be comprised of multiple sensing cells. Every type *A* sensor tries to establish a *sensing cell*, which is the area of its leadership. We also refer to a sensing cell as a *cluster*. Within a single sensing cell, there is only one type *A* leader and multiple type *B* sensor nodes. Since sensor deployment in certain areas and applications is random, there may exist multiple type *A* sensor nodes in a single sensing cell. In such cases, they resolve the contention and elect a single leader for the cell. The choice of election algorithms for this distributed system is purely an implementation choice. There are many traditional leader election algorithms in distributed systems [7]; any algorithm that can be implemented over these sensor nodes is acceptable. Since this is not hard to achieve in practice, we assume that leader election is completed without conflict. It is worth noting here that establishment of a sensing cell is crucial to this scheme, yet no assumption is made about the integrity of the leader who is elected, and no pre-installed secrets are required to complete this phase. Since we believe that position estimation algorithms should precede routing and authentication algorithms so that the latter can use position-related information to their advantage, we do not assume any routing capabilities in the network. As a result, only those sensor measurements are received at the leader that are within one-hop range from the leader node.

2.3.2 Distributed Aggregation and Moving Leader Design

This design feature helps improve fault tolerance, optimizes energy and bandwidth consumption, reduces chances of battery depletion attacks, and improves real time estimation by reducing unnecessary processing delays in the network. Most applications including tracking and sensing applications require data from multiple sources to be cooperatively aggregated together. In centralized approaches, as shown in Figure 3, all sensed data is relayed to a central base station for aggregation. This results in a lot of communication from the sensing locality to the locality of the base station. The nodes closer to the base station end up simply becoming relays for the rest of the network and quickly get exhausted and die. If the motion of the target is in a direction away from the base station, the situation becomes worse. From a security standpoint, relaying measurements from the sensing node to the base station through multiple hops opens up multiple points for intermediary nodes to corrupt data. Battery exhaustion is another valid attack adversarial nodes can launch upon nodes closer to the base station. If successful, it can result in the base station getting cut off from the rest of the network. Elaborate schemes for routing and path integrity maintenance will be required to mitigate these problems. For these

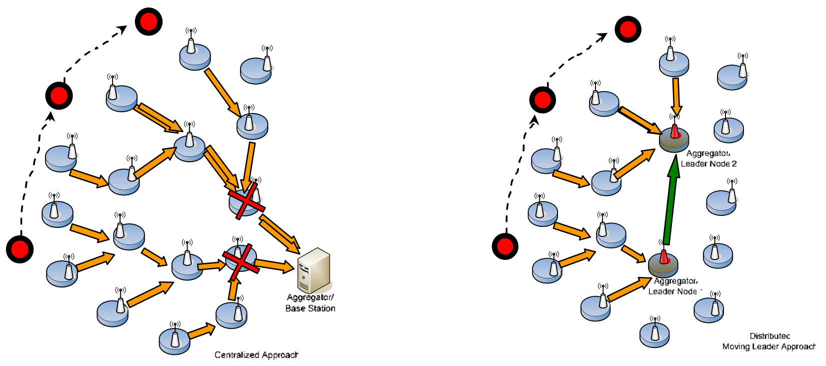


Figure 3: Moving Leader Approach

reasons, we discard the use of centralized processing schemes and adapt a distributed approach. The distributed aggregation approach provides intrusion tolerance to the protocol by moving the aggregation function as close to where the information is gathered as possible. This also helps minimize delays in computing the aggregate from the time the measurements are taken.

As the target moves through the sensing field and further away from the initial aggregator node, once again the sensor nodes start relaying information from the new locality of the target to the old locality where the aggregator resides. A lot of useful bandwidth is again wasted in this relaying process. It has been shown through experimentation that each bit transmitted consumes as much power as 800-1000 instructions executed [13] and hence is not an insignificant measure that can be overlooked. From a security perspective too, computing the aggregate at a single location becomes a single-point of failure if the only aggregating node fails, is compromised or the nodes closest to it are deliberately exhausted.

Therefore, the single aggregator is a high profile target for any attacker, and must be made fault tolerant. We adopt a *moving leader* approach, as shown in Figure 3 where the aggregating node is always moved to be within the sensing locality of the target. This is done by executing the aggregation function at the leader of the cluster in which the target is present at a given time. The leader performs the aggregation and *hands off* the target position it has estimated to the next leader in the predicted direction of the target's motion. Due to the hierarchical cluster arrangement of nodes in our protocol, the moving leader approach can be effortlessly implemented without any additional overhead. Clusters and their leaders have already been established, type B sensor nodes have been configured to send their measurements to their respective leaders. Therefore, the only change we need to make is that a leader has to send its position estimate to another leader. Since the particle filtering algorithm that does the aggregation has an apriori estimate component, this hand off becomes very useful. By incorporating a moving leader, we combine the goodness of data aggregation in the sensing locality with processing it within the sensing locality itself to provide intrusion tolerance, reduce delay and decrease load on sensor nodes.

2.3.3 SMC Method for Data Aggregation

This design feature helps improve real-time *online* target position estimation, accurate positioning and trajectory tracing. It also provides for low storage, communication, and computational costs as compact representation allows the storage and exchange of very little data without diminishing accuracy.

At the core of our protocol lies an algorithm that belongs to the class of sequential Monte Carlo methods (SMC), also known as particle filters because they maintain a set of state trajectories (or particles) that are candid representations of the system state. They have been information theoretically proven to be good filters for dynamical systems. We use one such particle filter to process input parameters otherwise known as *particles* obtained from multiple affiliated sources and aggregate them in a Bayesian manner that preserves previous information as well as incorporate the current to provide a *trajectory* of the target.

The first component of the particle filtering algorithm $<p.f.i>_j$ is the particle filter input from each reporting node j. For simplicity, we omit the notation *i* when we are talking of a single cell. An apriori estimate from a leader node adjacent to A_i is denoted by $<p.e>_{i-1}$. The second component is Γ the SMC particle filtering function that returns the final output after performing the three internal operations

(Initialization, importance and resampling). We describe these internal operations further in this sub section.

Sequential learning and inference methods are important in many applications involving real-time signal processing, where data arrival is inherently sequential. In our application, furthermore, due to the possible motion of the target, a sequential processing approach would be necessary to deal with non-stationary signals. This way, information from the recent past is given greater weightage than information from the distant past. From a logic perspective this makes more sense in our environment, as the last known location of a target is of more value than its previous locations for the purpose of computing its next possible location. To perform this type of computation using other conventional collaborative processing techniques would imply the storage and exchange of large amounts of state information, which defeats the purpose of using the distributed architecture and moving leader approach to keep communication overhead at a minimum. The particle filter also has a very compact representation, and very little data has to be comparatively exchanged to derive a true estimate of the target's position without diminishing accuracy. Thus computational simplicity in the form of not having to store all the data also constitutes an additional motivating factor for applying sequential methods.

Monte Carlo methods are very flexible in that they do not require any assumptions about the probability distributions of the data. Moreover, experimental evidence suggests that these methods lead to improved results [76]. From a Bayesian perspective, Sequential Monte Carlo methods allow one to compute the posterior probability distributions of interest on-line. Yet, the methods can also be applied within a maximum likelihood context. Though there are various implementations of particle filters, we describe the common approach and the generic steps involved:

Multiple copies (particles) of the variable of interest are used, each one associated with a weight that signifies the quality of that specific particle. An estimate of the variable of interest is obtained by the weighted sum of all particles. The particle filter algorithm is recursive in nature and operates in two phases: *prediction* and *update*. After each action, each particle is modified according to the existing model (*prediction* stage), including the addition of random noise in order to simulate the effect of noise on the variable of interest. This step is also called the Importance Sampling Step. Then, each particle's weight is re-evaluated based on the latest sensory information available (*update* stage). At times the particles with (infinitesimally) small weights are eliminated. This process is called resampling.

Step I: Initialization Step/ Sample Step (S):

In this step, the M particles, denoted by $\left\{x_n^{(m)}\right\}_{m=1}^{M}$ i.e., for m = 1….M are initialized by drawing samples from the initial distribution: $p\left(x_k \mid x_k = x_{k-1}^{m}\right)$ for every time instant *k*. x_{k-1}^{m} denotes the previous observation. Every importance weight is initialized to $w_0^{(m)} = 1/M$.

Step II: Importance Step (I):

In this step, we draw from an importance density function $\pi\left(x_k \mid x_{k-1}^{(m)}, z_{1:k}\right)$ and create a trajectory proposal as shown below:

$$\pi\left(x_{1:n}\right)=\pi\left(x_1 \mid z_1\right)\prod_{k=1}^{n}\pi\left(x_k \mid x_{1:k-1}, z_{1:k}\right).$$

Particle weights can be recursively computed as:

$$w_n^{*(m)} = w_{n-1}^{(m)} \frac{p\left(z_n \mid x_n^{(m)}\right).p\left(x_n^{(m)} \mid x_{n-1}^{(m)}\right)}{\pi\left(x_n^{(m)} \mid \pi x_{1:n}^{(m)}, z_{1:n}\right)}$$

And normalized as: $w_n^{(m)} = \dfrac{w_n^{*(m)}}{\sum_{j=1}^{M} w_n^{*(j)}}$

Step III: Resampling Step (R):

Since weights degrade, we resample so that trajectories with smaller weights can be neglected and those with higher weights can become more prominent. For a comprehensive understanding of SMC methods, additional resources are available at [54] [30].

In summary, we use the available indirect measurements (also called observations) from time 1 through k ($z_{1:k}$) and the most recent estimate of position (x_{k-1}) to compute the maximum likelihood of the next location using Bayesian inference.

2.3.4 Watchdogs: Additional Data Sources

This design feature helps disambiguate potential confusions, identify possibly malicious or malfunctioning nodes in a neighborhood, provides upper bounds on distance estimates without increasing the complexity of the protocol.

Here, we leverage the inherent property of a distributed wireless sensor network that all communication is seemingly open, spatially and temporally correlated and that both the observance and absence of data is a rich source of information. . We design watchdogs that observe and make inferences based on (1) communication between nodes and (2) the presence and absence of data. Watchdogs are uniformly distributed across the deployment network.

Node-to-Node (N2N) data Watchdogs: One of the advantages of having a distributed estimation system is that there is a lot of data around us that we can use to make our current estimates better and smarter. For example, due to the nature of the communicating medium, when sensor nodes within the same cell send messages to the leader, they can hear each other’s messages as well. This is a very important source of information, which can be used to identify potentially misbehaving nodes. For example, if a node is sending data inconsistently with respect to its neighbors (e.g., when there is no target sighted), the neighbors will be able to observe this inconsistency and report it to the leader. An important effect of this observation is that if a malicious node increases the frequency of sending false data (which is acceptable to the system till some extent), the watchdog nodes can observe and report this fact to the leader. This can prevent potentially large amounts of false data to be inserted into the target estimate.

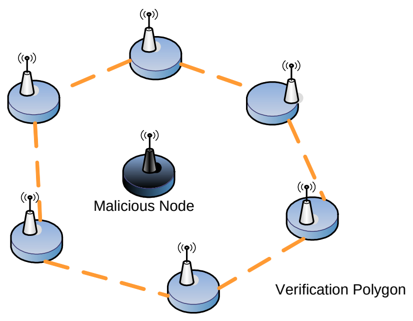


Figure 4: Disambiguation using Node-to-Node data.

Another example would be the alteration of the signal strength by the transmitting target or the adversary. If any entity changed its transmitting signal strength, it would seem to be closer or farther away from its neighbors than it actually is. If its neighbors form a closed polygon, as shown in Figure 4, the entity cannot appear to be closer to ALL or farther away from ALL its neighbors at the same time. If it appears closer than it is to a few neighbors, it must appear farther than it is to other neighbors and vice-versa. When neighbors relay information regarding this entity to the leader, they can hear each other's messages, detect any inconsistency and notify the leader. If the leader receives sufficient number of inconsistency reports for a node, it can choose to ignore the inputs received from the inconsistent node.

We formulate and prove succinctly how node to node data can be used to detect anomalies in node behavior and discard malformed input coming from such nodes using lemmas 1 and 2.

Lemma 1: *In a given neighborhood $\mathcal{N}_i$, repeated observations from any node q that deviate from the observations of a simple majority of q's neighbors in the same interval for the same target, indicate Byzantine behavior, malicious behavior, or play* *of contrasting environmental characteristic and such observations should not be included in aggregated estimates.*

Proof: In a benign environment, there is bound to be some deviation in measurement among nodes within a locality due to contrasting environmental factors like uneven terrain, shadows, presence of signal attenuators like trees nearer to some nodes than others, etc. Once measured and accounted for, this becomes the base line deviation for the deployment. Beyond this deviation, any observed deviation must be due to malfunction or due to node behavior under malicious influence. Repeated behavior can cause a deviant node to be reported and subsequently ignored. We call this threshold the observation threshold, crossing which results in a node being reported. It can be set depending upon the expected capabilities of adversaries and the baseline deviation. Existing vote based algorithms can be incorporated here to avoid innocent nodes being reported and ignored. However, it is not necessary to do so as long as larger-than-majority collusion does not occur within the neighborhood. If a node is reported for any reason including possible environmental factors by a number of watchdogs, it is best to ignore the inputs coming from this node as these inputs will result in dilution of the accuracy of the position estimate.

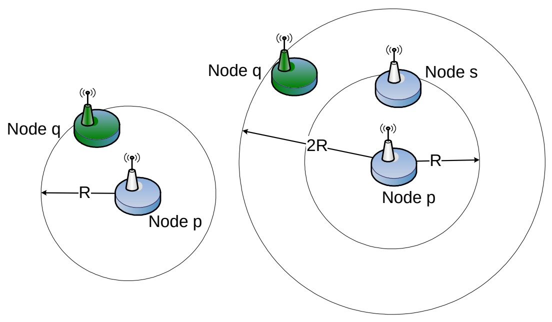


Figure 5: Inferring from Non-observed data

Non-observed data Watchdogs: This data comes, actually, from no data. In other words, just as the observance of data indicates something, in location determination and disambiguation, the non-observance of data is equally informative. For example, as shown in figure 5, if node *p* can hear node *q*, it would indicate that node *q* is within a radius of R of node *p*, R being the communication range of the nodes. Similarly, if node *p* does not hear node *q*, but knows that one of its neighbors *s* (who is within R of *p*) can hear node *q*, it would indicate to *p* that node *q* is within [R, 2R] of it. Similar information from few other nodes would help place node *q* or any other target more accurately.

Lemma 2: *If R denotes node communication range, and p q and s are three nodes deployed in the field under consideration, and* $\left|d_p - d_q\right|$ *being the absolute distance between p and q then the following statements are true:*

a) If p hears q, they must be at most R apart, i.e., $\left|d_p - d_q\right| \le R$

b) If s hears both p and q, they must be at most 2R apart i.e., $\left|d_p - d_q\right| \le 2R$

c) If p hears s, p does not hear q, but s hears q, then q must lie within $(R, 2R]$ *of p. i.e.,* $R < \left|d_p - d_q\right| \le 2R$

Proof a): We refer to figure 5 to prove this intuitive lemma. If the communication range of each node is R, then by virtue of this argument, any honest node that *p* can hear, must be within the range R. Therefore,

$$\left|d_p - d_q\right| \le R \qquad \text{q.e.d} \qquad (1.1)$$

Proof b): We use Lemma 2 a) to help us here. We start with Eq. (1.1)

$$\left|d_p - d_s\right| \le R \tag{1.2}$$

$$\left|d_s - d_q\right| \le R \tag{1.3}$$

Adding (1.2) and (1.3) we get

$$\left|d_p - d_s\right| + \left|d_s - d_q\right| \le R + R$$

$$\Rightarrow \left|d_p - d_q\right| \le 2R \tag{1.4}$$

Proof c): *p* hears *s*, *p* does not hear *q*, but *s* hears *p* can be rephrased as *s* hears both *p* and *q*, but *p* does not hear *q* (and vice versa is assumed).

From Lemma 2 a) we have

$$\left|d_p - d_s\right| \le R \text{, and } \left|d_s - d_q\right| \le R$$

From $\left|d_p - d_s\right| \le R$ we have

$$\begin{aligned} &\left\|d_p - d_q - d_s + d_q\right\| \le R \\ \Rightarrow &\left\|d_p - d_q\right\| - \left\|d_s - d_q\right\| \le R \\ \Rightarrow &\left\|d_p - d_q\right\| - R \le R \end{aligned}$$

$$\Rightarrow \left\|d_p - d_q\right\| \le 2R \tag{1.5}$$

And from *p* does not hear *q*, we have

$$\left\|d_p - d_q\right\| > R \tag{1.6}$$

Combining results (1.5) and (1.6), we have the mixed interval

$$R < \left\|d_p - d_q\right\| \le 2R \tag{1.7}$$

In summary, by using the observance and non observance of data from certain nodes, we can put an upper bound (and in some cases a lower bound too) on the distance estimates to a particular node, with negligible overhead. In the case of localizing a target, this information can be very useful to quickly bound a target to an upper and lower limit and then fine tune the estimate. These bounds also serve to disambiguate and reject impractical values quickly and easily without incurring much additional overhead. One important point to note is the distribution and density of watchdogs. If they are not uniformly distributed, then an adversary can take advantage of neighborhoods where watchdog nodes are sparse. If they are either too large or too small in number, they will not be very effective. A very large number of watchdogs is counterproductive since these nodes will add to the cost. On the other hand, having less number of watchdogs will fall short of serving the purpose. We leave this study of tradeoffs between their deployment density and distribution for the future.

Chapter 3 Analysis of Robust Target Localization

In this chapter, we analyze the resistance of our protocol against false data attacks from a robustness perspective. In other words, we examine how the protocol reacts to malicious behavior that is not detectable, i.e., that which has been crafted to look like noise, systemic variation, or environmental influence or cleverly crafted fraudulent data sand not specifically adversarial behavior. We quantify the maximum adversarial behavior the system can tolerate, and derive an expression for the least upper bound expected error under such circumstances. We then cast this as a multivariate optimization problem and solve it for the degree of robustness achieved by the protocol. We derive a lower bound on the number of particles that must be active in the particle filter in order to ensure that the solution is always δ-robust. Additionally, we examine the dependencies associated with this solution and their effects on the outcome. We then examine the various security properties of the system and showed that they are not violated in any run (malicious or non malicious) of the protocol.

WSNs are data centric networks, the prime objective being collection and processing of data say, for example, for strategic or military decision making. An intelligent adversary need not attempt sophisticated attacks to dislodge the network. It can intelligently craft bogus data acceptable to the system and negatively influence the outcome of the system or protocol. This in turn will negatively influence strategic decisions themselves. When data itself is falsified, strong cryptographic protocols cannot provide any resilience. Integrity check mechanisms will also fail because they only check for in-transit message corruption. They cannot prevent against falsified data in a legitimately created perfectly valid message. Source authentication also does not help as corrupt insider sources will be able to authenticate themselves successfully. Broadly, there are only two ways to resist the objective of such false data attacks. One is to be able to detect that the data is false and discard it, or second, to be robust against false data. In practice, it is hard to design perfectly robust protocols. We evaluate our protocol under worst case attack and show that it achieves δ-degree of robustness.

Recall our notion that the objective of an attack on a data centric network is to cause the network to either report incorrect or no data, or in the case of intelligence (inferential) operations, arrive at an incorrect or inconclusive outcome. In the case of the WSN under our consideration, an attacker's objective could be to adversely influence the resulting target estimate or result in no estimate of the target's position at all. This can be achieved through a variety of ways, directly or indirectly. We attempt to briefly classify these attacks. We refer to attacks like signal degradation, deliberately jamming a node's signal or withholding a measurement as *physical* attacks and attacks on the protocol like spoofing a leader's or aggregator's identity, lying about other nodes, misrepresenting one's location, falsely accusing honest nodes of malicious behavior, replay attacks, etc as *protocol* attacks. We refer to a third class of attacks as *data centric* attacks which includes attacks where a node sends incorrect belief values to a neighboring aggregator node, incorrect measurement values to the aggregator node, adversary increasing the frequency of its inputs to drown out the target estimate at the aggregator, or simply skew it towards a false outcome. Different classes of attacks warrant different treatments. While we do not consider physical attacks at this time, our protocol is capable, in some situations, of identifying if a direct physical attack is under way in some subsections of the deployment (using watchdogs, for example). We perform a security analysis of common protocol attacks that our protocol is susceptible too, and a robustness analysis to evaluate the effect of data centric attacks.

Undesirable behavior can be classified into *malfunctioning*, *malicious* and *compromised* nodes. They primarily differ in their intent, and hence have different detection probabilities in our model. We refer to a node as *malfunctioning* when the node disobeys protocol or supplies arbitrary measurements, without intent to harm the outcome or the working of the protocol. *Selfish* nodes come under this category too though they technically are not malfunctioning. We refer to a node as simply being *malicious* if it is not part of the sensing model, and is working with bad intent towards deliberately degrading and throwing the estimate off track. Finally, we refer to a node as being *compromised* if it is an authenticated party in the network and working with bad intent towards degrading the estimate.

Few threats and attacks that can cause an incorrect estimation of the target's position are enumerated below. We also discuss how these attacks are currently countered by our scheme.

3.1 Attack Model

We represent an honest principal by HP_b_i, and a dishonest one by DP_b_i, such that $\bigcup_{i \in m} \left(HP\_b_i \cup DP\_b_i \right) \in P_B$, is the set of all interacting type B principals in a cluster *i*. DP_b$_i$, includes all malicious and compromised nodes and HP_b$_i$ includes all non malicious nodes. If C_i represents the cluster *i* after a successful run of the cluster formation algorithm, then we simply state the following lemma without proof.

Lemma 3: *The precondition to a data centric attack, is satisfied if at the end of a successful run of the cluster formation algorithm,* $DP\_b_i \in C_i$ $\forall DP\_b_i \in P_B$, and $\forall i \in n$.

Proof: Initially, all nodes are assumed to be benign. Under this assumption, no dishonest node will be part of the protocol, and hence no data centric attacks can be launched. This is because our protocol only accepts data from sources that posses valid cryptographic keys required to sign message (1) in Phase 1 and message (1) in Phase 2 (Refer to protocol specification in Section 2.2.4.2). Therefore, it follows that if a dishonest node (malicious or compromised) is able to successfully penetrate the cluster formation process, then a necessary precondition to launch a data centric attack has been met.

We represent the actions of a malicious node that is a part of the cluster, within a single set of adversarial actions Λ. This includes disruptive actions such as non-forwarding, dropping, modifying data content, replaying, flooding, delay time-sensitive data packets selectively or inject bogus packets into the particle stream.

We further define Λ_d as the subset of actions in Λ which result in an identifiable unsuccessful run, i.e., those actions that have effects that hold in the next transition state, and which may or may not result in a successful termination of the protocol. .

A successful attack action $\overline{a} \in \Lambda$ requires that the preconditions of $\overline{a}$ hold at the start of the attack in say, state s1 and the effects of $\overline{a}$ hold in a subsequent state s2. S_{α} is one such precondition. If $\overline{a} \in \Lambda_d \in \Lambda$ then, the attack can be detected with non-negligible probability p_{Λ_d}. If $\overline{a} \in \Lambda \setminus \Lambda_d$ then, our protocol should result in a successful run with the target location output being within the tolerable MSE range, i.e., $\varepsilon^2 \leq \xi^2 \leq \xi_{MAX}^2$ with probability $1 - p_{\Lambda_d}$

Lemma 4: *In a scenario with multiple independent acting adversaries, the number of Byzantine nodes that can be tolerated depends upon the detection threshold* p_{Λ_d}*, and the failure rate γ, and is given by* $DP\_b_i \leq \dfrac{P_{B_i}\left[1-3\gamma\left(1-p_{\Lambda_d}\right)\right]-1}{3\left(1-p_{\Lambda_d}\right)\left(1-\gamma\right)}$

Here, clearly, $\overline{a} \in \Lambda$. In the worst case scenario, all dishonest principals DP_b$_i$ and some percentage of honest principals that are malfunctioning will be part of the adversary set. If the fail rate of the devices being employed is γ then, γHP_b$_i$ are the honest parties that contribute to the adversary set.

We start with the classic Byzantine *two-thirds majority*[2] result that in order for the non-malicious result to prevail, the following equation must hold:

$$\gamma HP\_b_i + DP\_b_i \leq \frac{P_{B_i} - 1}{3} \tag{3.1}$$

[2] The Byzantine simple majority result cannot be applied in our case since in order for the simple majority result to prevail, there needs to be an infrastructure that allows signed messages from each node B$_{ij}$ to prove not just message integrity but also authentication (source authentication)., for example like in a public key infrastructure. Since this is not the case in our scheme, we cannot use the simple majority result.

Factoring in the probability p_{Λ_d} that a malicious data is detected if $\varepsilon^2 \le \xi^2 \le \xi^2_{MAX}$ thereby resulting in the removing out of band data, implies that γHP_b_i can be detected with non negligible probability p_{Λ_d} if their estimates deviate more than ξ^2_{MAX}, say. Therefore, $\gamma HP\_b_i(1-p_{\Lambda_d})$ honest nodes are not detected and eliminated from the algorithms computation process.

Similarly for dishonest principals, we have $DP\_b_i(1-p_{\Lambda_d})$ as the number of participating dishonest nodes in a single cluster *i*.

Therefore, the equation from above effectively becomes

$$\gamma HP\_b_i + DP\_b_i \le \frac{P_{B_i}-1}{3(1-p_{\Lambda_d})}$$

which gives us

$$DP\_b_i \le \frac{P_{B_i}\left[1-3\gamma\left(1-p_{\Lambda_d}\right)\right]-1}{3\left(1-p_{\Lambda_d}\right)(1-\gamma)} \qquad (3.2)$$

Clearly, as the detection probability p_{Λ_d} increases the right hand side of Eq.(3.2) increases showing that the ability of the algorithm to withstand Byzantine behavior improves. In other words, we can tolerate more dishonest nodes and as a result become more resilient to false data as detection probability increases.

Figure 6 below shows the effect of varying p_{Λ_d} on the number of dishonest nodes tolerated (DP) for different values of γ. (γ is varied from 0.01 to 0.5, P_B=100 nodes, and p_{Λ_d} is varied from 0 to 0.6)

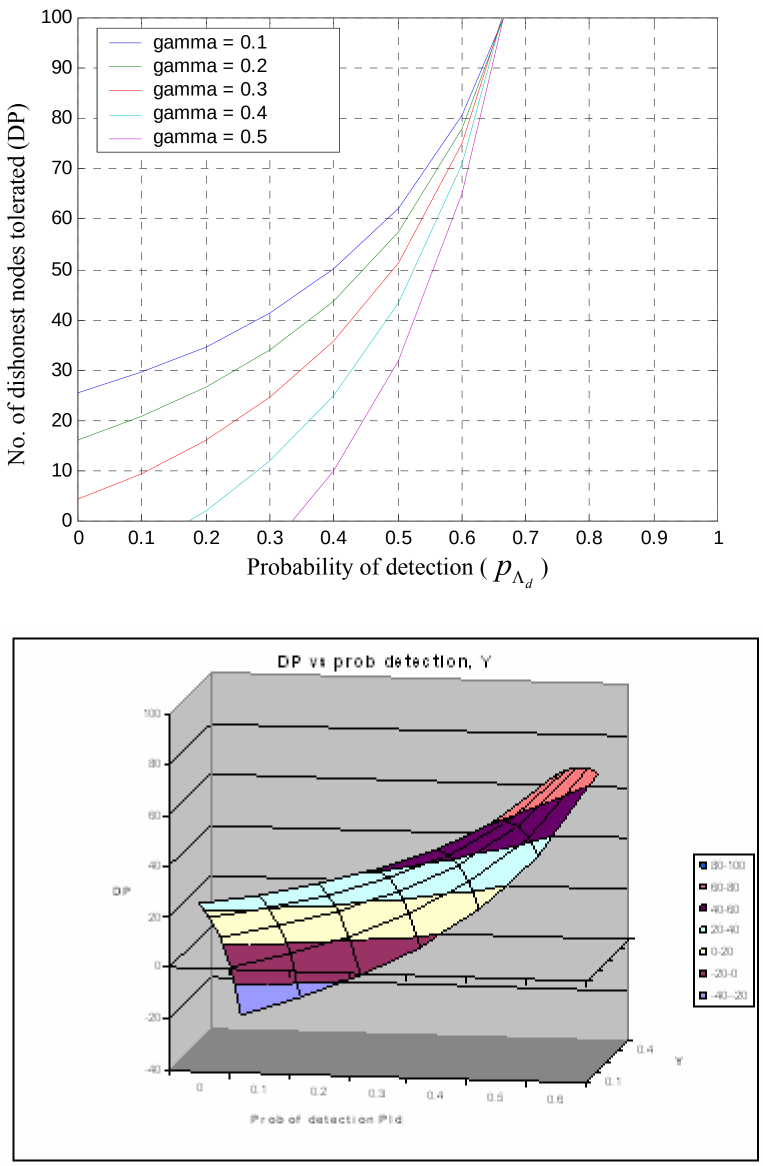


Figure 6: No. of dishonest nodes tolerated (DP) vs. Probability of detection ($P_{\Lambda d}$) & fail rate (γ)

Here we observe that as detection ability p_{Λ_d} increases, the system can tolerate more number of malicious entities. Detection ability p_{Λ_d} depends on the data validation mechanism employed as well on the particle filters ability to track and filter mis-data as noise. This directly translates to the particle filter threshold when it tracks performance vs. collective disturbances. The threshold can be modulated by modifying the time period of observations, reducing the update period, and increasing number of particles collected per time window by modulating the sleep awake distribution criteria of each cluster. Effects of these characteristics on the threshold and hence performance are discussed in a later section.

3.2 Robustness Analysis

To prove robustness we model the behavior of the protocol, its assumptions and dependencies as well as other sources of uncertainty. We then assess if the uncertain system satisfies a desirable property P for every admissible value of the uncertainty.

In our case, we define a robustness parameter δ, which is measured against location error as the desired property P. We model the uncertainties and observe their effect on the location error for all admissible values of the uncertainties in the constraint space. We examine the output for the worst-case malice from adversary within the given time window and given neighborhood. At this point, we would like to point out that the adversary is limited to the immediate neighborhood.

Input to tracking system is vector $x \in X$

Vector x is a set of observations from various entities in the neighborhood. Therefore, input is a continuous stream of input vectors $X = \{x[1], x[2], .., x[N]\}$ the collection of which at t+T (time window) results in the output position estimate $Z_{[t+T]}$.

If $Z_{[t+T]}$ is within δ of the true position $Z^*_{[t+T]}$, then the tracking estimate is useful.

We say that $Z_{[t+nT]}$ *is δ-robust if* $Z_{[t+nT]}$ *is* δ *within* $Z^*_{[t+nT]}$ *for all* $t, n \in R$ *for worst case malicious input.*

Phase 1: Aggregation and Position Estimation Phase at A$_i$

For every type B sensor that senses an active target in cell *i*:

(1) $B_{i*} \rightarrow A_i : \mathbb{M} : \left(< p.f.i >_j, t_k, B_{ij}, A_i, \{< p.f.i >_j, t_k, B_{ij}, A_i\} SK_{B_{ij}} \right) PK_{A_i}$

(2) $A_i : \forall j \;\; check < p.f.i >_j$ such that $\varepsilon^2 \leq \xi \leq \xi_{\max}^2$ AND $\mathfrak{V}(\mathbb{M})$ *is true*

If *true*, $A_i : \prod_{j=1}^{m} < p.f.i >_j = \langle Target\_Location \mid \theta \rangle_i$

(3) Process repeats from (1) for next time interval θ

Phase 2: Hand off Phase at A

(1) $A_i \rightarrow * : \left\{ A_i, t_k, \langle Target\_Location \mid \theta \rangle_i \right\} SK_{A_i}$

(2) A_{i+1} : If $\left\{ A_i, t_k, \langle Target\_Location \mid \theta \rangle_i \right\} SK_{A_i}$ is *true*,

$< p.e >_{(i+1)-1} = \langle Target\_Location \mid \theta \rangle$

Verification Function $\mathfrak{V}(\mathbb{M})$

Input: Message $\mathbb{M}$

If $\left(v \in [\hat{p} - v_l, \hat{p} + v_u] \,\&\, TS(\mathbb{M}) \,\&\, (sig\_val) \,\&\, N2N\_flag = 0 \right)$

Output (1); (Message verification PASS)

Else Output (0); (Message verification FAIL)

Figure 7: Secure and Robust Location Determination Protocol

Let us assume adversary can craft malicious input (attack feature) $\hat{x}[i]$. If the adversary's rate of sending attack particles is $\hat{p}$, probability p_α of setting its attack particles at rate $v = [\hat{p} - v_l, \hat{p} + v_u]$ where v_l and v_u are the lower and upper bounds on the allowable rate of particle inputs without raising suspicion and being detected, and probability p_β that the attack particles are set within the allowable error range, so as to not trigger outlier detection and rejection.

```
Adversary algorithm $\mathfrak{F}\left(p_\alpha, p_\beta\right)$
        Select $p_\alpha, \nu \in \left[\hat{p}-\nu_l, \hat{p}+\nu_u\right]$
        Select $p_\beta \in \left[0, p_{\Lambda_d}\right]$ such that $\xi_x \in \left[\varepsilon^2, \xi_{\max}^2\right]$
            Select $\xi_x \in \left[\varepsilon^2, \xi_{\max}^2\right]$
        $\hat{x} \leftarrow \mathfrak{E}\left(\langle p.f.c\rangle_{j|x}, \xi_x\right)$
        Output $\left(\hat{x}\right)$
```

If $\overline{a} \in \Lambda_d \in \Lambda$, attack detected with non negligible probability p_{Λ_d}

If $\overline{a} \in \Lambda \setminus \Lambda_d \Rightarrow$ protocol run successful with $\varepsilon^2 \leq \xi \leq \xi_{\max}^2$

Adversary outputs $\hat{x} \leftarrow \mathfrak{F}\left(p_\alpha, p_\beta\right)$

Definition 3-1: *At the end of a successful run, if adversary output* $\hat{x} \leftarrow \mathfrak{F}\left(p_\alpha, p_\beta\right)$ *produces no more than* δ *deviation from the estimate of the true position, then we say that the system is* δ*-robust.*

Let us calculate the effect of a single adversary injecting intelligently crafted malicious input to the tracking system at A. If the adversary does not exceed the rate of injection of particles $\nu = \left[\hat{p} - \nu_l, \hat{p} + \nu_u\right]$ then all its inputs $\hat{X} = \{\hat{x}[1], \hat{x}[2], .., \hat{x}[..]\}$ will be accepted with probability $1 - p_{\Lambda_d}$.

Therefore, at the end of the three stages, we calculate the derived position estimate $\hat{Z}_{[t+nT]}$ as:

After Initialization /Sample Step the M particles, denoted by $\left\{x_n^{(m)}\right\}_{m=1}^{M}$ i.e., for m = 1….M are initialized by drawing samples from the initial distribution: $p\left(x_n \mid x_n = x_{n-1}^m\right)$ for every time instant *n*. x_{n-1}^m is the previous observation. Every importance weight is initialized to $w_0^{(m)} = {}^{1}\!/_{M}$.

With probability p_β malicious input $\left\{\hat{x}_n^{(m)}\right\}_{m=1}^{\nu.T}$ with rate $\nu = \left[\hat{p} - \nu_l, \hat{p} + \nu_u\right]$ and MSE $\xi \le \xi_{\max}^2$ adversary can causes $\hat{N} = \hat{\nu}.T$ particles within the time window T to be accepted by the system. In the worst case scenario, $N = \hat{N}$ in which case, the output will be maximally deviated from the true estimate, and is not of concern to us. On the basis of the Byzantine result derived in the earlier sub-section, we can expect the worst case as a set of particles drawn with 50% malicious input.

In the second step, the proposed trajectory becomes $\hat{\pi}\left(\hat{x}_{1:k}\right) = \pi\left(x_1 \mid z_1\right)\prod_{k=1}^{n}\pi\left(\hat{x}_k \mid \hat{x}_{1:k-1}, z_{1:k}\right)$

And finally, after the resampling step, error propagation calculations show that the cumulative error in the estimate is as follows. Since we know the uncertainties parametrically from when they were introduced, we can estimate the uncertainties associated with the estimate as well.

$$\Delta\pi = \left(\sum_{i=1}^{n} \left(\frac{\partial\pi}{\partial x_i} \cdot \frac{\partial\pi}{\partial z_{i-1}} \right)^2 \right)^{\frac{1}{2}}$$

At each stage of the algorithm, the approximation admits a mean square error on the order of the number of particles.

Finally estimate $Z_{[t+nT]}$ becomes $\hat{Z}_{[t+nT]} = Z_{[t+nT]} + \Delta\hat{Z}_{[t+nT]}$

From here, we need to answer two questions. Firstly, what is the effect of this error on the output? If the error is not greater than the allowed tolerance δ, then the system is δ-robust as per *Definition 3-1*. Secondly, what is the upper bound on the frequency of crafted observations to cause the error to still be $\xi \le \xi^2_{\max}$ and $\Delta\hat{Z}_{[t+nT]} < \delta$, if any?

The probability distributions of the random variables and mean square error for the given system are known to converge. We therefore begin with the result that: $\lim_{N\to\infty} \pi_t^N = \pi_t \, (a.s)$ and again, at each stage of the algorithm, as the approximation admits a mean square error on the order of the number of particles, we can calculate an upper bound on the approximate mean square error introduced since the number of malicious particles is bound by the rate of delivery of the particles as determined by the adversary. In a given attempt, in the worst case an adversary can input atmost $\hat{v}.T$ malicious particles provided the input is within the tolerable error limit (to avoid being dropped). Thus the bounded error is a function of the input rate, the time window of operation and the probability of undetectability.

Lemma 5: *If the probability that particles pass undetected through the Byzantine detection and agreement algorithm is* p_{χ}*, and* p_{β} *is the probability that the attack particles are set within the allowable error range, then the maximum likelihood of undetectability becomes*

$$p_{\Lambda_{ud}} = \frac{p_{\chi_1} p_{\beta}}{p_{\beta}(p_{\chi_1} - p_{\chi_2}) + p_{\chi_2}}$$

Proof: We can easily derive the result above using Bayes conditional theory as

$$p_{\Lambda_{ud}} = \frac{p_{\chi_1} p_{\beta}}{p_{\chi_1} p_{\beta} + p_{\chi_2}(1 - p_{\beta})}$$

Rearranging, we get

$$p_{\Lambda_{ud}} = \frac{p_{\chi_1} p_{\beta}}{p_{\beta}(p_{\chi_1} - p_{\chi_2}) + p_{\chi_2}} \qquad \textit{(3.3)}$$

We have already defined our condition for robustness assessment in *Definition 3-1*. We have also seen briefly that the position estimate as well as the location error is dependent upon a few parameters like number of particles, node density, frequency of incoming particles, the time window of the filter etc. For now, all these parameters are variable in our setup. Solving this problem, therefore, becomes a multivariate optimization problem.

We can solve this problem in a few different ways. Commonly used optimization techniques are game theory/ decision theory approach, and parametric optimization approach. The parametric approach is better suited to our problem. One way to solve this problem parametrically is by combining all the parameters into a single parameter and optimizing the utility function (location error) in our case against it. We begin optimization by breaking the problem down into independent components each of which can be singularly optimized. The additive property of optimal solutions for independent events implies that the summation of these components will provide the optimal solution to the problem.

Lemma 6: *Given an initial estimate of location, initial parameters* $p_{\Lambda_{ud}}$ *,* p_{Λ} *and a derivable constant* $c_{t|t}$*, the least upper bound* δ *such that the expected error of the system is* δ*–robust is given by* $\delta \geq \inf E[e_1](p) + E[e_1](p') - E[e_3]$ *which is the difference of the MSE with and without the presence of malice.*

Proof:

We start with the definition of δ.

By definition, δ is the deviation in location error observed due to the introduction of malice in the tracking system, and is given by

$$\delta = \begin{bmatrix} \textit{Expected error} \\ \textit{with maximum} \\ \textit{malicious input} \end{bmatrix} \begin{pmatrix} \textit{Probability of} \\ \textit{malicious input} \end{pmatrix} - \begin{bmatrix} \textit{Expected error} \\ \textit{with no malicious} \\ \textit{input} \end{bmatrix} \begin{pmatrix} \textit{Probability of} \\ \textit{no malicious} \\ \textit{input} \end{pmatrix}$$

$$i.e. \quad \delta = f_1^* p_{\Lambda} - f_1(1 - p_{\Lambda}) \qquad (3.4)$$

Where f_1^* is the expected error with malicious input, given by $f_1^* = E[e_1].p + E[e_2].p'$ where $E[e_1]$ is the expected error with undetectable malicious input, $E\left[e_2\right]$ is the expected error with detectable malicious input and p is the associated probability. We will show shortly that f_1^* is actually the optimum value of the expected error with malicious input.

Similarly, f_1 is the expected error without malicious input, given by $f_1 = E\left[e_3\right].p\,|_{=1}$ where $E\left[e_3\right]$ is the expected error without any malicious input, and p=1.

Thus, we have

$$f_1^* = E\left[e_1\right] .p_{\Lambda_{ud}} + E\left[e_2\right].(1-p_{\Lambda_{ud}}) \tag{3.5}$$

Since f_1 is the expected error without malice, it follows that $f_1 = E\left[e_3\right] = \varepsilon^2$, a predefined semi-variant constraint which is the baseline error, and may be dictated by the sensitivity and requirements of the tracking system application.

E[e₂] being the expected error with detectable malicious input, it is inversely proportional to the number of particles *N*, and can be calculated as:

$$E\left[e_2\right] = C + \frac{C}{\sqrt{N}} \le c_{t|t}\frac{\|\varphi\|^2}{N}$$

for any bounded function φ and constants C, and $c_{t|t}$ (at time t.) The remaining term in f_1^* (Eq. (3.5)) i.e. the $E\left[e_1\right]$ term is an optimization problem in itself, where we require the minimum $E\left[e_1\right]$ for maximum malice in order for δ-robustness to be true. We now try to solve for this value. To get an intuitive idea, we present $E\left[e_1\right]$ in a min max setting:

$$\min_{\xi\in[\varepsilon^2,\xi_{\max}^2]\,\nu=[\hat{p}-\nu_l,\hat{p}+\nu_u]} \quad \max_{p_{\Lambda_{ud}},\mathfrak{F}(p_\alpha,p_\beta),\hat{N}} E^{p_\alpha,p_\beta,p_{\chi_i}\hat{N}}\left[e_1\right]^*$$

Frequency ν, as we have seen before, is already bounded by the system to $\nu=\left[\hat{p}-\nu_l,\hat{p}+\nu_u\right]$ which provides the range of error fluctuation. From this relation we can see that if the number of non malicious particles is increased, not through the rate but through increased number of participants (particle density) we can further reduce the impact of the malicious input. Intuitively moving a step further, if the particle density is increased in a non uniform manner (skewed towards cliques) the same nodes can help multiple clusters and disambiguate mis-data without increasing node participation. This is an important result. We can use frequency limitation, particle density and number of particles to narrow the constraint space and further eliminate solutions of the min max function above that fall outside of this space. This gives us a smaller solution space and reduces complexity by an order.

We will use Simultaneous Perturbation Stochastic Approximation (SPSA) method introduced by [37] to solve for $E\left[e_l\right]$ where the gradient is approximated using a randomized finite difference method. Compared to the standard finite difference method, SPSA is advantageous in that we only need to compute *2* estimates of the objective function per iteration, irrespective of p- the dimension of θ, instead of *2p* estimates. Also, under general conditions, SPSA and standard finite difference stochastic approximation methods achieve the same level of statistical accuracy for a given number of iterations even though SPSA requires 1/p times the measurements. SPSA converges to the optimal solution within a given level of accuracy (in our case δ) with p times fewer measurements of the objective function and is ideally suited to low computational cost, speed dependent real-time applications where both time and accuracy are important, and where all uncertainties and non-linearities cannot be accurately modeled.

The use of the SPSA technique results in an elegant optimization algorithm for SMC methods. For simplicity and clarity, we will follow the notations used in standard SPSA literature [37][38][39].

Assume (general assumption) the SMC algorithm is parameterized smoothly by a parameter $\theta \in \Theta$ where Θ is an open subset of $\mathcal{R}^m$. Under stability assumptions on the dynamic model of interest, the particles, their corresponding weights, the true state and the observation of the system form a homogenous and ergodic Markov chain. Performance measure can thus be defined as the expectation of a cost function with respect to the invariant distribution of this Markov chain parameterized by θ.

We now define the time average cost function $J(\theta)$ for our system as the expected error with undetectable malicious input $E[e_1]$, and represented as:

$$J(\theta) = E_\theta\left[f\left(Z, X, \tilde{X}, \tilde{W}\right)\right] = E[e_1]$$

where the expectation is with respect to the invariant distribution of the Markov chain $\left(Z, X, \tilde{X}, \tilde{W}\right)$ corresponding to the set of observations, true and estimated states, and estimated weights of the system. We are interested in estimating $\theta^* = \arg\min J(\theta)$ which will give us the desired minimum value of $E[e_1]$. Here, it is worth noting that the cost function is independent of the observations since the observation process is being integrated out. One important practical consequence of this is that the SMC algorithm can be alternately optimized off-line by simulating the data and then use the resulting optimized algorithm on real data.

The mean square error represented in the SPSA format is

$$\text{MSE } f\left(Z_n, X_n, \tilde{X}_n, \tilde{W}_n\right) = \left(X_n - \sum_{k=1}^{N} \tilde{X}_{n,k} \tilde{W}_{n,k}\right)^2 \qquad (3.6)$$

We are interested in estimating $\theta^* = \arg\min J(\theta)$.Using SPSA, the problem of minimizing a differentiable cost function $J(\theta)$, where $\theta \in \Theta \subseteq \mathcal{R}^m$ effectively translates into finding the zeros of the gradient $\nabla J(\theta)$. Recursively, we can estimate θ^* such that $\nabla J(\theta) = 0$ as follows:

$$\theta_n = \theta_{n-1} - \gamma_n \hat{\nabla} J_n \qquad (3.7)$$

Where $\hat{\nabla} J_n$ is the noise corrupted estimate of gradient $\nabla J(\theta)$ estimated at the point θ_{n-1} and $[\gamma_n]$ denotes a sequence of positive scalars such that $\gamma_n \to 0$ and $\sum_{n=1}^{\infty} \gamma_n \to \infty$. Under appropriate conditions, the iteration in (3.7) will almost sure (a.s) converge to θ^*.

In order to solve (3.7) we need to obtain the gradient estimate $\hat{\nabla} J_n$. In SPSA the gradient approximation is done via finite difference using the estimates of the cost function. Briefly, all elements of θ_{n-1} will be varied randomly simultaneously (hence the name simultaneous perturbation) to obtain two estimates of the cost function $J(\theta_{n-1} \pm \textit{perturbation})$. Only two estimates are required regardless of the dimension

p of the parameter. For a two-sided gradient approximation, the gradient estimate $\hat{\nabla} J_n = \left(\hat{\nabla} J_{n,1}, \hat{\nabla} J_{n,2}, \ldots\ldots, \hat{\nabla} J_{n,p}\right)^T$ is given by

$$\hat{\nabla} J_{n,i} = \frac{\hat{J}(\theta_{n-1} + c_n \Delta_n) - \hat{J}(\theta_{n-1} - c_n \Delta_n)}{2 c_n \Delta_{n,i}}$$

Where $\{c_n\}$ denotes a sequence of positive scalars such that $c_n \to 0$ and $\Delta_n = \left(\Delta_{n,1}, \Delta_{n,2}, \ldots., \Delta_{n,p}\right)$ is an p-dimensional random perturbation vector. Algorithm parameters $\gamma_n, c_n,$ and Δ_n require careful selection to ensure convergence. The γ_n and c_n sequence generally take the form of $\gamma_n = \frac{a}{(A+n)^\alpha}$ and $c_n = \frac{c}{n^\beta}$ respectively with non-negative coefficients a, c, A, α and β.

We find that $\alpha = 0.602$ and $\beta = 0.101$ (recommended values) are practically effective in our case too. Δ_n is a symmetric Bernoulli ± 1 distribution. We set a and c low initially (recommended for high noise settings) and our final stable values used were a= 0.16, c= 1, A=100, $\alpha = 0.602$ and β=0.101 .

We now incorporate the two-sided SPSA optimization algorithm into our SMC framework. Recall the steps involved in the filtering algorithm from Section 2.3.3

Step 1: *Sequential importance sampling with SPSA*

For n=1 to N, sample $\tilde{X}_{n,k} \sim q\theta_{n-1}\left(\tilde{X}_{n-1,k}, Z_n, \bullet\right)$ is the perturbed observation function. We then compute the normalized importance weights with perturbation as

$$\tilde{W}_{n,k} \propto \tilde{W}_{n-1,k} \frac{g\left(Z_n \mid \tilde{X}_{n,k}\right) f\left(\tilde{X}_{n,k} \mid \tilde{X}_{n-1,k}\right)}{q\theta_{n-1}\left(\tilde{X}_{n-1,k}, Z_n, \tilde{X}_{n,k}\right)}$$

We now evaluate the cost function.

First, we generate an p-dimensional simultaneous perturbation vector Δ_n and compute

$(\theta_{n-1} - c_n\Delta_n)$ and $(\theta_{n-1} + c_n\Delta_n)$

For k=1 to N, sample . $\tilde{X}^{+}_{n,k} \sim q\theta_{n-1} + c_n\Delta_n\left(\tilde{X}_{n-1,k}, Z_n, \bullet\right)$

Compute the normalized importance weights as

$$\tilde{W}^{+}_{n,k} \propto \tilde{W}_{n-1,k} \frac{g\left(Z_n \mid \tilde{X}^{+}_{n,k}\right) f\left(\tilde{X}^{+}_{n,k} \mid \tilde{X}_{n-1,k}\right)}{q\theta_{n-1} + c_n\Delta_n\left(\tilde{X}_{n-1,k}, Z_n, \tilde{X}^{+}_{n,k}\right)}$$

Again, for k=1 to N, sample . $\tilde{X}^{-}_{n,k} \sim q\theta_{n-1} - c_n\Delta_n\left(\tilde{X}_{n-1,k}, Z_n, \bullet\right)$

and compute the normalized importance weights as

$$\tilde{W}^{-}_{n,k} \propto \tilde{W}_{n-1,k} \frac{g\left(Z_n \mid \tilde{X}^{-}_{n,k}\right) f\left(\tilde{X}^{-}_{n,k} \mid \tilde{X}_{n-1,k}\right)}{q\theta_{n-1} - c_n\Delta_n\left(\tilde{X}_{n-1,k}, Z_n, \tilde{X}^{-}_{n,k}\right)}$$

We evaluate the cost function $J(\theta_{n-1} + c_n\Delta_n)$ and $J(\theta_{n-1} - c_n\Delta_n)$ from $\left\{\tilde{X}^{+}, \tilde{W}^{+}\right\} and \left\{\tilde{X}^{-}, \tilde{W}^{-}\right\}$ respectively.

Step 3: Gradient approximation

For i=1 to p, we evaluate the gradient components as

$$\hat{\nabla} J_{n,i} = \frac{\hat{J}(\theta_{n-1} + c_n\Delta_n) - \hat{J}(\theta_{n-1} - c_n\Delta_n)}{2c_n\Delta_{n,i}}$$

Step 4: Parameter update

Update θ_n to the new value $\theta_n = \theta_{n-1} - \gamma_n \hat{\nabla} J_n$

Step 5: Sampling

Multiply (Discard) particles $\tilde{X}_n$ with respect to high/low importance weights $\tilde{W}_n$ to obtain N particle $\tilde{X}_n$.

It is possible to improve the algorithm in many ways, for example, by using common random number or other numerical approximates like iterates averaging to reduce the variance of the gradient estimate. The idea behind it being to introduce strong correlation between our estimates of $J(\theta_{n-1} - c_n\Delta_n)$ and $J(\theta_{n-1} + c_n\Delta_n)$ so as to reduce the variance. For further details and improvements upon SPSA we refer the reader to [39].

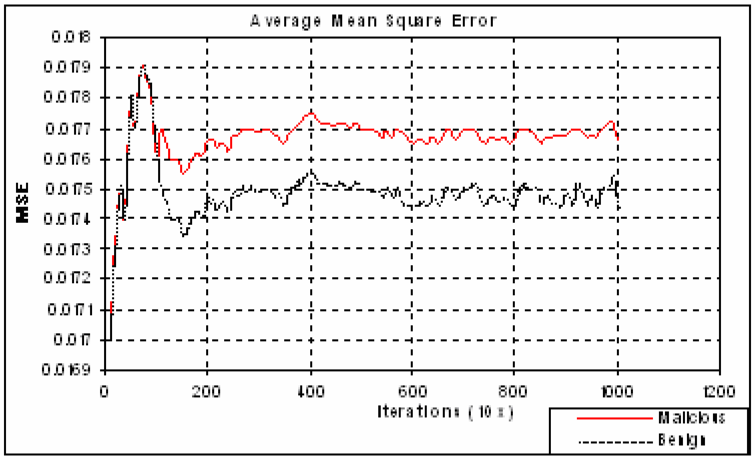


Figure 8 Sequence of average MSE estimates over time

The results obtained for this simulation are plotted above in Figure 8. for *a*= 0.16, *c*= 1, *A*=100, α = 0.602 , β =0.101 and for N = 100. Clearly, one can see that the MSE with maximum undetectable malicious input almost mimics the response of the original system, that is, there is no non-linear loss observed. Further, the difference in the error is almost constant, except for a few minor exceptions. If we set this finite difference in the error as $r.\delta$ where r is a safety factor $0.4 < r < 0.8$, then the system is guaranteed to be δ-robust as long as all the uncertainties modeled above do not violate their physical constraints.

We now try to answer the second question we asked earlier, *what is the upper bound on the frequency of crafted observations to cause the error to still be* $\xi \leq \xi_{\max}^2$ *and* $\Delta\hat{Z}_{[t+nT]} < \delta$*, if any?* This answer can be analytically derived. We formulate and prove it as a lemma thus:

Lemma 7: *For a given filter with known mean and variances for the importance function generating samples, the supremum value of frequency of crafted observations that can limit the maximum error in the approximation to under* $\xi_{\max}^2$ *and* $\Delta\hat{Z}_{[t+nT]} < \delta$ *is given by* $\min[\nu_{threshold}, \nu + \nu_u]$ where $\nu_{threshold} \leq \dfrac{N - N_{IS}}{T}$ and N_{IS} is the number of samples coming from the importance function.

Proof: We make use of KL distance sampling (Kullback-Leibler distance)[18] method from statistical theory that can be used to adaptively estimate the number of particles to represent the target posterior distribution without increasing the overhead to the normal operation of the filter.

KL distance sampling is used to adaptively estimate the number of samples needed to put an upper bound on the error of the particle filter. The error is measured by the KL distance between the true posterior distribution and the empirical distribution, which is a well known nonparametric maximum likelihood estimate. It is a standard measure of the difference between two probability distributions. It can never be negative, but a zero value is indicative of identical distributions.

The likelihood ratio converges to a chi-square distribution, and the bound for the number of particles N represented as:

$$N > \frac{1}{2\varepsilon_{KL}} \chi^2_{k-1,1-\delta} \qquad \textit{(3.8)}$$

where ε_{KL} is the upper bound for the error given by the KL distance, and $1-\delta$ is the quartile of the χ^2 distribution with *k-1* degrees of freedom. Equation 3.7 can be further expanded using the Wilson Hilferty transformation [24] but for our proof purpose, the form of equation 3.7 above will suffice.

A slight drawback of using KL distance sampling is the underlying assumption that the samples always come from a true distribution which we assume to be free of malicious input. Recall that our particle filter samples are drawn from an importance function π. In an adversarial scenario, some of these samples can be corrupt and misleading. Therefore, in statistical terms, the quality of the match (or rather, mismatch) between this function and the true distribution determines the accuracy of the filter in the presence of malicious samples, and in turn, the suitable number of particles required to uphold the correct estimates. The bound given by KL distance sampling only uses information about the complexity of the true posterior, but it ignores any mismatch between the true and the proposal distribution.

KL distance sampling, thus, does not provide the answer to our question directly, but provides an excellent start. We now need to quantify the degradation in the estimation using samples from the importance function instead of a uniform empirical distribution. This will give us the bounds we are interested in. We are interested in accurately finding an equivalent number of samples from the (possibly flawed) importance density function as that from the true density function that captures the same amount of information. Relative numerical efficiency (RNE) helps us derive such an accurate bound and adjust the KL distance sampling estimate by relating the two samples.

RNE in the context of Monte Carlo (MC) integration, introduced by Geweke [33], provides an index to quantify the influence of sampling from an importance function. RNE allows us to compare the relative accuracy of solving an integral using samples coming from both the true and the proposal density. This gives us the effect of sampling from an importance function as opposed to a true distribution. We follow the approach of [5] to equate the variance of the estimator estimated using KL distance sampling and RNE as follows:

Using Sequential Monte Carlo (SMC) integration to estimate the mean value of the state ($E_{MC}(x)$), the variance of the estimator given by [1] becomes:

$$Var\left[E_{MC}^{N}(x)\right] = \frac{Var_{\rho}(x)}{N} \qquad (3.9)$$

where N is the number of samples coming from the true distribution $\rho(x)$ with no malicious input samples.

When the samples come from an importance function π, the variance of the estimator actually corresponds to the variance of Importance Sampling (IS). This is given by [33] :

$$Var\left[E_{IS}^{N}(x)\right] = \frac{E_{\pi}\left(\left(x - E_{p}(x)\right)^{2} w(x)^{2}\right)}{N_{IS}} = \frac{\sigma_{IS}^{2}}{N_{IS}} \qquad (3.10)$$

where N_{IS} is the number of samples coming from the importance function containing both malicious and non malicious input samples, and w is the associated weight attached to the incoming particles.

Equating the variance of both estimators allows us to achieve similar levels of accuracy. This in turn allows us to find a relation that quantifies the equivalence between samples from the true and the proposal density. Equating both variances (i.e., equations 3.8 and 3.9), we get

$$N = \frac{N_{IS} Var_{\rho}(x)}{\sigma_{IS}^{2}} \qquad (3.11)$$

Replacing (3.11) in *(3.8)* allows us to obtain the correct bound given by KL distance sampling when the samples do not come from the true distribution but from an importance function. Therefore, we get

$$N_{IS} > \frac{\sigma_{IS}^{2}}{Var_{\rho}(x)} \frac{1}{2\varepsilon_{KL}} \chi_{k-1,1-\delta}^{2} \qquad (3.12)$$

$Var_{\rho}(x)$ and σ_{IS}^{2} can be estimated in the standard manner as:

$$Var_p(x) = E_p\left(x^2\right) - E_p\left(x\right)^2 \approx \frac{\sum_{i=1}^{N} x_i^2 w_i}{\sum_{i=1}^{N} w_i} - E_p\left(x\right)^2 \qquad (3.13)$$

and

$$\sigma_{IS}^2 \approx \frac{\sum_{i=1}^{N} x_i^2 w_i^2}{\sum_{i=1}^{N} w_i} - 2\frac{\sum_{i=1}^{N} x_i\, w_i^2 E_p\left(x\right)}{\sum_{i=1}^{N} w_i} + \frac{\sum_{i=1}^{N} w_i^2 E_p\left(x\right)^2}{\sum_{i=1}^{N} w_i} \qquad (3.14)$$

respectively, with $E_p\left(x\right) = \frac{\sum_{i=1}^{N} x_i\, w_i}{\sum_{i=1}^{N} w_i}$

Note from the above equation that the order of complexity of the filter is always maintained at *O(N)* making this a low complexity and lightweight approximation inline with our theme.

Equation (3.12) gives us the bound for the number of particles that can keep the error under bound $\xi_{\max}^2$ (as a function of quartile values). As long as N_{IS} is greater than the right hand side of equation (3.12), maximum error $\xi_{\max}^2$ will be bound by the quartile value $1-\delta$. Further, the minimum number of particles that can limit the maximum allowable error in the approximation $N_{IS_{\min}}$ will be $N_{IS}+1$.

Since the total number of incoming particles in a time window *T* is $\nu.T$, it further implies that the above result holds as long as the total number of malicious particles $\hat{\nu}.T \le N - N_{IS}$. This becomes our critical threshold frequency $\nu_{threshold}$ and

$\nu_{threshold} \leq \frac{N - N_{IS}}{T}$ reduces the allowable range of ν to $[\nu - \nu_l, \min[\nu_{threshold}, \nu + \nu_u]]$

q.e.d

3.3 Security Properties of our Scheme

To build a secure protocol, we must understand what it means to be secure. Security means different things to different people. Therefore, we first define what we consider are security properties for our protocol. *Security properties* essentially are characteristics that applications, protocols or programs must satisfy in order to be valid for all reasonable and unreasonable inputs. The violation of a security property for any input implies a vulnerability. Conscious application of this input by someone possessing this knowledge constitutes a legitimate attack on the system.

In this section, we will specify various security properties of the protocol, ascertain that they are preserved in the face of an attack and understand boundary conditions and dependencies that exist, if any, for each of the security properties under consideration.

For any protocol or system under design, security properties fall under the following broad categories: freshness, authenticity, secrecy, non-intrusion and resilience.

1. Freshness: freshness means that messages sent and received in a session are generated and used in the same session. An attacker cannot use messages from previous (or future) sessions in the current session without being detected.

Our target tracking protocol takes, as input, individual measurements coming from various nodes in the present neighborhood of the target, assimilates them and produces a Bayesian likelihood type estimate of the current target position. Targets may be moving. From this it is obvious that measurements must be time sensitive. Only measurements that are close to each other in time can be aggregated in the same iteration. The duration for which measurements are considered to be part of the same iteration is known as the *time window*. Only measurements received during the same time window can be used to estimate target location corresponding to that time window. For online processing, measurements are generated and aggregated closely in time whereas for offline or *passive* processing, measurements still need to be aggregated per window, but there may be a gap between the time measurements are taken and when they are aggregated. In this work, we consider the former case. The latter case is only a slight modification of the former and can be easily derived from the former.

Therefore, the freshness security property for our protocol states that a message containing a measurement regarding a target should only be acceptable if it is valid in the current time window. Since we assume that the nodes of the distributed system are loosely synchronized over single hop, the freshness property results in that time stamps associated with measurement messages should not be alterable without detection i.e., the time stamp must be integrity protected along with the rest of the message during transit. Messages should not be recorded and replayed later in a manner that is undetectable. Also within a time window messages need not be ordered since their ordering within the window is not relevant to the estimator. The size of the window of validity is tightly coupled with the synchronization degree (or rather tolerable synchronization error σ). If the window is smaller, the nodes need to synchronize more often, and more tightly (smaller allowable σ). If the window is larger, the nodes can synchronize less often and may have slightly larger values of tolerable synchronization error σ. This indirectly influences cost as well as storage since longer time windows result in larger storage requirement. We leave this as a future study goal.

2. Message Authenticity: Measurement messages should not be altered or corrupted in transit in a manner that is undetectable. Altered and corrupt messages should not be included in computing the target location estimate. Further, messages that come from sources that possess valid authentication material (like shared secrets) are accepted for use in computing the target location estimate. Non repudiation is not required. Note that based on this definition, select replays (fairly recent ones) of authentic messages are accepted by the system.
3. Uniqueness: Each node can take only one measurement at a given time instant. Alternatively, no node can have two or more measurements for the same time instant. Note that there is an implied assumption here. Since transactions are assumed atomic, no node can legitimately generate two messages for the same time instant.

4. Secrecy: In an honest scenario, contents of a message are kept secret between the intended participants of the protocol only i.e., they cannot be read by nodes other than the creator or the intended recipients. (A trusted third party, if existing, is regarded as the creator, and does not violate the secrecy property.)
5. δ-Robust computation: Computation of the target location estimate is robust against malicious input up to degree δ, where δ is the tolerable location error.

Our Approach

The correctness of the protocol is given in Appendix A, where we show that the protocol converges to the true result within a finite number of iterations. In other words, when we have reasonable inputs, the system provides reasonable outputs. We now examine the security of the protocol. For this we prove how the security properties identified above are preserved in the face of various attacks.

Our adversary can launch various attacks against the protocol as well as the data that is carried by the protocol. It can leverage deployment characteristics (for e.g., ad hoc nature, wireless medium, etc) and try to subvert the protocol. In this section, we will characterize these attacks and prove one or more of the following: (1) that the adversary cannot launch these attacks due to sufficient protection mechanisms, (2) the probability that such attacks can be successful is negligible due to time or computational infeasibility, (3) that the protocol is robust against certain attacks up to the desired degree of tolerance.

.

1. Replay, Redirect attacks:

   Replay attacks occur when an adversary stores a copy of a message and replays it at a later time after the original message was intended to be used. We examine the case where an adversary Trudy records an (encrypted) message (1) sent by B_j to leader A (Refer Fig 7.). After some time t' has elapsed, Trudy replays message 1, which is accepted by A. If the elapsed time t' (relative to A) is greater than the window of validity for the measurement, then the freshness verification will fail and A will discard the request as per protocol. If on the other hand, the elapsed time t' is smaller in value than the window in which the original message would have been accepted, then by virtue of *N* being a true crypto-quality nonce, A will detect the replay comparing the nonce with currently stored nonces. The existence of a match asserts that A has seen this message before. It is worth noting that a node only needs to store a nonce until the time window of validity of the measurement. A node need not store a nonce beyond this expiry since a message can be rejected on the basis that the time stamp is no longer fresh. We can do this because both the nonce and the time stamp are committed by the sender. Since timestamps can only advance from the previous message, and nonces are required to be unique across a large time window, replayed messages cannot be successful. Thus, the presence of timestamps and nonces in the messages serves as an adequate countermeasure to this type of attack. The same applies to message (1) of Phase 2 which can be stored and replayed by an adversary. Thus, we see that simply replaying an older message is not a successful attack.

A *redirect* attack occurs when a message is sent to a third entity instead of the intended participant. It can manifest in two ways, with original message suppression and without. In the former case, the original message is suppressed from reaching the receiver and redirect to a different entity, whereas in the latter case, the message is “replayed” to a different entity and not the originally intended recipient without having suppressed the original message. We argue that in a wireless medium only the latter is a valid attack because message suppression in a wireless medium is difficult to achieve. Subsequently, there are various physical and mac layer techniques like spread spectrum [69] etc that sufficiently mitigate these attacks. Therefore, we only consider the former case in our analysis.

Let us examine the case where adversary *Trudy* redirects message (1) to a different node A_2. Since the message is encrypted in the public key of A_1, A_2 will not be able to successfully decrypt the message and drop it.

2. Insert, Fabricate

   An insert is an attack where an adversary inserts a completely new message into the system (fabrication). Inserting a fabricated message essentially results in the message being accepted by the algorithm if an adversary possesses *any* legitimate signature key. This is because signatures are not tightly coupled with the identities of the nodes. Hence an adversary can supply any signed message and it will be accepted. The effect of fabricated messages on t he target estimate is considered under our robustness analysis.

3. Delay, False timing

   This type of attack occurs when nodes behave in a Byzantine manner (malfunctioning or exhibit arbitrary behavior). A node may send a measurement with incorrect timing values or send messages after a long delay. Two things happen here. One, when a node sends a message after a long delay, the watchdog nodes will flag the node. Second, if the delay is too long, the message may simply be discarded due to loss of freshness. False timing messages are rejected if the time is too distant (past or future) from the current processing window. If not, messages will be accepted and integrated into the target estimate.

4. Masquerade (Impersonation)

   A masquerade attack occurs when a user presents itself to the system as another user, usually a legitimate one. (Note that this attack is different from a compromise and takeover attack.) This may be done in order to gain unauthorized access to information or resources, to disseminate (mis)information in another's name, or to block or deny a system from operating correctly. [60]

   A malicious node Trudy may masquerade as a legitimate leader node to accumulate input messages or as a *B* node to insert false messages. In the former case, since messages are sent encrypted with the public key of the real leader node, a masquerading leader node can derive no benefit. Similarly, as a masquerading type B node, an adversary cannot forge the credentials of a legitimate type b node as it is against the property of the cryptosystem employed.

It can, however, falsely sends a fabricated message to $\mathcal{A}$ which is made to appear as if it came from $\mathcal{B}$. The message will be successfully received at $\mathcal{A}$ since the adversary can generate an arbitrary signature and encrypt the message using A's public key. When the fabricated message is received at A, A checks to see if the data is acceptable. Since all messages in the network are encrypted, an adversary does not know what an acceptable value is. It can only generate a message and hope that it is accepted with probability p_β (See Section 3.2) The result is that fabricated messages are accepted by our system if they are signed using any legitimate signing key. The impact of accepting fabricated messages is dealt with in the robustness analysis in the earlier section. (To summarize, we have seen that the algorithm can survive a large number of dishonest nodes provided the frequency of input is bounded.)

5. Man in the middle

   A man in the middle (MITM) attack is one in which the attacker intercepts messages in an exchange and then retransmits them, (sometimes substituting its own crypto primitives in place of the requested one) so that the two original parties still appear to be communicating with each other. The attack may be used to intercept, read or alter messages without the knowledge of either transacting party.

   As indicated in the previous attack capture, suppression, selective jamming are not easy to achieve in a wireless medium. Various techniques at the physical and mac layer sufficiently reduce the possibility and question the practicality of these attacks. One seemingly practical way of launching a MITM is if the adversary can send a message before the original is received at the recipient or to quickly compute a response and send before legitimate reply reaches the intended recipient. A multi-hop path is more conducive to this type of attack due to the practicality of creating a middle man message before the legitimate responder can create it. Since there is no multi-hop communication in our protocol, the practicality of this attack comes under question. Assuming that such an attack is possible, we analyze the effects of such an event occurring. The biggest deterrent for this type of attack is the use of cryptographic keys. Further, the protocol is asymmetric in both directions (in both Phase 1 and 2) and MITMs require symmetric message exchanges.

6. Capture/compromise

   A capture attack is whereby an adversary can subvert a legitimate node and take control of it. Capture and Compromise are used interchangeably. If Trudy captures a type b node, it can easily generate legitimate messages (with fraudulent data) using B's shared keys. This will result in lots of false data passing undetected through the particle filtering system. When falsified data is accepted by the system, the robustness analysis comes into play. We have seen as per our robustness analysis that as long as the frequency of inputs from a B node does not exceed vmax (the allowable maximum frequency without being detected as an outlier), the target estimate (output) will be within δ of the true estimate.

Capture multiple nodes: We have seen from our robustness analysis that if an adversary wants to disrupt the target estimate, then it has to compromise a very large number of nodes in a single neighborhood. Since our algorithm incorporates a moving leader approach whereby the aggregator function moves from leader to leader as the target moves through the sensing field, at any given time only those leaders and type B nodes are active which are in the neighborhood of a target. Therefore, in order to disrupt the target estimate an adversary has to undertake the Herculean task of guessing correctly which leader node (and cluster) will be active next (alternatively guess the pattern) and then compromise that cluster within the short amount of time that the target remains in the vicinity and the aggregation function is resident on that leader node. This is very difficult to achieve in practice even for an extremely capable adversary (due to the small time duration in which such a massive attack needs to be completed). Therefore, such an attack is not practically possible (unless there exist a large number of *colluding* adversaries).

7. Lying nodes, anomalous behavior: If a node behaves in a manner inconsistent with the nature of the neighborhood (for example, if none of the nodes in the neighborhood report spotting a target but only one node does) then watchdog nodes will make a note of this anomaly. Repeated anomalous behavior can result in flagging the node, and rejection of the data sent by it.

3.4 *Factors affecting performance, reliability and accuracy*

3.4.1 Topological Dependence

Figure 9 depicts common sensor network topologies like the star, cluster, tree and hierarchical arrangement of sensor nodes.

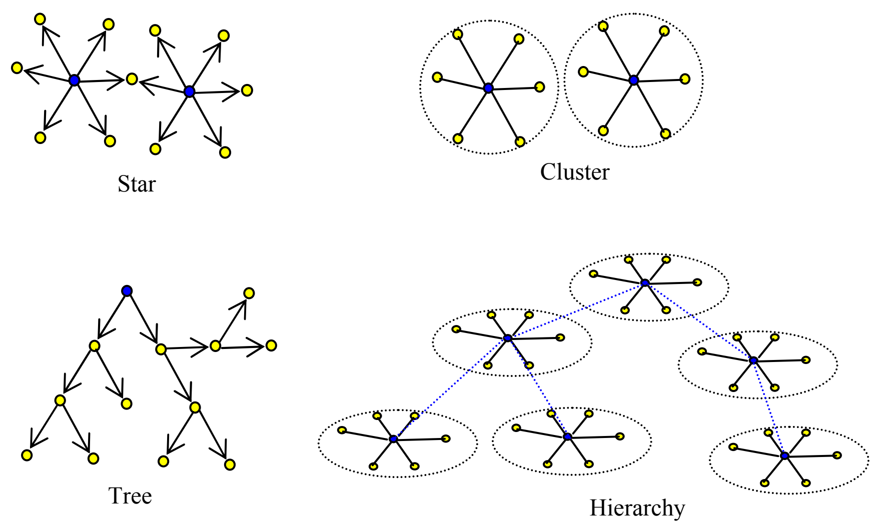


Figure 9: Basic Topologies

Few additional derivatives of these structures are binary tree structure, fan outs, linear, mesh, and ring. While the ring arrangement is the most uncommon form for a sensor network, it provides multiple (at least two) distinct paths to a destination with roughly the same cost and is used in some sensor network algorithms like CHORD. Full mesh and star topologies have higher overheads than tree based or hierarchical, but provide a rich assortment of data which is very helpful for particle filtering based techniques. Also this arrangement is least affected by mobility. Star topologies are great for broadcasts to percolate quickly through the network. Clusters are the most popular topological arrangements as they are more ‘hybrid’ and well adapted. They form a semi hierarchical, semi star/mesh arrangement, where members of a cluster/cell have a unique arrangement with each other (star or mesh) and each cluster has a connection with neighboring clusters to form an overlay that eventually connects and brings together the entire topology of the network. Based on our convergence results and error analysis we find that linear and tree type topologies are weaker than cluster or star/fan out arrangements. The former converge slower, the effective loss probability is higher, and error propagation causes the results to be more pronouncedly inaccurate than a cluster based approach that is well grounded due to its hybrid nature.

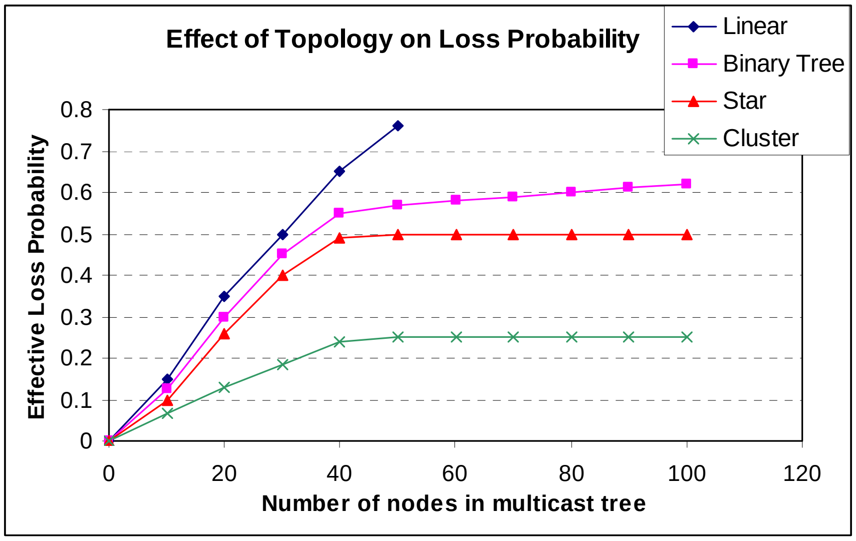


Figure 10: Effect of various topological configurations on loss probability.

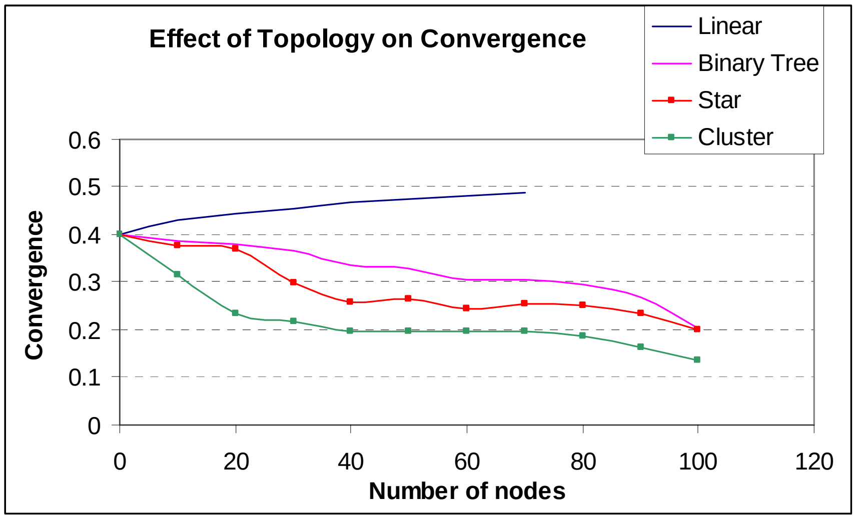


Figure 11: Effect of Topology on Convergence

The expected MSE for our distributed particle filter is of the form $E\left[\left(\langle \pi_t^N, \varphi\rangle - \langle \pi_t, \varphi\rangle\right)^2\right] \le c\frac{\|\varphi\|^2}{N}$ where c is a constant, π is the posterior distribution importance function at time t from which particles are drawn, φ is the transition kernel of the filter, and N is the number of particles per target in the given computation window. From this equation, we can see that at each approximation, the MSE is inversely proportional to the number of particles N. As the number of particles increase, the expected MSE tends to decrease. This tendency which is the convergence of the filtering algorithm thus is also proportional to the number of particles N. As the number of contributing nodes increases, N increases. A direct result of this is observable in Figure 11, where we see the trend of convergence of the particle filter for various topologies.

The cluster based topology has the closest coupling between the leader of the cluster and the sensing nodes (all single hop). Therefore, the time taken for each particle to reach the leader node for aggregation into the filter is minimal in this topology. Further, the single hop close knit structure also ensures that collisions and retransmission errors also will not be pronounced. For a cluster topology, the localized effect is that the density of particles will result in a proportional increase in the rate of convergence of the particle filter.

One could argue that the while the number of particles in a binary tree topology cannot be increased, it can be increased in a star as well as a hierarchical network so as to provide better convergence rates. We argue that in a localized context the star topology can create the dense particle effect similar to a cluster arrangement and produce equally good convergence rates, but on a larger scale, the star topology advantage will be subdued by the increased transmission delays due to relaying data to a far-away base station for aggregation, whereas this problem is absent in the cluster arrangement due to single hop aggregation. For a hierarchical network, transmission delays from the sensing element to the aggregator will become pronounced, as the data gets relayed further and further away from the sensing area. The cluster arrangement has an advantage in this regard, that it can not only create clusters of single hop neighbors but also form them closer to the sensing area, thereby minimizing transmission delays. From a global view point, a cluster arrangement can be viewed as a hierarchical formation of star arrangements, where each cluster has a star arrangement, and the aggregated information from each cluster is transferred to a decision making location through the leader nodes of the cluster.

Periodic Selection: A final remark is that periodic selections are very efficient and have a specific interpretation in nonlinear filtering settings. We have seen in other literature that in this situation the fitness functions are related to the observation process. Roughly speaking the selection transition evaluates the population structure and allocates reproductive opportunities in such a way that these particles which better match with the current observation are given more chance to reproduce. This stabilizes the particles around certain values of the real signal in accordance with its noisy observations. It often appears that a single observation data is not really sufficient to distinguish in a clear manner the relative fitness of individuals. For instance this may occur in high noise environments. In this sense the particle filtering system with periodic selections allows particles to learn the observation process between the selection dates in order to produce more effective selections

3.4.2 Effect of topology on Robustness

We have shown earlier that our tracking solution is δ-robust to maximum malice caused by an adversary local to the neighborhood. We can further reduce this problem to show its dependence on the topological configuration of the neighborhood. Essentially, δ-robustness for $Z_{[t+nT]}$ comes from the continuous stream of input observation vectors that create the robust estimate $Z_{[t+T]}$ at any time window [t+T]. The stream of input observations is dependent on the connectivity graph i.e., topological configuration (node degree) and the probability of receiving the observation vectors from the nodes in the neighborhood. Thus, the robustness of $Z_{[t+T]}$ becomes a simple problem dependent on the physical node degree and the probability of receiving observations from a minimum threshold number of honest nodes within the neighborhood.

Mathematically,

If $P_n\ (i=1)$: Probability that a sensor node a has connectivity with node b within a single logical hop in sensor a's information range.

$P_n\ \left(i=1\,|\,\tilde{A}\right)$: Probability that a sensor node a has connectivity with node b within a single logical hop in both sensor a and b's information range.

D_p is the average physical node degree.

Therefore, we can calculate $P_n\ (i=1)$ as follows:

$$P_n\ (i=1)=(1-P_n(1)).\sum_{n_1=1}^{D_p-1}\left(\binom{D_p-1}{n_1}(P_n(1))^{n_1}(1-P_n(1))^{D_p-1-n_1}.\left(1-(1-P_n(1))^{n_1}\right)\right)$$

We can also calculate the minimum guaranteed degree that must be maintained in order for the above observation $Z_{[t+T]}$ to be δ-robust as:

$$D_{\min}=D_p\left[1-\left(1-\sum_{i=1}^{\infty}P_n(i)\right)\left(\frac{1-\left(\sum_{i=1}^{\infty}P_n(i)-\sum_{i=1}^{\infty}P_n\left(i\,|\,\tilde{A}\right)\right)}{\left(1-\sum_{i=1}^{\infty}P_n\left(i\,|\,\tilde{A}\right)\right)}\right)\right]$$

From the above equation, we can see that minimum degree requirement is dependent on the each node's information range, as well as the intersection of their connection ranges as denoted by $P_n\left(i \mid \tilde{A}\right)$.

From here it follows that, in general, topological configurations with higher $P_n\left(i \mid \tilde{A}\right)$ i.e., dense connectivity areas also known as cliques, will show greater robustness to malicious input observations. Cliques are areas of common connectivity between neighborhoods. Among the configurations we have studied, cliques are not commonly observable in tree and linear configurations, are rarely observed in star configurations and most commonly observed in cluster arrangements. Hierarchical clustering topology is therefore a special case of clustering with maximal cliques. This is consistent with our observation in Section 3.4.1 where we examined the relationship between particle density profile and ambiguity resolution as well as our topology simulations.

3.4.3 Particle Density Profile

We have seen how topology models changes in the convergence and robustness of the particle filtering based algorithm. A closer look at topology also revealed that the internal configuration with respect to the topology also directly relates to the precision with which the algorithm tracks the target, resolves ambiguities and the resilience of the algorithm from being misdirected by malicious entities. A particle density profile is the distribution of particle channels across a topological entity like a cluster, representing a transition of particle systems. For a discrete particle system, the particle density profile can be represented as $\{\eta_{t_n}^N ; n \geq 0\}$ where

$$\eta_{t_n}^N = \frac{1}{N}\sum_{i=1}^{N}\delta_{\xi_{t_n}^i}$$

Where ξ_{t_n} is the particle at t$_n$.

For a cluster with uniformly distributed nodes and uniformly distributed anchor nodes, with density ρ_n the particle density profile can be represented as $\eta_{t_n}^N = 1/\rho_n$

The figure below denotes two clusters set up in a wsn. Each cluster has a leader and multiple sensing nodes.

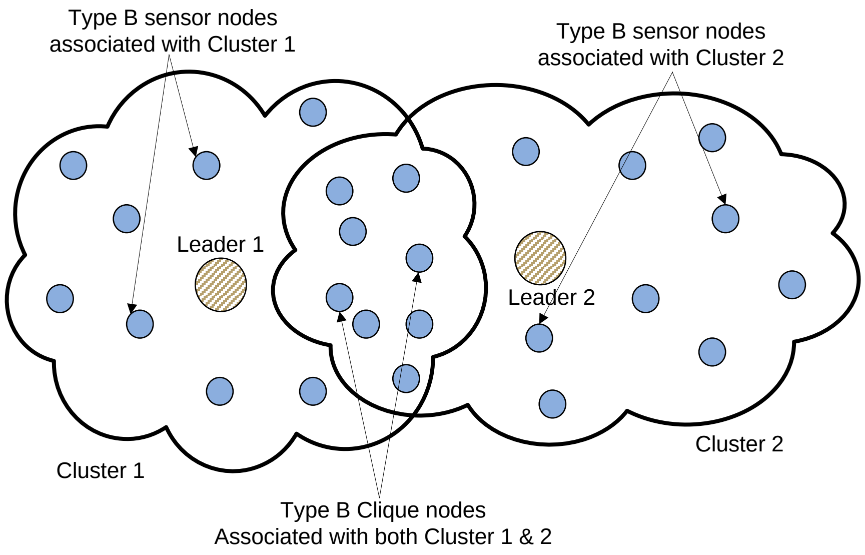

Type B sensor nodes associated with Cluster 1
Type B sensor nodes associated with Cluster 2
Leader 1
Leader 2
Cluster 1
Cluster 2
Type B Clique nodes Associated with both Cluster 1 & 2

Figure 12: Clusters with Cliques

As is obvious from the figure, a slightly skewed node distribution will lead to more precisely tracking a moving object than a uniformly distribution of nodes in the cluster. If every cluster maintains a minimum density ρ_{min} below which the low density will cause gaps in measurement and insufficient no. of particles to localize on a target location in the presence of noise, then a node distribution that is lightly skewed to have higher representation in areas common to multiple clusters will consistently produce results close to the true value of the target's position. By nature of being in the communication range of multiple clusters and their leaders, these clique nodes can observe and validate data that is being sent by its neighbors in both clusters, thereby resolving ambiguities, eliminating false and outlier data, as well as counteract malicious nodes. Their exact position with respect to malicious colluding nodes can possibly help detect collusions.

3.4.4 Seed Infrastructure

In many applications, sensor networks have to be deployed in remote, unexplored, or hostile regions. Often in such deployments, there may not be an existing infrastructure for the nodes to rely on. A similar story exists for ad hoc deployments. In the absence of GPS type absolute positioning systems, nodes must rely on other nodes to determine their whereabouts. Nodes that are aware of their own positions using some external means (often GPS) are known as *seed* or *anchor* nodes. These seed nodes help other nodes locate themselves. As a condition of accurate localization, seed nodes must be localized with a great deal of precision to ensure that nodes that are based off of these seed nodes will have a certain degree of precision in their location co-ordinates. As nodes localize themselves through seed nodes, some error creeps into their measurements. This error in turn propagates through to the tracking measurements a fully localized deployment takes. The greater the number of reference seed nodes present in a deployment, more accurate is the localization of the remaining nodes in the deployment. On the other hand, increasing the number of seed nodes results in a significant amount of overhead as more number of nodes are being localized through an external means. For example, GPS requires additional hardware receivers. More the number of sensor nodes localized through the GPS system, more the expensive hardware requirement, and hence more the cost to deploy. In order to keep costs practical, a definite balance between the density of seed nodes to regular nodes needs to be determined. In such a case, few nodes can be used as seed nodes. Nodes can be localized with respect to these seed nodes in a tier-like fashion. In the first round few nodes can localize themselves with seed nodes. Other nodes can localize themselves with respect to the secondary seeded nodes, and so on. In such a case, however, all errors in measurements made by the deployment must take note of this factor too, as error propagation results in slight decrease in precision of subsequently localized nodes.

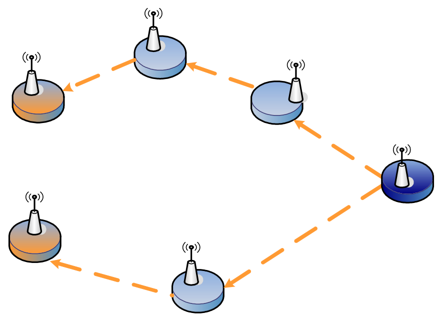


Figure 13: Collocated neighbor nodes (orange) using different reference node chains

Along with their numbers, location and distribution of anchor nodes is also critical. Distribution of anchor nodes must be such as to create an unambiguous coordinate system. For example, using three anchor points for multi-lateration results in two possible solutions for a node’s position, whereas adding a fourth node results in successfully disambiguating it down to one (within acceptable error bounds). Therefore, the distribution of anchor nodes is also as vital as their density. Sometimes, as seen in the figure below, having anchor nodes in a particular configuration results in a large error creeping into the measurement associated with a node due to using a different reference path for localization.

3.4.5 Mobility

Mobility models are used to formally describe the pattern of motion of mobile sensor nodes. They are useful for various reasons like predicting the next location of an entity based on its movement pattern if an earlier location is verifiably known, or for disambiguation purposes like ruling out a location. Mobility models thus improve precision. For a stationary deployment, the node mobility model is *static*. Node positions never change arbitrarily. Dynamic mobility models, on the other hand, can have many interesting representations, and vary from *constrained path* to *random walk* models. Essentially, in a constrained path model, a moving entity can follow select *paths* to get from point A to point B (for example a road, walkway, etc). All other areas that do not fall on this path are designated as non-traversable, and ruled out as possible locations for an entity to exist at. Constrained path models take into account location geography and obstacles and are a good representation of a practical deployment environment. In the random walk model, on the other hand, an entity can travel in any haphazard manner across the deployment field. This type of model is most common for flat, unvarying deployment environments like deserts, water surface, etc. Mobility models are entity specific since they contain information about not only the pattern of movement but also movement characteristics like speed, acceleration, etc and can be used to even differentiate various types of entities on the basis of this information (for example, a battle tank and a foot soldier will have different speeds and acceleration restrictions).

Chapter 4 Secure Time Synchronization

Synchronization among the collaborating entities is of paramount importance to a distributed system. Synchronization is essential to put measurements into context, for general ordering of events, for tracking, tracing a trajectory, for building a historical perspective, for replay detection, and much more. For example, in a wireless sensor network, synchronization is important to achieve a global view using measurements made within a single time frame. Measurements need to be aggregated within the same time window in order to be a meaningful representation of the network at a given point in time. An isolated event will be noticed by multiple sensor nodes within quick succession. This phenomenon needs to be represented in a single window of time. At times, an agreement needs to be reached regarding the ordering of sensed events, and at other times, a rough time of occurrence needs to be established.

Different types of synchronization are required based on the need of the system. In general, synchronization is done in two ways: Logical clock ordering and actual time synchronization. When events need to be placed in real-time context, synchronization based on actual time is required, whereas, when only the relative ordering of events matters rather than absolute time, logical clock ordering is used. Time synchronization can also be achieved locally or globally. Individual distributed entities can maintain coherent time with respect to each other using GPS-enabled receivers. These GPS receivers allow each entity to individually synchronize with a coherent source. Such approaches, however, are very expensive as each node requires a GPS receiver, increasing per node cost (and battery life). This makes it an unsuitable solution for low cost or cost-conscious networks. NTP is another popular protocol which is used in wired networks as well as over the internet, but is neither secure nor applicable to wireless sensor networks. The adversarial model, system model and the assumptions secure-NTP is based on is not applicable to wireless sensor networks and is hence not adequate for our purpose. Further, optimization for energy and bandwidth consumption was not in mind when secure-NTP was designed.

Other protocols designed specifically for distributed and collaborating networks have been summarized below. These protocols were not built with security applications in mind [28][32][40][41][42]**Error! Reference source not found.**[56][67][74][83]. For example, in RBS and TPSN type networks a hierarchy is created where a downstream node must synchronize through its upstream neighbor. Therefore, if an upstream node were to send malicious timestamps to its downstream node, the latter would fail to synchronize correctly. Similarly, all subsequent downstream nodes would fail to synchronize correctly. A node can claim to be closer to a downstream node and cause disruption as well as failure. Lack of cryptographic mechanisms can facilitate nodes to *relay* synchronization messages to its downstream neighbors even without actually receiving it, causing the downstream nodes to fall out of sync. A further aggravated version of this attack could cause wide-spread battery depletion among the downstream nodes due to repeated mis-synchronization. Further, these protocols are not very efficient under stringent energy constraints. Few protocols have been built with security in mind [43][44][45][55][75]. Some of these protocols, for example [75] require pre-shared keys to exist among multi-hop neighbors, which in an impractical and potentially insecure assumption. Further, they are not light-weight in terms of computation and energy consumption. [55] states various possible alternatives for securing commonly used time synchronization protocols. In [75] some sub-schemes are resilient to only external attackers, while others are resilient to only internal attackers. The group synchronization schemes are computationally not light weight. The authors use the notion of lightweight synchronization towards usage of less number of messages. We argue that for a deployed sensor network running a on finite energy supply, protocols with less number of messages but very high computational requirement are not as feasible as running a real lightweight protocol in terms of resource consumption and mission life sustenance. Since the ultimate aim of the sensor network is not just time synchronization, but utilization of the time synchronization mechanism to facilitate other services provided by the network, it would be wise to invest in a low energy consuming scheme. The authors have also stressed, time and again, on power saving schemes to offset the high-energy cost of their protocol. However, they do not address the eventuality that an adversary can take advantage of the sleep scheduling/power saving schemes and disrupt the time synchronization protocol. [45][43][44][45] are good solutions but for their hardware dependency. For example, [45] is dependent on very specific hardware modules for timestamping. [43] is a cluster based approach which is very promising, but is not resilient to wormhole attacks, and also requires time synchronization as a dependency. We have briefly summarized current work in the area in Section 4.1.

In section 5.1 we have investigated various attacks against synchronization schemes. While there has been a lot of research in the area of time synchronization for wireless sensor networks, most solutions either have practical limitations that restrict their widespread use, or have hardware dependencies that inhibit large scale deployment. Our approach is to create a time synchronization protocol that is robust against most practical in-field attacks, does not have practical limitations, expensive hardware dependencies and is lightweight enough to be a backbone service. Not requiring apriori knowledge of the network deployment, and not having too many trust dependencies would be an added plus. We now examine, in detail, various time synchronization protocols for wireless sensor networks and their characteristics.

4.1 Current Research in Time Synchronization

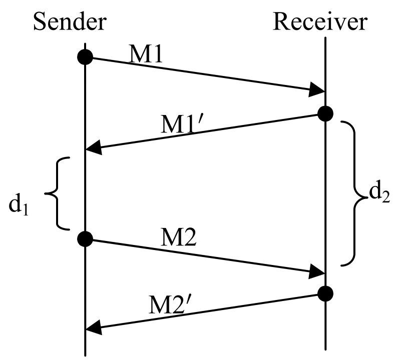
4.1.1 Review of existing Time Synchronization Protocols

In the following subsections, we discuss few synchronization algorithms from current literature that have impacted our work. We have categorized our literature review in the area of time synchronization into two types: review of time synchronization protocols, and review of *secure* time synchronization protocols. Due to the body of work in the time synchronization area, we attempt to summarize the security pitfalls and shortcomings of the secure time synchronization schemes only. Time synchronization protocols built for non adversarial environments suffer from all of the attacks listed in section 5.1.

4.1.1.1 Timestamp Synchronization (TSS)

TSS [42] is a local internal synchronization service where timestamps are received and converted into the local timescale of the receiver.

Synchronization is achieved by calculating the age of each timestamp from its creation to its arrival at each sensor node, which includes the time the message is resident at a node, and the time taken to propagate the timestamp message from one node to another. For multi-hop synchronization, the time taken to propagate the last message gets added to the total hold time of the previous node. Figure 14 is a simple representation of TSS.



Peer 2

Figure 14: Timestamp Synchronization (TSS)

4.1.1.2 Reference Broadcast Synchronization (RBS)

RBS [32] provides synchronization for a multiple nodes at a time. The time source node sends a *reference broadcast* to a set of client nodes in its one-hop neighborhood. The client nodes exchange their receipt times of the broadcast messages and compute relative offsets and rate differences with respect to each other. This way, they are able to relate their local time clocks to the clocks of their neighbors and reduce their offsets with each other. In the end, a cluster of nodes is relatively synchronized with each other with respect to the broadcast source.

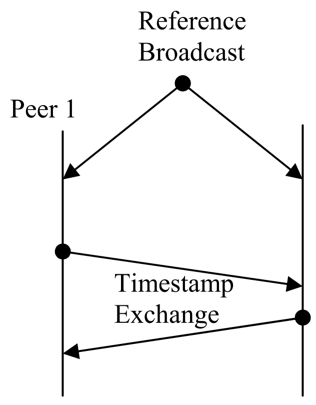


Figure 15: Reference Broadcast Synchronization (RBS)

4.1.1.3 Lightweight Time Synchronization (LTS)

LTS [40] is a time synchronization scheme that was designed to provide a light weight, scalable means of synchronizing nodes in a network. The on-demand version which is the more lightweight of the two, provides synchronization selectively to those that require frequent resynchronization than other nodes in the network. The broadcast synchronization which synchronizes all nodes proactively is the other type. Both schemes are built to exploit a spanning tree structure, where the root of the tree is synchronized through an out-of-band technique. In the proactive approach, the root floods all the nodes with a broadcast, and all child nodes synchronize with their parents. In the on-demand approach, child nodes request synchronization, and synchronize with the root node through the reverse path (reply messages) using round trip times. Synchronization messages can be further reduced by piggybacking with neighbors who have pending synchronization messages. In such cases, nodes simply synchronize laterally with their neighbors.

4.1.1.4 TimingSync Protocol for Sensor Networks (TPSN)

TPSN [74] is similar to LTS. A leader node is elected and a spanning tree is dynamically created with the leader as its node. The root node now floods the network with a broadcast message following which all nodes synchronize with their parents using round trip measurements. Message-delay uncertainties are reduced by time-stamping at the MAC layer. However, in case of node failures and topology changes, the entire process from root node election and tree construction must be repeated.

4.1.1.5 TSync

TSync [28] is an external time synchronization technique that uses independent radio channels for synchronization. It does so in order to avoid packet collisions and any inaccuracies resulting from the same. TSync comprises of two protocols for external synchronization: the Hierarchy Referencing Time Synchronization Protocol (HRTS) which is a proactive synchronization scheme, and the Individual-Based Time Request Protocol (ITR) which is an on-demand synchronization scheme. Nodes synchronize with a root node that has access to global time, in a spanning tree structure.

4.1.1.6 Interval Based Synchronization (IBS)

IBS [41] uses finite time intervals to set bounds on the current time. Nodes that wish to synchronize maintain upper and lower bounds on the current time. They each exchange their bounds and determine a new reduced interval than their previous interval by choosing the lower of the two exchanged upper bounds and the higher of the two exchanged lower bounds. They also keep track of the elapsed time and update their bounds for the next round accordingly.

4.1.1.7 Flooding Time Synchronization Protocol (FTSP)

FTSP [56] achieves time synchronization by a combination of agreement and regression. First, a node is selected as the leader based on its ID. (Lowest ID becomes the leader). The leader periodically sends synchronization messages to the nodes in the network. All nodes update their timestamps and relay the message to their neighbors. Each neighbor collects and compares eight pairs of messages and computes its new offset using linear regression. This scheme suffers from heavy message exchange and additional time loss. In order to minimize some of this loss, timestamping is done at the MAC layer.

4.1.1.8 Asynchronous Diffusion (AD)

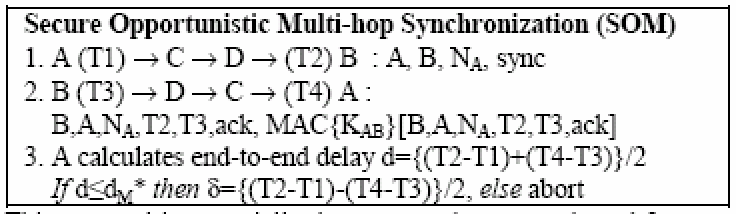
AD [67] is a simple time synchronization scheme that essentially averages the offsets among neighborhoods. Each node periodically sends a broadcast to its neighbors, which reply with a message containing their current time. The receiver then averages all received timestamps, and broadcasts *its* average to its neighbors who adopt the newly sent timestamp. In order to avoid random synchronizations, the nodes follow a predetermined synchronization order.

4.1.2 Review of existing Secure Time Synchronization Protocols

4.1.2.1 Secure Time Synchronization Protocols SOM, SDM, STM, L-SGS

In [75] the authors create four sub-protocols for secure time synchronization. Their main goal is to create a secure time synchronization protocol that is resilient to insider attacks, with minimal overhead in terms of messages exchanged as well as energy consumed. They provide three secure time synchronization protocols to satisfy their goals. They also provide a secure group time synchronization protocol that is resilient to attacks from external attacker as well as to attacks from a subset of compromised group nodes. While these protocols are not fully resilient to some insider attacks, they can detect malicious attacks on the time synchronization mechanism.

Ganeriwal et al Secure Time Synchronization Scheme #1: Secure Opportunistic Multihop (SOM)



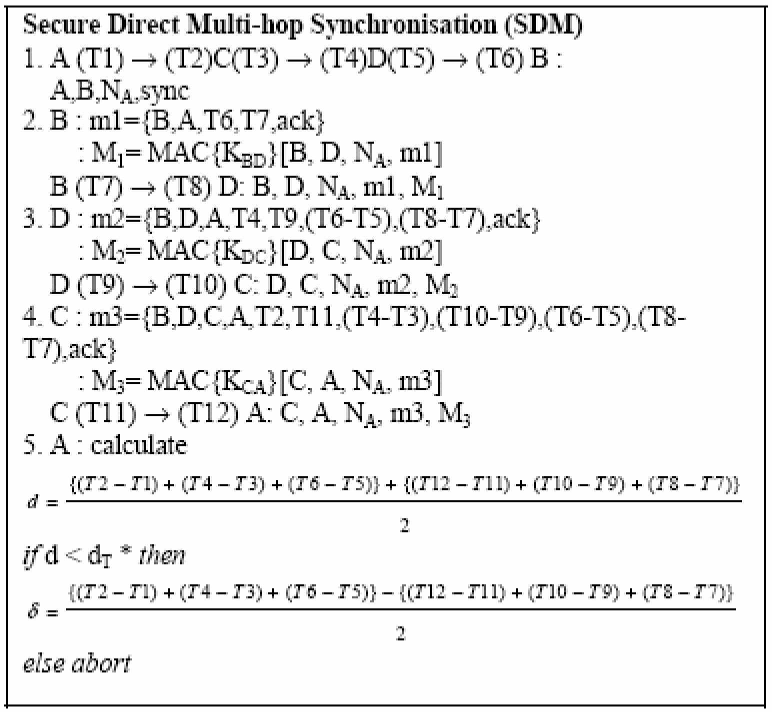
Secure Opportunistic Multi-hop Synchronization (SOM)

1. A (T1) $\rightarrow$ C $\rightarrow$ D $\rightarrow$ (T2) B : A, B, N_A, sync
2. B (T3) $\rightarrow$ D $\rightarrow$ C $\rightarrow$ (T4) A :
   B,A,N_A,T2,T3,ack, MAC{K_{AB}}[B,A,N_A,T2,T3,ack]
3. A calculates end-to-end delay d={(T2-T1)+(T4-T3)}/2
   If $d \leq d_M$* *then* δ={(T2-T1)-(T4-T3)}/2, *else* abort

Secure Opportunistic Multihop (SOM) assumes a shared secret key K_{AB} between two nodes say, A and B that are several hops away, and wish to synchronize. This assumption is not very practical due to the nature of deployment of sensor networks in that one can never guarantee the existence of a shared key between nodes that are multi hops away or in different neighborhoods (i.e., out-of-radio range of each other).

At best, one can only say that nodes A and B several hops away, can probabilistically share a secret key. Hence the practicality of this scheme is weak.

Ganeriwal et al Secure Time Synchronization Scheme #2: Secure Direct Multihop (SDM)



Secure Direct Multi-hop Synchronisation (SDM)

1. A (T1) $\rightarrow$ (T2)C(T3) $\rightarrow$ (T4)D(T5) $\rightarrow$ (T6) B :
   A,B,N_A,sync
2. B : m1={B,A,T6,T7,ack}
   : M_1= MAC{K_{BD}}[B, D, N_A, m1]
   B (T7) $\rightarrow$ (T8) D: B, D, N_A, m1, M_1
3. D : m2={B,D,A,T4,T9,(T6-T5),(T8-T7),ack}
   : M_2= MAC{K_{DC}}[D, C, N_A, m2]
   D (T9) $\rightarrow$ (T10) C: D, C, N_A, m2, M_2
4. C : m3={B,D,C,A,T2,T11,(T4-T3),(T10-T9),(T6-T5),(T8-T7),ack}
   : M_3= MAC{K_{CA}}[C, A, N_A, m3]
   C (T11) $\rightarrow$ (T12) A: C, A, N_A, m3, M_3
5. A : calculate

$$d = \frac{\{(T2-T1)+(T4-T3)+(T6-T5)\}+\{(T12-T11)+(T10-T9)+(T8-T7)\}}{2}$$

if d < d_T * *then*

$$\delta = \frac{\{(T2-T1)+(T4-T3)+(T6-T5)\}-\{(T12-T11)+(T10-T9)+(T8-T7)\}}{2}$$

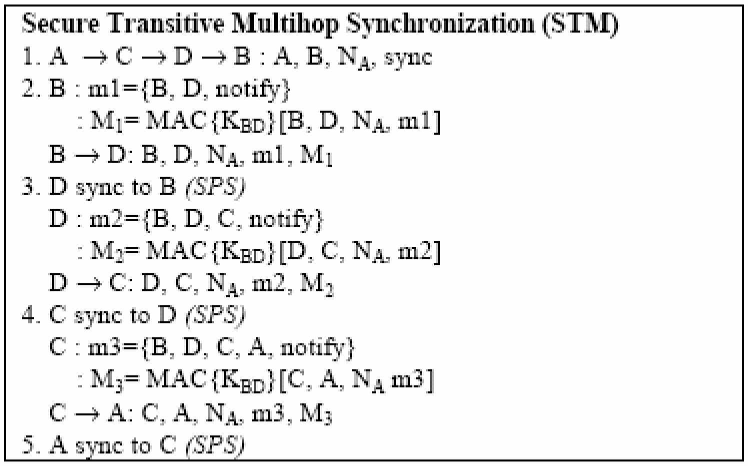
else abort

Both the SDM and STM (which follows) have additional trust assumptions associated with intermediary forwarding nodes. They assume that these nodes are trustworthy and hence are susceptible to insider, colluding and compromised nodes. (already acknowledged by authors)

Further, end-to-end delay between A and B is calculated as the cumulative end-to-end delay between each intermediary hop. For example, if A-> C ->D -> B then

$d_{AB} = d_{AC} + d_{CD} + d_{DB}$ is the (minimal) end-to-end delay incurred. Also, as per the assumptions, the expected message delay d is pre-calculated and known for any given path traversal. Since *d* which is determined on the entire route, is a function of per hop expected delay plus an additional factor for inter hop delays due to mac layer scheduling, channel disruption, etc, an intelligent adversary can cause enough delay at each hop, that is only slightly lesser than the per hop expected delay for that hop, allowing the pulse delay attack to go undetected for the single hop, and cause the cumulative end-to-end delay to be higher than the expected and possibly discarded at the end.

Ganeriwal et al Secure Time Synchronization Scheme #3: Secure Transitive Multihop (STM)

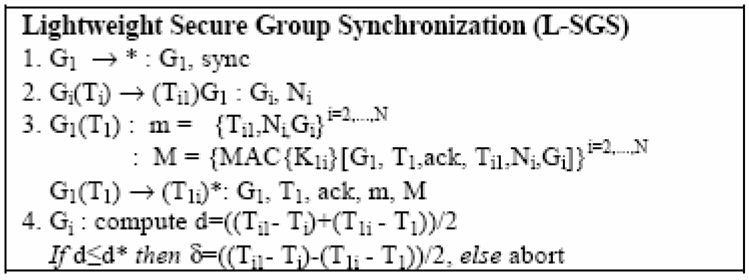


Secure Transitive Multihop Synchronization (STM)

1. $A \rightarrow C \rightarrow D \rightarrow B$: A, B, N_A, sync
2. B : m1={B, D, notify}
   : M_1= MAC{K_{BD}}[B, D, N_A, m1]
   $B \rightarrow D$: B, D, N_A, m1, M_1
3. D sync to B *(SPS)*
   D : m2={B, D, C, notify}
   : M_2= MAC{K_{BD}}[D, C, N_A, m2]
   $D \rightarrow C$: D, C, N_A, m2, M_2
4. C sync to D *(SPS)*
   C : m3={B, D, C, A, notify}
   : M_3= MAC{K_{BD}}[C, A, N_A m3]
   $C \rightarrow A$: C, A, N_A, m3, M_3
5. A sync to C *(SPS)*

In this scheme, multi-hop synchronization is achieved by transitively synchronizing each pair of nodes along the path from the source to the destination. Since only pair-wise delay is being considered, a practical attack would be to cause delays on each link such that the delay would be lesser than the maximum expected delay associated with that link, but the cumulative end-to-end delay for the entire path would too high to admit successful synchronization. Further, since each node synchronizes pair-wise with its down-stream neighbor, by the time the synchronization process propagates to the initiator, there is already a considerable drift between the clocks of the first and pen-ultimate node. For example, if the synchronization request was sent along A->C->D->B, D synchs with B, then C synchs with D, and when A synchs with C, there will already be a small skew and/or a small drift from the clock of the source B. This skew and possible drift must be accounted for in order for the synchronization error to remain bounded and practical.

Ganeriwal et al Secure Time Synchronization Scheme #4: Lightweight Secure Group Synchronization (L-SGS)



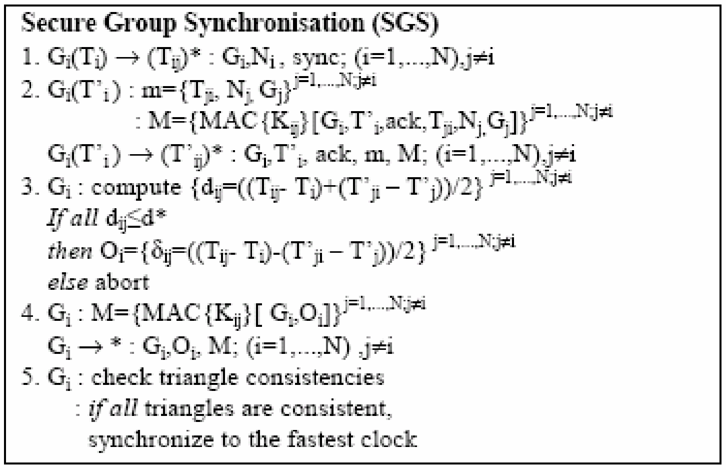
Lightweight Secure Group Synchronization (L-SGS)

1. $G_1 \rightarrow * : G_1$, sync
2. $G_i(T_i) \rightarrow (T_{i1})G_1 : G_i, N_i$
3. $G_1(T_1)$: $m = \{T_{i1}, N_i, G_i\}^{i=2,\ldots,N}$
   : $M = \{MAC\{K_{1i}\}[G_1, T_1, ack, T_{i1}, N_i, G_i]\}^{i=2,\ldots,N}$
   $G_1(T_1) \rightarrow (T_{1i})*: G_1, T_1$, ack, m, M
4. G_i : compute $d=((T_{i1} - T_i)+(T_{1i} - T_1))/2$
   If $d \leq d^*$ *then* $\delta=((T_{i1} - T_i)-(T_{1i} - T_1))/2$, *else* abort

This protocol is not resilient to internal attacks if G_1 is malicious. (acknowledged by authors). An implied assumption in this scheme is that every node must trust every other node in the cluster. Thus, it is very easy for an attacker that can capture a single node within the cluster.

This protocol, requires, at a minimum, the computation and transmission of N-1 MACs for a single synchronization request and hence is impractical for energy constrained deployments. The modification of replacing this requirement with a single MAC signed by a secret key created for the entire cluster is highly insecure. Now, the system has a single point of failure if even a single node is compromised. Also, due to the use of a symmetric key for the entire group, the MAC cannot be verified as having come from G_1 (the synchronization source) hence reliability can be further decreased.

Ganeriwal et al Secure Time Synchronization Scheme #4: Secure Group Synchronization (SGS)



Secure Group Synchronisation (SGS)

```
1. $G_i(T_i) \rightarrow (T_{ij})^* : G_i, N_i$, sync; $(i=1,\dots,N), j \neq i$
2. $G_i(T'_i) : m=\{T_{ji}, N_j, G_j\}^{j=1,\dots,N; j\neq i}$
            $: M=\{MAC\{K_{ij}\}[G_i, T'_i, ack, T_{ji}, N_j, G_j]\}^{j=1,\dots,N; j\neq i}$
   $G_i(T'_i) \rightarrow (T'_{ij})^* : G_i, T'_i$, ack, m, M; $(i=1,\dots,N), j\neq i$
3. $G_i$ : compute $\{d_{ij}=((T_{ij}-T_i)+(T'_{ji}-T'_j))/2\}^{j=1,\dots,N; j\neq i}$
   *If all* $d_{ij} \leq d^*$
   *then* $O_i=\{\delta_{ij}=((T_{ij}-T_i)-(T'_{ji}-T'_j))/2\}^{j=1,\dots,N; j\neq i}$
   *else* abort
4. $G_i : M=\{MAC\{K_{ij}\}[G_i, O_i]\}^{j=1,\dots,N; j\neq i}$
   $G_i \rightarrow * : G_i, O_i, M; (i=1,\dots,N), j\neq i$
5. $G_i$ : check triangle consistencies
       : *if all* triangles are consistent,
         synchronize to the fastest clock
```

In this protocol, 2(N-1) MACs are required to be computed and transmitted per synchronization transaction. Hence this protocol also energy-inefficient for the purposes of deployment in an environment where energy consumption should be optimal. Further, though the authors claim that this protocol is resilient to insider attacks, we can show that this protocol is susceptible to the classic *Byzantine Agreement Attack* [49]

Using the same example of nodes *i*,*j*,*k* enumerated by the authors, we can carry out the attack as follows:

Nodes *i*,*j*,*k* form a closed triangle where each node has calculated the offset between itself and its paired node. Thus each node only lacks the offset value calculated between the other two nodes. For the closed triangle, the sum of the offsets in a cycle in one direction yields zero if no node is malicious. Practically, the offset may not always be zero, due to inherent drift and skew error (as per the authors).

Attack formulation: We formulate an attack with the offset values set as represented in Figure 16.

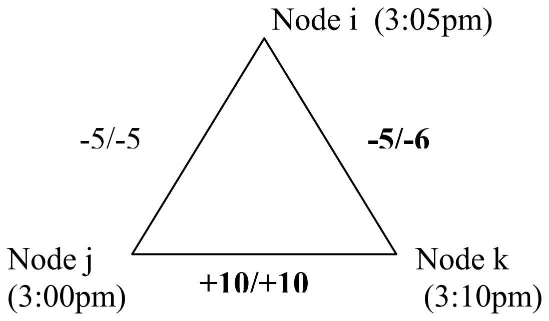


Figure 16: Synchronization Example

At the end of Step 2, each node possesses the following Offset set O_i:

Table 1: Offset values at each node at the end of Step 2

	$i \leftrightarrow j$	$j \leftrightarrow k$	$k \leftrightarrow i$	
i: {	**-5**	**?**	**-6**	} i.e., {i->j, j ↔k , k->i}
j: {	**-5**	**+10**	**?**	} i.e., {i->j, j ↔k , k->i}
k: {	**?**	**+10**	**-6**	} i.e., {i->j, j ↔k , k->i}

Each node depends on Step 3 to receive the missing element of its set O_i (which is represented as a *?* for simplicity) from its neighboring nodes.

At the end of Step 3, malicious node k sends the following message in place of its original message:

k: { **?** , **+12** , **-6** } i.e., {i->j, j ↔k , k->i}

Since the sum of the offsets is not zero, nodes *i, j* will only detect that there is a malfunction in the synchronization mechanism. It fails to identify the malicious node, and there will not be an agreement. On the other hand, if instead of a closed triangle, a closed quadrilateral was enforced, then a single malicious node can be easily identified. In general, the order of the polygon determines the maximum number of malicious nodes that can be identified. The number of malicious nodes must be less than a third of the degree of the polygon formed, in order to detect the malicious node.

4.1.2.2 TinySeRSync: Secure and Resilient Time Synchronization

In this paper [45], the authors develop a two phase secure and resilient time synchronization scheme called TinySeRSync for wireless sensor networks. They use hardware assisted source authentication to authenticate source, content and timeliness in the single pair-wise synchronization in the first phase and μ-TESLA based rebroadcast authentication to ensure timeliness and authenticity to achieve global synchronization in the second phase. The single pair-wise time synchronization is achieved using hardware assisted, authenticated medium access control (MAC) layer timestamps. Global time synchronization is achieved using μ-TESLA for local authenticated broadcasts. The $2t+1$ distinct paths between the sender and receiver ensure resilience against compromised nodes and Byzantine behavior.

Though this scheme is significantly better than others, it has some practical limitations, scalability issues and high overhead that limits its large scale deployment. For example, it may not always be practically possible for a node to have $2t+1$ distinct paths to it from the source or an upstream parent node for synchronization. By following a $2t+1$ approach, the onus of correctly synchronizing rests on the comparison a node makes with atleast t other pair-wise synchronization attempts. This can be very wasteful if the phenomenon manifests on a large scale in the network. In this scheme, a potential way of mitigating DoS attacks is by decreasing the synchronization intervals to a short time interval, thereby reducing the window available to an adversary to launch such an attack before the timestamps on the messages become obsolete and are discarded. The authors also point out that this approach comes with significant cost, both in terms of energy spent for synchronizing at such short intervals as well as storage requirements due to the fairly long key chain generated in the short interval.

Further, authenticated MAC layer timestamping requires secret keys be exchanged between communicating parties, which is a problem in itself, and a dual problem because reliable timestamps may be required to create and share keys post deployment. Further, μ-TESLA also requires certain parameters be exchanged apriori that has not been handled here.

Finally, this scheme does not mitigate rushing attacks or wormholes that advance messages.

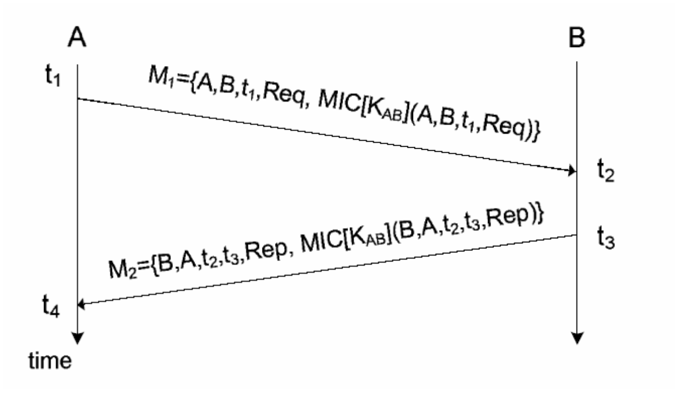

A
B
t_1
$M_1=\{A,B,t_1,Req, MIC[K_{AB}](A,B,t_1,Req)\}$
t_2
$M_2=\{B,A,t_2,t_3,Rep, MIC[K_{AB}](B,A,t_2,t_3,Rep)\}$
t_3
t_4
time

Figure 17: Revised Secure Pair-wise Synchronization in TinySeRSync

Figure shows the revised SPS protocol, in which all messages are timestamped and authenticated with the key K_{AB} shared by nodes A and B. Node A initiates the synchronization by sending message M_1. The message contains M_1's sending time t_1. Node B receives the message at t_2. After verifying the message, at time t_3, node B sends a message M_2 that includes t_2, t_3 to node A. When node A receives the message at t4, it can calculate the clock difference $\delta_{A,B} = (t_2-t_1)-(t_4-t_3)/2$, and the estimated one-way transmission delay $d_A = t_2-t_1+t_4-t_3$. Since all messages are authenticated, any modification to any message will be detected. To prevent the pulse-delay attacks [75] and wormhole attacks [85] , node A verifies that the one-way transmission delay is less than the maximum expected delay. In fact, this approach can detect any attack that attempts to mislead single-hop pairwise time synchronization by introducing significant extra message delays. Thus, sender A can easily detect attempts to affect the timeliness of the synchronization messages.

While this approach is significantly better than most approaches, their hardware dependency, $2t+1$ independent path requirement, and high overhead limit their widespread use.

4.1.2.3 Fault-Tolerant Cluster-Wise Clock Synchronization for Wireless Sensor Networks

In [43] the authors propose a synchronization scheme for nodes based in clusters. Nodes within a cluster communicate through authenticated broadcasts and only one synchronization message per cluster is sent. In each round, one node serves as the synchronizer and sends the broadcast. All nodes synchronize with this node if it is rightfully the turn of that node to be the synchronizer, and the clock difference between the synchronizer is not more than the clock difference between any two non faulty nodes. Fault tolerance is achieved through rotation of cluster heads and synchronizers. However, if colluding nodes in a cluster take turns to become synchronizers, they can in each round, cause nodes to synchronize to values very close to the extremal values of the acceptable range (say, close to upper bound $k\Delta$).

Consecutive rounds of synchronization with malicious synchronizers can cause the clock difference to cross this acceptable range.

4.1.2.4 Secure and Resilient Clock Synchronization in Wireless Sensor Networks

In [44], time synchronization is achieved through two means: the difference in clock measurements between nodes and their parents across hierarchical levels, and through diffusion through the network. In both cases, the authors claim that they can tolerate upto t malicious colluding nodes as well as upto s colluding source nodes. However, the approach dwells on the availability of $2t+1$ independent paths from a single source to any node in order to successfully synchronize to correct values. Each node needs to compute $2t+1$ clock differences before it can determine which clock difference values are acceptable. The message overhead is $O(|E|)$ which for a network with $2t+1$ independent paths to each node becomes quite significant. The total number of messages in one round is $n_1 + (|V| - n_1 - 1)(3t+1)$ where n_1 is the number of nodes at each level which for certain topologies could be disastrous to scale.

Clearly, this method not only has a heavy overhead, but also time taken to complete a single synchronization round is large enough for significant clock skew to creep in before synchronization completes. For this reason and more, the authors indicate that they require a high precision pair-wise synchronization scheme for their scheme to work, which becomes a catch 22 situation. Finally, to tolerate s colluding sources, the nodes must have access to $2s+1$ clock differences from *s different* source nodes. This makes it highly impractical to deploy.

4.2 Properties of a Robust Time Synchronization Scheme

Properties of a synchronization system that make it robust and dependable are:

(P 1) Must be robust to single point failure (except reference source)

(P 2) Must be robust to node failures

(P 3) Must always complete in the absence of active adversaries and communication errors and in the presence of honest participants who are compliant with the scheme.

(P 4) Must be resilient to active adversaries in that an active adversary cannot cause the protocol to deviate from the final outcome by more than the tolerable upper limit.

(P 5) Must allow for selective synchronization in the interest of efficiency for nodes that send time-sensitive data.

(P 6) .Freshness property: This property states that a message must be acceptable only while it is *fresh*. A message is considered fresh if the commitment associated with the message is not disclosed yet.

4.3 Components of our Secure Time Synchronization Scheme

4.3.1 One way Key Chains and Authenticated Broadcasts

One-way Key chains: A one way hash is a cryptographic primitive that is, simply put, a series of consecutive hashes created from a random *seed*. The notion behind a one way hash chain is that it is easy to compute up till the end value in one direction if either a seed or an intermediate value in the chain is known but computationally infeasible to compute in the reverse direction. This property of a one way hash chain makes it a very popular primitive in applications that are resource conscious, and where parties at each end (producer and verifier) can compute the chain in the same (efficient) direction. In other words, this primitive is easy to create and easy to verify. If the initial value of the chain that is disclosed can be uniquely and non-repudiably attributed to an entity then we can achieve source authentication as well. Many secure protocols for resource conscious applications like mobile devices and sensor networks employ one-way hash chains as core primitives. These chains can be computed within few milliseconds as opposed to tens of seconds to generate and verify signatures. Recently, researchers also proposed a variety of improvements to one-way hash chains to make storage and access more efficient O(log n) [14][88][87], or to make setup and verification more efficient O(n) and $O(\log^2 n)$ respectively [19][86].

In our time synchronization protocol, we use one way hash chain to provide an efficient means of providing message integrity and source authentication. The security of the technique vests in (1) the computational infeasibility of an adversary to compute a hash in the reverse direction and, (2) the infeasibility of an adversary to find a message $m' \neq m$ such that $H(m) = H(m')$.

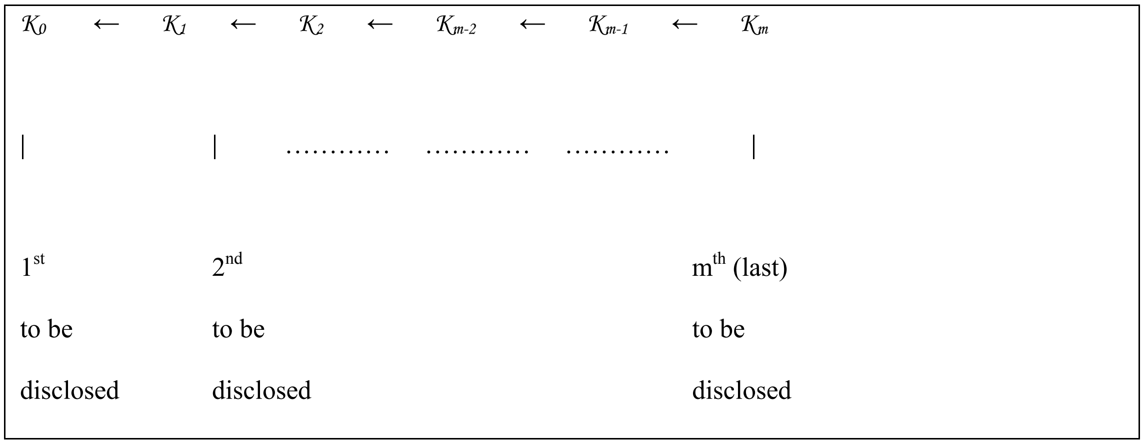
Usage: We assume a secure weak collision resistant one-way function $\mathcal{F}$ (to derive the one-way chain), and a secure one-way function $\mathbb{H}$ (to produce commitments). The generator then generates a one-way chain $\mathcal{V}_{\mathcal{N}}, \ldots\ldots, \mathcal{V}_o$, where $\mathcal{V}_i = \mathcal{F}(\mathcal{V}_{i+1})$ at times

$\mathcal{T}_{\mathcal{N}}, \ldots, \mathcal{T}_0$ respectively. We assume that the generator and verifiers are *at least* loosely time synchronized, with a maximum synchronization error of $\mathcal{T}_\Delta$. The generator creates and specifies a disclosure schedule for the one-way chain by selecting and specifying $\mathcal{T}_0$ and $\mathcal{T}_d$, where $\mathcal{T}_0$ is the time of disclosure of *end value* $\mathcal{V}_0$ and $\mathcal{T}_d$ is the time delay between the disclosure of two consecutive values. As per the disclosure schedule, the generator will disclose value $\mathcal{V}_i$ at time $\mathcal{T}_i = \mathcal{T}_0 + i * \mathcal{T}_d$, To authenticate a value *r* as being unaltered in transit, the generator publishes $r' = \mathbb{H}(\mathcal{V}_j \| r)$, where $\mathcal{V}_j$ is a value that will be disclosed in the future. When a verifier gets *r*, *r'*, *j* at time *t*, it verifies that the generator did not yet disclose $\mathcal{V}_j$ by checking that current time $t + \mathcal{T}_\Delta < \mathcal{T}_j$ (disclosure time of $\mathcal{V}_j$). If this condition holds, the verifier accepts *r'* and waits for the disclosure of $\mathcal{V}_j$ to authenticate *r'*. The verifier first verifies the authenticity of $\mathcal{V}_j$, by following the one-way chain to the last authentic value. If $\mathcal{V}_j$ is authentic then *r'* is authentic if $r' = \mathbb{H}(\mathcal{V}_j \| r)$. *r* can be any value that needs to be authenticated or verified. In our scheme, r is the time-stamp being sent by each node. Additionally, if $\mathcal{V}_j$ is uniquely and non-repudiable associated with an entity *A* (usually the generator) then the hash chain also provides source authentication i.e., provides the assertion that *A* is the generator of the one way hash chain (and hence the message that is tied to the hash chain).

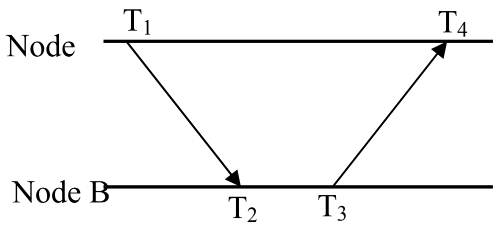
The optimum values of $\mathcal{T}_\Delta$, and $\mathcal{T}_d$ are dependent on the specific requirements for an application as well as the deployment topology as we will show in our analysis later. Also, special attention to the security properties is required during the synchronization error marginal interval ($\mathcal{T}_m + \mathcal{T}_\Delta$) from the time of disclosure of key $\mathcal{K}_m$. We leave this as a future exercise.

Authenticated broadcast: If source authentication is desired hen a node generates its own hash chain, the first element of the hash chain to be disclosed i.e., $\mathcal{K}_0$ should be authenticated. Thus, the initial element $\mathcal{K}_0$ gets coupled with the identity of the node generating this commitment. Since hash chains are self-committing, (every element disclosed is committed to all subsequently disclosed values) every subsequently disclosed element is also authenticated and tied to the generating node's identity.

Thus, authenticated broadcast is only required for the first disclosed element of the hash chain. After that, each element of the hash chain subsequently disclosed is also authenticated due to the initially disclosed key being committed to all future keys.


$\mathcal{K}_0 \leftarrow \mathcal{K}_1 \leftarrow \mathcal{K}_2 \leftarrow \mathcal{K}_{m-2} \leftarrow \mathcal{K}_{m-1} \leftarrow \mathcal{K}_m$
| | ………… ………… ………… |
1st to be disclosed
2nd to be disclosed
mth (last) to be disclosed

4.3.2 Basic Pair-wise Time Synchronization



Clock Drift $\Delta = \frac{T_2 - T_1 - (T_4 - T_3)}{2}$

Propagation Delay $d = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$

Figure 18: Basic Pair-wise Time Synchronization

This is a simple synchronization technique using Christian's algorithm [6] for node $\mathcal{A}$ to synchronize itself with node $\mathcal{B}$. Two messages are required for this synchronization (three if $\mathcal{B}$ also wishes to synchronize with $\mathcal{A}$). If $\mathcal{A}$ desires to synchronize with $\mathcal{B}$, it sends a synchronization request message to $\mathcal{B}$ at time T_1 and records the time (T_1). $\mathcal{B}$ receives the message at, say time T_2. $\mathcal{B}$ records the time and sends $\mathcal{A}$ a synchronization reply message at time T_3 which $\mathcal{A}$ receives at time T_4. The process of synchronization shown in Figure 18 is achieved by the sender $\mathcal{A}$ calculating its clock drift $\Delta = \frac{T_2 - T_1 - (T_4 - T_3)}{2}$ and propagation delay $d = \frac{T_2 - T_1 + (T_4 - T_3)}{2}$ with respect to the receiver using the time values exchanged.

For a network wishing to synchronize its nodes with a global time, the nodes will synchronize with the node(s) that have access to a global time reference. Throughout our work, we assume node $\mathcal{B}$ has access to global time and the other nodes, namely $\mathcal{A}$, $\mathcal{C}$, $\mathcal{E}$, $\mathcal{D}$ and $\mathcal{F}$ synchronize with $\mathcal{B}$ (Figure 19).

4.4 Our Secure Time Synchronization Scheme

4.4.1 Adversary Model

In the case of an internal adversary, we assume that the adversary can be only as powerful as the most powerful node in the network, for e.g., in the case of a heterogeneous network, the adversary maybe as powerful as the highly capable nodes in the network as opposed to the low-end dust-type sensor nodes. While considering an external adversary, we do not put a bound on the capabilities of the adversary since an external adversary could have the latest and greatest resources at its disposal. However, we can restrict the possible attacks that an external adversary can launch from outside the network since it does not have the obvious advantage that an insider may have (for e.g., shared secret keys if any, passive eavesdropping on a secured channel, etc) Finally, if an (external) attacker is able to compromise existing nodes, its capabilities of injecting and extracting information from within the network is limited by the capability of the compromised node(s). Besides these, the adversary can record, alter, reuse, insert, masquerade, replay, rush, fabricate messages, and collude with other adversaries. An adversary, however, cannot redirect, jam, capture, stop and delete messages that have been transmitted into the medium.

4.4.2 Assumptions

We start with a few practical and simple assumptions. Firstly, at least one node has access to a global time reference. All nodes will synchronize with this source eventually. If there are multiple root/leader nodes, they should all have access to an external unalterable coherent time reference. Nodes that have access to global time references are always stationary. Global time kept by any node is always orders of magnitude more accurate than the accuracy achieved by single-hop synchronization. We require no trust assumptions, except the obvious one, where the node that has access to a global time reference is trustworthy. Nodes must store nonces for as long as the average key disclosure times. Average key disclosure times and synchronization interval together affect storage cost. Shorter synchronization intervals result in more nonces being stored while key are pending disclosure. The tradeoff is simply determined based on application specific requirements.

4.4.3 Protocol Specification

4.4.3.1 Notations and Definitions

1. We define all participating principals $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\} \in \mathcal{P}$ where $\mathcal{P}$ is the set of all principals desiring to synchronize. Further, $\mathcal{P}$ comprises of all honest ($\mathcal{H}$) as well as all corrupt ($\mathcal{C}$) principals inside the network. No principals share secrets apriori.
2. The initial authentication element that allows each node to authenticate any commitment it generates is established prior to time synchronization and is not dependent on it.
3. Principals are aware of their upstream and downstream neighbors as well as the source they synchronize with.

4. It is highly desirable but not necessary for a node to know its distance from the source it synchronizes with.
5. $\mathcal{A}_{\mathcal{T}_1} \rightarrow_{\mathcal{T}_2} \mathcal{B}: \{\mathcal{A}, \mathcal{B}, \mathcal{T}_1\}$ denotes a message $\mathcal{M}$ sent from principal $\mathcal{A}$ to principal $\mathcal{B}$, and reads "At $\mathcal{T}_1$ node $\mathcal{A}$ sent message $\{\mathcal{A}, \mathcal{B}, \mathcal{T}_1\}$ to $\mathcal{B}$, which was received at $\mathcal{B}$ at local time $\mathcal{T}_2$. In general, *Node X* *(send time at sender)* → *(receive time at receiver)* *Node Y: {Contents of message}* represents a single synchronization message.
6. *n* is the known maximum depth of the tree and *m* is the # hops to a target node if known. If not known sender assumes *m=n.*
7. $\mathcal{N}_{\mathcal{X}}$ is the crypto-quality nonce from principal $\mathcal{X}$.
8. $\mathcal{K}_j$ is the key disclosed at time *j*.
9. $\mathbb{H}$ is the secure one-way function used to derive verifiable *time commitments*.
10. $\mathcal{F}$ is a secure weak collision resistant one-way function to derive one-way key chains.
11. $\mathcal{T}_i$ represents the local time of node at instance *i*, and $\mathcal{TC}_{\mathcal{AB}}$ which represents a time commitment between $\mathcal{A}$ and $\mathcal{B}$ is the collision free hash that contains a temporary secret and a time value that the sender commits itself to upfront.
12. $(\mathcal{A}.\mathcal{B}.\mathcal{C}.\mathcal{D}....)$ is a non alterable path component created by the time source ($\mathcal{B}$ in our case).

4.4.3.2 Our Secure Time Synchronization Protocol

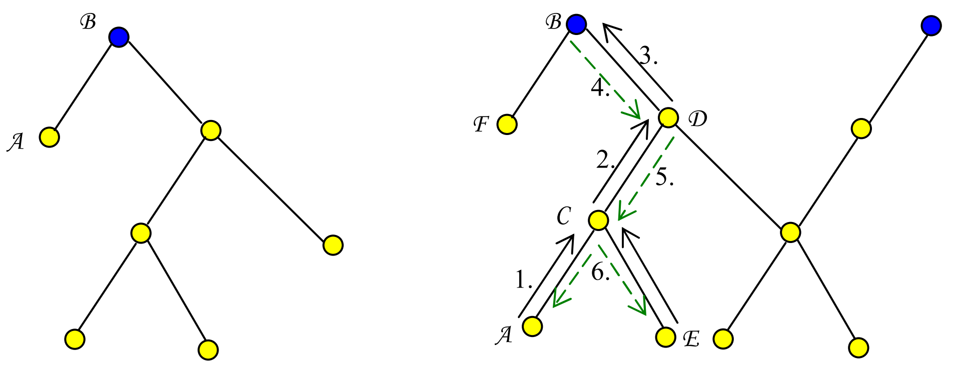


Figure 19: (a) Single hop (b) multi-hop

Protocol Description

Case 1: Single hop synchronization

Figure 19 (a) represents the single hop time synchronization scenario. If $\mathcal{B}$ is the source of global time for a subset of the network (as shown), then $\mathcal{A}$ synchronizes with $\mathcal{B}$. As per the basic description earlier, $\mathcal{A}$ sends a synchronization request message to $\mathcal{B}$. Through this message $\mathcal{A}$ additionally *commits* its current time value T_1 publicly by creating a *time commitment* $\mathcal{TC}_{\mathcal{AB}}$ which locks in the value of $\mathcal{A}$'s time apriori. Specifically, the commitment contains the identities of the sender and receiver (in our example $\mathcal{A}$ and $\mathcal{B}$), the synchronization request time T_1, the depth of the tree from source $\mathcal{B}$ to $\mathcal{A}$ (which for a single hop is 1), a crypto quality nonce $\mathcal{N}_{\mathcal{A}}$, and a one way hash chain value $\mathcal{K}_m$ which serves as the temporary secret. When $\mathcal{B}$ receives the synchronization request, it checks to see if the hashed commitment is stale i.e., if the hash chain value $\mathcal{K}_m$ has already been disclosed, and that the time remaining till exposure of the secret is not too large. This ensures that the message is not a replay, not too old to process, not too early that it amounts to futile storage cost for $\mathcal{B}$, and also to avoid clocks to fall out of synch if the interval is too large. If the message is acceptable, then $\mathcal{B}$ waits for the key to be disclosed. At a suitable time as per the disclosure schedule, $\mathcal{A}$ discloses a value from the hash chain $\mathcal{K}_j$ at time $\mathcal{T}_j$. On disclosure, $\mathcal{B}$ can verify that the time commitment was indeed produced legitimately by the generator of the secret key by following the hash chain to the end. If it verifies correctly, $\mathcal{B}$ sends the synchronization reply message which includes *$\mathcal{B}$'s* timestamps T_2 (message receipt time), T_3 and (response message send time). Similar to $\mathcal{A}$, $\mathcal{B}$ also authenticates its time values by including a commitment of its receive time T_2 and send time T_3 in the synchronization reply message. The steps are shown below:

<u>**Case 1: Single Hop**</u> $\mathcal{A} \rightarrow \mathcal{B}$

<u>At $\mathcal{A}$:</u>

Compute time commitment [A-B] $\mathcal{TC}_{\mathcal{AB}}$:

$$\mathcal{TC}_{\mathcal{AB}} = \{ \mathbb{H}\ (\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P}) \}\ ;\ m<n,\ \mathcal{P}=(\mathcal{A} \mid \mathcal{B} \mid m)$$

$$(1)\ \mathcal{A}_{\mathcal{T}_1} \rightarrow_{\mathcal{T}_2} \mathcal{B} :\ \{ \mathcal{A}, \mathcal{B}, \mathcal{T}_1, \mathcal{N}_{\mathcal{A}}, \mathcal{TC}_{\mathcal{AB}}, m, \mathbb{H}(m) \}$$

<u>At $\mathcal{B}$:</u>

If hash chain key has not been exposed, and time interval is not too much in the future or in the past, process the request:

(2) $\mathcal{B}_{\mathcal{T}_3} \rightarrow_{\mathcal{T}_4} \mathcal{A}: \ \{\mathcal{B}, \mathcal{A}, \mathcal{T}_2, \mathcal{T}_3, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{B}}, m', \mathcal{TC}_{\mathcal{BA}}\}$

Verify Time $\mathcal{T}_1$ on key disclosure. If *true*

Synchronize A.

Else, discard request.

<u>At $\mathcal{A}$:</u>

Calculate offset and delay (Synchronize with $\mathcal{B}$)

If bi-directional,

Repeat procedure at $\mathcal{B}$.

Case 2: Multi hop synchronization

Figure 19 (b) represents the multi hop time synchronization scenario. If $\mathcal{B}$ is the source of global time for a subset of the network (as shown), then $\mathcal{A}$ attempts to synchronize with $\mathcal{B}$ through $\mathcal{C}$, $\mathcal{D}$. Similar to the single hop case, each node on route to the source will perform time commitments (MACs) over the previous node's commitment. Thus there will be a nested series of commitments that serve to not only authenticate the time values of each node along the path but also to assert and verify the path and set a temporal order amongst node along the same path to the source. We show subsequently how these play an important part in mitigation some special attacks.

In the multi hop case, $\mathcal{A}$ now sends a synchronization request message *for $\mathcal{B}$ to $\mathcal{C}$.* As before, through this message $\mathcal{A}$ *commits* its current time value T_1 publicly by creating a *time commitment* $\mathcal{TC}_{\mathcal{AB}}$ which locks in the value of $\mathcal{A}$'s time apriori. The commitment contains the identities of the sender and receiver (in our example $\mathcal{A}$ and $\mathcal{B}$), the synchronization request time T_1, the depth of the tree from source $\mathcal{B}$ to $\mathcal{A}$ if known to $\mathcal{A}$, a crypto quality nonce $\mathcal{N}_{\mathcal{A}}$, and a one way hash chain value $\mathcal{K}_m^{\mathcal{A}}$ which serves as the temporary secret. If $\mathcal{A}$ does not know the number of hops to the source, it can assume *m* to be equal to the maximum depth of the tree *n*. When $\mathcal{C}$ receives the synchronization request, it checks to see if the hashed commitment is stale i.e., if the hash chain value $\mathcal{K}_m^{\mathcal{A}}$ has already been disclosed, and that the time remaining till exposure of the secret is not too large. If the message is acceptable, then $\mathcal{C}$ computes the next message and adds its own nested commitment to the message $\mathcal{TC}\left(\mathcal{C}, \mathcal{TC}_{\mathcal{AB}}\right)_{\mathcal{C}}$ as shown below. This process continues for all nodes along the path until the synchronization message reaches $\mathcal{B}$. $\mathcal{B}$ waits for the keys to be disclosed. At a suitable time as per their disclosure schedules, $\mathcal{D}$, $\mathcal{C}$ and $\mathcal{A}$ disclose a value from their respective hash chains $\mathcal{K}_j^{\mathcal{D}}, \mathcal{K}_k^{\mathcal{C}}, \mathcal{K}_l^{\mathcal{A}}$ at times T_j^D, T_k^C, T_l^A. On disclosure, $\mathcal{B}$ can verify that the time commitment was indeed produced legitimately by the generator of the message by following the hash chain to the end. If it verifies correctly, $\mathcal{B}$ sends the synchronization reply message which includes $\mathcal{B}$*'s* timestamps T_6 (message receipt time), T_7 and (response message send time). The downstream messages follow the same pattern as the upstream messages. If all nodes along the path are honest, this technique allows not only $\mathcal{A}$, but also all nodes along the path to $\mathcal{B}$ to correctly synchronize with the source $\mathcal{B}$ in the same iteration. The steps are shown below:

Case 2: Multi-Hop $\mathcal{A} \rightarrow C \rightarrow \mathcal{D} \rightarrow \mathcal{B}$

At $\mathcal{A}$:

Compute time commitment [A-B $\mathcal{TC}_{\mathcal{AB}}$]:

If # hops to B known, select m=# hops

Else, m= n

Time Commitment [A-B] $\mathcal{TC}_{\mathcal{AB}} = \{\mathbb{H}(\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_\mathcal{A} \mid \mathcal{P})\}; m<n, \mathcal{P}=(\mathcal{A} \mid \mathcal{B} \mid m)$

$$(1)\ \mathcal{A}_{\mathcal{T}_1} \rightarrow_{\mathcal{T}_2} C: \ \{\mathcal{A}, \mathcal{B}, \mathcal{T}_1, \mathcal{N}_\mathcal{A}, m, \mathcal{TC}_{\mathcal{AB}}, \mathbb{H}(m)\}$$

At C:

If hash chain key has not been exposed, and time interval is not too much in the future, or in the past, process the request:

$$(2)\ C_{\mathcal{T}_3} \rightarrow_{\mathcal{T}_4} \mathcal{D}: \ \left\{\mathcal{A}, \mathcal{B}, C, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_1, \mathcal{N}_\mathcal{A}, \mathcal{N}_C, m, \mathcal{TC}\left(C, \mathcal{TC}_{\mathcal{AB}}\right)_C\right\}$$

Else, discard request.

At $\mathcal{D}$:

If hash chain key has not been exposed, and time interval is not too much in the future, or in the past process the request:

$$(3)\ \mathcal{D}_{\mathcal{T}_5} \rightarrow_{\mathcal{T}_6} \mathcal{B}: \ \left\{\begin{array}{l} \mathcal{A}, \mathcal{B}, C, \mathcal{D}, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_1, \mathcal{N}_\mathcal{A}, \mathcal{N}_C, \mathcal{N}_\mathcal{D}, \\ m, \mathcal{TC}\left(\mathcal{D}, \mathcal{TC}\left(C, \mathcal{TC}_{\mathcal{AB}}\right)_C\right)_\mathcal{D} \end{array}\right\}$$

Else, discard request.

At $\mathcal{B}$:

If hash chain key has not been exposed, accept message and wait for disclosure.

On disclosure, first verify key by following hash chain.

If hash verifies correctly, process the request:

Time-stamp is authentic if $\mathcal{TC}_{\mathcal{AB}} = \mathbb{H}\left(\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P}\right)$ *and so on*

Observed Propagation time/Transit time = True Transit time if all hashes verify correctly.

Else, discard request.

(4) $\mathcal{B}_{\mathcal{T}_7} \rightarrow_{\mathcal{T}_8} \mathcal{D}: \ \{(\mathcal{A}.\mathcal{C}.\mathcal{D}.\mathcal{B}), \mathcal{B}, \mathcal{A}, \mathcal{T}_6, \mathcal{T}_7, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{B}}, \mathcal{N}_{C}, \mathcal{N}_{\mathcal{D}}, m', \mathcal{TC}_{\mathcal{BA}}\}$

Synchronize with $\mathcal{A}$ *in the reverse order.*

<u>At $\mathcal{D}$:</u>

If hash chain key has not been exposed, accept message and wait for disclosure.

On disclosure, first verify key by following hash chain.

If hash verifies correctly, process the request:

Time-stamp is authentic if $\mathcal{TC}_{\mathcal{AB}} = \mathbb{H}\left(\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P}\right)$ *and so on*

Observed Propagation time/Transit time = True Transit time if all hashes verify correctly. Synchronize with B

Else, discard request.

(5) $\mathcal{D}_{\mathcal{T}_9} \rightarrow_{\mathcal{T}_{10}} C: \begin{Bmatrix} (\mathcal{A}.\mathcal{C}.\mathcal{D}.\mathcal{B}), \mathcal{B}, \mathcal{D}, C, \mathcal{T}_6, \mathcal{T}_7, \mathcal{T}_8, \mathcal{T}_9, \mathcal{N}_{\mathcal{A}}, \\ \mathcal{N}_{\mathcal{B}}, \mathcal{N}_{C}, \mathcal{N}_{\mathcal{D}}, m', \mathcal{TC}\left(\mathcal{D}.\mathcal{TC}_{\mathcal{BA}}\right)_{\mathcal{D}} \end{Bmatrix}$

<u>At $\mathcal{C}$:</u>

If hash chain key has not been exposed, accept message and wait for disclosure.

On disclosure, first verify key by following hash chain.

If hash verifies correctly, process the request:

Time-stamp is authentic if $\mathcal{TC}_{\mathcal{AB}} = \mathbb{H}\left(\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P}\right)$ *and so on*

Observed Propagation time/Transit time = True Transit time if all hashes *verify correctly. Synchronize with B*

Else, discard request.

$$(6)\ \mathcal{C}_{\mathcal{T}_{11}} \rightarrow_{\mathcal{T}_{12}} \mathcal{A} : \begin{Bmatrix} (\mathcal{A}.\mathcal{C}.\mathcal{D}.\mathcal{B}), \mathcal{B}, \mathcal{D}, \mathcal{C}, \mathcal{A}\mathcal{T}_6, \mathcal{T}_7, \mathcal{T}_8, \mathcal{T}_9, \mathcal{T}_{10}, \mathcal{T}_{11}, \mathcal{N}_{\mathcal{A}}, \\ \mathcal{N}_{\mathcal{B}}, \mathcal{N}_{\mathcal{C}}, \mathcal{N}_{\mathcal{D}}, m', \mathcal{TC}\left(\mathcal{C}.\mathcal{TC}\left(\mathcal{D}.\mathcal{TC}_{\mathcal{BA}}\right)_{\mathcal{D}}\right)_{\mathcal{C}} \end{Bmatrix}$$

<u>At $\mathcal{A}$:</u>

If hash chain key has not been exposed, accept message and wait for disclosure.

On disclosure, first verify key by following hash chain.

If hash verifies correctly, process the request:

$$\mathcal{TC}_{\mathcal{AB}} = \mathbb{H}\left(\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P}\right)$$

Extract values $\mathcal{T}_1$ through $\mathcal{T}_{12}$ and synchronize with $\mathcal{B}$.

Of the potential attacks listed in section 5.1 this scheme mitigates attack M1 (False timing data/Insertion Attack) by the use of hash chains. M3 (Replay attacks) are mitigated by the use of nonces and timestamps. The use of nonces and verifiable hashes require a node to wait for the original message from its upstream nodes and in the case of a reply, a node needs to wait for the original reply from the downstream nodes. Hence M4 (rushing attacks) can be averted. Since this scheme is not dependent on shared secrets, compromised nodes do not provide any adversarial advantages and hence M5 (Compromised nodes) is avoided. A more thorough analysis would be required to comment on attacks M8 (colluding node attack) and M9 (Power save mode attacks). The analysis is described in detail in the following chapter.

Chapter 5 Analysis of Secure Time Synchronization

In this section, we will analyze the working of the protocol under special attack scenarios. We show that in the face of each of the attacks mentioned below, the properties of the designed time synchronization scheme are satisfied and that the protocol does not terminate at an undesirable state.

5.1 Attacks against Time Synchronization

The schemes discussed above in Section 4.1.1 are meant for non-adversarial scenarios due to which simple attacks by malicious nodes to foil the synchronization process will be successful. Few schemes discussed in Section 4.1.2 which are intended for adversarial environments are also susceptible to various attacks. In this section, we summarize some of the common attacks that can be launched against time synchronization schemes. These are:

(M 1) Malicious Nodes send false timing data/Insertion Attack: In RBS & TPSN type of schemes where hierarchical synchronization is done, if a non root node at the upper level sends malicious timestamps to the nodes below it, all the downstream nodes will fail to synchronize correctly. For e.g., if n_2 synchronizes through n_1, then if n_1 sends malicious data to n_2, n_2 will end up synchronizing to the incorrectly inserted value. Thus the entire synchronization process can be disrupted.

(M 2) Malicious nodes jam data &/or delay messages: Sometimes, a malicious node does not have to alter a timing message before forwarding. It can simply jam/block the message from being received by a node under attack. Another attack would be simply delaying the message instead of jamming it.

(M 3) Malicious nodes replay older messages: An older message maybe replayed to trick the downstream nodes into synchronizing with an incorrect timestamp.

(M 4) Malicious receiver can send next message before receiving the request: A malicious receiving node, can forward a time-stamped request to its downstream node without receiving a legitimate request from its upstream node. Another variation would be when a malicious receiver sends a reply message back to the upstream node before receiving a message back from its downstream nodes.

(M 5) Compromised node: If a legitimate node gets compromised, all secrets in possession of the node can be used by the attacker to carry out new attacks. These include sending legitimate requests, fabricating or masquerading as a legitimate user, etc A clever adversary can very easily use this to its advantage and foil the synchronization procedure. A worse scenario would be if the adversary is able to deplete the battery of other nodes in the otherwise secured network by tricking the nodes into legitimately synchronizing with itself repeatedly.

(M 6) Colluding nodes: Any number of colluding nodes can cause worm holes or collectively fabricate data in the network causing time synchronization protocols based on propagation delay and neighbor time values to fail.

(M 7) Power Save Mode Attacks: These types of attacks are possible in power-conscious networks where some power-saving schemes are being used for optimizing and prolonging battery life of the nodes. An adversary that has knowledge of the power-saving / sleep scheduling schemes used in a network can optimize its own behavior to take advantage of it and foil the synchronization procedure. Thus, if such schemes are being employed, we stress that an analysis under such conditions is equally important.

In this work, we do not consider the attacks on the hash chaining method itself, (for example, reusing hash key indices, reusing older hash keys, etc), which are well documented and protected against in works like [3].

5.1.1 Replay and Redirect Attacks

Replay attacks occur when an adversary stores a copy of a message and replays it at a later time after the original message was intended to be used. We examine the single hop case where T is an adversary that records message 1 sent by A to B (Refer section 4.4.3.2). After some time t' has elapsed, T decides to replay message 1 to B. Since the message is neither signed nor encrypted B has no way of knowing that the message originally came from A. As a result, B accepts the message initially. If the elapsed time t' is greater than the disclosure time, and the hash chain commitment (key K_m) has been disclosed already, the freshness verification will fail and B will discard the request as per protocol. If on the other hand, the elapsed time t' is lesser than the time left for disclosure (i.e., key disclosure corresponding to this message has not occurred yet) then if N is a true crypto-quality nonce, B will detect the replay comparing the nonce with currently stored nonces. The existence of a match makes the assertion to B that it has seen this message before. It is worth noting here that a node only needs to store a nonce until a little longer than the time window of validity of the nonce. A nonce expires when the commitment key associated with a message (and hence with the nonce) is disclosed. A node need not store a nonce beyond this expiry since a message can be rejected on the basis that the time stamp is no longer fresh. We can do this because both the nonce and the time stamp are committed by the sender. Since timestamps can only advance from the previous message, and nonces are required to be unique across a large time window, replayed messages cannot be successful.

Thus, the presence of timestamps and nonces in the messages serves as an adequate countermeasure to this type of attack. The multi-hop case is similar to the single hop case and no new additional information is made available to the adversary. In both the single hop as well as the multi-hop case, we see that simply replaying an older message is not a successful attack.

A *redirect* attack occurs when a message is sent to a third entity instead of the intended participant. It can manifest in two ways, with original message suppression and without. In the former case, the original message is suppressed from reaching the receiver and redirect to a different entity, whereas in the latter case, the message is "replayed" to a different entity and not the originally intended recipient without having suppressed the original message. We argue that in a wireless medium only the latter is a valid attack because message suppression in a wireless medium is difficult to achieve. Subsequently, there are various physical and mac layer techniques like spread spectrum [69] etc that sufficiently mitigate selective suppression attacks. Therefore, we only consider the latter case in our analysis.

Let us examine the case where adversary *T* redirects message (1) to a different node *D*. *D* will initially accept the message and wait for key disclosure to validate the message. Since the key commitment *TC* contains the identities of both the original sender and intended recipient, on key disclosure *D* will find that the message is not intended for itself and discard it.

5.1.2 Masquerade Attack

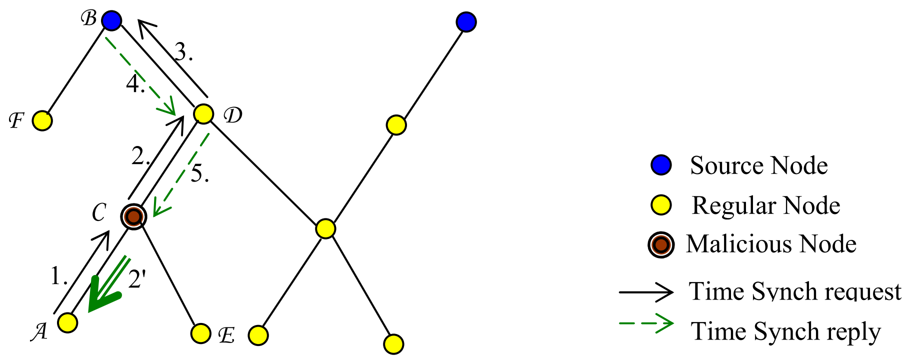


Figure 20: Node C masquerading as the source B

A masquerade attack occurs when a user presents itself to the system as another user, usually a legitimate one. This may be done in order to gain unauthorized access to information or resources, to disseminate (mis)information in another's name, or to block or deny a system from operating correctly. [60]

A malicious node may masquerade as a legitimate upstream or downstream node or as the source of global time itself in order to mislead other (mainly downstream) nodes. In our example, $\mathcal{C}$ is the malicious node that is trying to masquerade as the source of global time $\mathcal{B}$ to $\mathcal{A}$. (Refer Figure 20) If $\mathcal{A}$ wants to resynchronize its time after a finite and pre-configured interval, it sends out a synchronization request message as per protocol by creating its Time Commitment for $\mathcal{A}\ \mathcal{B}$ and creating the synchronization message as per the protocol description in the earlier section. To masquerade as $\mathcal{B}$, $\mathcal{C}$ cannot simply replay part of $\mathcal{B}$'s earlier message (as shown above in section 6.1). The only way for $\mathcal{C}$ to trick $\mathcal{A}$ into falsely believing it is communicating with $\mathcal{B}$ is by sending a fabricated synchronization reply message to $\mathcal{A}$ which is made to appear as if it came from $\mathcal{B}$. Since disclosed keys cannot be reused [3], and $\mathbb{H}$ is a strong pre-image resistant one-way function, while key K_m is undisclosed, we argue that as per the birthday paradox, adversary C cannot find a suitable $\mathcal{K}_m{}'$, X such that $\mathbb{H}\left(\mathcal{K}_m{}' \mid X\right) = \mathcal{TC}_{\mathcal{BA}}$ - the true reply commitment without a significantly large number of tries. Also, the adversary has to attempt this before a disclosure is made by $\mathcal{B}$, otherwise $\mathcal{A}$ will discover that the message it holds is not from $\mathcal{B}$. (Note that if an adversary instead of trying to generate an authentic time commitment simply replaces with an arbitrarily computed time commitment, the message will still be discarded. We discuss this premise in the MITM attack below.)

5.1.3 Man-in-the-middle (MITM) and message capture attacks

A man in the middle (MITM) attack is one in which the attacker intercepts messages in an exchange and then retransmits them, (sometimes substituting his own crypto primitives for the requested one) so that the two original parties still appear to be communicating with each other. The attack may be used to intercept, read or alter messages without the knowledge of either transacting party.

As indicated in the earlier section, message capture, suppression, selective jamming are not easy to achieve in a wireless medium. Various techniques at the physical and mac layer sufficiently reduce the possibility and question the practicality of these attacks. One seemingly practical way of launching a MITM is if the adversary can send a message before the original is received at the recipient or to quickly compute a response and send before legitimate reply reaches the intended recipient. A multi-hop path is more conducive to this type of attack due to the practicality of creating a middle man message before the legitimate responder can create it. Additionally, in a multi-hop, the intermediary node can easily make the end nodes believe that they are communicating with each other via a single hop by making itself *invisible* if the end nodes are not aware of the overall topology. We examine the single and multi-hop cases for this attack as follows:

Single hop case: Figure 21 shows a single and multi hop MITM. Nodes $\mathcal{B}$ and $\mathcal{F}$ are involved in the single hop case. $\mathcal{F}$ sends a synchronization request *i* to $\mathcal{B}$ $\mathcal{F}_{\mathcal{T}_1} \rightarrow_{\mathcal{T}_2} \mathcal{B}: \left\{\mathcal{F}, \mathcal{B}, \mathcal{T}_1, \mathcal{N}_{\mathcal{F}}, m, \mathcal{TC}_{\mathcal{FB}}\right\}$, X_1 alters the message by altering the time commitment $\mathcal{TC}_{\mathcal{FB}}$ to $\hat{\mathcal{T}}C_{\mathcal{FB}} = \mathbb{H}\left(\hat{K}_m \,|\, \hat{\mathcal{T}}_1 \,|\, \hat{\mathcal{N}}_{\mathcal{F}} \,|\, \mathcal{P}\right)$, where $\mathcal{P}: \left(\mathcal{F} \,|\, \mathcal{B} \,|\, m\right)$ and sends $\hat{i}: \mathcal{F}_{\hat{\mathcal{T}}_1} \rightarrow_{\mathcal{T}_2} \mathcal{B}: \left\{\mathcal{F}, \mathcal{B}, \hat{\mathcal{T}}_1, \hat{\mathcal{N}}_{\mathcal{F}}, m, \hat{\mathcal{T}}C_{\mathcal{FB}}\right\}$ to $\mathcal{B}$ before *$\mathcal{F}$'s* message reaches B. (In practice this is a very hard thing to achieve). $\mathcal{B}$ receives the altered synchronization request message $\hat{i}$ instead of *i* and initially accepts it. When X_1 finally discloses the commitment key $\hat{K}_m$ that can validate $\hat{\mathcal{T}}C_{\mathcal{FB}}$, $\mathcal{B}$ finds that the hash key $\hat{K}_m$ validates the hash commitment but fails the authenticity test. i.e., it is not tied to $\mathcal{F}$'s identity. Since $\mathcal{B}$ does not need to synchronize itself with $\mathcal{F}$, (since $\mathcal{B}$ is the source), in the interest of speeding up communication, it can simply go ahead and send the synchronization response *ii* to X_1

whom $\mathcal{B}$ believes to be $\mathcal{F}$. $\mathcal{B}_{\mathcal{T}_3} \rightarrow_{\hat{\mathcal{T}}_4} \mathcal{X} : \{\mathcal{B}, \mathcal{F}, \mathcal{T}_2, \mathcal{T}_3, \hat{\mathcal{N}}_{\mathcal{F}}, \mathcal{N}_{\mathcal{B}}, m, \mathcal{TC}_{\mathcal{BF}}\}$. Again, $\mathcal{X}_1$ intercepts this message and replaces it with $\hat{ii} : \mathcal{B}_{\mathcal{T}_3} \rightarrow_{\hat{\mathcal{T}}_4} \mathcal{X} : \{\mathcal{B}, \mathcal{F}, \mathcal{T}_2, \mathcal{T}_3, \hat{\mathcal{N}}_{\mathcal{F}}, \mathcal{N}_{\mathcal{B}}, m, \mathcal{TC}_{\mathcal{BF}}\}$ ii'. As in the former case, $\mathcal{F}$ initially accepts the message, but later discards it without synchronization since the hash chain broadcast will render the message invalid. $\mathcal{TC}_{\mathcal{AB}} =$

$\{\mathbb{H}\ (\mathcal{K}_m \mid \mathcal{T}_1 \mid \mathcal{N}_{\mathcal{A}} \mid \mathcal{P})\}$; $m<n$, $\mathcal{P}=(\mathcal{A} \mid \mathcal{B} \mid m)$.

(Alternately, as we have seen in the earlier case, adversary can attempt $2^{\left(\frac{\mathcal{N}o.\ of\ \mathcal{TC}\ bits}{2}\right)}$ to look for a collision in the time commitment to be substituted, for the message to pass the authenticity test, but this is even harder and resource intensive on the adversary for a single time synchronization operation. An operation that repeats very often.)

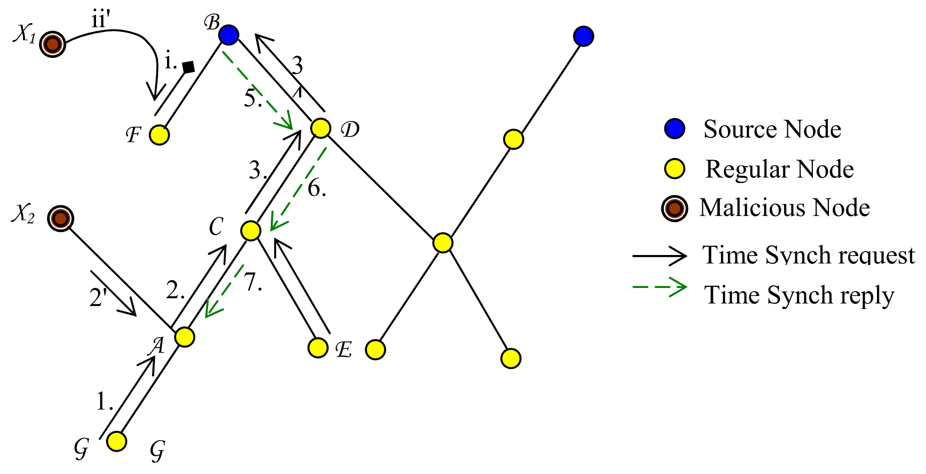


Figure 21: Man-in-the-middle attacks

Multi-hop: Again with reference to Figure 21, when G sends a synchronization request to A, adversary $\mathcal{X}_2$ replies to G before A does (similar to single hop case, but much more practical). As in the first case, X_2 cannot create legitimate hash chain commitments (in a timely manner) matching A or B's identity and the attack fails.

5.1.4 Simple Collusion

Collusion occurs when multiple entities, usually with malicious intent, work in tandem to achieve more prominent results than when each acting alone.

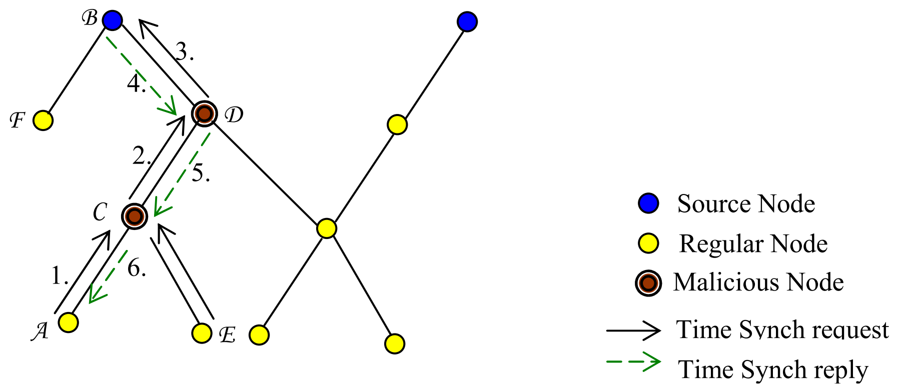


Figure 22: Simple collusion between C and D

In our example Figure 22, $\mathcal{C}$ and $\mathcal{D}$ collude to give the illusion of a shorter path depth and thereby 'advance' a message as follows. During the initial synchronization request phase, $\mathcal{A}$ sends a synchronization request message to $\mathcal{C}$. To give the impression of a shorter path, and hence incorrect synchronization values, $\mathcal{C}$ simply forwards the message to $\mathcal{D}$ without adding its own time commitment or following the format of the protocol. Now, $\mathcal{D}$ appends its time commitment to the message directly after $\mathcal{A}$'s and forwards the message to $\mathcal{B}$. From $\mathcal{B}$'s point of view, the path from the leaf node $\mathcal{A}$ now looks like $\mathcal{A}\rightarrow \mathcal{D}\rightarrow \mathcal{B}$. $\mathcal{B}$ builds the response message accordingly and send it to $\mathcal{D}$. Now, to maintain hop count, and number of time commitments, $\mathcal{D}$ forwards the message as is (without appending its commitment) to its colluding partner $\mathcal{C}$. $\mathcal{C}$ appends its commitment to the original message from $\mathcal{B}$ and sends to $\mathcal{A}$. To $\mathcal{A}$ the view of the network from the source would look like $\mathcal{B} \rightarrow \mathcal{C} \rightarrow \mathcal{A}$.

The presence of the path information component ($\mathcal{A}, \mathcal{D}, \mathcal{B}$) that is included by the source $\mathcal{B}$ in the synchronization response message in our protocol mitigates this attack. A downstream node cannot alter this component without invalidating the integrity check of the message. It is worth noting here that if there are three or more colluding nodes along the same path, for example if there exists another compromised node between $\mathcal{C}$ and $\mathcal{D}$, they can effectively mask the presence of this intermediary node along the path and advance the synchronization message. This form of attack is a type of wormhole attack and hence is discussed in the next sub section.

5.1.5 Wormhole attack

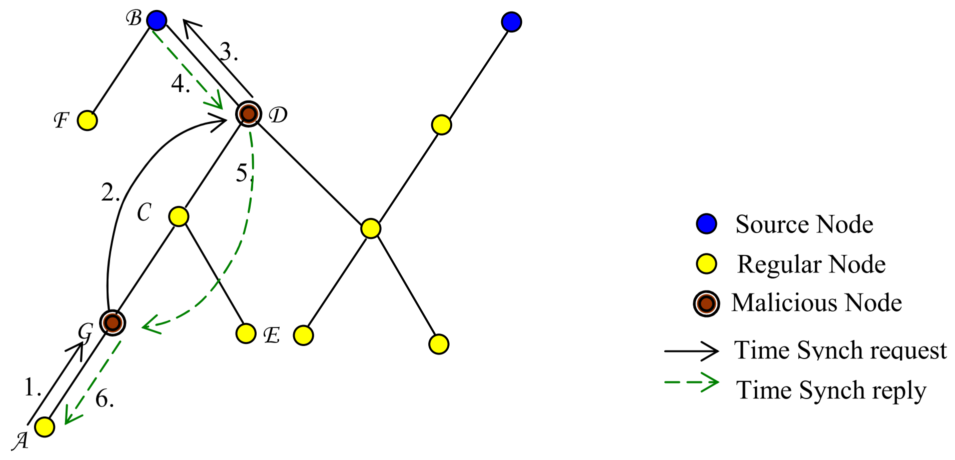


Figure 23: Wormhole between nodes G and D

A wormhole is a tunnel in a network, which allows signals from nodes to travel faster than normal, or sometimes, gives the impression that messages are traveling through the *shortest path (lesser hop count) than actual. Malicious nodes frequently use this technique to cause a large number of messages to be directed through them without actually deviating from protocol.*

Consider the case when two nodes significantly apart tunnel messages between themselves, either through a side channel or due to the compromise of all the nodes existing between them to give the impression that they are next hop from each other. This will result in the synchronization path appearing smaller than actuality. In our example (Figure 23) $\mathcal{G}$ and $\mathcal{D}$ collude resulting in a wormhole in the network. Thus, if $\mathcal{A}$ sends a synchronization message to $\mathcal{G}$, $\mathcal{G}$ tunnels it to $\mathcal{D}$ bypassing $\mathcal{C}$. $\mathcal{D}$ sends the shorter (faster) message to $\mathcal{B}$ and on receiving a response from $\mathcal{B}$ sends it back through the tunnel to $\mathcal{G}$. Since $\mathcal{B}$ sees a legitimate view of the network, it sends a response that reflects the same path in the path component, viz., $\mathcal{B} \rightarrow \mathcal{D} \rightarrow \mathcal{G} \rightarrow \mathcal{A}$ back. The message will pass all validation checks and will be accepted at $\mathcal{A}$ *if* $\mathcal{A}$ is unaware of *m* the no. of hops between itself and its source $\mathcal{B}$. If $\mathcal{A}$ is aware of the no. of hops between itself and $\mathcal{B}$ then it can calculate the maximum expected message delay (since it is a function of the send time, propagation time, and computation time at each intermediate node). If the total transmission time is greater than the maximum delay, then the timeliness of the message is under question, and an anomalous behavior may be suspected. Therefore, in the current state of the protocol, a wormhole attack is possible *if* a node does not know the distance (in number of hops) between itself and its source. (Recall that our protocol specification dictates that either a node knows the no. of hops *m*, or then assumes $m=n$ and generates a hash chain of appropriate length.) If the condition $m=n$ is exercised, then a wormhole attack may be successful.

The attack can be mitigated by the following two ways: As indicated, if $\mathcal{A}$ had additional topological information, for example, if the degree *m* was known to $\mathcal{A}$, it may detect the anomaly in degree using message delay characteristics. Additionally, if either $\mathcal{A}$ or $\mathcal{B}$ have topological/deployment knowledge like the existence of $\mathcal{C}$ between $\mathcal{G}$ and $\mathcal{D}$, the wormhole fails with high probability. With knowledge of system delays, processing times, etc and by comparing the actual time taken by a message to traverse the said route, an intelligent wormhole detection algorithm can detect a wormhole if one exists (within limits of its false detection rate). Employing watchdogs in the deployment that can observe multiple

5.1.6 Compromised node exhibiting Byzantine Behavior

Once a node is compromised, an adversary has access to all information available to the compromised node, all data stored on it, as well as legitimate use of the identity of the compromised node. In this sub section, we try to examine the effect of such behavior on the functioning of the secure time synchronization protocol.

In the example shown in Figure 24, C is the compromised node that shows Byzantine behavior.

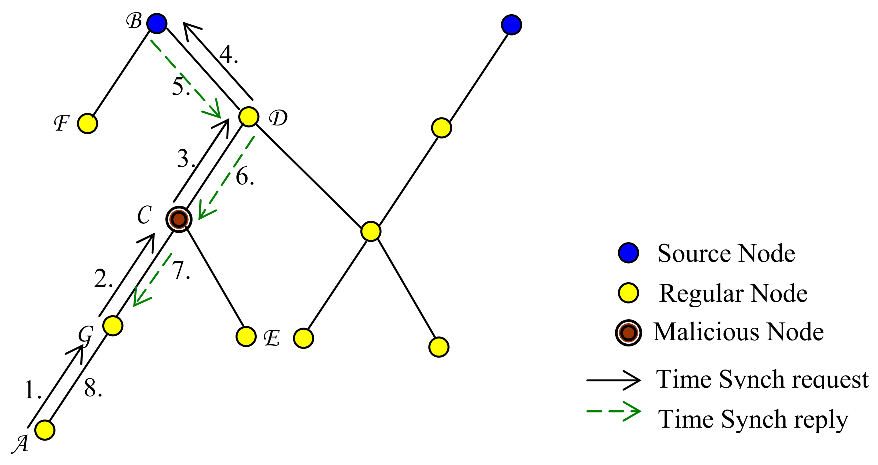


Figure 24: Arbitrary Byzantine behavior

Consider the following case: $\mathcal{A}$ sends a synchronization request message to $\mathcal{G}$ who appends its own commitment and forwards the message to $\mathcal{C}$. As per protocol, $\mathcal{C}$ appends its own commitment to the message and forwards it upstream to $\mathcal{D}$. The message is constructed in a legitimate manner with a legitimate commitment, however, with incorrect timing values. In this case, the messages will validate correctly, and timing data will be successfully accepted by all the nodes in this transaction if the timing value is intelligently fabricated to be within error limits for this deployment. However, due to the nature of the scheme, the damage done by this attack is minimal to zero, depending on the location of the Byzantine node in the tree. Nodes upstream from the Byzantine node will not be affected as they synchronize with the source through their upstream neighbors. All downstream nodes will be impacted, but the effect is more pronounced closer to the Byzantine node and reduces as we traverse deeper into the tree away from the Byzantine node. The deviation in the synchronization error from the average error will be bounded on the upper side by the maximum synchronization error. It is also worth reiterating here, that a Byzantine node can not advance timing values, only delay them, that too by a limited range. Thus, in this example, node $\mathcal{D}$ synchronizes with source $\mathcal{B}$ directly and hence does not get affected by the fabricated timing values from $\mathcal{C}$. $\mathcal{C}$ itself, synchronizes with $\mathcal{B}$ through $\mathcal{D}$ and does not benefit from causing itself to synchronize incorrectly. Nodes downstream from $\mathcal{C}$ will be marginally affected. If the fabricated data is within band (timing values cannot be reduced, only increased, due to the presence of previous node commitments) i.e., $\sigma < \sigma_{max}$, relative synchronization error between $\mathcal{G}$ and $\mathcal{C}$ will increase compared to the average synchronization error in the absence of an adversary. However, the delta synchronization error between $\mathcal{G}$ and $\mathcal{A}$ will reduce slightly due to $\mathcal{G}$'s committed time values prior to the faulty synchronization. Thus in the figure shown, we recreate the messages from A to G to C.

(1) $\mathcal{A}_{\mathcal{T}_1} \rightarrow_{\mathcal{T}_2} \mathcal{G}: \ \left\{\mathcal{A}, \mathcal{B}, \mathcal{T}_1, \mathcal{N}_{\mathcal{A}}, m, \mathcal{TC}_{\mathcal{AB}}\right\}$

(2) $\mathcal{G}_{\mathcal{T}_3} \rightarrow_{\mathcal{T}_4} \mathcal{C}: \ \left\{\mathcal{A}, \mathcal{B}, \ \mathcal{G}, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_1, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{G}}, m, \mathcal{TC}\left(\mathcal{G}, \mathcal{TC}_{\mathcal{AB}}\right)_{\mathcal{G}}\right\}$

(3) $\mathcal{C}_{\mathcal{T}_5} \rightarrow_{\mathcal{T}_6} \mathcal{D}: \ \left\{\begin{matrix} \mathcal{A}, \mathcal{B}, \ \mathcal{G}, \mathcal{C}, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_1, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{C}}, \mathcal{N}_{\mathcal{G}}, \\ m, \mathcal{TC}\left(\mathcal{C}, \mathcal{TC}\left(\mathcal{G}, \mathcal{TC}_{\mathcal{AB}}\right)_{\mathcal{G}}\right)_{\mathcal{C}} \end{matrix}\right\}$

If C is dishonest, it can change the timing value T_4, T_5, (synch request) and T_{12}, and T_{13} (on the return route). All other time values are already committed, and any alteration will result in detection and the message being discarded. Also, these values can only be changed within a small limit (lower bounded by the previous commitment and upper bounded by the allowable synchronization error), as D may not accept the message if the time value is too far out compared to its own clock. Similarly on the return route, C can lie about the true values of T_{12} and T_{13} within a small limit only. The result of this is that every node between C and the source B will synchronize correctly irrespective of the downstream values. C knowing its own true time values will also synchronize correctly (since it has no motive to desynchronize itself). Node G will synchronize with B using time values T_3 through and T_{14}. As a result, the false values of C will cause some effect but will get averaged out due to the presence of time values from other nodes in the path. Similarly, A synchronizes with B through values T_1 through T_{16} and the effect of C's false values is further drowned out. In other words, as the path length increases, the effect of bounded false timing values on the entire multi-hop synchronization decreases provided the false values are not in majority along the path. Note that, there is a trade off here. As the path length increases, the uncertainties associated with each node's individual processing time adds up and creates a non-negligible amount of uncertainty over the entire path. We would like to study this trade off as a future goal.

Multiple Byzantine adversaries

While the presence of a single Byzantine adversary does not significantly impact the security of the time synchronization protocol, this may not hold true in the presence of multiple Byzantine adversaries. Traditionally, Byzantine behavior is tackled using redundancy and thresholding techniques. Common approaches include multi-path multiple message approaches, randomizing and splitting data into chunks that travel through disjoint paths and are assembled at the recipient node to circumvent multiple Byzantine adversaries. Also, impossibility arguments render this problem unsolvable unless the activity of Byzantine adversaries is limited to the simultaneous corruption of a small number of nodes within a certain time window. We leave the study of multiple Byzantine adversaries as a future study goal.

5.1.7 False timing data insertion Attack

Simpler false timing data/insertion attacks are thwarted due to the usage of hash chains. As each node computes a committed time value that is corroborated using its authenticated hash value, any other node cannot insert or replace legitimate values with false data as doing this will simply result in the hash values not verifying correctly. Alternatively, in order for a node to generate a *correct* hash value it needs to attempt a large number of hash computation operations (birthday problem) for a single synchronization operation.

5.1.8 Rushing Attack

The use of nonces and verifiable hashes require a node to wait for the original message from its upstream nodes and in the case of a reply, a node needs to wait for the original reply from the downstream nodes. Hence (rushing attacks) can be averted.

5.1.9 Forging Messages

Again, message forgery is mitigated due to the usage of one way authenticated hash chains. While a forged message can get accepted by a recipient for not failing any integrity checks or lack of obvious anomalies like incorrect nonces, time reversal etc, forged messages will fail the authenticated hash chain broadcasts and will be discarded without using the forged values.

5.2 Communication Overhead

We calculate the overall communication overhead in terms of messages sent for each synchronization transaction to complete. As per the protocol description in Section 4.4.3, n is the known maximum depth of the tree and m is the number of hops to a target node if known. Since for every synchronization transaction, each intermediate node sends three messages, the first is the synchronization response upstream, the second, the synchronization response downstream and the third being the authenticated broadcast. Also, each leaf node sends only message, the synchronization request, and a source sends two messages, the synchronization response and the authenticated broadcast.

Thus, for a given transaction, the number of message required per path = $3n$.

Maximum number of messages (for max depth m) per path = $3m$.

5.2.1 Communication Overhead over a time period $\mathcal{T}$

Since sensor network deployments run on finite power, it would be worthwhile to assess what the communication overhead looks like over a long period of time $\mathcal{T}$.

Therefore, if the Synchronization interval is set to be $\mathcal{T}_{\text{int}}$, then in a given finite time interval $\mathcal{T}$, the number of messages would be

$3n\frac{T}{T_{\text{int}}}$ with a maximum of $3m\frac{T}{T_{\text{int}}}$.

Additionally, there is one more broadcast downstream, (final synchronization source time disclosure) which makes the total number of messages in the above equation to $4n$ and $4m$ respectively.

5.2.2 Communication Overhead during synchronization in tree

In the interest of optimal time synchronization, we can make use of the tree topology so as to reduce redundant downstream between a node and all its children. In this case, when a node at level i sends its synchronization request, all sibling nodes that can hear its request will cache it and compare their own timestamps with their sibling. When the response is received from the parent node, and verification is complete, If an upstream node synchronizes with more than one downstream node, then the number of messages required is as follows:

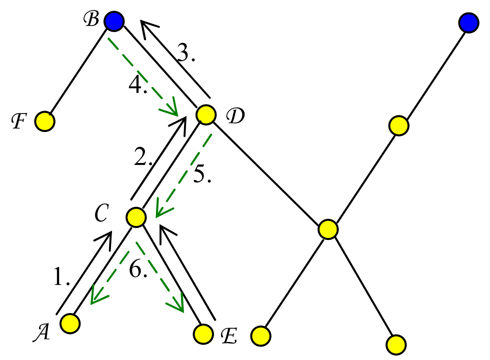


Figure 25: Tree structure

Let N_i be the number of nodes at each level i, and E_i be the total edges between nodes at level i and the level above (i-1) Therefore, the total number of messages per transaction can be calculated as: $\sum_{i=1}^{n} E_i + 2\sum_{i=0}^{n-1} N_i$ for total number of upstream messages plus total number of downstream messages.

$$\text{Total \# messages} = \sum_{i=1}^{n} E_i + 2\sum_{i=0}^{n-1} N_i \tag{5.1}$$

If every node has uniform in-degree per level *i*, then $E_i = \theta_{IN\_i}.N_i$ where θ_{IN_i} represents the degree of each node at level *i*.

$$= \sum_{i=1}^{n-1} \theta_{IN\_i}.N_i + \theta_{IN\_i}.N_n + 2\sum_{i=1}^{n-1} N_i + 2N_0$$

$$= 2N_0 + \theta_{IN\_i}.N_n + \sum_{i=1}^{n-1} N_i\left(2 + \theta_{IN\_i}\right) \tag{5.2}$$

Adding the broadcast requests from upstream nodes, we have

$$\text{Total \# messages} = 2N_0 + \theta_{IN\_i}.N_n + \sum_{i=1}^{n-1} N_i\left(2 + \theta_{IN\_i}\right) + \sum_{i=1}^{n-1} N_i + N_n$$

$$= 2N_0 + N_n\left(\theta_{IN\_i} + 1\right) + \sum_{i=1}^{n-1} N_i\left(3 + \theta_{IN\_i}\right) \tag{5.3}$$

For our example shown, this becomes 4*n* messages.

5.2.3 Storage Overhead

From [14], we have seen that computation and communication cost with hash chains is O(log *m*) and O(log *m*) where *m* is the length of the hash chain. We now calculate the remaining storage cost associated with the protocol. Recall that for every transaction, a node has to store values of all nonces occurring within a certain time period T. If T is large, then synchronization interval is large and as a result undisclosed nonces will have to be stored for a longer time. Also, until a key is disclosed, entire packets (message contents) of time synchronization messages must be stored as well at each node that performs verification). Recall that our synchronization message is very lightweight in comparison with most other schemes. Each message only contains a single hash value in spite of traversing a long multi-hop path. At each hop in the path, the nested hash chain commitment replaces the previous commitment with an *aggregated* commitment for the entire path. Therefore, storage costs associated with each run of the protocol is of the order of no. of messages stored at each node, which is constant at O(1) for each run.

Chapter 6 Conclusion

In this dissertation, we addressed the issue of creating robust services for infrastructure-less distributed networks, the objective being to create robust infrastructure base over which other services can be provided. We selected a wireless sensor network as our distributed network of choice. We chose to create a robust target localization service that is primarily robust against large amounts of falsified data. Most protocol based attacks were thwarted using cryptographic protection techniques like encryption, integrity protection using signatures, etc. However, we have shown that not all attacks can be mitigated using cryptographic protection techniques. Falsified data attacks are certainly immune to cryptographic techniques, and are a real threat to any data centric network. Our approach in this thesis has been to minimize the effect of falsified data on the outcome of the protocol.

We showed that by using a particle filtering algorithm at the core of the protocol we were able to create a model for an adversarial environment that tackles adversarial behavior as noise. By modeling data falsification attacks as statistical variances, and limiting *undetectable* adversarial behavior to within certain bounds, we reduce the amount of malicious data that can be inserted into the target estimating algorithm. We then showed how our target estimating algorithm is robust against the amount of malicious data that is inserted into the algorithm without detection. Our experiments yield that the algorithm is highly robust against large amounts of malicious data. We were able to provide a bound on the minimum number of particles required to be active in the filter for the resilient behavior to prevail. While particle filters have excellent tracking capabilities they are very complex to implement and computationally intensive. We reduced the implementation complexity and computational intensity per node by distributing the particle filter into two components, viz., the measurement sampling component and the aggregation component. Through a novel use of various additional elements like watchdog nodes and randomization features like moving the aggregation from one leader node to another resulted in a significant increase in complexity for an adversary to launch a successful attack without actually increasing the complexity or the cost to the system. This is directly in line with our philosophy of hardening the service by leveraging existing aspects of the distributed network. Watchdog nodes provided sanity checks in terms of distance bounds, frequency of messages, and anomalous behavior in a neighborhood to help eliminate data from such sources at the aggregator. Moving the aggregator function across various nodes in an unpredictable manner also increased the complexity for an adversary to launch attacks since now an adversary has to first guess where the aggregator function will move and then compromise a large subset in the vicinity of that leader node. This also improves the robustness of the algorithm to temporary failures as target estimation resumes as soon as the target moves into the vicinity of the next leader node. Further, using SPSA to cast adversarial behavior as perturbation resulted in solving the multi-variate optimization problem with only two measurements of the objective function per iteration (irrespective of the dimensions of the optimization problem). This resulted in a significantly lightweight solution compared to regular particle filtering that is also real-time efficient and facilitates online target location estimation.

As a secondary problem, we studied time synchronization since it is an important service for a distributed data centric network. While we did not follow a similar approach as with the first problem with a robustness study for time synchronization, we addressed the problem of what it implies to be a robust time synchronization service and how to create one that is lightweight and reliable for other services to rely on. We used a simple cryptographic mechanism called hash chain that helps prevent many attacks like replay, redirection, man-in-the-middle etc and is computationally lightweight (O(log N)). We also identified attacks that our time synchronization protocol is not robust against and will continue to work on hardening it as a future goal.

Through this thesis, we have developed an interesting paradigm of leveraging strong cross disciplinary technologies as the foundation of a robust service and tightly coupling it with cryptographic protection mechanisms. We have understood what it means to create robust protocols, that are communication and computationally feasible to implement in a power conscious environment. The tight coupling of cryptographic mechanisms was complimentary to the target estimation algorithm, and together they provide the protocol with stronger properties than what they can individually.

As a future goal, we would like to address some of the tangential issues that were identified throughout this thesis. Understanding how watchdog distribution and their density of deployment affects the accuracy of the scheme and how an adversary can take advantage of this knowledge is an interesting study. Further, we claimed that the size of the window of validity of target position measurements was tightly coupled with the tolerable synchronization error, and affects storage and cost. As another study we would like to quantify this relationship. Currently, we consider all nodes except the target to be static. We would like to explore the effect of introducing select mobility models, particularly constrained path model, on the target localization protocol and its robustness. As concerns time synchronization, hash chains have a pattern of disclosure and a predetermined schedule of disclosure. We would like to study the tactics an adversary can use and the attacks it can launch if it has knowledge of this schedule. The selection of values for T_Δ and T_d (synchronization error and disclosure time interval) is also interesting as an improper selection can provide appropriate windows of opportunity to an artful adversary. Finally, we would like to understand how Byzantine behavior affects time synchronization and what lightweight protection mechanisms can provide resilience to the service against such an attack.

References

[1] A. Doucet., N. de Freitas, N. Gordon, eds.: Sequential Monte Carlo Methods in Practice. Series: *Statistics for Engineering and Information Science.* Springer-Verlag, New York (2001).

[2] A. Mainwaring, J. Polastre, R. Szewcyzk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring", *In Proc of Wireless Sensor Networks and Applications (WSNA'02)"*, September 2002.

[3] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", IETF RFC 4082, June 2005.

[4] A. Savvides, C. Han and M. Srivastava, "Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors", *In Proc of MOBICOM 2001*, Rome, Italy, July 2001.

[5] A. Soto, "Self Adaptive Particle Filter", *In the Proceedings of the nineteenth International Conference on Artificial Intelligence (ICAI), Scotland, August 2005.*

[6] A. Tanenbaum, M. V. Steen, "Introduction to Distributed Systems", *Distributed Systems Principles and Paradigms*, Prentice Hall, NJ, 2002.

[7] A. Tanenbaum, M. V. Steen, "Leader Election Algorithms", pgs 262-271, *Distributed Systems Principles and Paradigms*, Prentice Hall, NJ, 2002.

[8] A. Want, A. Hopper, V. Falcao, and J. Gibbons, "The Active Badge Location system," *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, 1992.

[9] Alternative Paradigms in Wireless Networking, RFI SN07-09 , Jan 18th, 2007

[10] B. Hofmann-Wellenhof, H. Lichtenegger and J. Collins, "Global Positioning System: Theory and Practice", Fourth Edition, Springer-Verlag, 1997.

[11] B. Krishnamachari, D. Estrin, and S. B. Wicker, "Modeling Data-Centric Routing in Wireless Sensor Networks", *Technical Report CENG 02-14*, Dept. of Computer Engineering, USC, 2002.

[12] C. Karlof and D. Wagner, “Secure Routing in Sensor Networks: Attacks and Countermeasures”, *In Proc. of First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[13] C. Karlof, N. Sastry, D. Wagner, “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks”, *in Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems*, p-162-175, Baltimore, 2004

[14] D. Coppersmith and M. Jakobsson, “Almost optimal hash sequence traversal”, In *Proceedings of the Fourth Conference on Financial Cryptography (FC’02)*, Lecture Notes in Computer Science, 2002.

[15] D. Crisan and A. Doucet, “A survey of convergence results on particle filtering for practitioners”, *IEEE Transactions on Signal Processing,* Vol.50, Issue 3, pp:736-746, March 2002.

[16] D. Crisan and A. Doucet. “Convergence of sequential Monte Carlo methods,” *Technical Report Cambridge University,* CUED/FINFENG/TR381, 2000.

[17] D. Crisan. “Sequential Monte Carlo Methods in Practice”, Chapter 2, pages 17-41.Springer-Verlag, 2001.

[18] D. Fox, “KLD-Sampling: Adaptive particle filters”, In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 14, 2001.

[19] D. Liu and P. Ning, “Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks”, In *Network and Distributed System Security Symposium, NDSS ’03*, pages 263–276, February 2003.

[20] D. Liu, and P. Ning, “Multi level μTESLA: Broadcast authentication for distributed sensor networks,” *ACM Transactions in Embedded Computing Systems (TECS)*, Vol.3, no.4, pp 800-836, 2004

[21] D. Liu, P. Ning, and W. Du, “Attack-resistant location estimation in sensor networks”, *in Proc. of the Fourth International Workshop on Information Processing in Sensor Networks (IPSN)*, pp: 99-106, 2005.

[22] D. Niculescu and B. Nath, Ad Hoc Positioning System (APS) using AoA, *In Proc. of INFOCOM*, San Francisco, CA, USA, March 2003.

[23] E. Jovanov, D. Raskovic, J. Price, A. Moore, J. Chapman, and A. Krishnamurthy, "Patient Monitoring Using Personal Area Networks of Wireless Intelligent Sensors", *Biomedical Sciences Instrumentation*, Vol. 37, pp373-378, 2001.

[24] E. Wilson, M. Hilferty, "The distribution of chi-square," *In Proceedings of the National Academy of Sciences of the United States of America,* vol. 17, pp 684-688.

[25] F. Zhao, L. Guibas, "Wireless Sensor Networks, An Information Processing Approach," Elsevier Morgan Kaufmann Publishers, San Francisco, 2004.

[26] G. Lu, N. Sadagopan, B. Krishnamachari, A. Goel, "Delay Efficient Sleep Scheduling in Wireless Sensor Networks", *in Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies, (Infocom)*, Vol. 4, pp 2470-2481, March 2005.

[27] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," *in IEEE Symposium on Security and Privacy*, Berkeley, California, May 11-14, 2003, pp. 197-213

[28] H. Dai and R. Han. "Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks." *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 8/1 pp:125–139, January 2004.

[29] http://wins.rsc.rockwell.com: Sensor Specification

[30] I. Rekleitis, "A particle filtering tutorial for Mobile Robot Localization", *Technical Report TR-CIM-04-02*, Centre for Intelligent Machines, McGill University, Montreal, Quebec, Canada, 2004.

[31] IEEE Communications, Vol.6, number 9, September 2005 and http://www.coe.berkeley.edu/labnotes/0805/honicky.html

[32] J. Elson, L. Girod, and D. Estrin. "Fine-grained network time synchronization using reference broadcasts", *In Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

[33] J. Geweke., "Bayesian inference in econometric models using Monte Carlo integration." *Econometrica*, vol. 57, pp.1317–1339, 1989.

[34] J. Hightower, G. Boriello, and R. Want, "SpotON: An indoor 3D Location Sensing Technology Based on RF Signal Strength," *Technical Report*, University of Washington 2000-02-02, 2000.

[35] J. Jacod and P. Protter, Probability Essentials. Springer, 2000.

[36] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks", *In Proc. of Mobicom '99*, Seattle, WA, August 1999.

[37] J. Spall, 'An overview of the simultaneous perturbation method for efficient optimisation," *John Hopkins Technical Digest*, vol. 19, no. 4, pp. 482-492, 1998.

[38] J. Spall, "Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol 34, pp 817-823, 1998.

[39] J. Spall, "Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation," *IEEE Transactions on Automation and Control*, vol 37, pp 332-341, 1992.

[40] J. van Greunen, and J. Rabaey. "Lightweight time synchronization for sensor networks", *In Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 11–19, September 2003.

[41] K. Marzullo and S. Owicki. "Maintaining the time in a distributed system", *In Second annual ACM symposium on Principles of distributed computing*, pages 295–305. ACM Press, 1983.

[42] K. Rőmer, "Time synchronization in ad hoc networks", *In ACM Symposium on Mobile Ad-Hoc Networking and Computing*, pp 173-182, Long Beach California, October 2001.

[43] K. Sun, P. Ning, C. Wang, "Fault-Tolerant Cluster-Wise Clock Synchronization for Wireless Sensor Networks," in *IEEE Transactions on Dependable and Secure Computing (TDSC), Vol. 2, No. 3, pages 177-189,* July-September 2005.

[44] K. Sun, P. Ning, C. Wang, "Secure and Resilient Clock Synchronization in Wireless Sensor Networks," in IEEE Journal on Selected Areas in Communications (JSAC), *Vol. 24, No. 2,* February, 2006

[45] K. Sun, P. Ning, C. Wang, A. Liu, Y. Zhou, "TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks," In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06),* Alexandria, VA, November 2006.

[46] K.B. Rasmussen, and S. Capkun, "Implications of Radio Fingerprinting on the Security of Sensor Networks", In *Proceedings of IEEE SecureComm*, 2007

[47] L Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," *in Proceedings of the 9th ACM Conference on Computer and Communications Security,* Washington D.C., USA, November 18-22, 2002, pp 41-47.

[48] L. Lamport, "Time, Clocks, and the ordering of events in a distributed system", *Communications of the ACM,* Vol. 21(7), pp 558-565, July 1978.

[49] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Agreement Problem," *ACM Transactions on Programming Languages and Systems*, July 1982, pages 382-401.

[50] L. Lazos, and R. Poovendran, "SeRLoc: Secure Range Independent localization for wireless sensor networks", *In ACM Workshop on Wireless Security (WiSe)* 2004.

[51] L. Lazos, S. Capkun, and R. Poovendran, "Rope: Robust position estimation in wireless sensor networks", *In International Workshop on Information Processing in Sensor Networks (IPSN)*, 2005.

[52] L. Zhang, Y. Guan, "A Topology-aware Single Packet Attack Traceback Scheme," *In proceedings of Securecomm,* 2006.

[53] Location and Connection Aware Content Pushing (LOCO), Proposer Information Pamphlet (PIP) for Defense Advanced Research Projects Agency (DARPA) Strategic Technology Office (STO), BAA 07-12, July 20th, 2006, Reissue: Jan 17th, 2007.

[54] M. Coates, "Distributed Particle Filters for Sensor Networks", *in proceedings of the third International Symposium on Information Processing in Sensor Networks (IPSN),* pp: 99-107, 2004.

[55] M. Manzo, T. Roosta, S. Sastry, “Time Synchronization Attacks in Sensor Networks”, *In proceedings of the Security of Ad hoc and Sensor Networks,* 2005.

[56] M. Maroti, B. Kusy, Gyula Simon, and Akos Ledeczi. “The flooding time synchronization protocol”, *Technical Report ISIS-04-501*, Institute for Software Integrated Systems, Vanderbilt University, Nashville Tennessee, 2004.

[57] M. Miller, N. Vaidya, “Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio”, *in Proceedings of IEEE Wireless Communications and Networking Conference,* 2004.

[58] M. Miller, N. Vaidya, “Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio”, *in Proceedings of IEEE Wireless Communications and Networking Conference,* 2004.

[59] M. Valenti. Smart Sensors. “A Technology Assessment Impact (Technical Insights)”, *Technical report,* Frost & Sullivan, September 2004

[60] Masquerading web definition ac.bcc.ctc.edu/Policies/definitions.htm

[61] Mica2 Data Sheet: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[62] N. Bulusu, J. Heidemann and D. Estrin, “GPS-less Low Cost Outdoor Localization for Very Small Devices”, *In IEEE Personal Communications Magazine*, Vol. 7(5), pp:28-34, October 2000.

[63] N. Sastry, U.Shankar, D. Wagner, “Secure Verification of Location Claims”, *ACM Workshop on Wireless Security*, September 2004.

[64] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan, “The Cricket location-support system,” *in Proceedings of ACM MobiCom*, pp. 32–43, 2000.

[65] P. Castro, P. Chiu, T. Kremenek, and R. Muntz, “A Probabilistic Room Location Service for Wireless Networked Environments,” *in Proceedings of the Third International Conference Atlanta Ubiquitous Computing (Ubicomp)*, vol. 2201. Springer-Verlag Heidelberg, September 2001.

[66] P. Bahl and V. N. Padmanabhan, “RADAR: An In-Building RF-Based User Location and Tracking System,” *in Proceedings of IEEE Infocom,* vol. 2, 2000, pp. 775–784.

[67] Q. Li and D. Rus. "Global clock synchronization in sensor networks", *In Proc of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM),* Vol. 1,(7), pp 564-574: March 2004.

[68] R. Anderson and M. Kuhn. "Tamper Resistance - a Cautionary Note", *In Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1-11, November 1996.

[69] R. Pickholtz, D. Schilling, and L. Milstein, "Theory of Spread Spectrum Communications – A tutorial", *In the IEEE Transactions on Communications,* Vol. 30(5), pp855-884, May 1982.

[70] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The Active Badge Location system," *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, 1992.

[71] S. Capkun, J. Hubaux, "Securing position of wireless devices with application to sensor networks", *In Proc of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM),* 2004.

[72] S. Capkun, M. Hamdi and J. Hubaux, GPS-Free Positioning in Mobile Ad-Hoc Networks, In Proc. of HICCSS 2001,Maui, Hawaii, USA, January 2001.

[73] S. Capkun, M. Srivastava, and M. Cagali, "Securing localization with hidden and mobile base stations", *Technical Report, NESL-UCLA*, 2005.

[74] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks", *In First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[75] S. Ganeriwal, S. Capkun, C. C. Han, M.B. Srivastava, "Secure Time Synchronization Service for Sensor Networks", *ACM Workshop on Wireless Security (WiSe)*, October 2005.

[76] S. K Kim, R. Iltis, "Performance Comparison of Particle and Extended Kalman Filter Algorithms for GPS C/A Code Tracking and Interference Rejection", *36th Annual Conference on Information Sciences and Systems,* Princeton, March 2002.

[77] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," *In Proc. of Fifth*

Symposium on Operating Systems Design and Implementation (USENIX - OSDI '02), December 2002.

[78] S.B. Wicker, and M.D. Bartz, “Type II Hybrid – ARQ Protocols Using Punctured MDS Codes”, *In Proceedings of IEEE Transactions on Communications,* April 1994.

[79] T-H. Lin, H. Sanchez, H. Marcy, and W. Kaiser. “Wireless Integrated Network Sensor nodes (WINS) for Tactical Information Systems”, *In Proc. of the Government Microcircuit Applications Conference*, 1998.

[80] V. Gligor, P. Donescu, “New modes of encryption – A perspective and proposal”, *NIST modes of operation workshop*, Baltimore, MD, USA, October 2000.

[81] V. Hingne, A. Joshi, T. Finin, H. Kargupta, and E. Houstis. “Towards a Pervasive Grid”, *To Appear in NSF Next Generation Systems Program Workshop at International Parallel and Distributed Processing Symposium (IPDPS' 03),* April 2003.

[82] V. Mhatre and C. Rosenberg. “Homogeneous vs. Heterogeneous Sensor Networks: A Comparative Study”, *In Proc. of IEEE International Conference on Communications*, volume 6, pages 3646.3651, June 2004.

[83] W. Su and I. F. Akyildiz. “Time-diffusion synchronization protocol for sensor networks”, *IEEE/ACM Transactions on Networking*, 2004.

[84] Wireless Network after Next (WNaN) Adaptive Network Development (WAND), Broad Agency Announcement for Defense Advanced Research Projects Agency (DARPA) Strategic Technology Office (STO), BAA 07-07, February 23rd, 2007.

[85] Y. C. Hu, A. Perrig, and D.B. Johnson. “Packet leashes:A defense against wormhole attacks in wireless ad hoc networks”, *In Proceedings of INFOCOM 2003*, April 2003.

[86] Y. Hu, A. Perrig, and D. B. Johnson, “Efficient security mechanisms for routing protocols”, In *Network and Distributed System Security Symposium, NDSS ’03*, pages 57–73, February 2003.

[87] Y. Hu, M. Jakobsson, and A. Perrig, "Efficient Constructions for One-way Hash Chains", *SCS Technical Report Collection*, 2003.

[88] Y. Sella, "On the computation-storage trade-offs of hash chain traversal", In *Proceedings of Financial Cryptography 2003 (FC 2003)*, 2003.

[89] Y. Yao and J. E. Gehrke. "The Cougar Approach to In-Network Query Processing in Sensor Networks", *SIGMOD Record*, Vol. 31(3), pp:9-18, September 2002.

[90] Z. Li, W. Trappe, Y. Zhang and B. Nath, "Robust Statistical Methods for securing wireless localization in sensor networks", *In the International Workshop on Information Processing in Sensor Networks (IPSN)*, 2005.