ABSTRACT

Title of dissertation: WISE Abstraction Framework
For Wireless Networks

Seungjoon Lee, Doctor of Philosophy, 2006

Dissertation directed by: Professor Samrat Bhattacharjee
Department of Computer Science

Current wireless networks commonly consist of nodes with different capabilities (e.g., laptops and PDAs). Link quality such as link error rate and data transmit rate can differ widely. For efficient operation, the design of wireless networks must take into account such heterogeneity among nodes and wireless links.

We present systematic approaches to overcome problems due to heterogeneous node capability and link quality in wireless networks. We first present a general framework called WISE (Wireless Integration Sublayer Extension) that abstracts specific details of low-level wireless communication technologies (e.g., modulation or backoff scheme). WISE provides a set of common primitives, based on which upper-level protocols can operate efficiently without knowing the underlying details.

We also present a number of *protocol extensions* that employ the WISE framework to enhance the performance of specific upper-level protocols while hiding lower-level heterogeneity (e.g., link error rate). Our *multihop WLAN architecture* improves system performance by allowing client nodes to use multihop paths via other clients to reach

an AP. Our *geographic routing extension* considers both location and link quality in the next hop selection, which leads to optimal paths under certain conditions. To address heterogeneity in node capability, we consider *virtual routing backbone construction* in two settings: cooperative and selfish. In the cooperative setting, we present a protocol extension that constructs an optimal backbone composed of a small number of high-capability nodes, which can be generalized to a more resilient backbone. For the selfish case, we use game theory and design an incentive-compatible backbone construction scheme.

We evaluate our work from multiple perspectives. We use theoretical analysis to prove that our extensions lead to optimal solutions. We use simulations to experiment with our schemes in various scenarios and real-world implementation to understand the performance in practice. Our experiment results show that our schemes significantly outperform existing schemes.

WISE ABSTRACTION FRAMEWORK FOR WIRELESS NETWORKS

by

Seungjoon Lee

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
2006

Advisory Committee:
Professor Samrat Bhattacharjee, Chair/Advisor
Professor Udaya Shankar
Professor Samir Khuller
Professor François Guimbretière
Professor Mark Shayman

DEDICATION

To my family.

ACKNOWLEDGMENTS

I deeply thank my advisor Professor Bobby Bhattacharjee for his constant support, encouragement, and guidance. He always believed in me and encouraged me based on positive aspects from me. He also spent a great amount of his time and energy helping me improve on my weak points. I also thank my committee members for their feedback and support. Professor Udaya Shankar provided me with valuable feedbacks throughout my thesis work. I am indebted to his attention and encouragement throughout my graduate study. Research discussions with Professor Samir Khuller about various projects were very helpful, and conversations with him on various issues were always pleasant and provided me with surprising insight. Professor Mark Shayman was willing to be on my defense committee and share his expertise in wireless networking from electrical engineering background. Professor François Guimbretière suggested different perspectives and helped me think in a broader way.

I was extremely fortunate to work with Professor Aravind Srinivasan, which was a pure pleasure. He always welcomed scholarly discussions as well as other conversations about various issues. I thank Professor Neil Spring for his valuable feedback on various parts of my graduate work. I am also indebted to my former advisor Professor Chongkwon Kim at Seoul National University for his continuous guidance and support.

During my graduate study, I was fortunate to work with many talented individuals. I learned a tremendous amount working with Professor Suman Banerjee, who also provided

valuable feedback and support. Working and writing papers with Professor Richard La, Dr. Girija Narlikar, Dr. Lisa Zhang, Dr. Gordon Wilfong, Dr. Martin Pàl, Minho Shin, Dr. Hyojun Lim, Dr. Bohyung Han, Professor Yoo Ah Kim, Vijay Gopalakrishnan, Rob Sherwood, Dave Levin, Ryan Braud, Yijie Han, and Bo Han was a great experience. I was fortunate to have weekly research meetings and discussions with KGSYS members and want to acknowledge them: Professor Hyeonsang Eom, Joon-Hyuk Yoo, Jihwang Yeo, Minho Shin, Soobum Lee, Sunghyun Chun, Professor Yoo Ah Kim, Jae Hwan Lee, Minkyong Cho, Ji Sun Shin, and Jinhyuk Jung.

I am grateful that Mind Lab generously allowed me to use some of their equipments for various experiments. I am especially indebted to Bao Trinh, who willingly helped me with various technical issues during the experiments. I thank Dr. Christopher Kommareddy, Arunchandar Vasan, and Sunyoung Ju for helping me with various wireless experiments. I also thank administrative and technical staff members of the department for their support.

I want to thank many friends in Korea and those who I met in Maryland. They supported me during different phases of my life in the graduate school. There are too many, and I will not list all their names to avoid making a mistake of leaving out someone. Nevertheless, I wish my most sincere gratitude to reach each of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Current wireless networks typically consist of heterogeneous devices. For example, a laptop and a PDA can be in the same wireless local area network (WLAN). However, the disparity in CPU speed between them can lead to orders of magnitude difference in performance for cryptographic operations [1]. Wireless links connecting these nodes have different qualities as well. When a node in a multihop wireless network communicates with its neighbors using an IEEE 802.11g network interface card, the data transmission rates of wireless links can range from 1 Mbps to 54 Mbps with widely varying link error rates [2, 3]. With the increase of mobile devices and the evolution of communication technologies, the degree of such heterogeneity within a single network will grow further in the future.

We develop systematic strategies to overcome problems due to heterogeneous node capacity and link quality in wireless networks. Such problems result mainly from the binary abstraction (viz, a link up/down) exported by the link layer. While such an abstraction has served well in the wired network domain, wireless links exhibit diverse properties, and network operations need to adapt to these underlying differences to achieve satisfactory system performance. In Figure 1.1, suppose that S is communicating with T using the shortest path composed of links with high error rates. This causes repeated packet errors, which can lead to operation inefficiency and even unsuccessful commu-

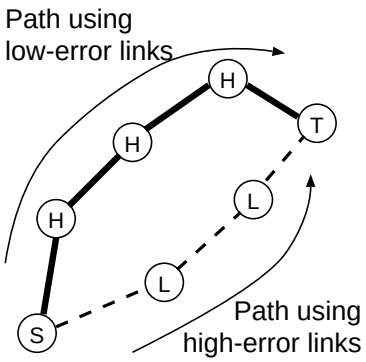


Figure 1.1: One path is composed of low-error links (thick lines on the top), while the other uses high-error links (dotted lines in the bottom).

nication. To solve this problem, upper-layer routing protocols need to differentiate link quality and find an alternate (possibly longer) path made of links with low error rates. In this dissertation, we provide efficient and effective mechanisms for upper-layer protocols to differentiate node capability and link quality.

In some prior approaches, individual upper-layer protocols explicitly consider lower-level details in their operations for heterogeneity adaptation [3, 4]. For example, the SP-Power routing scheme [4] considers the path loss exponent value to find an energy-efficient path. However, in these approaches, upper-layer protocols have to deal with various details possibly for a number of link-level protocols (e.g., CSMA, TDMA). As a result, the design and implementation of upper-layer protocols inevitably becomes more complicated. In addition, to reflect changes at the lower level such as new MAC protocols or better quality estimation techniques, all the relevant upper-layer protocols need to be modified. Consequently, this approach of modifying individual upper-layer protocols leads to complexity and extensibility issues.

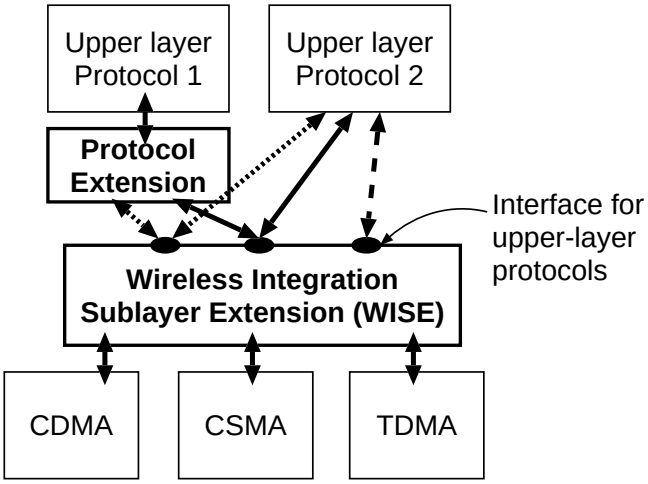


Figure 1.2: The proposed WISE abstraction framework. The WISE abstracts link-level details and provides well-defined interfaces to upper-layer protocols. *Protocol extensions* use WISE primitives to solve specific problems due to heterogeneity.

1.1 Our Approach

We first propose a new framework called Wireless Integration Sublayer Extension (WISE) that abstracts specific details of underlying wireless communication technologies. (See Figure 1.2.) WISE defines and provides a set of common primitives, which enables structured access to link-level details. To differentiate node capacity or link quality, upper-layer protocols simply use exported primitives without knowing the underlying details. For example, WISE exports the available link bandwidth using the state-of-the-art estimation technique for the underlying MAC protocol, and an upper-level protocol uses the estimate to find a high-bandwidth path. To realize the defined service, a WISE *implementation* deals with underlying characteristics specific to lower-level protocols. To estimate the available bandwidth, for example, a WISE implementation may use underlying details such as data transmission rate, backoff scheme, and contention level. A

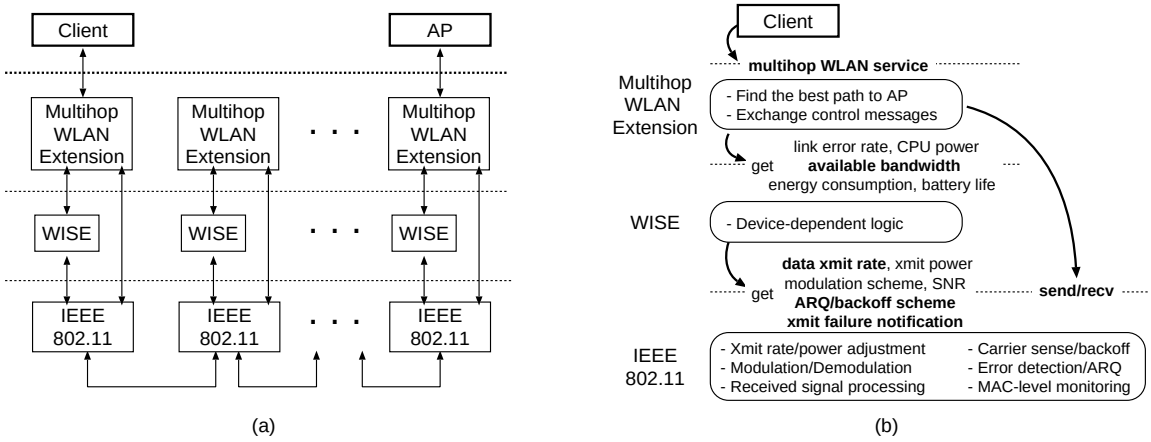


Figure 1.3: Diagram for the proposed multihop WLAN architecture. (a) Unaware of the lower-level details, a client and an AP software use the transparent multihop WLAN service to communicate through a multihop path. (b) A more detailed view of provided and used services between different layers. (Names in bold are used in the multihop WLAN extension.)

specific WISE instantiation depends on the underlying MAC and physical-layer (PHY) protocols, interface card, device driver, operating system, etc. However, the WISE provides the same set of common service despite the underlying differences. Thus, the WISE framework provides the flexibility to incorporate later changes (e.g., new estimation techniques) in its implementation without having to modify upper-layer protocols.

We also propose a number of protocol mechanisms called *protocol extensions* that handle underlying heterogeneity in node capacity and link quality (Figure 1.2). A protocol extension addresses a specific problem (e.g., disparate link quality as shown in Figure 1.1) and hides underlying heterogeneity from upper-level protocols. Using the enhanced ser-

vice by the protocol extension, upper-layer protocols can focus on their own objectives without considering underlying diversity. Actual mechanisms in a protocol extension may use one or more WISE primitives and possibly exchange control messages between neighbors. We further elaborate on adaptation extensions with an example below.

We use our design of a multihop WLAN (Wireless Local Area Network) architecture to illustrate the usage of an extension and its interaction with an upper-layer protocol and the WISE. In the IEEE 802.11-based WLAN environment, a client communicates with an AP using a direct link. We extend the current WLAN system to enable the selective use of multihop paths to an AP in case the direct link is of low quality. In the proposed architecture shown in Figure 1.3(a), instead of directly using an 802.11 MAC/PHY entity as in the current WLAN system, a client uses the service provided by the multihop WLAN extension. To find a better multihop path than the direct link, the multihop WLAN extension uses the bandwidth estimation primitive offered by the WISE. It also exchanges control messages between neighbors (Figure 1.3(b)). The WISE in turn estimates and exports the available bandwidth using lower-level details such as current data transmit rate and backoff scheme. When the client requests a packet transmission, the multihop WLAN extension uses the data transmit service by the IEEE 802.11 interface card, and depending on the direct link quality, this packet may go through multiple nodes to reach an AP. However, the client is completely unaware of the underlying details and receives the same WLAN service as in the single-hop WLAN case, only with significantly improved performance.

1.2 Contributions and Organization

In this dissertation, we make a number of contributions in various areas of wireless networking systems.

- We propose a general abstraction framework that provides a uniform set of access interfaces to low-level details. We present a number of interfaces we employ in our protocol extensions and describe how to implement them in the context of IEEE 802.11 systems.

- As discussed above, we design a novel *multihop WLAN architecture*, where unlike the current WLAN systems, end users (or clients) can act as proxies and use multihop paths as necessary to reach their access points (APs). We perform a measurement study in the current IEEE 802.11 WLAN environment, and the results show that a carefully designed multihop WLAN can improve the system performance significantly in terms of end-to-end throughput as well as extended coverage. We present protocol mechanisms for the new multihop WLAN architecture with incremental deployment paths. Our simulation results show that when a node adopts the multihop extension, it improves the performance of non-adopting nodes as well as the adopting node itself. (Chapter 4)

- We propose *an extension for efficient geographic routing* as a general framework for cost-aware geographic routing in multihop wireless networks. In geographic routing (or position-based routing), nodes use location information for packet delivery [5–9]. Our new link metric considers link cost as well as location information

and leads to optimal paths in idealized environments. Our simulation results show that in more realistic scenarios, the new metric achieves significant performance improvement when compared to the current geographic routing scheme. For example, in harsh environments with frequent packet losses, the extended scheme delivers six times more data packets than the current geographic routing protocol. (Chapter 5)

- We consider scenarios where nodes have different resource levels and develop a *routing backbone extension* that uses the concept of connected dominating set (CDS). This backbone can allow low-capacity nodes not in the backbone to save their resources (e.g., save energy to increase their lifetime). We prove that our distributed algorithm can construct a connected backbone that is essentially best possible approximation to a minimum connected dominating set. The resulting backbone also maximizes the minimum capacity node in the backbone. We also generalize this scheme such that it builds a resilient backbone that is more suitable in dynamic networks. Our experiment results show that compared to best existing schemes, the resulting backbone achieves significant energy saving and network lifetime increase. It also provides end-to-end connectivity in high-mobility scenarios. (Chapter 6)

- We consider backbone construction when wireless devices are selfish; unlike cooperative scenarios assumed in Chapter 6, nodes do not want to join the backbone since it consumes more energy. We apply and generalize a game theoretic model for multihop wireless networks and present a backbone construction protocol. Our simulation results show that the resulting backbone is comparable to existing schemes

that assume cooperative nodes. We also have implemented the scheme using real hardware and present experiment results on a testbed. This is the first evaluation of routing backbone used in real systems, and the results show that compared to the case without a backbone, we can achieve a similar level of performance when we use a backbone. (Chapter 7)

We discuss potential areas of future research and conclude in Chapter 8. We first describe some of related work in the next chapter.

Chapter 2

Related Work

In this chapter, we describe previous work related to this dissertation. We first review prior approaches to handling heterogeneity in wireless link quality and node capacity. Then, we describe several schemes proposed to employ multihop paths in infrastructure-based wireless networks (e.g., WLAN, cellular systems). We then present some previous work related to geographic routing and virtual routing backbone construction in multihop wireless networks. We finally review some recent work that considers selfish nodes in wireless networks.

2.1 Heterogeneous Wireless Networks

One of the earliest schemes that attempt to hide peculiarities of wireless links is in the context of TCP throughput enhancement [10]. Since a TCP source interprets packet losses as congestion and reduces the congestion window size, frequent wireless link errors significantly degrade TCP performance. To address this problem, the *snoop* scheme uses TCP-level information at the link layer [10]. When a snoop agent, located at a base station, detects duplicate acknowledgments or local timeouts, it initiates local retransmissions using cached packets. It also suppresses duplicate TCP acknowledgments to prevent the source from initiating the fast retransmit algorithm. The snoop scheme attempts to hide wireless link losses, but does not differentiate qualities between wireless

links.

There are many experiment results that report diverse wireless link quality in practice, and the differentiation of wireless links based on the link quality has become popular. Lundgren et al. [3] identify *gray zone*, where due to high bit error probability, nodes cannot exchange long data packets in most cases. Banerjee et al. [11] propose the use of a link metric based on link error probability. De Couto et al. [2] propose to use a similar routing metric called *ETX (Expected Transmission Count)*, which considers link error probability. They incorporate ETX into DSDV and DSR routing protocols, and the experiment results in a real rooftop network show that paths with smaller ETX perform better than shortest paths. Most IEEE 802.11 products support multiple transmission rates, and Heusse et al. [12] report that the sender with the lowest transmission rate acts as the limiting factor for the throughput of other senders in a WLAN. It is because transmissions by the slowest node take disproportionately long time, and the other nodes must wait before they can transmit next packets. Sadeghi et al. [13] propose a new MAC protocol called OAR (Opportunistic Auto Rate), in which a node transmits multiple packets consecutively if the channel condition is good. This scheme provides *temporal* fairness among nodes and significantly improves the throughput of links at higher data transmission rates.

Since most wireless devices are powered by exhaustible batteries, energy-efficient operation is an important issue in wireless networks. One intensive research area has been about how to find intermediate nodes that minimize power consumption along the paths in multihop wireless networks [4, 14–16]. This line of work is based on the fact that required transmit energy increases super-linearly to the distance [17]. Rodoplu et al. [14] present a localized algorithm that preserves network connectivity and achieves

the globally minimum-energy topology. In PARO [15], a node becomes a relay node if it finds that the relaying leads to lower energy consumption. Given traffic flows and node energy levels, Chang et al. [16] find a set of routes that maximize the system lifetime. On the other hand, according to many measurement studies on current wireless interface cards, an idle or receiving wireless card consumes a comparable amount of energy to a transmitting one, while a card in sleep mode expends far less energy [18]. To exploit this finding, many backbone construction algorithms have been proposed, such that backbone nodes stay awake and maintain network connectivity, and non-backbone nodes can be in sleep mode to save energy [19, 20]. We describe this line of work in more detail in Section 2.4.

2.2 Using Multihop Paths in Infrastructure-based Wireless Networks

Most wireless networks mainly use direct paths to infrastructure such as access points or base stations. In Chapter 4, we present a multihop wireless LAN architecture and describe some of related work here. Lin and Hsu [21] define a new multihop cellular architecture for wireless communication. They examine the general principles of using multihop paths to base stations in cellular networks. Based on useful but simplifying assumptions (e.g., static configurations, centralized routing table construction at all nodes based on an all-pair shortest path algorithm), they demonstrate that such a multihop architecture is beneficial in improving data throughput of cellular systems. In contrast, our work in Chapter 4 significantly builds on these general observations made in [21]. We propose multihop extensions at the *MAC-layer*, define detailed protocol mechanisms for

interoperability with existing IEEE 802.11 standards, and present detailed performance evaluation studies through actual measurements as well as simulations involving both static and mobile scenarios. Wu et. al. [22] proposed an ad-hoc relaying system on top of existing cellular networks. Focused on reducing the call blocking probability, the iCar system uses dedicated ad hoc relaying stations (ARSs) at vantage points. In contrast, our multihop WLAN architecture is based on the cooperation of enhanced clients and works without additional dedicated infrastructure.

Dousse et al. [23] have proposed a hybrid network to improve the connectivity of an ad-hoc network. In their definition, a hybrid network is an ad-hoc network which is interconnected by a sparse set of wired backbone nodes. Liu et al. [24] analyze the capacity of such hybrid networks and identify the scaling behavior of capacity with increasing number of wireless and wired nodes. Kozat and Tassiulas [25] also analyze the scaling behavior of hybrid networks where nodes and access points are randomly distributed.

Hsieh and Sivakumar [26] present performance comparisons of conventional cellular networks with ad-hoc wireless networks, and briefly introduce another hybrid network model that switches between a purely cellular network and an ad-hoc network. The base station of the cell uses a centralized algorithm to compute all routes as in multihop networks and disseminates this information to the wireless nodes. In their proposed scheme, at any instant, all wireless nodes operate in the same mode (i.e., either cellular mode or ad-hoc mode, but not both at the same time). In contrast, in our proposed architecture, direct paths and multihop paths co-exist at the same time.

Miller et. al. [27] propose a routing protocol in a hybrid network that uses both APs and multihop relaying clients. The protocol has both proactive and reactive components,

and multihop relaying is restricted to K hops, where K is a small constant (e.g., 2 or 3). As in our proposed mechanism, this work attempts to extend the reach of infrastructure. However, their approach is based on network layer routing, while our work uses MAC-specific information. LUNAR [28] is an ad-hoc routing protocol that also limits the number of intermediate nodes (up to three hops). LUNAR is similar to our work in that it places the ad-hoc routing between MAC layer and IP layer. More recently, Luo et al. [29] have proposed an architecture called UCAN that utilizes ad-hoc routing over 802.11-based interfaces to improve the performance of 3G cellular networks. All nodes in the UCAN architecture are equipped with both 3G cellular and 802.11 interfaces, and a node that observes very low bandwidth on its 3G interface connects to another node with higher 3G bandwidth using multihop relaying over 802.11 capable nodes.

Ben Salem et al. [30] have examined the construction of a multihop wireless packet forwarding technique in the context of cellular networks. The goal of their work was to define incentive-based mechanisms such that cellular users provide multihop forwarding services for each other. Therefore the techniques developed in [30] define a solution to a useful and complementary problem (in the context of cellular networks) to what we address in Chapter 4. Our work can leverage such an approach to provide incentives for mobile clients to serve as proxies in a multihop WLAN.

2.3 Geographic Routing in Multihop Wireless Networks

In Chapter 5, we present a protocol extension for geographic routing such that we can find a low-cost path in multihop wireless networks. In geographic routing (or

position-based routing), nodes use location information for packet delivery in multi-hop wireless networks [5–9]. Neighbors locally exchange location information obtained through GPS (Global Positioning System) or other location determination techniques [31]. Most geographic routing protocols use one-hop information, but generalization to two-hop neighborhood is also possible [32]. Traditional geographic routing schemes use only geometric information such as the length of projection (called *progress*) and angle value against the straight line between source and the destination (please see [9] and the references therein). However, the most straightforward and popular strategy for geographic routing is simply forwarding data packets to the neighbor geographically closest to the destination [5–7].

Although the above *greedy* method is effective in many cases, packets may get routed to where no neighbor is closer to the destination than the current node. Many *recovery* schemes have been proposed to route around such *voids* for guaranteed packet delivery as long as a path exists [5–7, 33]. These techniques typically exploit planar subgraphs (i.e., Gabriel graph, Relative Neighborhood graph) and specific rules to recover from such local minima. For example, *Face Routing* [5] uses the right-hand rule in Gabriel graph, and GPSR employs a similar scheme in its *perimeter mode* [6]. Terminode routing uses *Anchored Geodesic Packet Forwarding (AGPF)* similar to loose source routing [33]. Kuhn et al. present GOAFR+, which is efficient on average cases and worst-case optimal [7]. These recovery schemes are orthogonal and complementary to the use of our proposed link metric.

As in other table-driven and on-demand routing work described in Section 2.1, more recent geographic routing schemes consider link costs in the next hop selection.

Stojmenovic et al. [34] propose a routing metric for power-efficient routing, as discussed in Section 5.4. Seada et al. [35] focus on the minimum energy consumption in lossy environments and propose threshold-based schemes as well as a link metric in Eq. 5.4. Zorzi and Armaroli also independently propose the same link metric [36]. Our work in Chapter 5 is different from them in that we present a more general framework and provide the rationale behind the use of new link metric by proving the optimal tradeoff between hop count and link cost.

2.4 Virtual Routing Backbone in Multihop Wireless Networks

In multihop wireless networks, end-nodes are typically responsible for relaying traffic [37]. However, we often utilize a "connected dominating set" of nodes that form a routing backbone [38, 39]. Many distributed algorithms are proposed to find a connected dominating set. Das and Bharghavan [39] directly apply well-known centralized algorithms [40]. Using the unit-disk graph model, Wan et al. [41] propose a message-optimal algorithm that achieves a constant approximation ratio. Dubhashi et al. [42] propose a distributed algorithm that finds an $O(\log \Delta)$ approximation to the minimum connected dominating set in $O(\log n \log \Delta)$ running time. None of them consider backbone maintenance or node capacity. Since backbone nodes consume more resource (e.g., energy), it is beneficial to include only high-capacity nodes in the backbone. There are a few prior schemes that consider remaining energy level when finding a connected backbone [19, 20, 43]. However, they use node capacity only as a secondary metric, and the resulting backbones often include low-capacity nodes. (See Chapter 6.) In contrast, our scheme described

in Chapter 6 can build a backbone composed only of high-capacity nodes. Also, under reasonable assumptions, we can show the backbone is essentially the smallest possible.

In the context of sensor networks, HEED [44] selects cluster-heads based on the residual energy and other parameters such as node degree. However, HEED assumes that the network is quasi-stationary, whereas in Chapter 6, we consider the problem of backbone construction and maintenance in dynamic wireless networks. More recently, given different node cost, Wang et al. independently propose a backbone construction scheme that attempts to minimize the *sum* of node cost in the backbone [45]. In contrast, our scheme maximizes the minimum capacity node to increase the network lifetime, which we validate by analysis and simulation experiments based on our proposed distributed protocol. There are schemes that exploit the sleep mode operation, but are not based on the connected backbone approach. Zheng and Kravets [46] propose an on-demand power saving scheme, where nodes stay awake according to traffic load and their soft-state timers. As mentioned in Section 2.1 we also can achieve energy saving through transmission power control at each node [14, 47, 48], and such topology control schemes are complementary to our backbone construction scheme.

2.5 Protocol Design in Selfish Environments

In Chapter 7, we present a backbone construction scheme for wireless networks composed of selfish nodes. We describe related prior schemes here.

Among systems that enforce cooperation in wireless networks, the vast majority make use of external incentive mechanisms. Ad hoc-VCG [49] finds a minimum-energy

path by carefully determining the payment amount. Each forwarding node is rewarded (e.g., using money) depending on their forwarding cost announcement, and the authors present a strategy-proof mechanism by applying the game-theoretic principle of Vickrey, Clarke, and Groves auctions [50]. Zhong et al. propose a credit-based system in Sprite [51]. They assume the existence of a centralized Credit Clearance Service (CCS). Each node receives a receipt for each packet forwarded, and submits these receipts to the CCS for compensation. Buttyán and Hubaux [52] use a similar approach using virtual currencies, but rely on tamper resistant hardware to store information about the remaining currency. Zhong et al. [53] design protocols that stimulate cooperation for routing and forwarding using cryptographic techniques. Note that all of these approaches require a public key infrastructure for correctness. In contrast, our work in Chapter 7 uses internal incentives only and does not require external money or security infrastructure. Catch [54] is closely related to our work in two aspects: (1) it uses internal (dis)incentive and (2) requires a detection mechanism such as Watchdog [55], which utilizes the broadcasting property of wireless medium for misbehavior detection.

A few recent papers consider the scenario where selfish nodes do not follow the IEEE 802.11 MAC protocol, for example, by using a small contention window. Cagali et al. [56] apply the bargaining game theory to derive an optimal contention window size that each of multiple cheaters should use depending on the total number of cheaters. Kyasanur and Vaidya [57] present modifications to the IEEE 802.11 protocol to facilitate the detection of such selfish nodes using RTS and CTS frames. Raya et al. [58] classify different MAC level misbehavior techniques and present a monitoring system that runs on access points to detect and prevent selfish nodes from achieving higher performance. All

these protocols consider the issue of a node deviating from the MAC protocol to achieve gain (*e.g.*, higher throughput). In contrast, our work in Chapter 7 considers the existence of selfish nodes in the context of backbone construction and correct message forwarding in multihop wireless networks. Hence, these prior schemes are complementary to our work.

Chapter 3

Wireless Integration Sublayer Extension (WISE)

WISE provides a structured set of mechanisms for upper-layer protocols to access lower-level information. WISE exports a number of primitives independent of lower-level communication technologies. To differentiate node capability and link quality, upper-layer protocols use the exported WISE primitives without knowing the underlying mechanisms. In some cases, the WISE processes several low-level details to return a value of interest. For example, the WISE can return link error probability inferred from the current modulation scheme, data transmission rate, and signal-to-noise ratio (SNR). The implementation of WISE depends on the underlying communication technology, hardware architecture, operating system, and device driver.

The WISE framework nominally exports all interesting behaviors and characteristics of a generic underlying wireless link. However, depending on the capability of underlying wireless interface card and device driver, a specific WISE implementation may implement only a subset of primitives. Therefore, upper-level applications should first check which WISE primitives are implemented in the current node. In this section, we present the exported WISE primitives and implementation techniques that we have used in our work. Although we can have different link-level protocols under the WISE, in this chapter we focus on the implementation techniques based on the IEEE 802.11 standard. In the future, we plan to expand the WISE framework with more primitives, which

WISE Interface	Return Value
`get_per(from, to, plen)`	Packet error rate (PER)
`get_delay(from, to, plen)`	Link delay of wireless link
`get_link_bw(from, to, plen)`	Link bandwidth of the link
`get_pwr_required(from, to, plen)`	Power consumption required for packet transmission
`get_remaining_battery()`	Remaining battery level

Table 3.1: Current primitives exported by WISE.

operates on top of different wireless technologies (e.g., TDMA).

3.1 WISE Interfaces and Implementation

We first present a set of primitives that WISE exports for interesting lower-level details. In Table 3.1, we list the WISE primitives we have defined and used in the design of proposed extensions described in Chapters 4, 5, 6, and 7. In the table, `from` is the one end of wireless link, `to` is the other end, and `plen` is the packet length. There can potentially be other useful WISE primitives (e.g., `get_CPU_power()`), but we focus on those we use later in this dissertation.

We now demonstrate how the WISE implements these exported primitives. Note that the techniques used to implement the primitives are hidden from upper-level protocols, and we can later reflect any future advances into the system-specific implementation without modifying upper-layer protocols.

3.1.1 Packet Error Rate (PER) Estimation

Wireless links are typically more prone to packet errors than wired ones, and many theoretical and empirical schemes are proposed to estimate packet error probability on a wireless link [2, 17, 59, 60]. Some of previous work uses additional probe messages for packet error rate estimation in the bootstrapping phase [2, 61]. However, such control messages consume already scarce network resources. Also, network environments may change over time (e.g., due to mobility), and old link estimates may become obsolete.

We first describe two simple packet error models that can be easily used in real wireless networks. We then provide multiple estimation techniques thus enabling nodes to choose the best scheme for the current network and system setting. In a resource-rich network, for example, nodes can use a method that uses probe messages. In the case of a dense large-scale network with limited resources, such probe messages may prove to be costly, and nodes can use an alternate scheme that uses no extra control messages.

3.1.1.1 Error Models

Many theoretical models for wireless packet error have been proposed [2, 17, 59]. The *independent bit error model* is among the simplest for wireless packet errors. In this model, each bit is corrupted independent of other bits in the packet. Specifically, if the bit error rate is p_b, then the error probability for an L-bit packets is:

$$PER(L) = 1 - (1 - p_b)^L. \tag{3.1}$$

A number of previous results show that bit errors are often correlated and occur in a bursty fashion [17, 59, 62, 63]. Some previous works use finite-state Markov models to

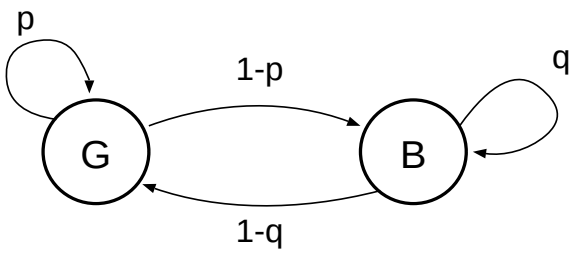


Figure 3.1: Gilbert/Elliot model. G denotes *good* state, and B denotes *bad* state.

model such correlated bit errors. Although such a model can have an arbitrary number of states, for simplicity, we use the *two-state Markov model* proposed by Gilbert and Elliot [62, 63] in this chapter. In the Gilbert/Elliot (GE) model, a wireless channel is in one of the following two states: *good* and *bad* (Figure 3.1). If the channel is in good state, then a bit transmission error occurs with the probability of e_g. On the other hand, if the channel is in bad state, the probability of bit transmission error is e_b. Prior to the transmission of each new bit, the channel may change states or remain in the current state. Figure 3.1 shows the GE model representation with state-transition probabilities. In this chapter, we assume that $e_g = 0$ and $e_g = 1$ for simplicity.

For this model, we can calculate the steady-state probability of being in *good* state (P_G) and *bad* state (P_B) as follows:

$$P_G = \frac{1-q}{2-p-q}, \qquad P_B = \frac{1-p}{2-p-q}.$$

Then, the error probability of an L-bit packet is:

$$\textit{PER}(L) = 1 - (P_G\, p^L + P_B\, (1-q)p^{L-1}). \tag{3.2}$$

Note that there are two cases where no bit error occurs in a packet. First, the channel is initially in good state and remains there for all bit transmissions, and the probability

is $P_G\ p^L$ for L-bit packet. In the other case, the channel is initially in bad state, but the channel changes into good state for the first bit transmission and remains in good state. This probability is $P_B\ (1-q)p^{L-1}$. A packet error occurs if none of them happens, and hence Eq. 3.2.

3.1.1.2 Estimation Techniques

Using Signal-to-Noise Ratio (SNR) for PER Estimation We can estimate the link bit error rate (BER) using SNR measurement by the wireless card and theoretical error models for different modulation schemes [17]. Assuming an AWGN (Additive White Gaussian Noise) channel, the bit error rate p_b of the BPSK (Binary Phase Shift Keying) modulation scheme is given by:

$$p_b = 0.5 \times \textit{erfc}(\sqrt{\frac{P_r \times W}{N \times f}}), \tag{3.3}$$

where P_r is the received power, W the channel bandwidth, N the noise power, f the transmission bit rate, and *erfc* the complementary error function. Most wireless cards typically measure SNR = $10 \log \frac{P_r}{N}$ (dB) for each received packet, and using such SNR values and Eq. 3.3, a node can calculate p_b for its neighbors and corresponding packet error rates, for example, by using Eq. 3.1. Due to potential asymmetry in link quality, a node may need to inform its neighbors of respective SNR values. This can be done either via additional control messages or by modifying the beacon message format to include the information.

This scheme is useful primarily in free-space environments, but not applicable for indoor environments, where signal path characteristics are more complex. The measure-

ment results using a rooftop mesh network show that it is hard to predict link quality using SNR even in outdoor environments [64]. However, in a different measurement study using a sensor network, Zuniga et al. [60] report that empirical results closely match their analytical models.

Using Probe Messages for PER Estimation If any upper-level protocol is already using probe messages [2,6,61], the WISE can extract the link error probability from them. However, since such probe messages are usually shorter than data packets, a node may experience higher PERs for actual packets than the observed PER [2]. To obtain more accurate link cost estimation, we need to adjust PER depending on the data packet length. We next describe how to adjust PERs for longer data packets.

Suppose we use the independent bit error model and know only one observed PER value for l-bit probe messages denoted by $PER(l)$. Then, from Eq. 3.1, we can infer $p_b = 1 - (1 - PER(l))^{1/l}$, and for a L-bit data frame we use the following estimation:

$$PER(L) = 1 - (1 - PER(l))^{L/l}. \tag{3.4}$$

In case we want to use the two-state Markov model shown in Figure 3.1, we need at least two distinct PER values observed for different probe message types. In addition to $PER(l)$, consider $PER(m)$ for m-bit probe messages. Using Eq. 3.2, we can get one of the state transition probabilities in Figure 3.1 as follows:

$$p = \left(\frac{1 - PER(l)}{1 - PER(m)} \right)^{\frac{1}{l-m}}.$$

Then, we can estimate the PER of L-bit data messages using the following formula:

$$PER(L) = 1 - (1 - PER(m)) \left(\frac{1 - PER(l)}{1 - PER(m)} \right)^{\frac{L-m}{l-m}} \tag{3.5}$$

In Section 3.2, we present measurement results to illustrate how we can employ these estimation approaches in practical wireless systems.

Neighborhood Monitoring for PER Estimation The WISE also can use passive monitoring to infer link PERs. For example, in IEEE 802.11 networks, node A in *promiscuous* mode can monitor all frames sent by neighbors. In that case, A can infer the PER of link $B \rightarrow A$ by using the MAC sequence number and counting how many frames from neighbor B it has missed. Again, since the quality of two directional links may differ, A needs to inform B of the PER estimation as in the previous scheme.

Self Monitoring for PER Estimation The three methods above require either additional control messages or the modification of beacon message format. When these are not possible, we suggest the following technique. Whenever a node transmits a data frame to neighbor n, the MAC-layer informs the WISE whether the transmission was successful or not. Let us define an indicator variable F; $F = 1$ when a frame exchange failed, and $F = 0$ otherwise. Then, WISE infers the PER of wireless link to neighbor n as follows:

$$\mathit{PER}_n \leftarrow (1 - \alpha)\mathit{PER}_n + \alpha F, \tag{3.6}$$

where α denotes the weight parameter. In the simulation study in Section 5.4, we use $\alpha = 0.1$, and the default PER value is set to 0. Note that $F = 1$ even when an ACK frame failure occurs in IEEE 802.11 networks [2].

To track the link quality change even when no packets are forwarded to n, we use an *aging* scheme and periodically reduce PERs of unused links. When this reduction makes the estimated PER become lower than the actual one, packets may be forwarded to n, but

the estimated PER will increase after transmission failures. The magnitude and frequency of reduction should balance such overhead and prompt adjustment. In the simulation in Section 5.4, we multiply PERs of unused links by 0.9 every 30 seconds.

3.1.2 Link Delay Estimation

We can think of two types of link delay. First, due to the broadcast nature of wireless medium, it is desirable to minimize the *medium time*, the time spent in sending a packet over the link [65]. When the underlying physical medium supports multi-rate transmissions (e.g., the IEEE 802.11 standard), it is a function of the current transmission rate. The WISE can easily retrieve the current value of transmission rate from the MAC layer and calculate the necessary medium time to the neighbor as follows. Consider the IEEE 802.11 RTS/CTS access method, where RTS and CTS are transmitted at 1 Mbps. We can calculate the medium time τ as follows:

$$\tau = \frac{L_{RTS} + L_{CTS}}{1.0 \times 10^6} + \frac{L_{DATA} + L_{ACK}}{b_{data}} + 4T_{PHY} + 3\mathit{SIFS} + \mathit{DIFS}, \qquad (3.7)$$

where L_X denotes the length of respective frame in bits, T_{PHY} is the transmission time for 192-bit PHY header, and *SIFS* and *DIFS* are inter-frame intervals [66]. In Table 3.2, we present typical values for IEEE 802.11b/g. L_{DATA} is determined by `plen` (See Table 3.1).

Another delay metric of interest to upper-layer protocols is *total delay*, which denotes the time from the packet insertion into the interface queue until the notification of successful transmission. It includes queueing delay, backoff timeout, contention period, and retransmissions due to errors or collisions. Ideally, a routing scheme can use this

Interval	Time (μs)	Frame	Length (bytes)
T_{PHY}	192	L_{RTS}	20
SIFS	10	L_{CTS}	14
DIFS	50	L_{ACK}	14

Table 3.2: Constants used to calculate medium time in Eq. 3.7.

value as link cost to enable packets to detour congested areas, which is an interesting area of future work.

3.1.3 Link Bandwidth Estimation

For the estimation of link bandwidth, we use the following heuristic based on the IEEE 802.11 standard. We use the backoff algorithm as well as the medium time τ and estimated PER. The IEEE 802.11 standard uses a backoff counter, which corresponds to the time a client should wait before transmitting. The backoff counter value is chosen uniformly at random between [0, *CW*], where a "contention window" parameter *CW* is initialized to CW_{min} and doubled on each transmission failure until it reaches a maximum value. Let us denote $\beta = CW_{min}/2$. With some simplifying assumptions, the average data delivery latency on the wireless link is given by (see [67] for details):

$$l = \sum_{i \geq 1} \{(2^i - 1)\beta + i\tau\} PER^{i-1}(1 - PER) = \frac{\tau}{1 - PER} + \frac{\beta}{1 - 2PER}. \quad (3.8)$$

Then, the WISE implementation can return the following as link bandwidth:

$$b = \frac{L_{DATA}}{l}. \quad (3.9)$$

3.1.4 Energy Consumption Estimation

Many wireless systems have a MAC-level control mechanism for transmission power adjustment to save battery and reduce interference [17, 68]. The WISE can retrieve the value and calculate the total system power consumption considering additional components of power consumption [69].

3.1.5 Remaining Battery Level

To achieve energy efficiency, some protocols use the remaining battery level in their operation [19, 20, 43, 70]. WISE provides a uniform access mechanism across various underlying platforms and implementations such as APM (Advanced Power Management) and ACPI (Advanced Configuration and Power Interface). In our current WISE implementation, we use ACPI on Linux, where battery information is exported as `/proc` entries typically under `/proc/acpi/battery/BAT`. There can be multiple ways to use this information, and we currently divide the remaining battery value by the maximum capacity and use the normalized value.

3.2 Testbed Experiments for PER Estimation

In this section, we present our experiment results on two wireless testbeds and investigate performance of some of the estimation strategies described in Section 3.1.1.2.

Figure 3.2: The floor map of Emulab wireless testbed with ten nodes.

3.2.1 Experiment Setup

We have performed our experiments on two open access wireless testbeds: Emulab (`http://www.emulab.net`) and ORBIT (`http://www.orbit-lab.org`). Although Emulab is often used to provide emulated network environments for wired networks experiments, the Emulab wireless testbed uses *real air communication* through IEEE 802.11 wireless interfaces between stationary PC nodes scattered around a typical office building. Figure 3.2 shows testbed nodes we use in our experiments. We use nine nodes on the third floor and one node (11) on the fourth floor to experiment with high-loss links. Each PC has two Netgear WAG311 wireless interface cards based on the Atheros 5212 chipset. It uses Redhat 9.0 with 2.4 kernel and the MadWifi open-source device driver[1]. The ORBIT testbed currently consists of 400 wireless nodes, each equipped with two IEEE 802.11 wireless cards laid out in a 20-by-20 grid with approximately one meter spacing between nearby nodes. Due to the relatively small deployment area, observed packet error rates in ORBIT show less diversity, and estimation results on ORBIT show

[1] `http://www.madwifi.org`

trends similar to those obtained from Emulab. Thus, in this chapter, we focus on results from Emulab to illustrate the performance of estimation techniques under both low-error and high-error settings.

In our experiments, a sender alternately broadcasts 16, 32, 64, 128, 256, 512 and 1024-byte UDP packets every 0.05 seconds to minimize the effect of link condition variation over time. In our experiments, we use only one sender at any instant to minimize the interference and collisions. Each sender broadcasts 10000 packets for each size (70000 packets total). All nodes receiving the packets record the packet size and sequence number. In this chapter, we use the fixed transmission rate of 1 Mbps for all messages. Investigating the impact of different data rates will be an interesting area of future research. The transmit power is fixed at 31 mW, which is the default value in the device driver.

We compare the estimation performance of the following strategies:

- BASIC(m): This scheme uses the average error rate of m-byte probe messages for data packets of all sizes.

- INDEP(m): This scheme assumes the independent bit error model and extrapolates the expected packet error rate based on the statistics of m-byte probe messages.

- GE(m, n): This is the estimation scheme based on the GE model, which uses the statistics of m-byte and n-byte probe messages.

- OBSERVED: This is the actual observed packet error rate.

While only one measurement value is required for INDEP, GE uses two parameters, and there can be more possible combinations of the two parameters. For both schemes, proper

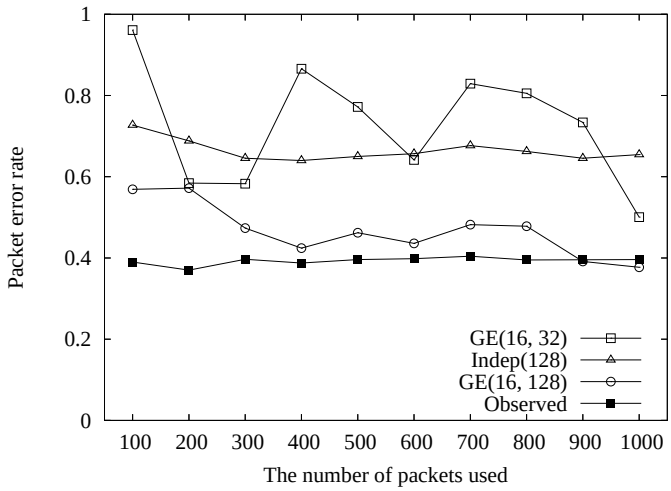


Figure 3.3: PER estimation based on 1000 packets. We use link from node 1 to node 4 in Figure 3.2.

parameter choice can be crucial to correct PER estimation. We consider three different combinations of parameters for GE and two different cases for INDEP and compare the estimation performance.

3.2.2 Experiment Results

3.2.2.1 Estimation Accuracy of Different Schemes

We first consider how well the above estimation strategies perform. In Figure 3.3, we plot the observed error rate for 1024-byte packets and estimated error rates by different schemes[2]. We use a representative experiment sending 1000 packets for each probe type, and each point in the figure is based on cumulative packet error rates after every 100 packets. (We use a smaller number of packets (1000) to illustrate how quickly the estimated

[2]We include additional 84 bytes of lower layer headers in the calculation.

PER converges to the observed PER.) We use the link from node 1 to node 4 in Emulab (Figure 3.2). The estimation by GE(16,128) closely matches the actual average PER. However, GE(16,128) still requires several hundred probe messages before achieving reasonable accuracy, which takes around ten minutes when we send a probe message every second. This amount of time is acceptable for more static wireless mesh networks [64], while more dynamic wireless networks such as ad hoc networks may require faster convergence. One possible approach to reducing the number of required probe messages is to experiment with regression analysis techniques combined with observed error rates for actually transmitted data packets. Addressing this issue will be an interesting area of future research.

In our experiments, GE(16,32) does not perform as well as GE(16,128); there is considerable fluctuation in the estimated value, and the measurement error is relatively large even when we use a larger number of probe messages (See Table 3.3). Specifically, with 10000 probe messages (ten times the size shown in Figure 3.3), the absolute estimation error by GE(16,32) stays around 19% (52.4% vs. 33.4%). One possible explanation to this error is that the estimation by GE(16,32) is less robust because we use extrapolation based on two relatively nearby sample points; a slight measurement error can amplify the estimation error. Also, Kopke et al. [59] find that there is difference in bit error probability depending on the bit position, and bit errors occur more frequently at the beginning of a packet. In that case, estimation using short probe messages alone can potentially lead to higher estimation errors. In the figure, INDEP does not estimate PER correctly, and although not shown in the figure, the estimation error by INDEP(16) is larger than that of INDEP(128). Although we do not show all the results here, we have experimented with

	Emulab Links						
	8→9	1→13	1→7	1→4	1→8	11→16	16→5
OBSERVED	0.018	0.135	0.145	0.334	0.375	0.548	0.738
GE(16,128)	**0.021**	**0.131**	**0.145**	**0.385**	**0.393**	**0.526**	**0.754**
GE(16,64)	0.025	0.222	0.247	0.465	0.332	0.415	0.791
GE(16,32)	0.046	0.154	0.043	0.524	0.243	0.594	0.677
INDEP(128)	0.052	0.222	0.255	0.629	0.645	0.907	0.996
INDEP(16)	0.092	0.332	0.383	0.816	0.831	0.993	1.000
BASIC(128)	0.010	0.047	0.055	0.173	0.180	0.385	0.646

Table 3.3: Comparison of different estimation techniques against actual packet error rates. We use 10000 packets for each of probe and data message types. Values in bold represent the cases with minimum estimation error.

other links and performed multiple experiments for each link, and the results are similar. We next present some results obtained from other links.

3.2.2.2 Experiments with Various Links

In previous results, we considered results only from one particular link. We now present results from various wireless links with diverse link quality. In Table 3.3, we report estimated PERs by different schemes as well as observed error rates for 1024-byte packets based on 10000 packets for each message type. We observe that the GE(16,128) estimation is the most accurate in all cases (highlighted in bold), and the estimation error is small regardless of link quality. GE(16,64) often performs better than GE(16,32), but

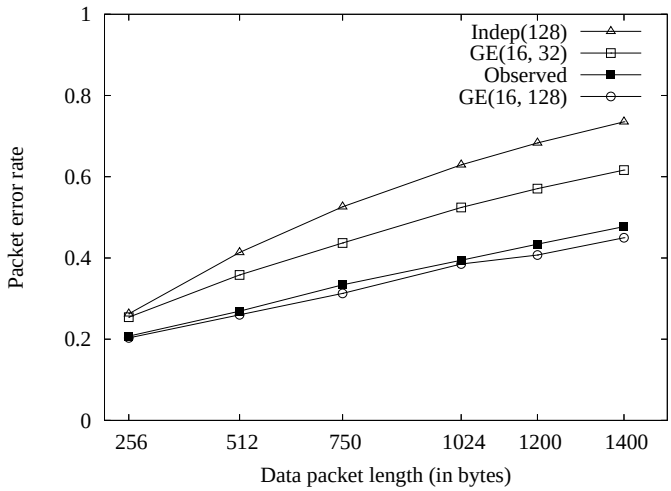


Figure 3.4: Estimation of PER when we vary the size of data packets. We use the link from node 1 to node 4 in Figure 3.2

both of them result in larger estimation errors than GE(16,128). As in Figure 3.3, INDEP always overestimates packet error rates, and the degree of overestimation is higher with INDEP(16) than INDEP(128). Although the independent bit error model has served as a reasonable model in [60], it does not seem to reflect the channel characteristics correctly in our indoor experiments. In BASIC(128), we use the error rate of 128-byte probe messages as the estimation for 1024-byte packets, which results in significant underestimation of PERs.

3.2.2.3 Varying Data Packet Sizes

In the previous experiments, we fixed the data packet length to 1024 bytes. In this set of experiments, we vary the data packet size and compare the estimated and observed error rates. In this experiment, we use additional packet sizes (750, 1200, and 1400 bytes). In Figure 3.4, we plot the estimated and actual packet error rates with varying packet sizes.

We use the statistics of 10000 message for each probe type. Not surprisingly, average PER increases as data packets become larger. We observe that GE(16,128) again performs best in estimating error rates for other packet sizes. Other schemes show similar trends to Figure 3.3. This result illustrates that our proposed estimation technique estimates error rates for various packet sizes.

3.3 Summary

In this chapter, we have defined a number of WISE interfaces and presented techniques needed for implementation. We also presented measurement experiments to compare possible strategies for packet error rate estimation. In the following chapters, we describe how we can use the WISE interfaces in actual wireless systems.

Chapter 4

Protocol Extension for Multihop Wireless Local Area Networks

IEEE 802.11 based wireless LANs (WLANs) are one of the primary enablers of untethered access to the Internet. A typical WLAN consists of two different entities—Access Points (APs) and stations (STAs), which we refer to as clients in this chapter. A client associates itself with an AP within its *direct* communication range. The set of all such clients for a specific AP is known as the Basic Service Set (BSS) for that AP. A single WLAN can consist of a number of such BSSs, one corresponding to each AP. The APs are connected via a backbone distribution system (DS), which provides a conduit to the external network. All the BSSs together with the DS are known as the Extended Service Set (ESS). The entire ESS is identified by a single ESSID.

In this chapter we (1) define a *multihop* 802.11-based WLAN architecture (Sections 4.1 and 4.3), (2) demonstrate how such a system can provide significant performance benefits over existing single-hop counterparts (Sections 4.2 and 4.4), and (3) describe a deployment path that will enable it to seamlessly interoperate with existing WLAN infrastructures (Section 4.3). We first present the multihop WLAN architecture and discuss the advantages and potential issues.

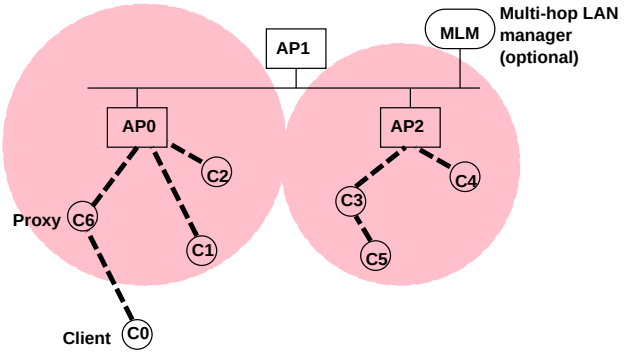


Figure 4.1: The multihop 802.11 architecture. The circles represent the communication range for the specific APs.

4.1 Multihop WLAN Architecture

In Figure 4.1 we illustrate our proposed multihop 802.11 architecture. In this architecture, each client can directly associate itself with an AP in the WLAN. Additionally, the client can also have a multihop path, via other clients acting as intermediaries or proxies, to indirectly associate with the AP. In a typical scenario we expect the proxies to be "resource-rich" clients that take data forwarding responsibilities on behalf of "resource-depleted" clients.

4.1.1 Advantages

There are a number of benefits of a multihop wireless LAN architecture. We discuss them in turn.

4.1.1.1 Enhanced Performance

Some clients in a WLAN are resource-depleted. Consider the case when a specific client (say client C_5 in Figure 4.1) is low on battery power. The energy required for it to communicate directly with AP_2 is prohibitively expensive. However, the availability of a nearby client that can serve as a proxy (e.g., client C_3) significantly reduces the energy requirements for communication. Therefore the multihop path leads to increased lifetime for C_5.

Similarly, consider another scenario where the direct channel between C_5 and AP_2 is very noisy. Therefore, data transmitted on this channel will encounter significant errors and losses. Typical implementations of the IEEE 802.11 protocol reacts to such losses by reducing the data rate. Alternatively we can use the 802.11 protocol and maintain the higher data rate by using a higher transmit power. Increasing the transmit power increases the signal to noise ratio, which in turn reduces the bit error rate on the channel and allows the 802.11 protocol to operate at the higher data rate. This high power solution leads to increased interference in the WLAN. For example, transmissions from C_5 may now interfere with data transmissions between AP_0 and its clients, thus reducing the data throughput of the WLAN. In a multihop system, C_5 can use a "better-located" client (e.g., C_3) to communicate with the AP. We performed detailed measurements in existing WLANs to study the benefits of a multihop approach to clients. Our results in Section 4.2 indicate that in many such cases clients can leverage a multihop path to significantly improve their data throughput. Additionally, the performance improvement of these "resource-depleted" clients also positively impacts the performance of clients in the

same WLAN that are not even aware of multihop extensions.

4.1.1.2 Extended Wireless Coverage

In the usual single-hop WLANs, a client must be located within the coverage area of some AP to receive wireless services. A multihop WLAN leverages participating proxies to extend the coverage area, e.g., client C_0 in Figure 4.1. Such a solution is particularly useful in *handling flash crowds.* If a transient user population moves into an area with no wireless coverage, a multihop WLAN can be used to provide immediate wireless services. Obviously, the long-term solution to provide wireless connectivity in a popular user location is to add more APs in that area. However, the multihop solution is more appropriate to handle transience. This is because it requires no setup, administrative overhead, or additional hardware.

4.1.1.3 Enabling Automated Re-organization of AP Distribution

The goal of a WLAN designer is to ensure that each location in the area is visible to at least one of the APs of the WLAN. WLAN administrators currently use various techniques to monitor the expected performance of WLANs. One of the more popular methods is to perform signal strength measurements at various locations of the coverage area from the nearby APs. Such an approach is tedious and cannot be performed very frequently. As a result, WLAN administrators often do not have accurate radio maps that reflect the existing conditions in the wireless environment [71].

The multihop WLAN presents a new opportunity for the online performance moni-

toring. For example, when proxies in a specific location get heavily used (e.g., due to poor channel conditions in the direct path to the APs), the system can trigger alerts to the LAN administrators to appropriately add or re-distribute the APs in that location. In the proposed multihop 802.11 architecture, the proxies provide such information to the Multihop LAN Manager (MLM) and the latter is responsible for providing such notifications.

In some of the above examples such as extended wireless coverage, the long term solution is to add more APs to the WLAN. In such cases the multihop architecture can be leveraged to (1) provide a short term solution, (2) handle transient situations, e.g., flash crowds, (3) provide performance benefits in cases where re-organization of the WLAN is too expensive, and (4) allow administrators to discover performance problems in the WLAN which can trigger the long-term re-deployment based solutions. In other cases, the multihop architecture provides the only logical solution to improve the performance of resource-depleted devices (e.g., a device with low residual battery power).

4.1.2 Potential Pitfalls

While there are a number of benefits of the multihop architecture, it is important to evaluate some of the potential pitfalls that may arise in this environment.

4.1.2.1 Increased Channel Contention

When a packet follows a multihop path to an AP, it uses the wireless channel two or more times. This may increase the contention of the channel and potentially allow reduced data throughput for the source as well as other clients in the vicinity. To quantify the

effect of multihop paths on data throughput, we have performed detailed measurements as well as simulations. The results show that in many cases the data throughput increase due to better (multihop) path choices more than compensates for the loss due to channel contention. Our proposed mechanisms take channel contention effects into account when making such multi-path choices.

4.1.2.2 Resource Consumption at Proxies

Packets following multihop paths consume resources at the proxies, e.g., battery power, bandwidth. Clearly, there is no incentive for wireless clients to operate in such an altruistic mode. Each client in the WLAN can choose independent policies on when it is willing to serve as a proxy. For example, some users may volunteer their laptop clients when they are powered from an electric outlet, and when the laptops are idle, i.e., not actively generating network traffic. Additionally, it is possible to define incentive based packet forwarding rules in such multihop environments as shown in [30].

4.1.2.3 Security Threats

Allowing an intermediary to forward data packets on behalf of a client may potentially open the WLAN to new security threats. For example, a malicious proxy can (1) mount a denial of service attack by dropping all frames forwarded to it by the clients, or (2) tamper sensitive data sent through it. However, we believe that multihop extensions do not add any *new* threat that is not already present in WLAN environments. For example, in current WLANs it is relatively easy to mount a denial of service attack by using sim-

ple channel jamming techniques. Similarly, all sensitive data should be encrypted using end-to-end mechanisms even in existing WLANs, since the entire network between the endpoints should be considered to be untrusted for such applications.

4.1.3 Incremental Deployment

IEEE 802.11 based WLANs are currently widely deployed. Therefore a new multihop architecture that requires a change to existing entities (e.g., clients and APs) is not always feasible. Therefore, we explore the potential paths of deployment of multihop WLANs that require various degrees of change to existing entities. The proxies are new entities in the system, and any client that acts as a proxy needs to implement the multihop extensions. However, to maintain backward compatibility with existing systems we consider cases where the other entities (i.e., regular clients and APs) are not aware of multihop extensions to the WLAN. We consider the four different cases—(1) unaware-AP, unaware-client, (2) unaware-AP, aware-client, (3) aware-AP, unaware-client, and (4) aware-AP, aware-client—and define techniques for implementing a multihop 802.11 WLAN for each of these cases. While the basic principles of the protocols in these cases are similar, the mechanisms required to achieve the desired effect vary from case to case. The protocols and mechanisms for the aware-client cases are more interesting, and we primarily focus on these two cases in this chapter.

4.1.4 Comparison to Routing-based Solutions

One way to construct this multihop access infrastructure is to use a routing layer based solution. In fact, a number of on-demand routing protocols have been defined to provide network level connectivity between arbitrary pairs of wireless nodes in an ad-hoc wireless network (e.g., DSR [37], AODV [72], TORA [73], ZRP [74]). While these protocols can be used to construct appropriate multihop paths from the wireless clients to the Access Points (APs) of an 802.11 WLAN, we believe that the benefits of a multihop wireless access infrastructure can be better realized when implemented at the wireless medium access layer due to the following reason.

In most popular wireless environments (e.g., office buildings, homes, and WiFi hotspots), wireless clients typically need mechanisms to access the wired infrastructure. Consequently, the goal of the access infrastructure is to construct appropriate (single-hop or multihop) paths to the nearest AP of a WLAN. A full routing protocol that allows flexible routing between arbitrary pairs of nodes is not necessary for such purposes. Note that some of the proposed route construction mechanisms (e.g., network-wide flooding to locate destination nodes) are based on arbitrary separation between the source and the destination. In contrast, the clients in a WLAN are in a much more limited region, where typically the clients are in direct communication range of the APs. In fact, as our experimental results show, most data performance benefits are gained by using short paths (using one or two proxies) between the clients and the APs. We next present our measurement and simulation results to validate the proposed architecture.

4.2 Measurement-based Evaluation

In the previous section, we identified some of the potential pitfalls of a multihop WLAN architecture. In particular, we identified the issue of increased channel contention as a potential disadvantage of multihop WLANs. In this section we primarily examine the channel contention effects and their impact on data throughput. Our results indicate that a carefully designed multihop WLAN protocol can lead to significant data performance benefits in all cases.

4.2.1 Experimental Setup

We performed our experiments on the 4th floor of A.V. Williams building (which hosts the Computer Science Department at the University of Maryland). The map of the floor is shown in Figure 4.2. In the experiments described in this section, we performed the experiments with respect to a representative AP running the 802.11b protocol and located at the position marked in the figure. We measured the data throughput achieved by clients using both direct and multihop configurations. In both these configurations, the client performed a reliable data transfer (using TCP) of 51.12 MB of data to a sink, located in the same wired subnet as the AP. (This translates to 100,000 IP packets of size 536 bytes each, generated at the source.) In each experiment we measured the data transfer latency as observed at the application layer.

For the multihop measurements, we did not implement the full version of our proposed protocol (to be described in Section 4.3). Instead we emulated the multihop link layer mechanisms using statically assigned IP addresses and routes, as shown in Fig-

ure 4.3. In this setup, the proxy device used two separate wireless cards—one to associate with the AP and operate in the managed mode, and the other to interact with the source client and operate in the ad hoc mode. Due to physical constraints of the PCMCIA slots of laptops, we found it convenient to use two laptops, connected by 100 Mbps Ethernet, to operate as a single proxy as shown in the figure. Note that such an arrangement is actually disadvantageous to the multihop experiment. Unlike multihop link layer mechanisms, the data packets encounter additional delay due to network layer processing. This setup also leads to an additional latency due to data transfer between laptops A and B via Ethernet. In these experiments we used IBM Thinkpad laptops running Linux with kernel version 2.4.19, equipped with Orinoco Silver PC cards.

The IEEE 802.11 standard allows multiple channels to be used simultaneously. In the multihop experiments there are two wireless links, one from the source to the proxy, and the other from the proxy to the AP. We experimented with using the same channel as well as two independent channels for these two links and compared the performance of both these scenarios with the single-hop case. In an actual deployment, specific network conditions and other administrative factors will determine whether multiple channels can be used.

4.2.2 Results

We performed this measurement study throughout the month of June 2003, in which we observed the data throughput of more than 30 sample positions. Not surprisingly, we found that the wireless data throughput fluctuated between different measurements.

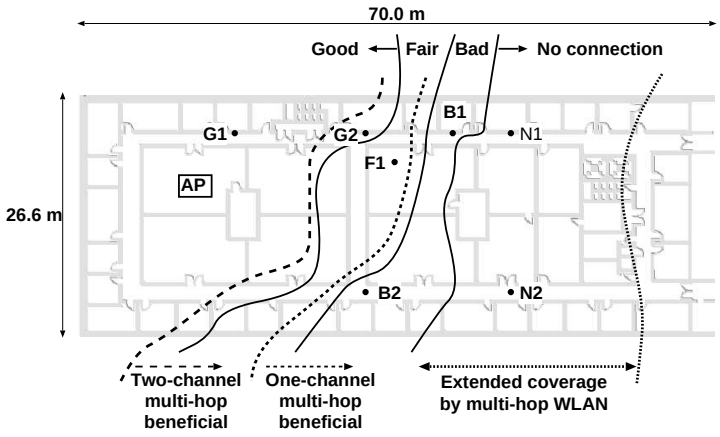


Figure 4.2: Potential data throughput improvement by using multihop extensions to the currently deployed WLAN on the 4th floor of the A.V. Williams building. The "Good," "Fair," "Bad," and "No Connection" marks the performance of the single-hop WLAN. The multihop benefits shown in this figure are obtained using two hop paths.

However, it was easy to identify a consistent ordering among the data throughput achieved at different locations.

In Figure 4.2 we present an *approximate* wireless coverage and direct-hop data throughput from different locations to a representative AP (marked in the figure). In the area marked "Good" users can get data throughput of more than 4 Mbps. (Although the maximum data rate in the 802.11b WLAN is 11 Mbps, it is not possible to achieve an 11 Mbps data rate due to control overhead such as ACK frames, backoff mechanism, etc.) In the area marked "Fair" the throughput varies between 1 and 4 Mbps. In the area marked "Bad" the throughput is less than 1 Mbps, and finally the users lose connectivity with the AP in the area so marked.

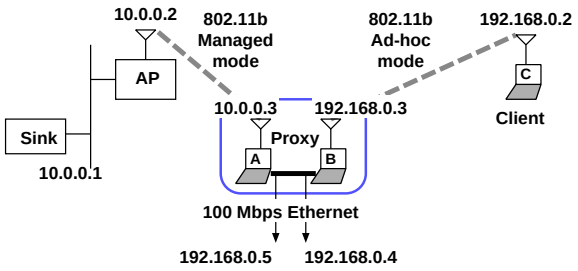


Figure 4.3: The experimental setup to measure performance of a multihop WLAN.

In Figure 4.2, the two dotted lines on the left identify the regions where the emulated multihop wireless paths lead to improved performance over the existing infrastructure (e.g., > 2 times higher bandwidth in the "bad" region). The two-channel multihop paths are useful even when users are located within the good wireless coverage region (e.g., location G2). It provides considerable performance improvement for users in "fair" and "bad" areas (e.g., F1, B1) as well as in "no connection" area. The single-channel scenario is expected to have worse performance than the two-channel case due to greater contention effects in the single channel. The results indicate that in spite of these effects, single-channel multihop wireless paths provide significantly improved performance in the areas marked "Bad" and "No connection" (e.g., B1, N1). Finally we can observe that the multihop WLAN considerably extends coverage, as shown in the figure.

In Table 4.1 we tabulate some of the representative measurements at selected locations on the floor.

Using three hops We also have conducted some experiments with three-hop paths. We observe that the bandwidth achieved in these experiments are similar to the two-hop

Position	Direct	Multihop	
		One-channel	Two-channel
G1	4.94	2.42	4.56
G2	4.12	2.58	4.50
F1	2.46	2.50	4.60
B1	0.84	2.26	4.30
B2	0.83	2.37	4.24
N1	-	1.83	3.77
N2	-	2.50	2.96

Table 4.1: Actual throughput values (Mbps) measured at representative points

measurements. For example, at location N1, the end-to-end throughput is 1.70 Mbps for a single channel experiment and 3.79 Mbps when three channels were used, which are similar to the two-hop results shown in Table 4.1. As discussed in previous study [75], when we use a single channel for a three-hop path, all links in the path interfere with one another, which cancels the benefit of using high-quality links. Even when we use different channels for each of the three links, a packet needs to access the shared medium multiple times (e.g., binary backoff and retransmissions due to collision [76]). Our results show that due to such overhead, in typical WLANs scenarios with reasonable AP coverage, the additional benefits of using three or more hops are marginal.

Overall, we believe that these experiments serve as evidence that multihop WLANs can be useful to clients in many cases. In the next section, we describe the network architecture and its deployment path in more detail.

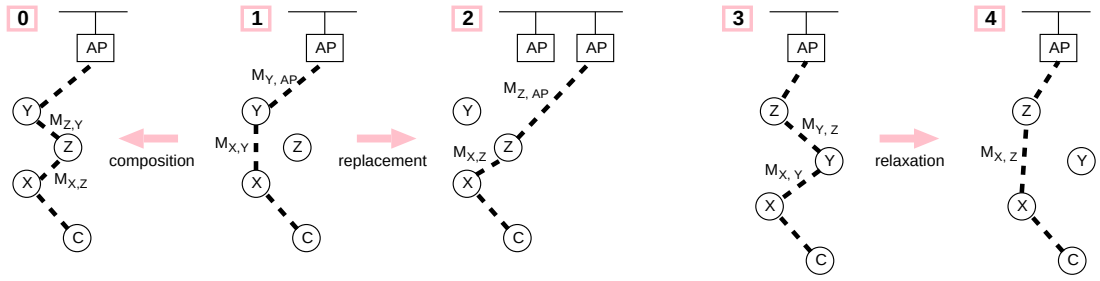


Figure 4.4: The Composition, Replacement, and Relaxation constructs. C is a client. X, Y, and Z are proxies.

4.3 Multihop WLAN Architecture and Deployment

In this section, we describe the proposed multihop WLAN architecture and protocol mechanisms. We first define three important constructs necessary to implement a multihop WLAN. We call them *composition*, *relaxation*, and *replacement* of proxies (Figure 4.4). In the examples in the figure we use three or more hops for the multihop paths. The protocol mechanisms generalize to an arbitrary number of hops. However, our measurements (Section 4.2) indicate that in most typical scenarios, two hop paths are sufficient for performance benefits, and benefits of additional hops are marginal.

Let us consider any general metric, $\mathcal{M}$ (e.g., bandwidth, loss rate, latency, energy consumption). *Composition* defines the protocol mechanisms to add a proxy on the path from the client to the AP (Panel 1 $\rightarrow$ Panel 0). Such an addition is performed if and only if the path improves with respect to the given metric, $\mathcal{M}$, i.e., in the figure:

$$\mathcal{M}_{X,Z} \bigoplus \mathcal{M}_{Z,Y} \text{ better than } \mathcal{M}_{X,Y}$$

(We use the $\bigoplus$ operator to denote composition.) The definition of "better than" depends

on the specific metric.

Replacement describes mechanisms where one proxy replaces another (Panel 1 $\rightarrow$ Panel 2) and leads to an improvement of the path quality with respect to $\mathcal{M}$. In the figure this implies that:

$$\mathcal{M}_{X,Z} \bigoplus \mathcal{M}_{Z,AP} \text{ better than } \mathcal{M}_{X,Y} \bigoplus \mathcal{M}_{Y,AP}$$

Note that the proxy Z may be associated with a different AP within the same WLAN.

Finally, *relaxation* defines protocol mechanisms to remove a proxy on the path between the client and the AP (Panel 3 $\rightarrow$ Panel 4), to improve the path quality. In the figure this requires:

$$\mathcal{M}_{X,Z} \text{ better than } \mathcal{M}_{X,Y} \bigoplus \mathcal{M}_{Y,Z}$$

In this chapter, we describe the implementation of the constructs with respect to an example metric—bandwidth available on the path from the client to the AP. We use the following notation. For a link $X \rightarrow Y$, $b_{X,Y}$ denotes the bandwidth on that link. For a client C, we represent the end-to-end bandwidth on its single or multihop path to the AP, by b_C. Thus $b_C = \min\{b_{X,Y}\}$ over all $X \rightarrow Y$ hops on this path. b_C is our objective of maximization.

Note that there are two key components that determine the bandwidth of a wireless path: (1) noise on the wireless channel, and (2) contention with other clients. As the noise on the channel increases, the 802.11b implementations on the wireless cards reduce the data rate, thus increasing the path latency and reducing the path bandwidth. Similarly as collisions occur on the wireless channel, the 802.11b clients perform contention resolution which leads to reduction in bandwidth and increase in latency.

In order to compute multihop paths with good bandwidth or latency performance, we need to estimate these metrics for individual wireless hops. One possibility is to use periodic message exchange as in [2]. However, this method may introduce considerable control overhead even when we do not need to use multihop paths. In this chapter, we focus on a WISE interface based on passive observations (See Chapter 3). There are two advantages of this proposed heuristic: (1) it requires no active measurement traffic and hence does not increase the contention of the data channel, and (2) an endpoint of a wireless link or any external entity with the capability to snoop packets can use this technique to estimate the the metrics for that link. Although it is possible that nodes occasionally do not detect nearby transmissions, such link quality estimation techniques based on passive observations are widely used in practical wireless applications [77].

As explained in Section 4.1.3, we have considered four different scenarios for deployment of a multihop WLAN. We now describe the multihop architecture that implements the composition, relaxation, and replacement constructs for improved performance in these scenarios.

4.3.1 Aware client

We first describe the protocol mechanisms for clients that are aware of multihop extensions. We independently consider the path from the client to the AP (forward path) and the path from the AP to the client (return path).

4.3.1.1 Forward Path

Let us assume that a client C currently uses some forward path (either direct or multihop) to an AP, where the client is the source of traffic. Consider a specific hop on this path, $X \rightarrow Y$ as shown in Panel 1, Figure 4.4. X computes the bandwidth available on that hop, $b_{X,Y}$, using the WISE interface call. Also, when a client C uses a multihop path, C (or proxy P on the path) periodically advertises the value of b_C (or b_P) in its single-hop neighborhood. This periodic advertisement can be done either using local broadcasts of an additional packet type at a low frequency, or piggy-backing onto data packets. In Panel 1, Figure 4.4, Y advertises b_Y, and using this information, X can calculate b_X as $\min\{b_{X,Y}, b_Y\}$. Then, X also advertises b_X, and any proxy in the vicinity snoops and uses this information to determine if the multihop path through itself is better than the current one.

For example, consider another proxy Z that is within direct communication range of X. Z receives the bandwidth advertisement, b_X, on this path. X has a better path to an AP through Z rather than its existing path through Y, if

$$\{\min(b_{X,Z}, b_Z) - b_X\} > b_{thresh} \tag{4.1}$$

where b_{thresh} is the bandwidth advantage threshold. Note that Z is also a regular client in the system and therefore computes and maintains the available bandwidth, b_Z, to its AP. Z estimates the value of $b_{X,Z}$ using the passive estimation technique implemented by the WISE interface call as described in Chapter 3.

If using Inequality 4.1, Z detects that the path $C \rightarrow \ldots X \rightarrow Z \rightarrow \ldots \rightarrow AP$ has higher bandwidth, it sends a *ForwardProxyBid* message to X. This message includes the

values of $b_{X,Z}$ and b_Z. If X receives multiple such *ForwardProxyBid* messages, it chooses a proxy that leads to the best bandwidth improvement. X sends a *ForwardProxyAccept* message to the chosen proxy and starts forwarding data packets to Z.

If the path from Z to the AP has Y as its first hop, then this operation would be a *Composition* (shown in Panel 1 $\rightarrow$ Panel 0, Figure 4.4). If the first hop from Z is some node other than Y, this would be a *Replacement* operation (Panel 1 $\rightarrow$ Panel 2, Figure 4.4). Finally, if in the original multihop path from the client C to the AP, Z was the next hop to Y, then the operation describe above is equivalent to a *Relaxation* (Panel 3 $\rightarrow$ Panel 4, Figure 4.4).

The proxy state is soft. Therefore, in absence of data packets, X is required to periodically refresh the state at Y by sending gratuitous *ForwardProxyAccept* messages. Y can revoke proxy services to its previous hop X by using a *ForwardProxyRevoke* message. This can be invoked due to many reasons. For example, the laptop serving as the proxy is unplugged from the electric outlet and, hence, is no longer willing to serve as a proxy. Alternatively it can also be that the proxy is dissociated from its AP. As a final fallback mechanism, X can also detect the failure of Y, when it fails to acknowledge a threshold number of consecutive RTS packets forwarded to it (in the RTS/CTS access method).

Clients outside direct range of AP In case no AP is directly reachable from C, it attempts to set up an initial multihop path to an AP. For this, C monitors messages in its single-hop wireless neighborhood. If C finds a proxy in its vicinity, e.g., by detecting any proxy-specific control message, then it sends an unsolicited *ForwardProxyAccept*

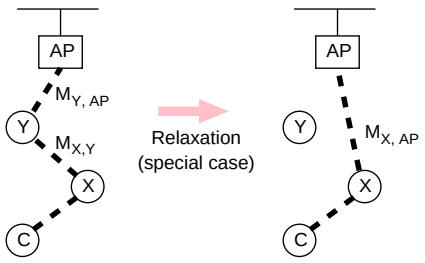


Figure 4.5: Relaxation of the last proxy on a multihop path.

message to it to set up a multihop path. (Subsequently C might discover better multihop paths to some AP using the mechanisms described above.) Otherwise, C probes each neighboring client by sending an unsolicited *ForwardProxyAccept* until it finds a proxy-enabled client. When a proxy-enabled client receives such an unsolicited message, it sends *ForwardProxyBid* and a multihop path is established.

Special case for unaware-AP All the above operations work independent of whether the AP is aware or unaware of multihop extensions to the MAC protocol, except one special case. This special case arises for the unaware-AP case, when the original multihop client path was $C \rightarrow \ldots \rightarrow X \rightarrow Y \rightarrow AP$, and a relaxation operation is required to eliminate the last proxy, Y, from the path (Figure 4.5).

Note that in the aware-AP case, we implement the same *ForwardProxyBid* mechanism in the AP that leads to this relaxation operation. We call such a relaxation step *Relaxation assisted-by Access Point (RAP)*. However, if the AP is unaware, such an operation is not feasible. An unaware-AP will not attempt to evaluate the bandwidth of the $X \rightarrow AP$ link, nor send a *ForwardProxyBid* message to eliminate Y from the path.

Therefore to enable the elimination of the last proxy from a multihop path, if and when necessary, we need to define additional mechanisms for the unaware-AP case.

In the unaware-AP case, X actively probes the quality of the direct path between itself and the AP. In this active probe technique, X periodically sends a *NULL* frame to the AP. The *NULL* frame is a special frame which is automatically dropped by the AP and therefore does not add any extra load on the Distribution System. However, like any data packet, the AP will perform the four-way handshake to receive this packet (i.e., RTS-CTS-NULL-ACK). Using this low frequency stream of *NULL* frames, X estimates two parameters: (1) the packet error rate, p, on this link, and (2) the latency of the four-way handshake for a successful data transfer across the link, τ. Estimation of these two parameters is sufficient for X to infer $b_{X,AP}$ using the WISE interface call. If $b_{X,AP} - b_X > b_{thresh}$ then X directly eliminates Y from the multihop path, by sending a *ForwardProxyRevoke* message.

MAC Address Translation Consider a forward multihop path from the client C to the AP, $C \rightarrow P \rightarrow AP$, where P is a proxy. When P forwards data frames to the AP, on behalf of the client C, it uses its own MAC address as the source address for those data frames.[1] (Alternatively P can use a specially chosen independent MAC address when forwarding packets for each specific client.) The proxy therefore performs *MAC-level Address Translation (MAT)* for data frames transmitted by C. This is true for a multihop return path as well, as described next.

[1] If P spoofs the MAC address of C, it can lead to ambiguities and incorrect operation at the MAC layer.

4.3.1.2 Return Path

Typically, the return path from the AP to the client should use the direct single hop path if available. This is because the AP is usually a resource-rich device and can transmit with adequate power to tide over moderate noise levels in the channel. However, there may still be cases where the bandwidth of direct return path from AP is poor, or no direct path is available. In such cases we use the reverse of the forward multihop path, as described below. The client C makes a decision on which path to use. If it chooses to use the reversed forward multihop path, it sends a *ReverseProxyRequest* along the multihop path. The return multihop path is activated by the last proxy on the path (i.e., the proxy Z in Panel 4, Figure 4.4) using ARP mechanisms. However, C continues to stay associated with its AP and continually estimates the bandwidth on the direct hop from the AP to itself. On detecting an improvement of this single-hop path, it reverts back to this path. C sends a *ReverseProxyRevoke* message to its first-hop proxy to effect this change. Alternatively C stops refreshing the proxy state on the reverse path, and the states at the proxies time out.

It is also possible to consider an independent return path. For example, in Panel 4, Figure 4.4, while the forward path is $C \rightarrow X \rightarrow Z \rightarrow AP$, it is possible that $AP \rightarrow Y \rightarrow C$ is the best return path. However, although C can infer the bandwidth of link $Y \rightarrow C$, it is not always able to know the link bandwidth from AP to Y (e.g., C is beyond the AP coverage area). As a result, to find the best independent return path, all candidate proxies such as Y need to advertise their return path bandwidth (e.g., from AP to Y in the figure). Since such advertisement can be expensive, we do not further explore this alternative in

this chapter.

MAC Address Resolution Let us first consider the direct single-hop return path. In this case, when the AP sends an ARP request for C's IP address, C sends the ARP response with its own MAC address. Hence the AP transmits all data packets addressed to C using C's MAC address as the destination.

Next consider the case when the return traffic uses the multihop path. In this case, when the AP sends an ARP request for C's IP address, the last proxy (Y in Panel 0, Figure 4.4) on the multihop path sends proxied ARP responses with its own MAC address (or a specially chosen independent MAC address). Subsequently all traffic destined for C will be forwarded by the AP to Y's MAC address. Whenever C switches between the two paths, an explicit ARP response is sent to update the cache entry at the AP appropriately.

In the aware-client case, all interaction between the clients and proxies takes place using the *ad-hoc* mode of the Distributed Coordination Function (DCF) of 802.11b operation.

4.3.2 Unaware client

We now describe the implementation path for a multihop WLAN for the unaware client scenarios. In these scenarios since the clients are unaware of multihop extensions, they will not associate with any entity other than APs with the designated ESSID. Therefore the key problem in this scenario is to compose a proxy on the path from the client to the AP.

In these scenarios a multihop path can only be constructed if the proxies operate

as APs in the WLANs. All these active proxies (acting as APs) need to interact with the actual APs in the WLAN to form a Wireless Distribution System (WDS). Some implementations of WDS are already commercially available today, e.g., Orinoco AP-2000 from Agere Systems[2] and WX-1520 from SparkLAN[3].

If all possible proxies act as APs, then the number of APs in the system can become very large. Therefore, unlike existing implementations of WDS, the proxies in our proposed system emulates AP functionality on-demand, i.e., only when it is needed by low-performance clients.

Consider a client C that is directly associated with an actual AP (which we call wired AP in this description). A proxy X emulates AP functionality when it detects that the path $C \rightarrow X \rightarrow AP$ has a higher bandwidth than the direct path $C \rightarrow AP$. As in the aware client scenarios, X maintains the estimate of bandwidth from itself to its wired AP (i.e., b_X) and computes the direct bandwidth from C to itself (i.e., $b_{C,X}$). X also estimates the direct bandwidth from C to AP (i.e., b_C) by snooping the wireless traffic sent by C to the AP. Note that only proxy X estimates link bandwidths, but unaware client C does not.

Let us first consider the *Composition* operation in the unaware-AP, unaware-client case. Low link quality between the client and AP is typically due to two reasons: (1) poor channel conditions, i.e., high noise in the wireless medium on the path from C to AP, or (2) high network traffic which leads to significant channel contention. 802.11b clients respond to both these scenarios by trying to identify a "better" AP and associating it. If

[2]See http://www.agere.com

[3]See http://www.sparklan.com

C attempts such a re-association and sends a probe message, the proxy P (operating as an AP) will receive the probe message and respond to it. Of course, in this unaware client scenario, it is possible that the client selects a sub-optimal proxy since it may consider the quality of the immediate link to the proxy, not the quality of the composite path. Hence, we cannot guarantee bandwidth-optimal paths in the unaware client case.

In the aware-AP, unaware-client case, the aware APs can actively participate in a *Composition* operation as follows. A proxy X, on detecting a better path for C, can optionally send a *ClientDissociateRequest* to the AP with which C is currently associated. The AP on receiving this message will explicitly dissociate C. This will force C to locate an alternate AP, and in the process will find proxy X. We call this process *Composition assisted-by Access Point (CAP)*. With CAP the *Composition* operation can be initiated before the link between the client and AP becomes very poor.

In the unaware client scenarios, the *Relaxation* step is also hard to guarantee. For the aware-AP, unaware-client case, we rely on the AP to initiate the relaxation step (RAP). When the AP detects that the direct path has better bandwidth than the composed multi-hop path, it sends a *ClientDissociateRequest* to the proxy, X, which has been emulating AP functionality. The proxy X subsequently dissociates the client, C, and C eventually re-associates directly with the wired AP. In the unaware-AP, unaware-client case, relaxation is possible only if the channel conditions on the path between the client and the proxy becomes bad, and the client automatically attempts to locate a better AP for itself. Therefore, to force the client to locate better alternate and possibly direct paths, the proxy should periodically dissociate the client, forcing the latter to locate a better AP. This is the only possible mechanism that can enable path relaxation when both a client and an

	Access Point	
Client	Unaware	Aware
Unaware	WDS	WDS + RAP/ CAP
Aware	MAT	MAT + RAP

Table 4.2: Mechanisms required to deploy multihop WLANs for the four different scenarios.

AP are unaware.

When a proxy is eliminated from a multihop path through the relaxation process, and it is not serving as a proxy for any other client, it stops operating as a wireless AP and reverts back to the regular client mode.

We summarize the mechanisms to implement all the four scenarios in Table 4.2.

4.3.3 Discussion

In this section, we discuss other issues relevant to our proposed multihop WLAN system.

Association Overhead As the results in Section 4.2 demonstrated, in many cases there are significant benefits of using multihop paths. However, transitions between the direct to the multihop path typically will incur some overhead at the clients. We expect this overhead to be equivalent to that experienced by clients when they re-associate from one AP to another in existing WLAN environments. We will need an actual measurement study from a prototype system to be able to quantify this aspect further.

Path Oscillation Since a client and its proxies attempt to find the best multihop path, it is possible that path changes occur too frequently. There are several ways to overcome this problem. First, we can use a reasonably large value for b_{thresh} in Inequality 4.1. Second, we can use a running average of bandwidth metric to mask temporary fluctuation of measurements. Finally, we can set an upper bound on the path changing rate (e.g., at most one proxy change every N seconds).

Loop-freedom The use of a reasonably large value for b_{thresh} ensures that any multihop path is loop-free in a stable environment. However, infrequent and transient loops may potentially occur in case of inconsistent metric measurements among mobile nodes. However, such loops will quickly disappear as the measurements converge to a consistent state. In addition, if we limit the number of proxies on the path to two (which is sufficient in most cases), we can trivially guarantee loop-freedom on all multihop paths.

4.4 Simulation Studies

To evaluate the performance of our proposed protocol in the aware-clients case, we have performed detailed simulations using the *ns-2* network simulator.[4] In addition to static scenarios, we have also performed detailed experiments that involve mobile clients. We use a circular topology with radius of 250 meters, which is the maximum transmission range of each node (Figure 4.6). The AP is located at the center of the circle, which does not move. Other nodes may or may not move depending on particular simulation settings. In this section we focus on the scenario of the aware client and aware AP. We primarily

[4]Available at: http://www.isi.edu/nsnam/ns

investigate the impact of our proposed techniques for the bandwidth and latency metrics, but we also report results on extended coverage due to the multihop extensions.

4.4.1 Simulated Environment

In our experiments, we use *ftp* traffic to model reliable TCP-based data transfer between sources and destinations to measure the end-to-end throughput. These data sources are typically mobile clients that sent traffic through APs to a wired sink node. Since our study focuses on the data performance of the WLAN, we assume that the link between the AP and the wired sink is not a bandwidth bottleneck. Typical simulation durations are between 300 and 600 seconds. In this section, we primarily present results for multihop extensions where all communication used a single channel. We present a brief summary of results for the two-channel experiments at the end of this section.

We model the environment as a noisy channel. We assume that the underlying physical layer uses the Binary Phase Shift Keying (BPSK) modulation scheme in which we can calculate the bit error rate using Eq. 3.3 in Chapter 3. We also assume that signal strength is reduced proportionally to the square of distance. Therefore the quality of the channel depends on the noise in the environment and the distance between the endpoints.

Most wireless cards incorporate a mechanism called Automatic Rate Fallback (ARF) to handle noisy channel conditions [78]. In this mechanism, each node initially uses a data transmission rate of 11 Mbps. On detecting repeated data transmission failures, it reduces its transmission data rate to 5.5 Mbps, 2 Mbps, and 1 Mbps successively. Later, if the node receives ACKs for several successive data packets, it increases its transmission bandwidth

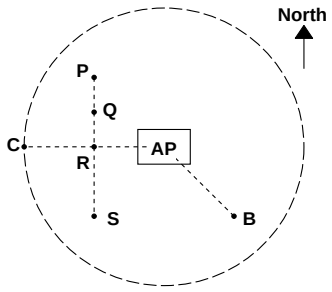


Figure 4.6: Location of clients and AP in the some of the experiments. The radius of the circle is 250 meters.

until the bandwidth reaches 11 Mbps. Note that the IEEE 802.11 standard does not specify any ARF algorithm, and implementations of this mechanism vary between different card vendors. We have incorporated an ARF mechanism into the *ns-2* simulator based on the description presented in [78]. We use Figure 4.6 to explain the relative locations of nodes in the following experiments.

4.4.2 Experiments with a Single Sender

In the first experiment, an *ftp* sender is placed at C in Figure 4.6. We consider two mobility cases for a proxy-capable client: (1) A proxy is initially co-located with the AP, and moves towards C (westbound), starting at time 25 seconds, at the speed of 1 m/s. It reaches C at 275 seconds. (2) It is initially at P, and moves towards S (southbound) with the same speed. Both these scenarios capture how the location of a proxy affects bandwidth performance at the client.

Figure 4.7 illustrates the achieved bandwidth averaged over 20 second intervals for

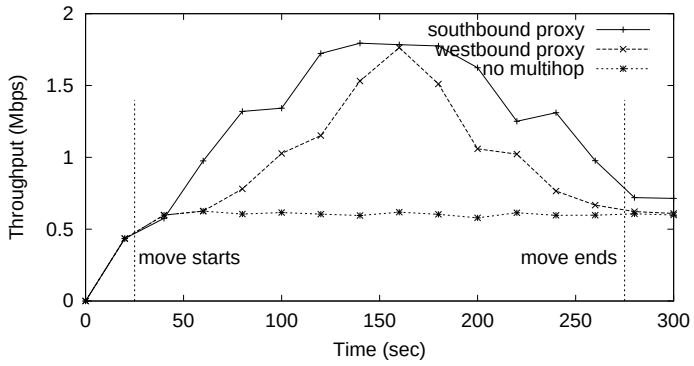


Figure 4.7: *Bandwidth benefits of multihop extensions for a single sender.* The sender is at C in Figure 4.6. The westbound proxy-enabled client moves from AP to C The southbound proxy-enabled client moves from P to S.

these two cases, and compares it with the single-hop scenario. In absence of multihop extensions, the client achieves a data throughput of about 0.5 Mbps. The data throughput achieved in the multihop scenario depends on the location of the proxy. For example, when the westbound client is close to the AP, it is not useful as a proxy to the sender. Therefore, the sender continues to use the direct path to the AP. At time 75 seconds, the westbound client has moved sufficiently away from the AP, and the sender starts using it as a proxy. Note that the bit error rate is higher for a channel with larger distance. Hence the best data performance is observed when the proxy is located at R (mid-way between the client and the AP) at time 150 seconds. As expected, we observe that the proxy-enabled client moving along the Y-axis is better located for bandwidth performance at C.

In the next experiment, we show that the proposed protocol allows a *Replacement* operation when a better proxy becomes available. In this experiment, the sender is at C as

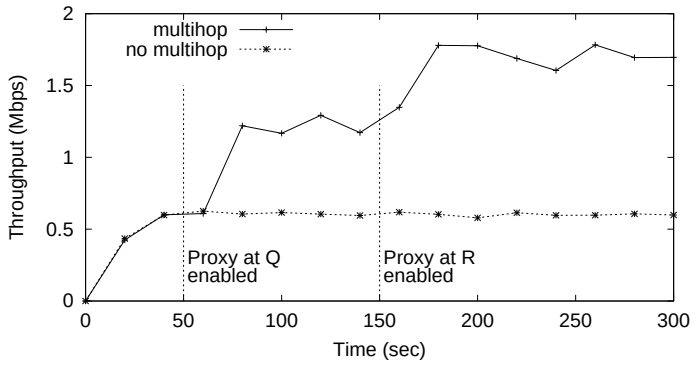


Figure 4.8: *Adaptation of multihop path using the* Replacement *operation.* The sender is at C in Figure 4.6, and two proxy-enabled clients are at Q and R, respectively. The two upward transitions in bandwidth, corresponds to the adoption of each proxy in the multihop path.

before. There are two proxy-enabled clients, at Q and at R, respectively. Furthermore, the client at Q is enabled to act as a proxy after 50 seconds from the start of the simulation. The other client (at R) is enabled to act as a proxy after 150 seconds. (We can imagine that these two proxy-enabled clients are plugged into the power source and become willing to serve as proxies at those respective time instants.)

In Figure 4.8 we present the results from this experiment. The sender starts to use the client at Q as a proxy starting at around 70 seconds. This corresponds to an increase in the bandwidth in the plot (from 0.5 Mbps to 1.3 Mbps). Subsequently, when R is available, it is evaluated to be a better proxy. R sends an appropriate *ForwardProxyBid* which is accepted by the sender in a *Replacement* operation. This happens at time 165 seconds, and the bandwidth increases to around 1.8 Mbps.

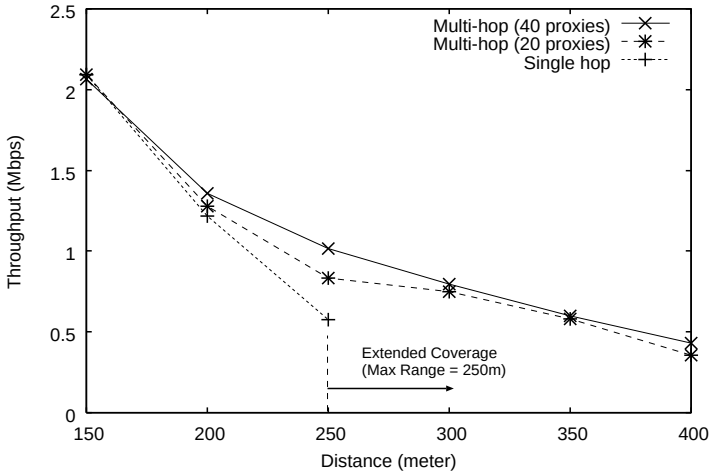


Figure 4.9: Average end-to-end throughput when we vary the distance between the source and the AP.

In the previous experiments, we placed proxies at interesting locations. In the following experiments, we place proxies uniformly at random within the coverage area of AP while experimenting with different numbers of available proxies. We also vary the distance between a source and the AP and examine how the end-to-end throughput changes. In some of the experiment scenarios, the sender client is located beyond the transmission range of the AP. We use ten different proxy placement scenarios for each distance and compare the average end-to-end throughput against the single-hop scenario.

In Figure 4.9, we plot the average end-to-end throughput when we vary the distance between the source and the AP. For clarity, we report only the results when there are 20 proxies and 40 proxies. As in the previous results, we observe that by using the multihop extensions the sender can achieve higher end-to-end throughput. For example, when the sender is 250 meters from the AP, the use of multihop extensions allows the sender to

achieve 45% throughput improvement with 20 proxies and 77% improvement with 40 proxies. With more proxies, the performance improvement is larger because the source can choose to use a better-located proxy. However, as the source becomes farther from the AP, the difference between the two multihop cases becomes smaller. As shown in the figure, without using proxies, the source farther than the maximum transmission range (250 meters) cannot communicate with the AP. In contrast, the use of multihop extensions significantly increases the coverage area. Specifically, with the multihop extensions, when the source is 350 meters away from AP, it still can communicate with the AP at a rate of 0.60 Mbps (with 40 proxies), which is higher than the single-hop throughput of the source at 250 meters. We also note that as the distance between the sender and the AP becomes larger, the sender is sometimes unable to find a multihop path using randomly placed proxies. For example, when there are 20 proxies, in two experiments out of ten, the source at 400 meters from the AP was not able to find a multihop path.

In the next section, we present the results when we use the multihop extensions in the presence of multiple senders in a WLAN.

4.4.3 Impact on Other Senders

We now examine the impact of such multihop paths on other sources. Intuitively it appears that a source using a multihop path incurs a higher channel contention in the common wireless medium and adversely affects the performance of other sources. However, in this set of experiments we demonstrate that when sources with poor bandwidth to the AP use multihop paths instead of the direct paths, they positively impact the performance

of other data sources sharing the same wireless medium.

We first consider a scenario with two senders, located at B ("near" sender) and C ("far" sender) respectively (in Figure 4.6). At time 200 seconds, a proxy-enabled client is activated at location R. At time 400 seconds, the far client starts to move eastbound from C (to R) at the speed of 2 m/s. We examine the bandwidth and latency experienced by the two clients in Figures 4.10 and 4.11 respectively.

In the first 200 seconds, both the clients achieve around 0.5 Mbps of data throughput on the channel (Figure 4.10). Note that the far client experiences higher error rate than the near client, and therefore due to ARF mechanisms, typically uses a lower data rate (1 Mbps) than the near client (which often can use 11 Mbps). Consequently when the far client gets access to the channel, it occupies the channel for a longer time duration than the near client to transmit the data packet of the same size. This is because it transmits the data frame at a lower data rate. Although the near client transmits at a higher data rate, the far client gets a larger time share of the channel, effectively canceling out the benefits of the higher data rate of the near client. Similar observations of 802.11 WLAN behavior were made in [12, 79].

Now we observe how multihop extensions used by the far client positively impacts the near client. Note that the near client itself does not use multihop extensions. At time 200 seconds the proxy-enabled client is activated at R and the far client starts using this proxy to enhance its own bandwidth. We can observe in Figure 4.10 that simultaneously, the bandwidth of the near client also improves. This can be explained as follows. With the availability of the proxy, the far client is able to use higher data rates, and consequently reduces the time occupancy of the channel. Consequently the near client is able to occupy

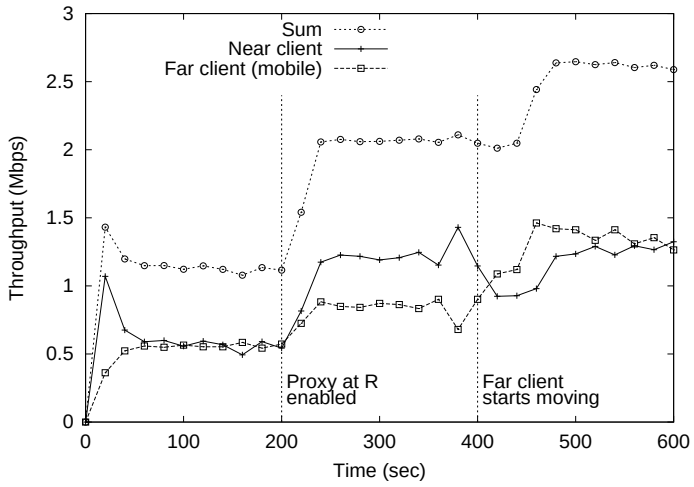


Figure 4.10: *Impact of multihop extensions on bandwidth at other senders.* Two senders are located at B (near client) and C (far client) in Figure 4.6. A proxy-enabled client (located at R) is activated at time 200 seconds. At time 400 seconds, the far client starts moving towards R at the speed of 2 m/s.

the channel for a larger proportion. This leads to the improved data throughput for the near client. In Figure 4.10 we can see that the availability of the proxy-enabled client increases the aggregate data throughput (line marked 'sum') from 1.2 Mbps to about 2.05 Mbps. The use of multihop paths by the far client also positively impacts the end-to-end latency experienced by both the clients (Figure 4.11). When the far client starts using the proxy, the latency of the two clients drop from 80 and 60 ms respectively to about 33 ms for both of them.

Finally, as the far client starts to move towards the AP (at time 400 seconds), the error rate on its direct path to AP further reduces. When it reaches location R, the direct path is obviously more efficient than the multihop path. It switches back to a direct single-

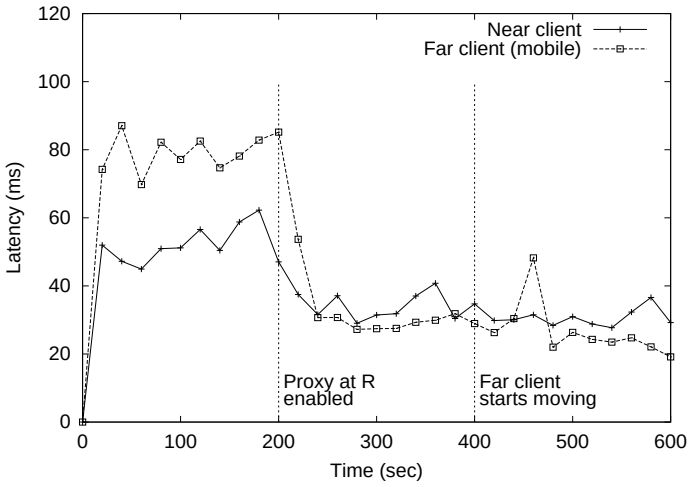


Figure 4.11: *Impact of multihop extensions on latency at other senders.* This is the latency plot corresponding to Figure 4.10.

hop path to the AP (*Relaxation*), and we observe another increase in aggregate bandwidth for the two clients (Figure 4.10).

Finally we report the results of experiments with a larger number of wireless senders to see the impact of multihop extensions in such a scenario. In these experiments there are 20 wireless clients randomly distributed around the AP. Five of these clients are *ftp* sources. We classify these sources into two groups—those that leveraged a multihop path ("proxied"), and those for which the direct hop path provided good bandwidth ("direct"). In Table 4.3 we present a summary of the bandwidth received by all these clients. All the values are averaged over 50 runs of the simulations.

In Table 4.3, we observe that multihop extensions lead to better bandwidth performance for both direct as well as proxied clients. For the single channel case the improvements are 61% and 16% for direct and proxied clients respectively. For the two-channel

	No multihop	Multihop 1-channel		Multihop 2-channel	
Client	Mbps	Mbps	Ratio	Mbps	Ratio
Direct	0.28 (0.02)	0.45 (0.07)	60.7%	0.48 (0.04)	71.4%
Proxied	0.32 (0.01)	0.37 (0.03)	15.6%	0.49 (0.04)	53.1%
All	0.30 (0.01)	0.41 (0.05)	36.7%	0.48 (0.04)	60.0%

Table 4.3: Performance improvement of multihop extensions in Direct, Proxied, and all clients for both 1-channel and 2-channel cases. Numbers in parenthesis indicate standard deviations.

case, they are 71% and 53% respectively. Note that the clients close to the AP use direct paths. Their data performance were significantly impacted by the distant clients in the single-hop WLAN. The distant clients used proxied paths in the multihop WLAN environment and allowed the near clients to significantly improve their path bandwidths.

Control Overheads The extra control overheads due to the multihop extensions was marginal. This is because most of the inference was done using passive measurement techniques. In all our experiments, the extra control traffic was < 1 packet per second.

4.5 Conclusions

In this chapter we have defined a multihop WLAN architecture and quantified its benefits. We also have defined deployment paths for these multihop extensions that can interoperate with existing deployed WLANs. Through detailed measurements and simulation studies we show that the proposed mechanisms benefit all WLAN users: those that use the proposed multihop extensions, as well as those who do not adopt these extensions.

Chapter 5

Protocol Extension for Multihop Geographic Routing

In geographic routing, nodes locally select next hop nodes based on location information of neighbors and destination. Therefore, neither route establishment nor per-destination state is required. As large-scale sensor networks become more feasible, properties such as stateless nature and low maintenance overhead make geographic routing increasingly more attractive [80].

The most popular strategy for geographic routing is simply forwarding data packets to the neighbor geographically closest to the destination [5–7]. In this chapter, we propose the use of a new link metric that considers both location and link quality. In Figure 5.1, for example, although A is closest to destination T among S's neighbors, the link between S and A is experiencing high packet error probability. B is slightly farther from T than A, but provides a higher quality link from S. In this scenario, forwarding packets to B is better, and our new metric lets us choose B over A.

The contributions of this work are as follows.

- We propose a new link metric called *normalized advance (NADV)* in geographic routing. Instead of the neighbor closest to the destination, NADV lets us select the neighbor with the best trade-off between link cost and proximity. We show that a path chosen by NADV approaches the optimal minimum cost path in networks with sufficiently high node density.

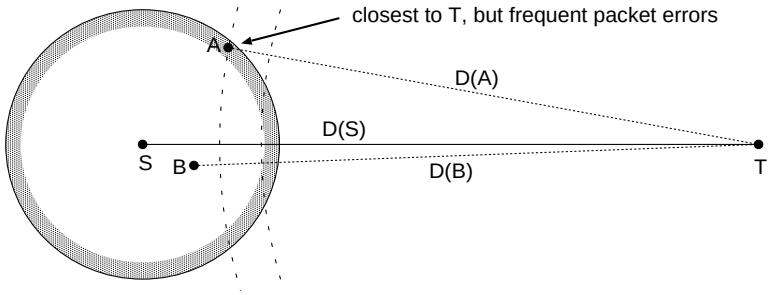


Figure 5.1: An example scenario for geographic routing. While among S's neighbors, node A is closest to T, the link between S and A is experiencing a high packet error rate. Consequently, higher performance can be achieved if S forwards packets to B.

- NADV presents a general framework for efficient geographic routing. Unlike prior schemes that consider only one link cost type [34–36], the NADV framework can accommodate a variety of different cost types.

- Due to the local rule for next hop decision, the use of NADV in geographic routing provides a unique opportunity for adaptive routing. In dynamic ad hoc networks, it is possible that the link costs change while the path is still in use (e.g., due to mobility or environment changes). As long as link cost estimation schemes employed by WISE can track link costs change, NADV immediately reflects the change, which in turn would result in the selection of the best next hop in geographic routing.

- Our experiment results show that NADV significantly improves network performance in various environments. For example, when compared to the current geographic routing scheme in challenging environments with frequent packet losses, NADV leads to 81% higher packet delivery ratio on average (from 16% to 97%).

The simulation results also show that when link costs change, the use of NADV in geographic routing enables adaptive path migration, where the quality of found paths is close to the optimum found by the centralized algorithm.

5.1 New Link Metric for Geographic Routing

In this section, we introduce a new link metric for geographic routing and discuss its optimality in an ideal setting. Here, we assume link cost is positive and known a priori. In practice, we use WISE interfaces to retrieve estimated link cost values.

5.1.1 Background

In this chapter we differentiate link *cost* and link *metric*. An example of link *cost* is the power consumption required for a packet transmission over the link. We define link *metric* as "degree of preference" in path selection. For example, even though two neighbors require the same power consumption, in geographic routing we prefer the neighbor closer to the destination. The goal of this section is to propose a new link metric for geographic routing that can be generalized to various cost types (e.g., power consumption, link delay).

In many geographic routing protocols, the current node S greedily selects the neighbor that is closest to destination T whenever possible [5–7]. The implicit goal of this strategy is to minimize the hop count between source and destination. Let us consider the amount of decrease in distance by a neighbor n, which we call the *advance (ADV)* of

n [81]:

$$\mathit{ADV}(n) = D(S) - D(n), \tag{5.1}$$

where $D(x)$ denotes the distance from node x to T. Then, the above strategy tries to maximize the ADV of next hop, and ADV is the link metric in this case. However, this link metric ADV does not take link cost into account, while different wireless links can have different link costs. For example, Lundgren et al. [3] identify *gray zone*, where due to high error probability, nodes cannot exchange long data packets in most cases. Therefore, the simple policy using ADV may use poor quality links and lead to unnecessarily high communication cost [2].

Clearly, when choosing next hops we want to avoid neighbors with very low quality links. At the same time, we want to gain as large advance as possible for fast and efficient packet delivery. The goal of our work is to balance the trade-off, so that we can select a neighbor with both large advance and good link quality. We can achieve this goal by using the new metric proposed next.

5.1.2 Normalized Advance

We now introduce a new metric called *normalized advance (NADV)*. Suppose we can identify the link cost $\mathit{Cost}(n)$ of the link to neighbor n. Then the normalized advance of neighbor n is simply:

$$\mathit{NADV}(n) = \frac{\mathit{ADV}(n)}{\mathit{Cost}(n)}. \tag{5.2}$$

Intuitively, NADV denotes the amount of advance achieved per unit cost. For example, suppose we know that only $P^{succ}(n)$ fraction of data transmissions to neighbor n are

successful. If we use $1/P^{succ}(n)$ as link cost, $NADV(n) = ADV(n) \times P^{succ}(n)$, which means the expected advance per transmission.

We propose to use NADV as link metric in geographic routing, such that a node forwards packets to the neighbor with largest NADV. Besides obvious simplicity, NADV has the following desirable properties:

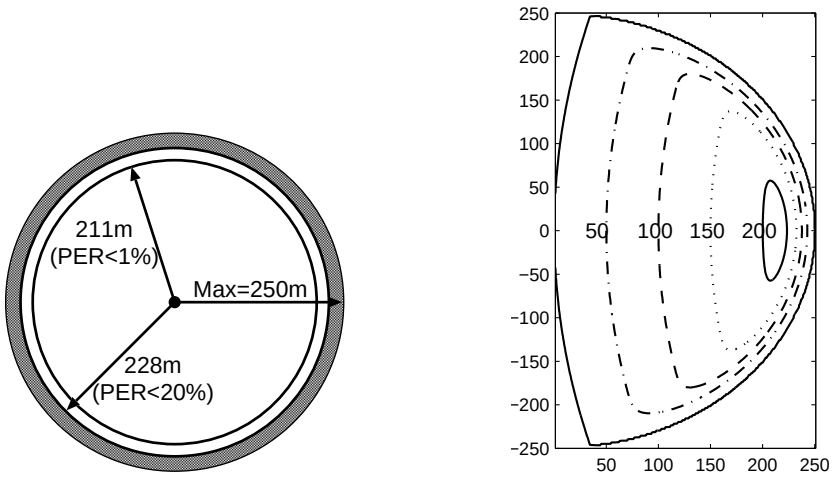
- As shown in Section 5.1.3, the path found by using NADV approaches the optimal path under certain conditions. The experiment results in Section 5.4 show that the use of NADV significantly improves path quality in realistic environments as well.

- It is general and accommodates various types of cost metrics, so that applications can utilize the NADV framework for different objectives. We further describe this feature in Section 5.2.

- Loop freedom is guaranteed as long as we select a node with positive NADV [81].

Using NADV, we can select neighbors that balance the advance against the link cost. Depending on the link cost values, NADV can select a neighbor with strictly less advance (e.g., node B over A in Figure 5.1). We further illustrate this feature in Figure 5.2. Figure 5.2-(a) shows the degree of packet errors to simulate a gray zone[1]. In Figure 5.2-(b), we present the corresponding contour map of NADV when link cost is a function of packet error probability. We can observe that compared to their ADV values, points within the gray zone have relatively low NADV values. As a result, by using NADV, we can easily avoid neighbors in the gray zone. We next provide the theoretical rationale

[1]We assume independent bit errors and use Eq. 3.3 as bit error probability function, which increases rapidly after a certain distance.



(a) Illustration of gray zone (b) NADV contour map

Figure 5.2: Illustration of gray zone and corresponding contour map of NADV. (a) Two inner circles represent the border lines for 1% and 20% packet error rates (PERs) for a 1024-byte frame, respectively. (b) The corresponding contour map of NADV when the packet error probability determines link cost. The current node is at (0,0), and the destination is 900 meters away on the X-axis. Values within the plot denote the NADVs of corresponding lines.

behind using NADV in geographic routing.

5.1.3 Optimality of NADV in an Idealized Environment

We now show that in an idealized environment, paths found by using NADV are optimal. The goal of routing in this discussion is to minimize the sum of link costs along the found path. We make two assumptions: (1) we can find a node at an arbitrary point, and (2) link cost is an *unknown* increasing convex function of distance (e.g., transmission power consumption [14, 34]). Let *DIST* be the distance between the source and the desti-

nation, which we assume is relatively large. Since the cost function is increasing, and we can find a node at an arbitrary point, an optimal path will use only nodes along the straight line between the source and the destination. Also, since link cost is a convex function of distance, the sum of link costs is minimized when all links have the same distance. As a result, the optimal policy is to choose nodes on an equidistant basis along the line that connects the source and the destination.

Now, it remains to find the optimal interval. Suppose ADV_X is an interval, and $Cost_X$ is the corresponding link cost. Then we want to minimize:

$$\begin{aligned} \textit{Total Cost} &= \textit{(Link Cost)} \times \textit{(Hop Count)} \\ &= Cost_X \times \left\lceil \frac{DIST}{ADV_X} \right\rceil \\ &\approx DIST \times \frac{Cost_X}{ADV_X}. \end{aligned} \tag{5.3}$$

The last line comes from the assumption of large *DIST*, which makes the rounding error negligible. From Eq. 5.3 we can find the minimum cost path by iteratively selecting nodes with minimum $\frac{Cost}{ADV}$, or equivalently *maximum NADV* $= \frac{ADV}{Cost}$.

In practical wireless networks, the above assumptions are unlikely to be true. In low-density networks, nodes may not be able to use the greedy forwarding rule, and the recovery procedure will likely result in performance degradation [7]. Also, although many existing schemes are based on the simplified model, and there usually exists strong correlation [3, 60], the link cost is not a strict function of distance in practice. In Section 5.4, we use simulations to show that NADV significantly improves performance in realistic environments as well.

Although the concept of NADV is simple, the implementation for practical use

involves a number of challenges. Link cost estimation is one of the most critical elements, and we use WISE interfaces to retrieve link cost values. In the next section, we describe how NADV framework can be used with various link cost types.

5.2 NADV with Various Link Cost Types

In this section, we describe how to use the NADV framework with various types of link cost. We consider three cost types: packet error rate, link delay, and energy consumption. We use WISE interfaces to obtain link cost estimation and find the next hop node based on the corresponding NADV values. This section focuses on the independent use of each link cost, and the issue of interdependence among multiple cost criteria is discussed in Section 5.5.

5.2.1 Packet Error Rate (PER)

Most recent attention has been on how to find a high performance path considering wireless link errors [2, 11]. In this scenario, we use the following as link cost: $C_{error} = 1/(1 - PER)$. It denotes the expected transmission count (ETX) proposed in [2].[2] We use the following link metric, which is the expected advance per transmission:

$$NADV_{error} = \frac{ADV}{C_{error}} = ADV(1 - PER). \tag{5.4}$$

An equivalent link metric is independently developed in [35, 36], as discussed in Section 2.3.

[2]The estimation techniques described here can easily incorporate ACK frame loss probability as in [2], but here we have simplified the description for brevity.

5.2.2 Delay

If link delay C_{delay} is used as link cost to reduce the path end-to-end delay, we can use $NADV_{delay} = \frac{ADV}{C_{delay}}$. Assuming multiple data transmit rates are available in the system, we use the medium time as C_{delay}. (See Section 3.1.2.) The WISE can easily retrieve the current value of data transmit rate from the MAC layer and export the requested value.

5.2.3 Power Consumption

Many wireless systems have a control mechanism for transmission power adjustment to save battery and reduce interference [17, 68]. We assume that using such a mechanism, nodes know the appropriate transmission power level (p_{tx}) to each neighbor. Then, the WISE can retrieve the p_{tx} value and calculate the actual system power consumption C_{power} considering additional components of power consumption [69]. If C_{power} is used as link cost, a geographic routing protocol can use $NADV_{power} = \frac{ADV}{C_{power}}$ to find a path that minimizes power consumption to deliver packets to a destination.

So far, we have listed interesting cost types and shown how the NADV framework can incorporate them. The NADV framework can include other types of link cost as well (for example, *reluctance* metric in [82]). However, in this chapter we limit our attention to the cost types discussed above and report simulation results in the following sections.

5.3 Simulation Model

We use *ns-2* simulations to evaluate the system performance when we employ the proposed NADV metric when coupled with WISE interfaces described in Chapter 3. In

this section, we describe various aspects of simulation in detail. We present the simulation results in Section 5.4.

We place nodes uniformly at random on a 1000m by 1000m square. Unless otherwise stated, 100 static nodes are used in the simulation.[3] We usually use only one source-destination pair to capture the individual performance effects accurately. In this scenario, denoting the lower left corner of the square as (0, 0), the static source is located at (50, 500). The destination is placed at (50+D, 500), where D is the distance between the source and the destination. We usually use D=900. The source generates a CBR (Constant Bit Rate) flow, which sends a 1024-byte UDP packet every two seconds from 10 seconds to 1000 seconds of simulation time. The maximum transmission range R is 250 meters.

For geographic routing, we use the simulation code for GPSR.[4] We have slightly modified the next hop selection algorithm to include NADV. The simulation code for GPSR provides an option about whether to exploit transmission failure notification from the MAC layer [6]. If a node exploits the option, then upon receiving a notification, it selects the next best neighbor for retry until the forwarding is successful. This option leads to higher delivery ratio with higher resource consumption. When not using the notification, a node does not attempt to retransmit to other neighbors. We explore both cases in the simulation. The beaconing period in GPSR is set to 1.5 seconds. We use the IEEE 802.11b standard for the underlying MAC layer protocol [66]. We assume the

[3]We also experimented using sparser networks with 50 nodes. However, in scenarios with high packet error rates, networks frequently became disconnected (e.g., due to repeated beacon message losses).

[4]Available at http://www-2.cs.cmu.edu/~bkarp/gpsr/gpsr.html

	Noise power ($\times$1.0e-12 W)				
	0.8	1.0	1.2	1.4	1.6
(dBm)	(-91.0)	(-90.0)	(-89.2)	(-88.5)	(-88.0)
BER at 220m	6.0e-8	1.1e-6	7.8e-6	3.2e-5	9.1e-5
BER at 240m	4.4e-6	3.5e-5	1.4e-4	3.9e-4	8.3e-4

Table 5.1: Bit error rate values with different levels of noise.

location of the destination is known to the source.

In the following subsections, we describe models of individual simulation components in more detail.

5.3.1 Error Model

To simulate a lossy channel, we use two different error models. First, assuming the use of BPSK modulation in the physical layer, we simulate packet errors using Eq. 3.3 as bit error model. (We assume independent bit errors for simplicity.) In the default *ns-2* propagation model, the signal strength is reduced proportionally to d^2 if the distance d is smaller than a certain threshold. Otherwise, the path loss is proportional to d^4. In this experiment scenario the transmit signal power is fixed at 20 mW (or 13dBm) supported in Cisco Aironet 350 interface cards [83]. Then the received signal strength for a node 250 meters away is -85dBm. The noise channel bandwidth in Eq. 3.3 is set to 2MHz. In this model, we use ambient noise environments, where the noise value is identical everywhere. Therefore the quality of a link depends only on the distance between two nodes, and C_{error} is a convex function of distance. In Table 5.1 we tabulate the used noise

values and corresponding bit error rates (BERs).[5]

To examine the performance of NADV in the presence of randomness in packet errors [64], we also perform simulations using a random packet error model. In this model, for each wireless link, we assign a packet error rate, which is distributed uniformly at random between 0 and a maximum value (*max-PER*). We vary the maximum packet error probability for different degrees of packet losses. In practice, shorter packets such as periodic beacons experience lower error probability [2], and we adjust the error probability for these packets according to Eq. 3.4. Clearly, link cost is not a function of distance in this model.

In some of our simulations, we compare NADV against another scheme called *blacklisting* [35, 84]. This scheme uses a fixed threshold, and when selecting a next hop, a node excludes neighbors with low-quality link based on the threshold. For example, if we use a threshold value of 0.5, then a node excludes neighbors that are closer to the destination and belong to the lower half in the link quality. Among the remaining neighbors, the blacklisting scheme selects the neighbor with largest ADV.

5.3.2 Transmission Rate Adaptation and Link Delay

Most IEEE 802.11b wireless cards dynamically adjust the data transmission rate b_{data} using Automatic Rate Fallback (ARF) [78]. In ARF, according to MAC transmission failures or successes, each node adjusts b_{data} to 1, 2, 5.5, or 11 Mbps. For control frames such as RTS and CTS, nodes use another transmission rate b_{basic}, which is fixed at 1

[5]Noise values from more than 20000 measurements in our building range from -91dBm to -73dBm, with the median at -89dBm. The noise value used in Figure 5.2 is -89.2dBm.

Mbps in the simulation study. We incorporate ARF algorithm to the MAC simulation code. Then, we can calculate the medium time as described in Eq. 3.7 in Chapter 3.

5.3.3 Power Consumption Model

As discussed above, the strength of transmitted signal decreases in proportion to d^n, where d is distance, and n is the *path loss exponent* (usually $2 \leq n \leq 6$). Suppose that a receiver network interface requires a received signal strength of at least S_{min} for successful packet reception. To simplify the description we assume that the transmission power should be at least $d^n S_{min}$ for successful reception at a receiver whose distance is d. Modeling after most wireless systems and products [17, 83], we assume in this section that the transmission power is restricted to one of L levels in the set $P = \{p_1, p_2, ..., p_L\}$. In this scenario, it is best for a node to use the smallest power level no less than $d^n S_{min}$:

$$p_{tx} = min\{p_m : d^n S_{min} \leq p_m,\ 1 \leq m \leq L\}. \tag{5.5}$$

Simplifying the *ns-2* propagation model, we fix $n = 4$ in the power consumption experiments. Also, we focus on the relative magnitude of power consumption and use $S_{min} = 1/(R)^n$, where R is the maximum transmission range. Based on the specification of the Cisco Aironet 350 card [83], we use the following set P = {0.01, 0.05, 0.2, 0.3, 0,5, 1.0}.

We also simplify a widely used power consumption model [14, 34, 69] and assume that each packet forwarding consumes the following amount of energy:

$$C_{power} = 1 + c\, p_{tx}, \tag{5.6}$$

where c is a proportionality constant to the transmission power component. Note that $c = 0$ degenerates C_{power} to the hop count metric. c is a hardware-specific constant, which we assume WISE can retrieve. Actual c values of different interfaces range between 0.17 and 1.30, and we use $c = 1.0$ in the simulation [69]. Previous measurement results show that the energy consumption of an idle or receiving wireless interface card is comparable to that of a transmitting one [19]. Although NADV can include these aspects of such energy consumption, for ease of comparison against an existing scheme, we focus on the energy consumption due to transmission. In this model, link cost is a non-decreasing step function.

In some of our simulations, we compare *NADV*$_{power}$ against SP-Power [34]. Given a power consumption equation, the authors of [34] derive a formula for link metric and prove that the node selection based on the metric is optimal in an ideal setting. If SP-Power uses Eq. 5.6 as power consumption equation, the current node selects the neighbor that minimizes the following formula:

$$E = (1 + c\,t_{px}) + \frac{D}{R}(c(n-1))^{\frac{1}{n}} + \frac{D}{R}c(c(n-1))^{\frac{1-n}{n}} \tag{5.7}$$

where D denotes the distance between the neighbor and the destination.

5.4 Simulation Results

In this section we present the results of simulation experiments. We begin with the effect of wireless link errors. We first assume the perfect knowledge of link error rates when we investigate the performance. Then we compare the performance when we use NADV with the WISE interfaces in Chapter 3. We then consider the cases when delay

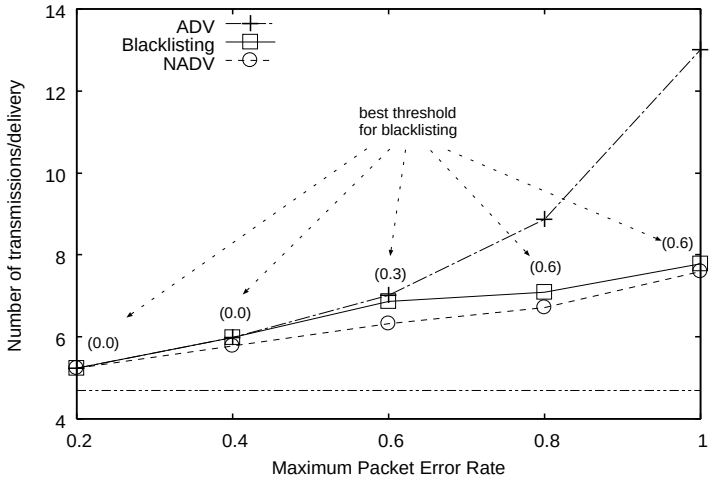


Figure 5.3: The number of MAC-level data transmissions per delivered packet with different degrees of packet errors. C_{error} is used as link cost. Unless the error rate is low, next hops chosen by ADV can cause multiple retransmissions, and ADV significantly increases the number of data transmissions. As marked in the bottom, when *max-PER*=0.2, the average path length using ADV is 4.7 hops.

and power consumption are used as link costs in turn. Finally we compare geographic routing using NADV against idealized routing.

5.4.1 Experiments with Perfect Estimation of Link Errors

We first present the results when nodes experience packet losses due to wireless link errors. In this section, we assume that there exists a perfect estimation scheme that provides accurate link cost values, and compare the performance when NADV and other geographic routing schemes operate based on the knowledge. We later present results when we combine NADV with link cost estimation schemes by WISE.

In the first set of experiments, we use the random packet error model described in Section 5.3.1, where packet error rates are distributed uniformly at random between 0 and *max-PER*. Although in this model the frequency of links at a given error rate is similar to the previous measurement results in [64], this model does not consider the correlation between the distance and link error. As a result, on average, packets sent to distant neighbors have the same error probability as those sent to close neighbors. In fact, this setting is in favor of ADV, because in practice, transmissions to neighbors with large ADV are likely to suffer from frequent errors [3, 60]. We use an average of ten runs, each with different placement of stationary nodes. To avoid the packet errors due to the ARF algorithm, we fix the data transmission rate at 1 Mbps in this set of experiments.

In Figure 5.3, we report the number of MAC-level data transmissions (including retransmissions) per delivered packet for each scheme when we vary the value of *max-PER*. In this set of experiments, GPSR employs MAC-level failure notification, and all results are based on 100% packet delivery. We can observe that as the packet error rate increases, the data transmission overhead of ADV increases abruptly (up to 71% higher than that of NADV). This is because ADV often selects neighbors with low-quality link, which causes repeated retransmissions. In contrast, NADV intelligently avoids nodes with high PER, and although the data overhead of NADV increases as *max-PER* increases, the number of data transmissions is much smaller than that of ADV. Each transmission requires network bandwidth as well as node resources (e.g. battery power), and NADV uses system resources more efficiently.

In Figure 5.3, we also compare the performance of NADV against the *blacklisting* scheme described in Section 5.3.1. Blacklisting uses a fixed threshold, and to find the

best threshold, we consider nine different blacklisting threshold values between 0.1 and 0.9 with an increment of 0.1. (The use of threshold value 0.0 in blacklisting corresponds to ADV.) In Figure 5.3, we plot the best result for blacklisting in each setting and mark the corresponding threshold value in parenthesis. We can observe that depending on the network environment, different threshold values lead to best results for blacklisting—a fixed threshold value in blacklisting does not work well. When packet errors are frequent, it is better to use a high threshold value in blacklisting and exclude many neighbors with low-quality links. However, when there are few low-quality links, the use of a high threshold value may exclude useful neighbors and lead to longer paths. In contrast, NADV adapts to the changing network environment and is able to achieve low data transmission overhead in all cases.

Repeated retransmissions also affect the packet delay. Although not displayed here, the end-to-end latencies of ADV also show an increasing trend similar to Figure 5.3. Specifically, as *max-PER* changes from 0.2 to 1.0, the average packet latency of ADV increases from 54.9ms to 151.6ms. The performance degradation by NADV is less severe (increase from 54.9ms to 81.8ms). We later present more results on end-to-end latency. Instead of NADV, we also experimented using different combination of ADV and link cost, and NADV outperformed them as well. A more conservative link metric (e.g., $ADV/Cost^2$) often results in longer paths, while a different metric such as $ADV/\sqrt{Cost}$ often underestimates high-cost links and causes more retransmissions due to packet errors.

In the previous experiments, we assumed the perfect knowledge of link cost. We next investigate the performance of NADV used with the proposed PER estimation tech-

Name	Description
NADV-Beacon	Using periodic beacon messages (Eq. 3.4)
NADV-SNR	Using SNR (Eq. 3.3)
NADV-Self	Using own data packets (Eq. 3.6)
NADV-Perfect	Assuming the perfect knowledge of link cost

Table 5.2: Different scenarios when NADV and different WISE PER estimation techniques are combined.

niques.

5.4.2 Experiments using WISE PER Estimation Techniques

In Table 5.2, we tabulate three schemes when we use NADV and three different PER estimation schemes by WISE together, in addition to the case with perfect estimation (NADV-Perfect). Note that none of the three estimation schemes use extra control messages. However, in the case of NADV-SNR and NADV-Beacon, we modify the periodic beacon format to include reverse link information, and the message length slightly increases (See Chapter 3.1.1.2). For simplicity, NADV-Beacon and NADV-SNR assume the independent bit error model for packet error estimation. Storage overhead for the link cost estimation is also negligible since each node in GPSR already maintains neighbor information.

In the first set of experiments, we use the random packet error model used in the previous section and compare the actual performance of NADV against the case with the perfect knowledge of link cost. In this model, there is no correlation between SNR

	max-PER		
	0.6	0.8	1.0
NADV-Self	7.0 (0.6)	7.8 (0.8)	8.9 (1.1)
NADV-Beacon	6.6 (0.7)	7.2 (0.6)	7.8 (1.2)
NADV-Perfect	6.3 (0.5)	6.7 (0.7)	7.6 (1.2)

Table 5.3: The number of data transmissions per delivered packet when NADV and different WISE PER estimation techniques are used. Values in parentheses are the standard deviations. When there are no link errors, the average path length is 4.7 hops.

and packet error rate, and we experiment with NADV-Beacon and NADV-Self only. In Table 5.3, we show the data transmission overhead of NADV schemes for high-error scenarios. In this table, the overhead of NADV-Beacon and NADV-Self is reasonably close to that of NADV-Perfect, and we can infer that WISE schemes described in Chapter 3 provide good link cost estimation. Although NADV-Self has the most flexibility in deployment (e.g., no modification of protocol message format), its performance is slightly worse than NADV-Beacon.

In the previous experiments, we used the random packet error model. In the next experiments, we use the independent bit error model, where a bit error occurs according to Eq. 3.3 (See Table 5.1). To identify the performance of proposed PER estimation schemes, we consider three simulation scenarios. First, we change the ambient noise power over time and observe how each estimation technique adapts to varying environments. Second, we increase the number of data flows to vary the network contention level. Third, we consider the scenario with node mobility.

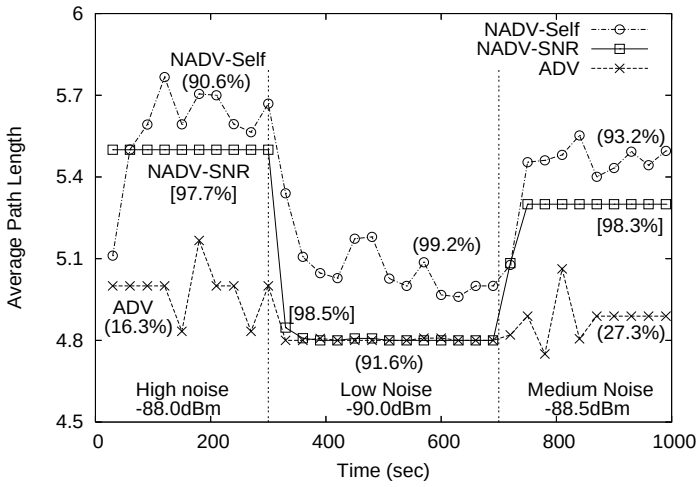


Figure 5.4: The average path lengths of NADV and ADV. The noise value changes from high (-88.0dBm) to low (-90.0dBm), and finally to medium (-88.5dBm). Numbers next to the lines are corresponding delivery ratios in each phase. PER estimation schemes enable NADV to choose appropriate neighbors and maintain high delivery ratios.

5.4.2.1 Changing Noise Power

In these experiments, we investigate the adaptiveness of WISE PER estimation schemes, and we start with a high noise value, change to a low noise value after 300 seconds, and change again to a medium noise value after 700 seconds. In Figure 5.4 we plot the average path lengths for ADV and NADV when we vary the ambient noise power value. We do not employ MAC-level failure notification in GPSR, and the values in the parentheses show the average delivery ratios for different scenarios. Since the performance of NADV-Beacon is similar to that of NADV-SNR, we do not show the result of NADV-Beacon for clarity.

In Figure 5.4, the length of the path chosen by ADV is always shortest, but the packet delivery performance is always worse than that of NADV. For example, in the high-error scenarios (noise value=-88dBm), the difference in packet delivery ratio is more than 81% (16.3% vs. 97.7%). In this scenario, although ADV does not adapt to the environment, beacons from far neighbors are frequently lost, and the path length increases. When we use NADV, we observe that the WISE PER estimation schemes successfully assign appropriate link costs in dynamic environments. As a result, NADV uses different neighbors according to the current environment, and the path length change is more noticeable. NADV-SNR explicitly utilizes the link characteristic value, and in this simulation model, NADV-SNR exhibits more accurate estimation and faster convergence. NADV-Self occasionally employs slightly longer paths than NADV-SNR, but it is also able to adapt to environment changes.

5.4.2.2 Varying the Number of Data Flows

In the previous experiments, we use only one pair of source and destination. When we have more source-destination pairs, the network contention increases, in which the proposed estimation techniques may estimate PER values incorrectly. For example, although received SNR values may be high, a node can experience high packet error rates due to increased collisions. To identify the performance of estimations schemes in this scenario, we vary the the number of data flows in the next set of experiments. We choose source-destination pairs uniformly at random. As in the previous experiments, we do not use the MAC-level notification of GPSR and report the average packet delivery ratio for

	Number of Flows			
	1	2	4	8
NADV-Beacon	98.6	98.7	97.3	97.6
NADV-SNR	99.2	99.3	97.9	98.1
NADV-Self	98.8	99.0	97.4	97.8
ADV	71.9	73.8	71.3	77.6

Table 5.4: Data delivery ratio (in %) when the number of data flows is varied.

each scenario. Among the values in Table 5.1, we fix the noise value at -89.2dBm, which is the closest to the median of noise measurements in our building.

In Table 5.4, we report the average data delivery ratios in different scenarios. In these experiments, all NADV schemes perform similarly. Although the delivery ratio appears to decrease when we increase the number of flows, the amount of degradation is small. When we experimented using 16 flows without packet errors, we observed low delivery ratios due to the network saturation, and eight data flows in this setting corresponds to relatively high network utilization. These results show that WISE estimation techniques work well in the presence of high network load. In contrast, when we use ADV, the average delivery ratio lies between 70% and 80%, depending on the random node placement.

5.4.2.3 Experiments with Mobile Nodes

In the previous results, we have shown that WISE PER estimation techniques perform well in static networks. We now investigate how well they adapt to network topology

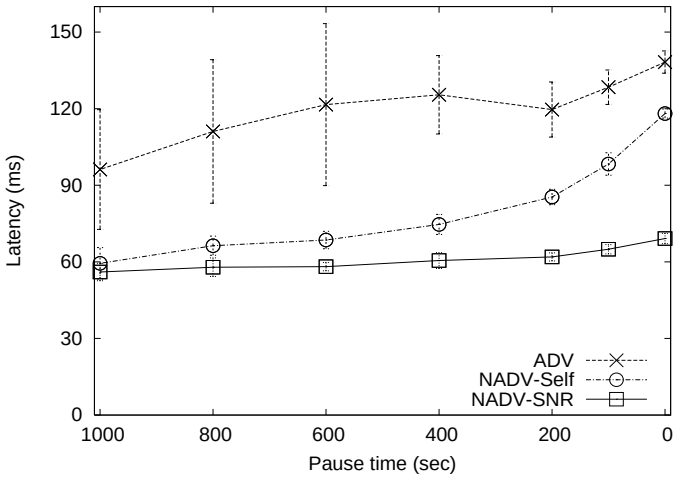


Figure 5.5: Average end-to-end latency when nodes are mobile. C_{error} is used as link cost. We changed the pause time for different degrees of mobility. NADV and WISE PER estimation schemes are effective with node mobility.

changes. In this scenario, the source and destination pair do not move, but the remaining 98 nodes move according to the random waypoint model. The speed is randomly chosen between 1 and 10 m/s, and we vary the pause time for different degrees of node mobility. We use the MAC-level failure notification and fix the ambient noise power at -89.2dBm.

In Figure 5.5, we present the end-to-end latency results with varying mobility. As mentioned before, the data transmission overhead shows a similar trend to Figure 5.5. We observe that average latencies increase as node mobility becomes higher. This is because frequent link failures cause more retransmissions. Compared to ADV, both NADV schemes achieve lower average latency. With NADV-SNR, PER estimation is more accurate, and the increase in end-to-end latency is minimal even with the highest mobility (50% lower compared to ADV). When NADV-Self is used in high mobility scenarios,

Distance	500m	600m	700m	800m	900m
ADV	22.9	26.5	31.7	36.2	42.9
NADV_{delay}	14.5	17.3	20.0	22.7	26.2

Table 5.5: Average end-to-end latency (in ms) with different source-destination distances.

most neighboring nodes move out of range before the estimated values can converge. As a result, the performance gain is smaller than in low mobility cases. Still, its average latency is 15% shorter than that of ADV when nodes move constantly. NADV-Beacon also requires a certain number of beacon messages for good estimation, and the results are similar to those of NADV-Self (within 5% difference in all cases), which we do not show here for clarity.

To summarize, we observe that WISE PER estimation schemes are effective even with node mobility, and NADV combined with them provides an efficient and adaptive geographic routing strategy. As the network environment becomes harsher, the performance of NADV degrades gracefully. In the next subsection, we discuss the results when link delay is used as link cost.

5.4.3 Using Delay as Link Cost

In this subsection, we use link delay as link cost, and $NADV \equiv NADV_{delay}$ in this scenario. We use the independent bit error model using Eq. 3.3, and ARF is used for transmit rate adjustment. In this model, due to the interaction with ARF, link cost is not a convex function. In this experiment, we use a low noise value of -91.0dBm in this

set of simulations. Note that this scenario is in favor of ADV because with high noise, ADV suffers from increased end-to-end latency as previously discussed. The MAC-level failure notification is used, and the delivery ratios are 100% in all cases. Each value in this experiment is an average of ten runs.

In Table 5.5, we report the average end-to-end latency of each scheme when we vary the distance between the source and the destination. As the distance increases, packets go through more relay nodes, and the latency increases accordingly. Compared to ADV, NADV significantly decreases the end-to-end latency (by up to 35%). It is because when we use ADV, we are likely to choose far neighbors to minimize the distance to the destination. However, the transmission rates to such nodes are usually 1 or 2 Mbps, which causes the transmission to take longer. With the use of NADV, the current node may choose a neighbor that is not the closest to the destination, but the corresponding link is good enough for a higher transmission rate such as 11 Mbps. This strategy eventually leads to shorter transmission time.

When using $NADV_{delay}$ in this simulation scenario, the current node usually selects neighbors close to itself, which leads to more relay nodes (e.g., 55% increase when the distance is 900m). Since this increase is based only on the local decision to minimize the medium time, it may degrade the overall performance, especially when multiple traffic flows exist in the network. To investigate this potential problem, we perform experiments using different numbers of source-destination pairs, which we select uniformly at random.

In Figure 5.6, we plot the average end-to-end latency when we change the number of flows in the network. We can observe that with more flows in the network, ADV increases the average latency noticeably. This is because ADV holds the wireless medium

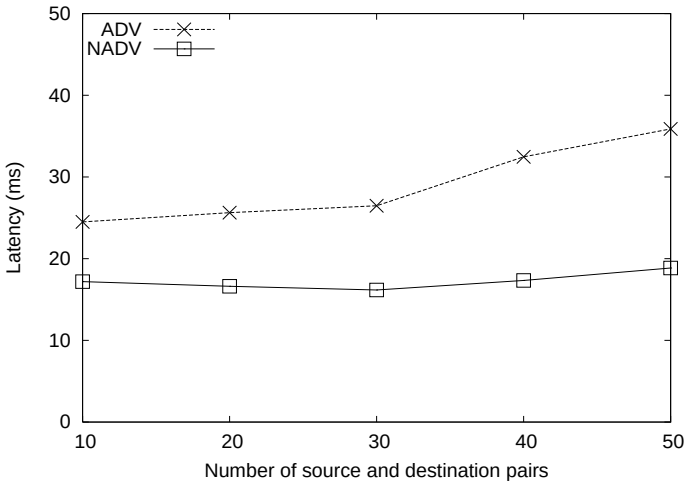


Figure 5.6: Average end-to-end delay with multiple flows. ARF and C_{delay} are used. When NADV is used, the network can support more flows without significant increase in the latency.

longer than necessary, leading to a higher level of network contention. In contrast, NADV maintains the aggregate medium time low enough, such that the network can support more flows without significant increase in the latency. Consequently, compared to ADV, NADV improves the latency performance even more with higher network traffic load. Specifically, in the case of 10 flows, NADV decreases the average latency by 30%, but with 50 flows the improvement is 48%.[6] In the case of 50 flows, only 2 flows experience slight increase ($<$ 2ms) in the end-to-end delay. This experiment result shows that the use

[6]In the experiments for Table 5.4, we use the fixed data transmission rate of 1 Mbps, and we observe network saturation when we send more than 8 packets per second. In the experiments for Figure 5.6, the data transmission rate can be up to 11 Mbps, and NADV can support more data flows without network saturation.

of *NADV*$_{delay}$ does not negatively affect the performance of other traffic in the network.

5.4.4 Using Power Consumption as Link Cost

We compare *NADV* ($\equiv$ *NADV*$_{power}$) against the metric proposed in the *SP-Power* scheme [34]. When the power consumption equation is $C_{power} = 1 + c\,t_{px}$, NADV needs to know the current transmission power p_{tx}, which we assume WISE exports using information from a control mechanism. SP-Power requires the exact value of path loss exponent, which we also assume is available. In practice, however, the path loss exponent estimation is not trivial, and depending on the measurement parameters, the estimated values can vary significantly [17,85]. We assume that both schemes know the proportionality value c, which is a hardware-specific constant. In the following set of simulations, the distance between source and destination is 900m, and there are no packet errors. We vary the node density and use average values of 20 runs for each case. We also compare the performance of optimal paths found by the centralized algorithm.

In Figure 5.7, we plot the average power consumption of each scheme with different node density. The amount of power consumption in each scheme decreases as node density increases. This is because with higher node density, more neighbors become available, and all schemes likely choose better next hops. We also observe that compared to ADV, both NADV and SP-Power find paths that reduce overall power consumption.[7] The performances of NADV and SP-Power are almost identical; NADV performs slightly

[7] In Figure 5.7, the performance difference between the optimal case and ADV is not large. It is because the constant term in Eq. 5.6 constitutes a signifi cant power consumption regardless of the transmission power, as is the case with most existing products [69].

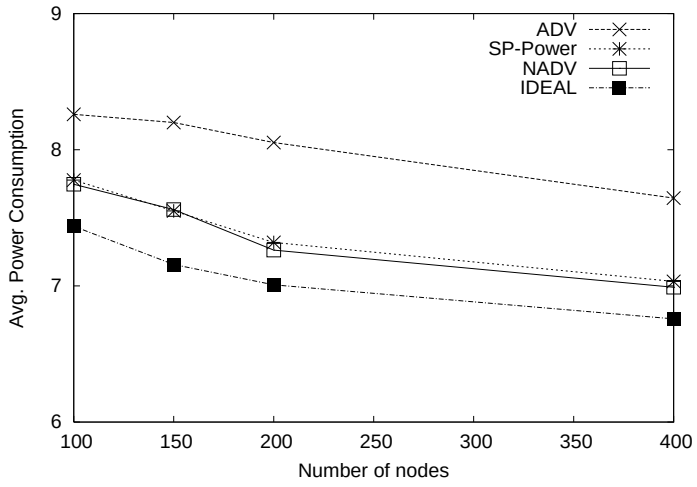


Figure 5.7: Average power consumption with different schemes. In dense networks, as more neighbors are available, power consumption decreases. The power consumption values by NADV and SP-Power are similar, which are close to the optimal value.

better. (NADV and SP-Power find the same path in 15 cases out of 20 in the 400-node scenarios.) Even though we do not report detailed results in this chapter, *NADV*$_{power}$ and SP-Power also show very similar performance in other settings (e.g., distance, continuous power adjustment, different path loss exponents, and proportionality constants c). For other aspects of energy consumption (e.g., in idle or receiving mode), we expect that *NADV*$_{power}$ and SP-Power will consume a similar amount of energy and that their performance will be close to each other as well.

When the goal of geographic routing is to minimize the path power consumption, we argue that *NADV*$_{power}$ is the metric of choice. *NADV*$_{power}$ and SP-Power are based on a similar rationale for next hop selection and exhibit almost identical performance. However, as mentioned above, SP-Power needs to estimate the path loss exponent, which

can be difficult in practice. In contrast, *NADV*$_{power}$ only requires t_{px}, which WISE can easily determine with the support of existing control mechanisms [17, 68].

5.4.5 Experiments with Generic Cost

Recently, new metrics are being proposed for various multihop routing purposes. For example, Draves et al. [86] propose the WCETT (Weighted Cumulative Expected Transmission Time) metric to improve network throughput in multi-radio mesh networks. As multihop wireless networks become more widely used for different objectives, we expect to see other new routing metrics proposed to achieve specific goals. In this section, we apply NADV to a generic cost metric to see whether the use of NADV can be generalized to other types of link cost. We use the following link cost:

$$C_{generic} = 1.0 + r\left(\frac{d}{R}\right)^2,\ 1 \leq r \leq 5, \tag{5.8}$$

where r is a uniformly distributed random number, d is the distance between two nodes, and R is the maximum transmission range. The above link metric attempts to capture both the correlation with distance and the random property of link quality [60, 64]. In this subsection, we assume the availability of accurate and up-to-date link cost information.

We use the following experiment scenario. The source and the destination are 900 meters apart, and the source starts to send data packets after 10 seconds. At 30 seconds, we assume that the environment of some part of the network changes (e.g., due to new obstacles, increased interference, node mobility), and we randomly select 50% of links and increase their link costs by 50%. For NADV, we additionally consider a geographic routing scheme that uses two-hop neighborhood information [32]. To compare NADV

	ADV	Non-adaptive (AODV)	NADV one-hop	NADV two-hop	IDEAL
Initial	14.43	11.14	11.28	10.82	10.32
After change	18.51	14.30	13.50	12.52	11.62

Table 5.6: The average costs of paths found by respective routing schemes when link costs change.

against AODV [72], we modify the AODV simulation code, such that AODV finds paths that minimize the sum of link costs along the paths, not hop count.

In Table 5.6, we report average path quality of each scheme before and after the link cost change. Each value in the table is an average of ten experiments. In this table, we can see that using NADV, geographic routing (both one-hop and two-hop) can find paths comparable to the optimal paths. Not surprisingly, utilizing two-hop neighborhood information leads to higher-quality paths than the one-hop case. The performance of initial paths by AODV lies between those by one-hop NADV and two-hop NADV. However, even after some link cost values increase after 30 seconds, AODV keeps using the initial path, and the path performance degrades accordingly. In contrast, the use of NADV enables localized geographic routing to detect the change and determine better next hops, which results in better paths.

In summary, geographic routing with NADV can find paths whose costs are comparable to the optimum. It is also able to adapt to network environment changes, due to the localized next hop decision.

5.5 Summary and Future Work

We have introduced NADV as link metric for geographic routing in multihop wireless networks. Geographic routing with NADV provides an adaptive routing strategy, which is general and can be used for various link cost types. We have presented how NADV can be used with multiple WISE interfaces. In the simulation experiments, the combination of NADV and WISE cost estimation techniques outperforms the current geographic routing scheme. NADV also finds paths whose cost is close to the optimum.

In this section, we have treated each link cost type independently. However, if we consider multiple interdependent costs simultaneously, choosing the next hop based on one cost type may not be always the best choice for other costs. One possible future direction will be to design a link cost model that balances multiple cost criteria, which would allow the NADV framework to leverage the combined link cost to find a low cost path.

Chapter 6

TRUNC-K: Virtual Backbone Construction for Wireless Networks

In this chapter, we present a backbone construction scheme to increase the lifetime of multihop wireless networks. The most popular model for backbone is *connected dominating set*. Nodes not in the backbone have at least one backbone neighbor (hence the backbone is a dominating set) and do not participate in routing and forwarding to save energy. Smaller backbones[1] lead to greater overall energy savings [39–42], but when nodes are battery-powered, the use of low-battery nodes in the backbone can shorten the overall network lifetime. Therefore, many schemes have been proposed that consider the residual battery power in selecting backbone nodes [19, 20, 43]. However, along with the battery capacity, these schemes often also use other criteria for including nodes in the backbone (e.g., randomized node selection for arbitration [19]). This leads to the inclusion of low-capacity nodes in the resulting backbone. The construction of small backbones composed of high-capacity nodes is the subject of this chapter.

The operating environments for multihop wireless networks can vary widely (e.g., minimal node mobility in sensor or rooftop networks [64] vs. higher mobility for rescue operation). Ideally, a backbone construction algorithm should work well in a wide range of network environments. In some existing backbone construction algorithms, nodes use only local information to build and maintain a backbone quickly [19, 20, 38, 43]. Although this class of backbone algorithms can be useful in dynamic networks, they do not

[1] In general, finding an optimal connected dominating set is NP-hard.

provide any guarantee on performance objectives such as backbone size or node capacity. Other backbone construction schemes find a "good" connected backbone, e.g., with provable bounds on backbone size or control overhead. However, this second class of algorithms typically have higher control overhead, require longer convergence times, and do not provide efficient mechanisms for backbone maintenance [39, 41]. Therefore, they are most useful in static environments, but in dynamic networks, the overhead of maintaining a "good" backbone can be prohibitive. Due to such inherent heterogeneities in the operating environments for multihop wireless networks, it is unlikely that a single fixed algorithm will work best in all situations. In this work, we develop a general solution that can be tailored to particular network environments.

The contributions of this work are as follows:

- We present a *parameterized backbone construction algorithm*, which permits explicit tradeoffs between different performance measures including backbone size, resilience to node movement and failure, node capacity, and path length. Our scheme has two logical steps. First, each node nominates its highest-capacity neighbor as its *leader* (Section 6.1). Next, we connect these leaders such that the resulting backbone achieves specific efficiency and resilience properties (Section 6.2).

- We prove that our scheme can construct essentially *best possible backbones* with respect to node capacity and backbone size (Sections 6.1 and 6.2). To the best of our knowledge, this is the first work that achieves both objectives at the same time.

- Based upon our backbone construction algorithm, we present a *distributed protocol* that builds and maintains a connected backbone in dynamic networks where nodes

are mobile, and node capacity constantly changes (Section 6.3).

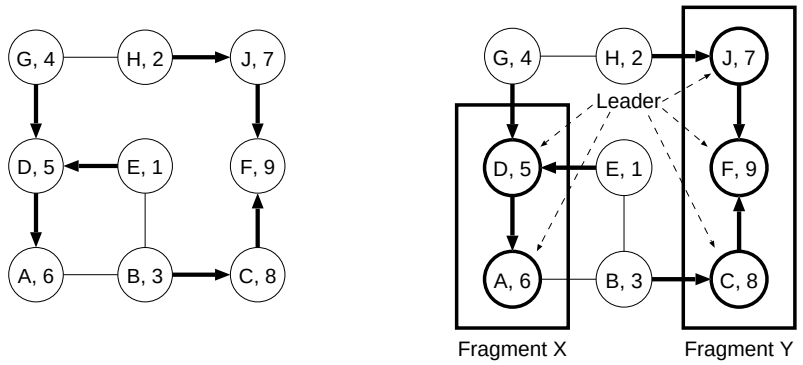
- We present *simulation results* that investigate different performance aspects of our proposed algorithm, including backbone size, network lifetime, backbone node capacity, and path length. Compared to previous energy-saving techniques, our scheme increases network lifetimes by 20–220% without adversely affecting data delivery or end-to-end latency (Section 6.4).

6.1 Leader Nomination

In this section, we first describe how each node nominates a *leader* in the initial phase of our algorithm, and then show desirable properties of the resulting set of leaders. We defer the description of connecting the leaders to Section 6.2.

We assume the network is connected and model it as undirected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges between nodes. (We discuss the issue of uni-directional links in Section 6.3.) We denote the total number of nodes in the network by $n = |V|$. We define $N(v)$ to be the set of neighbors of node v, and $N^+(v) = N(v) \cup \{v\}$. We denote v's degree by $d_v = |N(v)|$, and $\Delta = \max_{v \in V} d_v$. A node v has a unique ID and a capacity value c_v. Although we can consider various attributes for c_v (e.g., CPU speed, storage space), we focus on the battery capacity in this work.[2]

[2]For the ease of exposition, we assume distinct capacity values throughout this chapter. In practice, we use unique IDs to break ties.



(a) Leader nomination (b) Resulting fragments

Figure 6.1: Leader nomination and resulting fragments.

6.1.1 Algorithm Description

We assume that each node knows the capacity value of its neighbors. The algorithm proceeds as follows: each node nominates the node with highest capacity value in $N^+(v)$ as leader. Each node then informs its leader of its decision, and all nominated nodes constitute the set of leaders, which we denote by $\mathcal{L}$. For example, in Figure 6.1(a), the network has nine nodes. The number in each circle denotes the node capacity (e.g., $c_A = 6$). Thin lines between nodes represent wireless links, while thick lines with arrows represent leader nomination. In the figure, G nominates D because $c_D = 5$ is higher than $c_H = 2$ and $c_G = 4$. As a result, nodes A, C, D, F, and J become leaders, as shown in Figure 6.1(b). (Nodes A and F nominate themselves as leader, which we do not show here.)

The above algorithm requires only one-hop neighborhood information and constant time. A similar clustering scheme is proposed in [87]. Gao et al. [88] analyze the size of resulting set using specific geometrical properties. However, their analysis assumes that

all nodes have a square-shaped communication region of the same size, which is seldom the case in practice [64]. We next present new analysis results, based on more realistic assumptions.

6.1.2 Properties of the Leader Set $\mathcal{L}$

We show that (1) $\mathcal{L}$ forms a dominating set using high-capacity nodes, and (2) the cardinality of $\mathcal{L}$ is small under reasonable assumptions. Recall that a *dominating set DS* of $G = (V, E)$ is a subset of V, where each node in V either is in *DS* or has a neighbor in *DS* [40]. If all nodes in *DS* are connected, then it is called a *connected dominating set (CDS)*. A *minimum* (connected) dominating set is of smallest cardinality among all (connected) dominating sets. We define a *maximum-capacity* (connected) dominating set DS_M to be a (connected) dominating set that maximizes the *bottleneck* node capacity. Formally, DS_M satisfies:

$$\forall DS, \min_{v \in DS_M} c_v \geq \min_{u \in DS} c_u, \tag{6.1}$$

where *DS* denotes a (connected) dominating set.

Theorem 6.1.1 *$\mathcal{L}$ is a maximum-capacity dominating set.*

Proof: $\mathcal{L}$ is a dominating set by construction. We prove the maximum-capacity property by contradiction. Assume that $\mathcal{L}$ is not a maximum-capacity dominating set. Consider a maximum-capacity dominating set DS_M. Then, the minimum-capacity node $v \in \mathcal{L}$ satisfies the following: $\forall u \in DS_M, c_v < c_u$. By the leader nomination rule, there exists a node w for which v is the maximum-capacity node in $N^+(w)$. However, DS_M also has a node in $N^+(w)$, which is contradiction. ■

We now show that the expected size of $\mathcal{L}$ (denoted by $E[|\mathcal{L}|]$) is small. For the sake of simpler analysis, we first consider the case of D-regular graphs (i.e., $\forall v,\ d_v = D$) and analyze a more generalized case later in this section. In this analysis, we assume c_v is uniformly distributed between 0 and 1, and $\log(\cdot)$ denotes the natural logarithm.[3] (The proof of Theorem 6.1.2 is in Appendix A.1.)

Theorem 6.1.2 *Suppose $\forall v, d_v = D$ for a positive integer D. Then, there exists a constant $\epsilon > 0$ such that:*

$$E[|\mathcal{L}|] \leq (1+\epsilon)\frac{n}{D}\log(D+1). \tag{6.2}$$

Also, ϵ approaches 0 as D increases.

In practice, wireless nodes are likely to have different numbers of neighbors, and Theorem 6.1.2 does not hold in general. However, due to spatial locality in the node distribution, we expect that neighboring nodes in multihop wireless networks have a similar number of neighbors. Formally, for a constant $\alpha \geq 1$, we define $G = (V, E)$ to be α-*locally-regular* if $\forall (u,v) \in E,\ d_v \leq \alpha\, d_u$. In a 3-locally-regular graph, for example, the degree of v's neighbor is between $d_v/3$ and $3\, d_v$.

We now generalize Theorem 6.1.2 to show that in α-locally-regular graphs, $E[|\mathcal{L}|]$ is within an $O(\log \Delta)$-factor of the size of a minimum dominating set. (The proofs are in Appendix A.)

Lemma 6.1.3 *Suppose $G = (V, E)$ is α-locally-regular for constant $\alpha \geq 1$. Then, $E[|\mathcal{L}|] \leq c' \sum_{v \in V} \frac{1}{d_v}\log(d_v + 1)$, where c' is a constant that depends on α.*

[3]Our current analysis assumes a uniform distribution of node capacity, and we plan to examine other distributions in the future.

Theorem 6.1.4 *Suppose $G = (V, E)$ is α-locally-regular for constant $\alpha \geq 1$. Then, $E[|\mathcal{L}|] = O(\log \Delta)$ OPT, where OPT is the size of a minimum dominating set.*

Discussion Note that Theorems 6.1.2 and 6.1.4 are essentially best possible. Theorem 6.1.2 holds for any D-*regular* graph, and as shown in [89], there exist D-regular graphs whose minimum dominating sets are of size at least $(1 - \epsilon')\frac{n}{D} \log D$ for $\epsilon' > 0$. As D becomes large, this value becomes arbitrarily close to the upper bound in Theorem 6.1.2. Also, the *approximation ratio* of Theorem 6.1.4 to *OPT* is $O(\log \Delta)$. In general, finding a minimum dominating set for a given graph is NP-hard [40]. Furthermore, no polynomial time algorithm can achieve the approximation ratio of $(1 - \epsilon') \log \Delta$ for any $\epsilon' > 0$ unless NP has $n^{O(\log \log n)}$-time deterministic algorithms [90]. Thus, the bound in Theorem 6.1.4 is within a constant factor of best possible approximation.

6.2 Connecting the Leaders

In this section, we present the second phase of our algorithm that connects the leaders to construct a connected dominating set. We first describe how to represent the leader set using a multigraph before the algorithm description.

6.2.1 Multigraph Representation

The set of leaders form a forest in which edges are leader nomination relations. We refer to each tree in this forest as a *fragment*. For example, in Figure 6.1(b), there are two fragments: fragment X (nodes A and D) and fragment Y (nodes C, F, and J). Since $\mathcal{L}$ is a dominating set, as shown in [40], chains of up to two non-leader nodes are suffi-

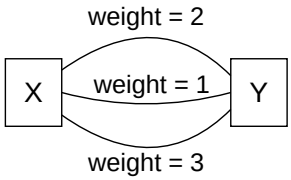


Figure 6.2: Multigraph representation of Figure 6.1(b).

cient to connect all fragments. We define a *virtual edge* to be such a chain of (up to two) non-leader nodes that connects two fragments. We transform the graph into a multigraph, where each fragment corresponds to a vertex with (possibly multiple) virtual edges connecting fragments. For a given virtual edge, we use the minimum node capacity as the weight of the edge.[4] In Figure 6.1(b), there are three virtual edges between fragments X and Y. The first one connects the fragments using nodes G and H, the second one uses nodes E and B, and the last one uses node B only. Since we use the minimum node capacity as virtual edge weight, the weights of these three edges are 2, 1, and 3, respectively. Figure 6.2 shows the corresponding multigraph representation. We next describe how we use this multigraph to find a connected backbone.

6.2.2 Spanning Tree-Based Algorithm

We begin with an approach based on the well-studied minimum spanning tree (MST) problem. This MST-based approach is a special case of our parameterized algorithm (Section 6.2.3). Recall that an MST of edge-weighted graph $G = (V, E)$ connects all nodes in V using a tree $T \subseteq E$, such that the sum of edge weights in T is

[4]It is possible that no nodes are involved in a virtual edge. In this case, we set the weight of the virtual edge to ∞.

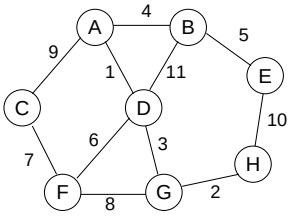


Figure 6.3: Example graph.

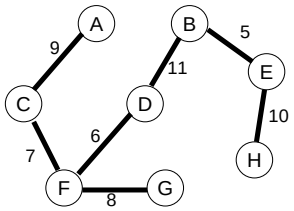
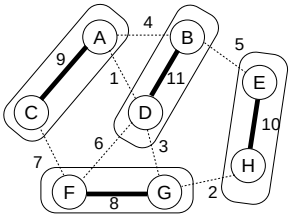
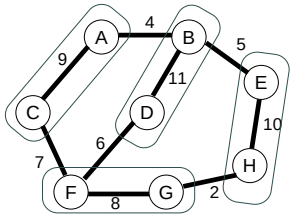


Figure 6.4: MST-based backbone



(a) After first round



(b) Resulting backbone ($\mathcal{B}_1$)

Figure 6.5: Illustration of truncated algorithm.

minimized [91].

In many algorithms that find MSTs, nodes select a minimum outgoing edge that does not result in a loop [91, 92]. However, since we want to select high-capacity nodes in the backbone, we need to use *maximum-weight* outgoing virtual edges. For example, in Figure 6.2, when connecting fragments X and Y to obtain high-capacity connected backbone, we should use the virtual edge of weight 3. We further illustrate this approach using an example graph in Figure 6.3. In this figure, each node corresponds to a fragment after the leader nomination phase, and each fragment is connected by virtual edges. (We show only the maximum-weight virtual edges between fragments for clarity.) Figure 6.4 shows the MST (as defined above).

Let $\mathcal{B}_{\text{MST}}$ denote the connected backbone obtained by using an MST algorithm. We next show that $\mathcal{B}_{\text{MST}}$ produces a *small* connected backbone using *high-capacity* nodes.

Theorem 6.2.1 *$\mathcal{B}_{\text{MST}}$ is a maximum-capacity connected dominating set.*

Proof: Please see Appendix A.4. ■

Lemma 6.2.2 *$|\mathcal{B}_{\text{MST}}| \leq 3|\mathcal{L}|$, where $\mathcal{L}$ denotes the leader set.*

Proof: Suppose that $\mathcal{L}$ initially consists of f fragments. We need to use $(f-1)$ virtual edges to find a spanning tree. Since there are at most two nodes in each virtual edge and $f \leq |\mathcal{L}|$, $|\mathcal{B}_{\text{MST}}| \leq |\mathcal{L}| + 2(f-1) \leq 3|\mathcal{L}|$. ■

Theorem 6.2.3 *For α-locally-regular graphs, $E[|\mathcal{B}_{\text{MST}}|] = O(\log \Delta)$ OPT, where OPT denotes the size of minimum connected dominating set.*

Proof: This follows from Theorem 6.1.4 and Lemma 6.2.2. ■

Discussion Theorem 6.2.1 states that $\mathcal{B}_{\text{MST}}$ includes a node with low capacity only when it is necessary in maintaining connectivity. We show in Theorem 6.2.3 that for α-locally-regular graphs, $\mathcal{B}_{\text{MST}}$ is an $O(\log \Delta)$-approximation to a minimum connected dominating set. As discussed in Section 6.1, this is within a constant factor of best possible approximation. However, if desired, we can further reduce the constant factor of approximation ratio by slightly modifying an existing distributed CDS algorithm, which we describe in detail in [70].

Although an MST-based approach achieves our desired goals (i.e., finding a small backbone using high-capacity nodes), the running time can be long. For example, a

Algorithm 1 Description of Truncated Algorithm (Centralized)

```
1: Round ← 0
2: while more than one fragment exists do
3:     if Round = K then
4:         Merge with all neighboring fragments
5:         Return
6:     end if
7:     Each fragment selects the best outgoing edge
8:     Merge fragments using the selected edges
9:     Round ← Round + 1
10: end while
```

distributed MST algorithm by Gallager, Humblet, and Spira (the GHS algorithm) takes $O(n \log n)$ running time [92]. Also, there is a clear trade-off between small backbones and shorter path lengths as well as resilience. In Figure 6.4, the backbone becomes disconnected even when a single link fails. Also, to reach a node in fragment G, a node in fragment H needs to use a path consisting of five virtual edges, compared to only one when no backbone is used. We address this issue next.

6.2.3 TRUNC-K: Our Parameterized Algorithm

We now describe our generalized scheme that balances the above-mentioned trade-off when connecting the leader set (Algorithm 1). It is based on a well-known MST algorithm by Boruvka [93]. In Boruvka's algorithm, each fragment finds and marks the best outgoing edge. Then, using those edges, fragments are merged into new larger fragments.

This step is repeated until there is no outgoing edge (i.e., there is only one fragment). During the first K rounds, our algorithm runs just as Boruvka's algorithm, where K is an algorithm parameter. However, in our truncated algorithm, all remaining fragments after K rounds mark edges to *all* neighboring fragments and are merged into one fragment. One extreme case is $K = 0$, where after leader nomination, each pair of neighboring fragments marks one virtual edge (e.g., all edges shown in Figure 6.3). Another extreme case is when $K = \infty$, which results in $\mathcal{B}_{\text{MST}}$.

Figure 6.5 shows the operations of the algorithm applied to the graph in Figure 6.3. Here, we set $K = 1$. In the first round, each individual fragment selects the best outgoing edge among neighboring fragments, and fragments are merged using selected edges. Then, as shown in Figure 6.5(a), there remain four fragments at the end of first round. Since $K = 1$ in this example, each remaining fragment after the first round connects to all neighboring fragments. For example, fragment FG chooses three edges to fragments AC, BD, and EH. The resulting connected backbone is shown in Figure 6.5(b).

We call this algorithm TRUNC-K and the resulting backbone $\mathcal{B}_K$. In contrast to $O(\log n)$ rounds in Boruvka's algorithm, TRUNC-K needs only a constant number (K) of rounds to complete, and the resulting backbone has higher redundancy than $\mathcal{B}_{\text{MST}}$. This eventually leads to both increased resilience against node mobility and decreased average path length. Note that the resulting backbone is not a maximum-capacity backbone and may include low-capacity nodes. However, by construction, nodes included in the first K rounds are part of a maximum-capacity backbone. After the K-th round, when connecting to each of remaining neighboring fragments, we choose the best virtual edge among typically multiple edges, and we are likely to include relatively high-capacity nodes. Also,

the resulting backbone includes more virtual edges than $\mathcal{B}_{\text{MST}}$, and Theorem 6.2.3 does not hold. However, we can adjust K to control the amount of increase. Our future goal is to analyze the performance trade-offs (e.g., backbone size, capacity distribution) when we vary K. We next use simulation experiments to illustrate that even with small values of K, the increase in backbone size is not significant.

6.2.4 Evaluation of the TRUNC-K Algorithm

In this subsection, we use simulations to understand performance trade-offs of the TRUNC-K algorithm (e.g., backbone size, capacity) when we use different values of the parameter K. In this simulation, stationary nodes are distributed on a square uniformly at random, and we vary the number of nodes and the size of square to experiment with various settings. Node capacity values are uniformly distributed between 0 and 1.[5] Nodes within the nominal transmission range (250 meters) become neighbors. For each set of parameters, we use 25 runs with different node placement scenarios and report the average.

6.2.4.1 Backbone Size

In Figure 6.6, we show the average size of the backbone with varying K. We use two different network settings—the one with 1000 nodes on a 2km$\times$2km square, and the other with 4000 nodes on a 4km$\times$4km square. In Figure 6.6, the use of extra rounds is most effective when K is small. Specifically, the first round ($K = 1$) leads to the

[5]For simulations in Section 6.2.4 and Section 6.4, we also experimented using various scenarios with non-uniform node placement and different capacity distribution, and obtained similar results.

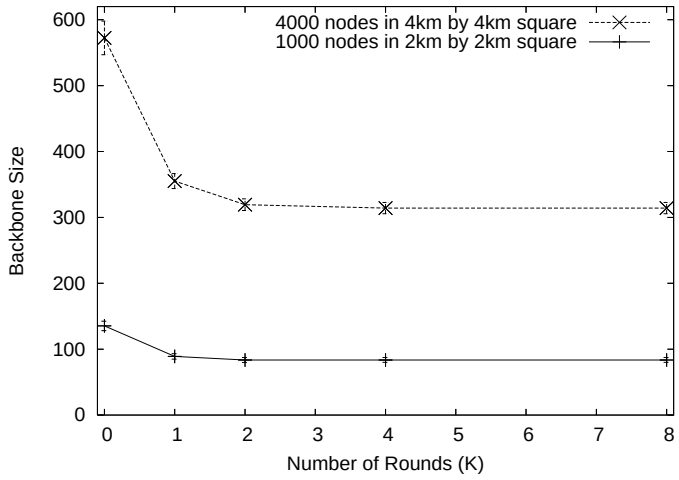


Figure 6.6: The size of the backbone with different K values. The error bars represent standard deviations.

largest reduction in backbone size. With larger $K > 2$, there are a small number of remaining fragments after K rounds. As a result, when compared to $\mathcal{B}_{\text{MST}}$, connecting all neighboring fragments does not significantly increase the backbone size.

6.2.4.2 Capacity Distribution among Backbone Nodes

We investigate another potential problem with TRUNC-K backbone—the resulting backbone may include low-capacity nodes. In this scenario, we use 1000 nodes but vary the square size, thus varying node density. In Table 6.1, we list the minimum and average capacity values depending on K values with different node density. Even with small K (1 or 2), the difference in minimum-capacity between $\mathcal{B}_K$ and $\mathcal{B}_{\text{MST}}$ is small. For example, in the case of 2km$\times$2km square, the difference between $\mathcal{B}_1$ and $\mathcal{B}_{\text{MST}}$ is around 10% (0.705 vs. 0.787). The difference in average capacity values is even smaller: less

	1.4 km×1.4 km		2.0×2.0		2.8×2.8		4.0×4.0	
	min.	avg.	min.	avg.	min.	avg.	min.	avg.
$\mathcal{B}_0$	0.349	0.913	0.111	0.854	0.021	0.771	0.007	0.666
$\mathcal{B}_1$	**0.860**	0.967	**0.705**	0.934	**0.307**	0.872	**0.054**	0.760
$\mathcal{B}_2$	0.888	0.970	0.787	0.942	0.573	0.892	0.188	0.796
$\mathcal{B}_{\mathrm{MST}}$	**0.888**	0.970	**0.787**	0.942	**0.574**	0.893	**0.200**	0.802

Table 6.1: Capacity values of backbone nodes with varying node density

than 1% for the same scenario. As discussed in 6.2.3, this is because fragments after K rounds choose best possible virtual edges to connect neighboring fragments, and very low-capacity nodes are not likely to join the backbone. However, in sparser networks, fewer virtual edges are available between neighboring fragments, and the difference in minimum capacity between $\mathcal{B}_K$ and $\mathcal{B}_{\mathrm{MST}}$ is slightly larger.[6] In the actual deployment of the TRUNC-K algorithm, we should choose an appropriate K value based on estimated node density and mobility.

6.2.4.3 Average Path Length

We consider the average shortest path length (induced by the backbone) between each of all possible node pairs. This measure provides a good lower bound for the performance of practical routing protocols such as [37]. In this chapter, we report results for the cases with 1000 nodes placed in a 4km×4km square only. When there is no backbone,

[6]When 5km×5km squares are used with 1000 nodes, only four cases out of 25 resulted in connected networks, and the use of 4km×4km squares with 1000 nodes corresponds to considerably sparse scenarios.

the average path length is 11.2. The use of any routing backbone inevitably increases path length since we are forced to find paths using a restricted set of nodes. $\mathcal{B}_0$ has most redundancy, and the average path length is 12.5 with minimum increase. As K increases, the redundancy decreases and the path length thus increases; the average path length of $\mathcal{B}_1$ is 14.8, while that of $\mathcal{B}_2$ is 18.8. In contrast, the average expansion in path length for $\mathcal{B}_{\text{MST}}$ is 23.0, which is more than twice the underlying shortest paths. We observe that small K values again offer a good trade-off. For example, compared to $\mathcal{B}_{\text{MST}}$, the average path length of $\mathcal{B}_1$ is up to 36% shorter, while the backbone size is only up to 13% larger.

To summarize, backbones obtained using small K (1 or 2) perform well and provide a reasonable balance among a number of performance measures. We next describe a distributed protocol that implements the TRUNC-K algorithm.

6.3 Distributed Protocol Description

In this section, we present our distributed protocol that implements the TRUNC-K algorithm to construct and maintain a connected backbone in *dynamic* network environments. Our protocol is based on the GHS algorithm, which is a distributed version of Boruvka's algorithm [92]. We assume that each node has a unique ID (e.g., IP address). We first describe the leader nomination and explain how to connect the fragments obtained after the nomination phase. We also present a backbone maintenance mechanism later in this section.

Field Name	Description
Node ID	node identifier
Capacity	current capacity of node
IsLeader	whether this node is a leader or not
Leader ID	highest-capacity neighbor
Level-0 fragment root	ID of level-0 fragment root
...	...
Level-K fragment root	ID of level-K fragment root

Table 6.2: Information about individual nodes in a HELLO message.

6.3.1 Leader Nomination Protocol

Each node broadcasts a HELLO message periodically that includes information about itself and its neighbors. Table 6.2 shows the fields for individual node information in HELLO messages. Using these fields, each node maintains information about two-hop neighbors (e.g., capacity, fragment root IDs).

Before broadcasting a HELLO message, node v checks which neighbor has the highest capacity (e.g., residual battery power). Suppose u is the highest-capacity neighbor of v. Then, v sets its *Leader ID* field to u in its HELLO message. Upon receiving a HELLO message from v, u becomes a leader and sets its *IsLeader* field to TRUE in subsequent HELLO messages until v changes leaders and there exist no other neighbors nominating u (e.g., due to later decrease in residual battery).

Suppose node u finds itself as the highest-capacity node in $N^+(u)$. Then, in addi-

Algorithm 2 Distributed operation of level-i fragments

1: Level-i fragment root periodically sends REQ_i message

2: Fragment members send REPLY_i messages with neighboring level-i fragment information

3: **if** level-i fragment root receives all REPLY_i messages, or a timeout occurs **then**

4: **if** i = K **then** {highest level}

5: Fragment root sends CONNECT_i messages to all neighboring level-i fragments

6: **else** $\{ i < K \}$

7: Fragment root sends CONNECT_i message only to best neighboring level-i fragment

8: **end if**

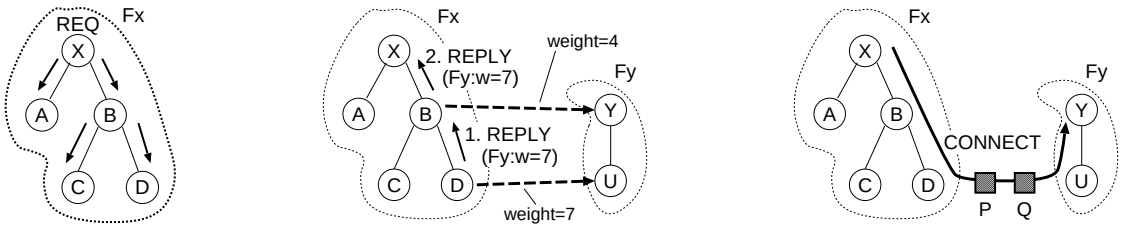
9: **end if**

tion to being a leader, u also becomes a *level-0 fragment root*, where a level-0 fragment is a set of leaders who are themselves connected via the leader-nomination relation. In Figure 6.1(b), nodes A and F are level-0 fragment roots.

6.3.2 Protocol for Fragment Members

As discussed in Section 6.2, the set of leaders form a forest consisting of multiple fragments, and the protocol described here merges the fragments to form one connected component. In Algorithm 2, we present a high-level protocol description. We begin with the operation of level-0 fragments and later discuss the operation of higher-level fragments.

We illustrate the protocol operations of level-0 fragments using Figure 6.7. Each



(a) REQ message

(b) REPLY message with outgoing virtual edges

(c) CONNECT message

Figure 6.7: Overview of protocol operations.

level-0 fragment forms a tree rooted at its fragment root. To discover neighboring level-0 fragments, level-0 fragment roots periodically send REQ_0 messages, which are forwarded down this tree to the leaves (who cannot forward the REQ message any further). The leaves then generate a REPLY_0 message that contains information about other fragments (if any) that they are connected to. The REPLY_0 messages are forwarded back towards the fragment root. For example, in Figure 6.7(b), node D generates a REPLY_0 message, which contains the ID of other level-0 fragments that D knows of (F_y in this example) along with the cost of the virtual edge to connect to F_y. (Recall that D keeps the information about fragment roots of two-hop neighbors.) At each hop, before forwarding the REPLY message towards the leader, nodes update the message if they know of a better virtual edge than the one carried in the message. Also, nodes add information about any new neighboring fragments that are not in the REPLY message. In our example, node B does not modify the REPLY message from D, since its path to F_y is worse than the one that D found (Figure 6.7(b)). (A and C also send REPLY_0 messages, which are not shown in the figure.)

Once the fragment root has accumulated all REPLY messages (or has timed out on some), it sends a CONNECT message using the best outgoing virtual edge. This is shown in Figure 6.7(c), where X connects to Y using the weight 7 edge through D. This virtual edge has two non-backbone nodes P and Q, and upon receiving the CONNECT_0 message, they become *bridges* and join the backbone. Using these bridge nodes, F_x and F_y are merged to form a new level-1 fragment.

Level-1 fragments also need to find neighboring level-1 fragments to form next-level fragments. To elect level-1 fragment roots that send REQ_1 messages, we use the following rule similar to [92]: If two fragments choose each other as their best neighboring fragment, then two fragment roots become candidates for the next-level fragment root. We choose the node with lower ID as the level-1 fragment root.

Level-i fragments ($0 < i < K$) operate similarly to above procedures until their level reaches K. At the highest level-K, instead of using only the best virtual edge, the level-K fragment roots send CONNECT messages to *all* neighboring level-K fragments, thus assuring a connected backbone.

In Figure 6.7, X is a both level-0 and level-1 fragment root, and it periodically sends both REQ_0 and REQ_1 messages. In general, a node can be a fragment root of up to (K+1) levels at the same time. Even if higher-level fragments are already found, lower-level fragment roots (e.g., Y in Figure 6.7) still send REQ_i messages periodically. This allows lower-level fragments to find new or better virtual edges to neighboring fragments in dynamic networks.

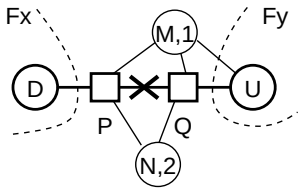


Figure 6.8: Local maintenance.

6.3.3 Backbone Maintenance

All of the protocol specific states (e.g., leaders, bridges, fragment roots) are "soft." A node removes a neighbor if it does not receive a HELLO message from the neighbor for a certain duration (e.g., four HELLO-PERIODS). If the capacity of the leader becomes lower (e.g., due to battery consumption), a node may choose a different node with highest capacity as leader. If a leader finds that no neighbors nominate it as leader for some time, it stops being a leader. When a bridge does not receive a CONNECT message for a certain period of time, it stops being a bridge.

In a dynamic network, however, the basic protocol mechanisms described above may not be sufficient for the timely maintenance of the connected backbone. We efficiently reconstruct the backbone using a simple local search protocol that exploits spatial locality. We illustrate its operation via an example. In Figure 6.8, node P detects a link failure to backbone neighbor Q. P looks up its neighbor table to find other nodes that also had Q as neighbor. (Note that these nodes need not currently be part of the backbone). In this example, P finds two such neighbors, M and N, and sends a RECOVER message to N, which has higher capacity. Upon receiving this message, N temporarily joins the

backbone and forms a bridge to Q. In the next REQ-REPLY phase, X might choose a different virtual edge (of higher weight) to connect to F_y. If that happens, N will leave the backbone after a timeout.

There are potential problems with the local recovery scheme. First, the repaired backbone may include lower-capacity nodes than necessary. However, as mentioned above, in the next REQ-REPLY phase, the fragment root will discover the best virtual edge and send the appropriate CONNECT message. Also, a node may not be able to find a common neighbor for recovery. However, in networks with reasonable node density, such events will likely be infrequent. Finally, the recovery scheme does not help when nodes fail. However, the TRUNC-K backbone should have sufficient resilience to maintain connectivity against infrequent recovery failures. We examine the effectiveness of this recovery scheme using simulations in Section 6.4.

6.3.3.1 Discussion

As shown in Table 6.2, HELLO messages in TRUNC-K contain (K+3) node IDs per neighbor: *Node ID, Leader ID*, and *fragment roots* for $K+1$ levels. For example, suppose that node U in Figure 6.7(c) is about to broadcast a HELLO message. When K=1, the information for neighbor Q should include (Q, U, Y, X). Including more information in HELLO messages increases the control overhead. However, many neighbors share leaders and fragment roots, and we can reduce the amount of increase by using a simple table-based indexing scheme. We describe this scheme in detail in Technical Report [70], and our simulation results in Section 6.4 show that the overall control overhead of TRUNC-K

is minimal.

Another issue is that wireless links in practice show a wide range of difference in their quality [64]. Thus, it is beneficial to use high-quality links when connecting backbone nodes. In the future, we plan to incorporate the link quality aspect into backbone construction and maintenance mechanisms. Also related is the existence of unidirectional links, which we discuss in more detail in Technical Report [70].

6.4 Simulation Study

In this section, we compare TRUNC-K with prior approaches using simulation experiments. Based on the results in Section 6.2.4, we consider only the case of $K = 1$. Although we performed experiments in other various scenarios using different topologies, traffic patterns, and capacity distributions, we present only a subset of representative results in this chapter. We first describe prior approaches and then compare the performance of our scheme against them.

6.4.1 Brief Description of Existing Schemes

In this section, we compare the performance of our algorithm with that of SPAN [19], GAF [20], and the scheme proposed by Wu et al. [43]. (We do not compare against other schemes that do not consider node capacity [38, 39, 41, 42].)

In SPAN [19], a node becomes a *coordinator* and joins the backbone when any two neighbors are not connected using up to two current coordinators. To minimize contention and give priority to high-energy nodes, SPAN uses a randomized backoff using the energy

level, number of neighbors, and a random number. A coordinator withdraws after some period of time to give other neighbors a chance to become coordinators.

In GAF [20], the area is divided into square-shaped virtual grids. GAF assumes the availability of location information (e.g., from GPS), and each node can know its virtual grid from the location information. Then, GAF elects the highest-energy node in each grid, and these elected nodes form a connected backbone due to the grid construction rule [20].

In the scheme by Wu et al. [43], a node initially joins the backbone if its two neighbors are not connected. Then, to reduce the size of this initial backbone, node v searches for a neighbor u, or two neighbors u and w, such that the (union of) neighbor set(s) includes the neighbor set of v. Due to symmetry, the above rule may lead to connectivity loss, and the authors of [43] also use the power level and degree of node to avoid the loss of connectivity.

6.4.2 Comparison Study in Large Networks

In this set of experiments, we use the same settings as in Section 6.2.4. We consider capacity values in $[0, 1]$; in GAF, the side length of the square grid is set to 100 meters (which is the value the authors of GAF use [20]). We measure the performance when the initial backbone stabilizes, and report the average of 25 runs each.

We first examine the size of backbones constructed by different schemes. In this set of experiments, we vary the number of nodes and the size of square, but maintain the average node degree constant. In Table 6.3, we present the average backbone sizes for

No. of nodes	500	1000	2000	4000
Square size (km×km)	(1.4×1.4)	(2.0×2.0)	(2.8×2.8)	(4.0.×4.0)
SPAN	54.5	113.9	227.7	467.4
Wu et al.	67.4	150.3	308.9	652.5
GAF	158.0	308.6	605.6	1236.9
TRUNC-1	**44.7**	**89.0**	**174.6**	**355.1**

Table 6.3: Backbone size constructed by different schemes.

various scenarios. The standard deviations are small (less than 6% of the average in all cases), which we do not present here. We observe that TRUNC-1 backbones are smallest in all cases. Specifically, when the network has 4000 nodes, the TRUNC-1 backbone has 355 nodes on average. This is 24% smaller than the SPAN backbone, which is the second smallest in all these experiments.

Our proposed scheme also builds a backbone consisting of higher-capacity nodes. In Table 6.4, we tabulate the minimum and average capacity values of backbone nodes. In all cases, the backbone by TRUNC-1 achieves the highest values for both minimum and average node capacity. For example, in 4000-node networks, the TRUNC-1 backbone does not include any of bottom 30% nodes, while the GAF backbone includes some of bottom 0.5% nodes. In the same scenario, the average capacity of TRUNC-1 backbone is also 30% higher than those of SPAN and GAF. When the routing backbone is used to reduce power consumption and increase the network lifetime, the use of low-capacity nodes can drain their energy unnecessarily. We later investigate this aspect using packet-level simulations.

	No. of Nodes (square size in km×km)							
	500 (1.4×1.4)		1000 (2.0×2.0)		2000 (2.8×2.8)		4000 (4.0.×4.0)	
	min.	avg.	min.	avg.	min.	avg.	min.	avg.
SPAN	0.056	0.700	0.046	0.686	0.025	0.704	0.011	0.708
Wu et al.	0.005	0.504	0.002	0.480	0.002	0.495	0.002	0.504
GAF	0.032	0.714	0.014	0.707	0.007	0.720	0.003	0.723
TRUNC-1	**0.752**	**0.937**	**0.705**	**0.934**	**0.502**	**0.933**	**0.335**	**0.933**

Table 6.4: Capacity value of backbone nodes by each scheme

In Figure 6.9, we present a detailed snapshot of a representative run with 1000 nodes. We sort all nodes in an ascending order of capacity value and cumulatively plot the number of backbone nodes whose capacity is less than or equal to that of a given node. For example, the GAF backbone includes 49 nodes out of 500 lowest-capacity nodes, while SPAN chooses 19 nodes from the lowest 500 nodes. In contrast, the TRUNC-1 backbone does not include any of the lowest-capacity nodes, but selects only 93 nodes among the top 330 nodes.

In Table 6.5, we report the average path lengths by different schemes as well as the case using no backbones. Not surprisingly, since TRUNC-1 backbones are smaller in size than any other scheme (Table 6.3), its average path lengths are the longest. However, the amount of reduction in backbone size is more than the increase in the path length, especially in larger networks. Specifically, in 4000-node networks, the difference in the average path length between SPAN and TRUNC-1 is around 20%, while the difference in

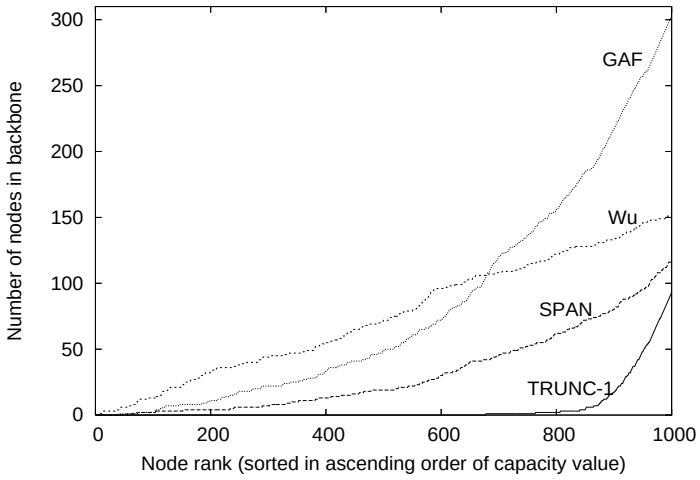


Figure 6.9: The capacity distribution of backbone nodes in different schemes.

backbone size is more than 30%. The other two schemes (GAF and Wu et al.) have shorter path lengths on average, but their backbones are substantially larger in size (Table 6.3). This result illustrates that TRUNC-1 backbones provide relatively good paths considering the small size.

6.4.3 Packet-level Simulations

In this subsection, we focus on saving energy and extending network lifetime using ns-2 simulations [94]. SPAN and TRUNC-1 performed best in Section 6.4.2, and we compare only these two schemes here. We use the SPAN simulation code written by the authors of SPAN.[7]. Due to high resource requirements, we have been able to perform simulations only with relatively small topologies (with 150 nodes). We first describe our simulation environment before reporting the results.

[7]Available at http://www.pdos.lcs.mit.edu/span/.

	No. of nodes (square size in km×km)			
	500	1000	2000	4000
	(1.4×1.4)	(2.0×2.0)	(2.8×2.8)	(4.0.×4.0)
No backbone	3.66	5.04	6.86	9.60
SPAN	4.39	6.17	8.44	11.89
Wu et al.	4.14	5.75	7.86	11.01
GAF	3.93	5.45	7.45	10.46
TRUNC-1	5.66	7.84	10.27	14.35

Table 6.5: Average path length by different schemes.

6.4.3.1 Simulation Environment

Both TRUNC-K and SPAN run on top of the IEEE 802.11 MAC layer [66], and non-backbone nodes stay in the power saving mode. In the IEEE 802.11 power saving mode, time is partitioned into *beacon periods*. All nodes stay awake in the beginning of each beacon period and exchange messages to inform neighbors of pending messages. If a node finds that there are buffered incoming messages, it requests the messages and stays awake during the beacon period. Otherwise, it goes back to sleep until the start of the next beacon period. Power saving mode usually leads to increased delay and reduced throughput (e.g., due to additional control packets), and Chen et al. [19] slightly modified the power saving mode in the 802.11 MAC to improve performance, which we use in our simulations.

We assume there are three classes for the node energy level. A low-energy node

has 300J of energy, which is used in the experiments in [19]. A medium-energy node has 600J, and a high-energy node has 2500J. (2500J is usually sufficient to last 3000 seconds of simulation time.) We vary the node percentage of each class to examine the performance in different settings. Node energy is constantly updated using the following power consumption values reported in [19]: 1.4W for transmission, 1.0W for receiving, 0.83W for idling, and 0.13W for sleeping. We place 150 nodes uniformly at random on a 1000 meter by 1000 meter square area. The transmission range of each mobile node is 250 meters.

We choose 10 pairs of source and destination nodes uniformly at random among high-energy nodes; each source generates traffic 50 seconds into the simulation at the constant rate of one 128-byte packet per second. The MAC-level transmission rate is 2 Mbps. As we discuss later, SPAN rotates backbone nodes frequently (e.g., 2 changes per second), which shortens path lifetimes. When we used on-demand routing protocols over SPAN, the path maintenance overhead was high. Instead, we use an idealized scheme for packet routing, where a path is found on top of the connected backbone using the centralized Floyd-Warshall algorithm [91] implemented in ns-2. This corresponds to a best case scenario for SPAN. Nodes move according to the Random Waypoint mobility model (pause time=400s, and maximum speed is 1–16m/s) [94]. We also set the minimum speed to be 0.1m/s to avoid speed decay [95]. Unless otherwise stated, we use mobile scenarios with the maximum speed of 1m/s.

In both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. For TRUNC-1, we set the period of REQ messages to 14 seconds, which leads to reasonable performance. In each case, we report the average of 5 runs.

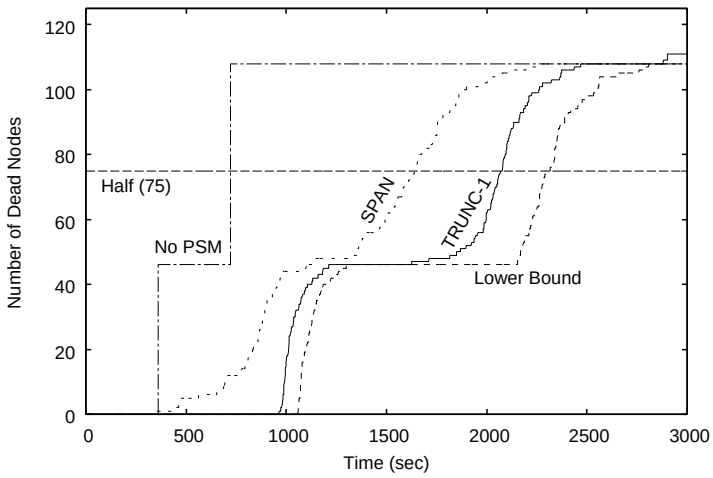


Figure 6.10: Number of dead nodes over time. L:M:H=3:4:3.

6.4.3.2 Simulation Results

For the first set of results, we examine two types of network lifetimes [96]. Network *1-life* is the time when the first node dies, and *half-life* is the time when the half of initial nodes die.[8] In addition to TRUNC-1 and SPAN, we use two other schemes for reference. The first one is to identify a lower bound, in which all nodes always stay in sleep mode except when they wake up at the beginning of beacon periods. Each node also sends a 128-byte HELLO message every two seconds. In the second scheme (No-PSM), no power saving operation is used, and all nodes always stay awake without sending any control messages. We send no data traffic in either of the two reference cases.

In Figure 6.10 we present a snapshot for the number of dead nodes over time. In this setting, approximately 30% of nodes are low-energy (L), 40% of them are medium-

[8]We assume that the network needs external support after this time (e.g., addition of fresh nodes in the case of sensor networks).

energy (M), and the rest 30% are high-energy (H) nodes. (To denote this ratio, we use an abbreviated notation L:M:H=3:4:3.) In Figure 6.10, the network 1-life of SPAN is similar to that of "No-PSM." This is expected from Figure 6.9 to some extent; SPAN includes low-energy nodes in the backbone, and their lifetime decreases significantly. Although SPAN rotates the backbone node responsibility, there exists an unfortunate low-energy node in most of our experiments that stays in the backbone during the first 350 seconds. In contrast, with TRUNC-1, the network 1-life is close to 960 seconds, which is 2.7 times longer than that of SPAN. This is because the TRUNC-1 backbone consists mostly of high-energy nodes plus a few medium-energy nodes, and low-energy nodes can stay in sleep mode and save energy. In the case of TRUNC-1, we observe a sharp increase in the number of dead nodes as the first node dies. This is the time (960 seconds) when all low-energy nodes in TRUNC-1 run out of power. Note that this is earlier than the case of lower-bound (around 1050 seconds). This is because with TRUNC-1, nodes consume more energy to exchange larger HELLO messages (in this experiment around 211 bytes on average) than the lower-bound case (128 bytes). Compared to SPAN, TRUNC-1 also increases the average lifetime of low-energy nodes by 28% (1038.1 seconds vs. 811.3 seconds).

We now consider the lifetime of medium-energy nodes in Figure 6.10. The use of low-energy backbone nodes in SPAN allows more medium-energy nodes to be in sleep mode and potentially increase their lifetime. Still, compared to SPAN, the TRUNC-1 backbone increases the network half-life by around 26%. We explain this as follows. In this network setting, a connected backbone needs to use several medium-energy nodes to maintain connectivity. Ideally, as the initial medium-energy backbone nodes expend

Ratio of	1-Life (sec)		Half-Life (sec)	
L:M:H	TRUNC-1	SPAN	TRUNC-1	SPAN
4:4:2	892.1	365.5	1911.0	1506.3
	(101.0)	(28.3)	(139.6)	(54.5)
3:4:3	946.6	375.0	2106.2	1689.8
	(22.0)	(29.1)	(33.7)	(40.5)
2:4:4	962.0	412.3	2208.3	1842.3
	(13.7)	(54.7)	(41.1)	(43.9)

Table 6.6: Network lifetimes when the proportion of nodes at different energy levels is varied. The values in parentheses are standard deviations.

their energy, they should be replaced with different medium-energy nodes, such that their lifetime does not decrease significantly. From Figure 6.10, we infer that TRUNC-1 evenly distributes the backbone responsibility among all medium-energy nodes, and no medium-energy nodes die until 1600 seconds. (In Figure 6.10, after all low-energy nodes die in the case of TRUNC-1 backbone, we observe a relatively stable period during which no node dies.) In contrast, in SPAN, medium-energy nodes start to die before 1200 seconds, and the network half-life of SPAN decreased.

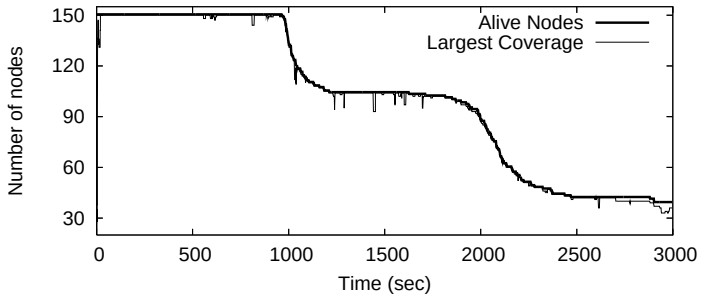
In Table 6.6, we tabulate the average network lifetimes and standard deviations while varying the proportion of nodes at different energy levels. We observe that in all scenarios, TRUNC-1 achieves longer network lifetimes than SPAN (133–152% longer for 1-life and 20–26% longer for half-life). We also experimented using different parameters (e.g., different initial battery capacity values and L:M:H ratios), and TRUNC-1

outperformed SPAN in all cases. In all these experiments, the average backbone sizes of TRUNC-1 and SPAN are very similar (between 21 and 23 nodes depending on the scenarios).

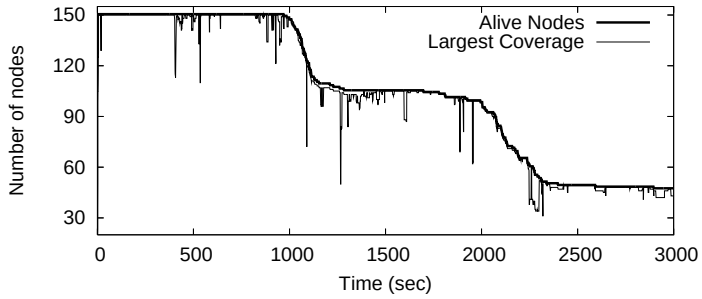
The previous results are based on the energy consumption values reported in [19]. We briefly report results that we obtained using different sets of energy consumption values in [18] and [20]. In general, as nodes in sleep mode consume less energy, TRUNC-1 achieves larger network lifetime extension over SPAN. Specifically, when using values in [20], idle nodes consume 40 times more energy than nodes in sleep mode, and TRUNC-1 achieves 220% increase in 1-life and 43% increase in half-life over SPAN. (In Table 6.6, the respective increases are 152% and 24%.)

Backbone Maintenance

We now examine the backbone resilience as well as the effectiveness of our proposed maintenance mechanisms against backbone partition. Let us define the *coverage* of a connected backbone to be the number of nodes that are in the backbone or have a neighbor in the backbone. For an ideal connected backbone, its coverage would always be equal to the number of nodes alive in the network. In Figure 6.11, we show the largest coverage of the TRUNC-1 backbone over time, while we change the maximum speed (1m/s and 16m/s). Due to node mobility or energy-level change, the backbone may get disconnected, and we see occasional drops in the coverage of TRUNC-1 backbone. However, our protocol detects such disconnections quickly, and the local maintenance scheme helps to regain the perfect coverage in a short period of time. As node mobility



(a) Maximum speed=1m/s. (TRUNC-1)

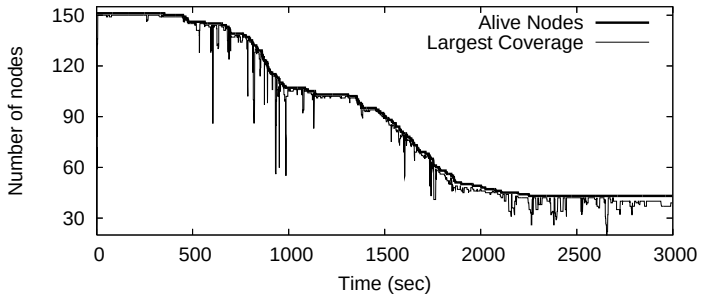


(b) Maximum speed=16m/s. (TRUNC-1)

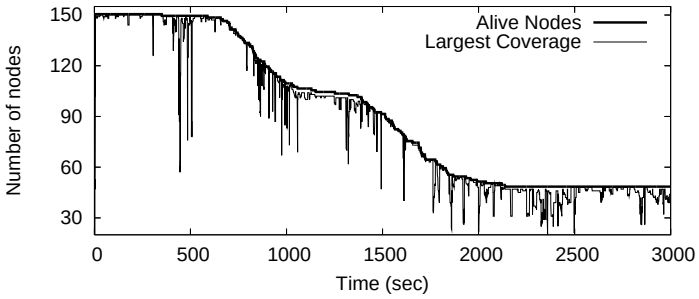
Figure 6.11: TRUNC-1 backbone coverage.

becomes higher, we observe modest increase in the number of partitions in the TRUNC-1 backbone. Compared to the TRUNC-1 backbone, the SPAN backbone results in more frequent coverage loss (Figure 6.12). In SPAN, nodes periodically leave the backbone only after ensuring that the departure does not cause backbone disconnection. However, it is possible that a node makes such a decision based on outdated information (e.g., due to node mobility), which occurs frequently, for example, once every 30 seconds on average in Figure 6.12(a).

Another aspect of backbone maintenance is the frequency with which nodes in the backbone change. In Figure 6.12(a), the SPAN backbone undergoes 674 membership



(a) Maximum speed=1m/s. (SPAN)



(b) Maximum speed=16m/s. (SPAN)

Figure 6.12: SPAN backbone coverage.

changes between 100 and 400 seconds. This is because backbone nodes in SPAN periodically leave the backbone. In the same scenario, TRUNC-1 causes 49 changes in the backbone membership. Suppose that an on-demand routing protocol such as DSR [37] found a path using backbone nodes. With SPAN, nodes on such a path are likely to leave the backbone about 12 times more frequently than TRUNC-1, and the source may need to find a new path consisting of backbone nodes frequently. (Recall that this is why we use the idealized routing in our experiments.)

	1 packet/sec	2 packets/sec	4 packets/sec
SPAN	0.96 (0.04)	0.88 (0.05)	0.62 (0.02)
TRUNC-1	0.98 (0.02)	0.92 (0.05)	0.58 (0.04)

Table 6.7: Average delivery ratio with varying traffic load. The values in parentheses are standard deviations.

Data Delivery

We briefly report the results about data delivery performance of TRUNC-1 backbone. In the previous light-load experiments, both TRUNC-1 and SPAN achieve near-perfect data delivery ratios. In this set of experiments we consider high-load scenarios using 1024-byte data packets with varying packet rates. We also use static networks and ensure the distance between source and destination is more than 500 meters such that all data packets go through at least two intermediate hops. In Table 6.7, we tabulate the average data delivery ratios and standard deviations with different sending rates. We observe that as the amount of data traffic increases, the average delivery ratio decreases in both schemes, and the difference between TRUNC-1 and SPAN is marginal. In these experiments, TRUNC-1 leads to shorter average end-to-end delays than SPAN, but the difference is not significant.

Control Overhead

In both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. HELLO messages in TRUNC-1 contain more information, and the average message

Max. speed	REQ	REPLY	CONNECT	RECOVER	HELLO
1m/s	1.80	2.15	1.64	0.12	52.73
8m/s	2.02	2.29	1.94	0.34	53.43
16m/s	1.83	2.04	2.14	0.49	53.65

Table 6.8: Average number of control packets per second in the entire 150-node network.

is longer than that of SPAN. Specifically, in TRUNC-1, the average length of HELLO messages is around 192 bytes, and in SPAN it is around 131 bytes. Note that the difference is due in part to more dead nodes in SPAN, which lead to fewer neighbors in HELLO messages.

TRUNC-1 uses additional control messages (e.g., REQ and CONNECT), and in Table 6.8 we tabulate the average numbers of those control packets per second used in the *entire* network. As shown in the table, the total number of non-HELLO control packets is only around 6 packets per second in the 150-node network, and their average sizes are 20 to 70 bytes. When each of 150 nodes sends a HELLO message every two seconds, the expected number is 75 per second. In Table 6.8, however, due to dead nodes, the number of HELLO messages is around 30% smaller. We also observe that the overall increase in control overhead due to higher mobility is marginal. We believe that the advantages of TRUNC-1 (e.g., longer network lifetime, better backbone coverage) outweigh the modest increase in control overhead.

6.5 Summary and Future Work

We have presented a parameterized scheme TRUNC-K that builds a connected backbone in multihop wireless networks. We have proved that our scheme can construct essentially best possible backbones with respect to backbone size and node capacity. We have generalized our scheme to construct and maintain a resilient backbone in dynamic networks. Through detailed simulations, we have demonstrated that our proposed scheme outperforms existing energy-saving techniques in many aspects.

In the future, we plan to investigate how to adjust the K value according to network environments (e.g., node mobility or density). Then, we will be able to include adaptive protocol mechanisms that can automatically change the K value when network parameters change over time (e.g., increased mobility or new node deployment). We also want to analytically investigate the backbone performance with different K values. As discussed earlier, another obvious extension to the current scheme is to consider the difference in link quality [64], such that we simultaneously consider node capacity and link quality in nominating leaders and connecting fragments.

Chapter 7

Backbone Construction in Selfi sh Settings

Most prior work for multihop wireless networks has assumed either (1) nodes in the network are inherently altruistic or cooperative, or (2) external mechanisms such as secure hardware or a central bank can be used to enforce cooperation. In this chapter, we consider a network in which participants have data to send (or receive) but are selfish—they are not inclined to relay packets for others. Thus, nodes in our system do not *want* to join the backbone (since they do not want to relay packets for others), but do want a backbone to exist (since they want their own packets to be forwarded). Under this assumption, we use tools from game theory and present a mechanism for backbone construction, without resorting to external mechanisms for enforcing cooperation.

We model the problem of building a backbone as that of creating a *public good*: a commodity from which all nodes benefit, but which they must collectively provide. We apply a well-known game-theoretic model called the *Volunteer's Dilemma* [97–99]. Each participant in the network needs *some* of the nodes to volunteer to provide the public good (i.e., backbone), but no one wants to be one of the volunteers. We extend the base model, and show how to apply it in the wireless network setting. Nodes compute an amount of time to *wait* before they volunteer to join the backbone, and we derive a dominant strategy that considers the node's remaining battery and its local neighborhood to compute this waiting time. The resultant protocol retains the goodness of cooperative backbone

construction algorithms: the backbone size is small, and nodes with greater capacity are preferentially added to the backbone.

We make the following contributions:

- We generalize Volunteer's Dilemma in Section 7.2, and use it to compute the optimal waiting time before a node volunteers. In Section 7.3, we present a protocol that uses these computed times to form connected backbones.

- We evaluate our protocol using extensive simulations in Section 7.5. Our results show that a backbone connects quickly and consists predominately of nodes with higher capacity, allowing low-capacity nodes to save their battery by sleeping.

- We present our implementation experience using real hardware and experiment results on a testbed. This is the first to evaluate the network performance when we use routing backbones in real wireless systems. Our results show that with a routing backbone, we can achieve a similar level of network performance (e.g., end-to-end throughput) when compared to cases without a backbone.

One potential issue after backbone construction is that a backbone node may refuse to forward messages as promised. We address this type of misbehavior in Section 7.4. We begin with a brief review of our network model and assumptions.

7.1 Model and Assumptions

We model the network as an undirected graph $G = (V, E)$, where V is the set of nodes in the network, and E is the set of bi-directional edges. As in previous protocols for

detecting neighborhood transmissions [54, 55], we assume that whenever a node u sends a packet, it is received by each node in its 1-hop neighborhood, $\mathcal{N}^1(u)$, as well as the packet's intended recipient.

Node Utility and Capacity We assume the primary concern of each node in our model is to ensure that its connections have high goodput. Since we do not consider any external incentives, if a node knows that it will not be sending or receiving packets for a significant amount of time, we cannot motivate the node to route and forward packets on behalf of other nodes. We assume that nodes do not collude, which is a standard assumption made in most game-theoretic analysis.

We use each node's remaining battery to model its capacity. We do not define a specific utility function that a node tries to maximize. Instead, we use the following preference relation: for each node v, if there is any data to be sent or received, v first attempts to maximize its goodput. If there are multiple paths that yield equal goodput, then v uses the path that minimizes the energy utilized. When v has nothing to send or receive, it strictly tries to conserve its battery.

Traffic Patterns To model traffic behavior, we assume that connections are periodically made between random source-destination pairs. Although this is common knowledge to all nodes, nodes do not have any further knowledge about traffic patterns *a priori*. Since we assume all the nodes wish to maximize their goodput, this assumption implies that it is selfish nodes' best interest to maintain end-to-end connectivity all the time.

7.2 Backbone Formation: Theory

In this section, we develop the background for our backbone formation protocol. We begin with a review of the well-known Volunteer's Timing Dilemma [98] and then extend it to work in a generalized setting suitable to a multihop wireless network.

7.2.1 The Volunteer's Dilemma

Consider the following social dilemma: a group of rational individuals want a *single* person from the group to volunteer to offer some service. This service expends some of the volunteer's resources but, if provided, *all* the individuals, including the volunteer, benefit from it. In other words, this service is a *public good*. Without loss of generality, let us assume that each node, $i = 1, .., N$, derives 1 unit of benefit from the existence of this public good and it costs node i $c_i \in [0, 1]$ to provide the service. Further, the *distribution* $F(c_i)$ of these costs c_i is public knowledge, but the cost to any individual node is private (i.e., node i knows how all costs are distributed, but only i knows the precise value of c_i).

Diekmann [97] presents this formally as a one-shot game called the *Volunteer's Dilemma* (VOD). Each node has two possible strategies it may play: volunteer or free-ride. Player v's utility is:

$$U_v \stackrel{\text{def}}{=} \begin{cases} 1 - c_v & \text{if } v \text{ volunteers} \\ 1 & \text{if someone, but not } v, \text{ volunteers} \\ 0 & \text{if no one volunteers} \end{cases}$$

That is, if at least one node volunteers, everyone obtains the public good and receives

utility 1, but each node i that volunteers must pay c_i. If no one volunteers, the public good is not available and no one gains any benefit.

Bliss and Nalebuff [98] consider a slightly different scenario, often called the *Volunteer's Timing Dilemma* (VTD) [99]. In their model, each player's strategy is no longer to "volunteer or not", but rather to determine a time $T \geq 0$ that denotes "when to volunteer". If no one volunteers until time t, then the public good is not available until then. To capture this, each player's utility decreases by the standard discount factor, giving player v utility $e^{-t}U_v$. As in the VOD, cost is private information but the distribution of costs is common knowledge. For the remainder of this paper, we assume the distribution of costs, F, is the uniform. (In our simulations, we experiment with cases when the actual cost distribution is different from the assumed distribution.)

Bliss and Nalebuff derive $T(n, c_i)$, the optimal time for node i to volunteer given its cost, c_i, and the total number of players, n:

$$T(n, c_i) = (n-1) \cdot \left(\frac{c_i}{1 - c_i} + \ln(1 - c_i) \right) \tag{7.1}$$

This derivation has several nice properties. First, when all players are rational, the node with highest capacity (lowest c_i) is the one to volunteer. Second, as n increases, each player's expected utility increases, as does their optimal time to volunteer. Last, since T is found by maximizing $e^{-t}U_v$ for each v, T defines a dominant strategy for all players.

7.2.2 Generalized VTD

In both Diekmann's and Bliss and Nalebuff's models, all players can observe and benefit from any volunteer. Here, we introduce the Generalized Volunteer's Timing

Dilemma (GVTD). An input to the game is an arbitrary, undirected graph $G = (V, E)$, where V is the set of players. Each player v continues playing until either v or one of its one-hop neighbors u volunteers where $(u, v) \in E$. Observe that when G is a clique, we have the original VTD.

Let us assume that each node (player) knows only its two-hop neighborhood. We now present a derivation of the optimal time that each node should wait before volunteering, and show that the final set of volunteers constitutes a maximal independent set.

Optimal Waiting Time To calculate the *optimal* time for u to wait before volunteering, u needs to know the global topology, which is typically costly. Instead, we assume that each node learns its two-hop neighborhood using techniques suggested in Catch [54] and computes its waiting time with this partial topology information. Our technique generalizes to the global optimal if a consistent view of the entire topology were available.

Each node u takes into account both u's remaining battery level and energy loss due to volunteering to obtain its volunteering cost c_u.[1] Nodes currently playing the game also use the information of their two-hop neighborhood to calculate their optimal time to volunteer. For a neighbor v of u, let n_v denote the number of one-hop neighbors of v who are *not* one-hop neighbors of u. Also, let $\mathcal{R}^1(v)$ be the one-hop neighbors of v who are currently playing the game. Then the optimal time for v to wait, as a function of its cost,

[1] An accurate prediction of the cost incurred by volunteering would require u to have an estimate on how much traffic its neighbors wish to send, which is often difficult. In our implementation and simulation, we let c_u be one minus u's relative remaining battery.

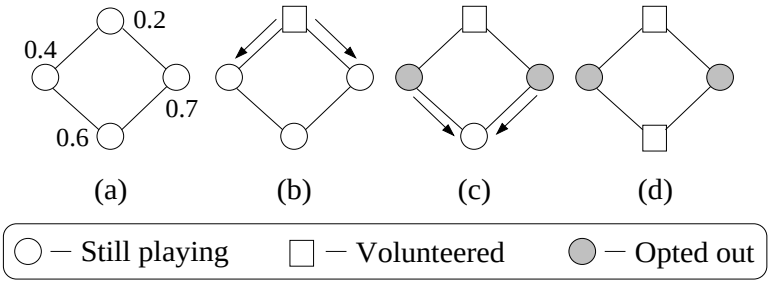


Figure 7.1: An example GVTD game run on (a) a sample graph. (b) The top-most node volunteers and notifies its neighbors who (c) opt out and inform their neighbor, who then (d) volunteers immediately.

is:

$$T_v(c) \approx \int_{x=0}^{c} \sum_{u \in \mathcal{R}^1(v)} \frac{n_u x(1-x)^{n_u-2}}{n_u - 1 + (1-x)^{n_u}} \; dx, \tag{7.2}$$

which is easily integrated numerically. (The derivation sketch is in Appendix B.)

Observe that T_v reflects the amount of time to wait to volunteer since the beginning of the game. When a node w has waited long enough and none of its neighbors have volunteered, w volunteers. Then, its one-hop neighbors *opt out* of the game; they stop playing because they are already receiving the public good (as shown in Figure 7.1). Note that $\mathcal{R}^1(v)$ can change over time with some volunteering nodes, and each node v needs to recalculate T_v. When a node v finds that the current value of T_v is less than the current time elapsed in the game, it volunteers immediately. In Figure 7.1(d), the bottom-most node recalculates T with no remaining neighbors ($\mathcal{R}^1 = \emptyset$) and therefore volunteers immediately.

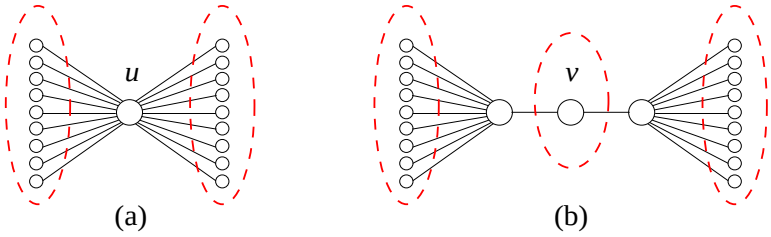


Figure 7.2: Dashed ovals represent likely volunteers. (a) A large one-hop neighborhood reduces u's probability of volunteering, whereas (b) a large two-hop neighborhood has the opposite effect.

7.2.2.1 GVTD Solution Properties

The derivation in Equation 7.2 yields many of the same properties of the original VTD. First, observe that, as in the model proposed by Bliss and Nalebuff, T_v in the GVTD is increasing in c_v — this ensures that (all other factors being equal), nodes with lower cost volunteer earlier. Next, T_v is decreasing in $|\mathcal{N}^1(v)|$; this implies each additional one-hop neighbor is another candidate to allow v to opt out instead of volunteer itself. For example, in Figure 7.2(a), node u is unlikely to volunteer, as it has many one-hop neighbors who may do so earlier.

New to the GVTD is the notion of a non-trivial $\mathcal{N}^2(v)$; as this grows, T_v decreases. To see this, note that each additional two-hop neighbor is another candidate for (at least) one of v's one-hop neighbors to opt out. In Figure 7.2(b), node v is likely to volunteer, since each of its one-hop neighbors is likely to opt out.

GVTD does not always result in the nodes with the lowest cost volunteering. For instance, suppose in Figure 7.2(a) that node u has cost 0.1 and every other node has cost

0.99. Though node u has the lowest cost, all of its neighbors have degree 1, hence they will all have significantly larger probability of volunteering and therefore smaller values of T. In this example, $T_u(c_u) = 0.1$ and, for each neighbor w of u, $T_w(c_w) = 0.003$. Such an effect is a necessary outcome of this game. This is due to both private information and the graph's topological constraints. Since each node u does not know any other nodes' cost to volunteer, it can at best estimate the probability that its neighbors will volunteer before it does.

GVTD Yields a Maximal Independent Set Recall that an independent set S of $G = (V, E)$ is a subset of V such that no two vertices in S correspond to an edge in E. S is a maximal independent set if no proper superset of S is an independent set. Recall also that for a *dominating set* D, each node in V either is in D or has a neighbor in D.

Theorem 7.2.1 *Given an input graph $G = (V, E)$, when the GVTD game ends, the set of volunteers constitutes a maximal independent set of G.*

Proof: Let $U \subseteq V$ denote the set of nodes that volunteered at the completion of the VTD game. We have, for each node $v \in V$, exactly one of the following: $v \in U$ or $\exists \alpha \in \mathcal{N}^1(v) \cap U$. By definition, U is an independent *dominating* set, and is therefore a maximal independent set. ■

In the unit-disk graph model [42], which is a simple yet popular model for wireless networks, a maximal independent set is a constant-factor approximation of a minimum dominating set. Finding a minimum-sized dominating set is a well-known NP-hard problem [40], and Theorem 7.2.1 proves that in the unit-disk graph model, the resulting dominating set of a GVTD game is essentially smallest possible. This implies that the

backbone has a minimum possible number of volunteers, which in turn minimizes the overall energy consumed. In Figure 7.2(a) and (b), the dashed ovals show a maximal independent set for each of two graphs.

These GVTD properties are now sufficient background to construct a protocol for backbone formation in selfish networks, which we describe next.

7.3 Backbone Formation: Protocol

Our backbone construction protocol consists of two logical steps: leader selection and the connection of the leaders. In the first phase, nodes play the GVTD game. We make the standard assumptions of incomplete information [100]: each node v knows that all volunteering costs are chosen uniformly at random from $(0, 1]$ and that its own precise cost is c_v. Given its two-hop neighborhood information, each node independently computes how patient it will be, and then waits for some other node to volunteer. When a node observes that there is no leader in the neighborhood for a long (enough) time, it volunteers as a leader to speed up the backbone construction and thus minimize loss of its own messages.

Nodes who volunteer become *leaders*. The GVTD game ensures that these leaders form a maximal independent set. In the second phase, we choose *bridge nodes* to connect the leaders and obtain a connected backbone (specifically, a connected dominating set). In this section, we describe the backbone formation protocol using the IEEE 802.11 terminology; however, the protocol can easily be generalized to operate with other schemes.

7.3.1 Leader Selection Protocol

Initially, we assume each node is in sleep mode. Nodes wake up periodically (*e.g.*, as in IEEE 802.11) and exchange their neighbor information. This information includes IDs of the transmitting node and all its neighbors. Each node also checks if any neighbor has volunteered. If node v does not observe any volunteers for a period longer than its optimal waiting time, $T_v(c_v)$, it volunteers as a leader and broadcasts a LEADERDECL message to its neighbor information. This message includes a "service duration," which specifies how long v is willing to be part of the backbone. Upon receipt of this message, all of v's neighbors know that they have a volunteer; they opt out of the game and include the leader information in subsequent periodic messages. Our leader selection protocol is *repeated* when the service duration expires.

The service duration is an important system parameter. It should be long enough for the backbone to amortize the overhead and function stably, and short enough for nodes to change roles without consuming too much battery at once. One potential issue with the service duration arises when a node wants to use a very short duration when volunteering. In particular, if the node knows it has data packets to send soon, it can try to set the duration such that it stays in the backbone just long enough for its own transmissions and leaves the backbone immediately after the completion. In this case, however, its neighbors can detect that the duration is below a certain threshold, and punish the node by not forwarding packets destined for the node for a while. Since we assume there can be incoming packets for the node at any moment, such a punishment can potentially lead to multiple packet losses for the misbehaving node, which significantly decreases the node's

utility.

Incentives for Truthfulness Obviously, the volunteering procedure described above is trivial — the interesting part is ensuring that even selfish nodes perform the protocol correctly.

Recall from Section 7.2 that v will volunteer sooner if its one-hop neighborhood is sparser. Thus, it may appear that v's neighbor u would choose not to broadcast its identity, so that v does not count u as a neighbor and volunteers earlier. However, if v becomes a leader, v will not regard u as a neighbor and will not provide the backbone service to u. Hence, "hiding" is of no use to u.

Another way for u to shorten v's waiting time is to pretend u has many neighbors by including fake neighbors in periodic messages. This is, in effect, a Sybil attack [101]. We consider this an orthogonal problem, and direct the reader to Newsome *et al.* [102] who detect and defend against Sybil attacks in a wireless setting.

Estimating Cost Distribution As discussed in Section 7.1, we assume that each node knows its own cost and also the distribution of costs of other nodes. In practice, various factors determine the cost, including remaining battery power, the degree of desire for communication, and altruistic tendency. A network composed of many nodes with cost close to 1 corresponds to the scenario where there are many selfish nodes that are reluctant to volunteer. In contrast, if there are many low-cost nodes, then the network is more altruistic and the backbone construction is faster. One way is to learn relative willingness of neighbors over time and use them as sample points to infer the distribution.

In our results, we quantify the convergence time and quality of the backbone when the estimated cost distribution is different from the actual. However, there exists little study on altruistic behavior in real multihop wireless networks, and inferring the overall cost distribution is an open question.

7.3.2 Connecting the Leaders

After leader selection, no two leaders will be adjacent. The second phase of our backbone construction consists of choosing nodes between leaders to act as *bridge nodes* and forward messages between them.

Since the final leader set is a dominating set (Theorem 7.2.1), each leader will be no more than three hops away from at least one other leader [40]. It therefore suffices for each leader node to learn about other leaders that are reachable within three hops (*i.e.*, via a path through at most two non-leaders). To accomplish this, node u broadcasts to its neighbors a message indicating that it has volunteered. u's neighbors then forward this message to their one-hop neighbors. (We defer the issue of enforcing the correct delivery of this message to Section 7.4.) Each neighbor v of u then requires its neighbors, $\mathcal{N}^1(v) \setminus \mathcal{N}^1(u)$, to forward this message to their one-hop neighbors. Node v has incentive to do this because, otherwise, the backbone may not be constructed quickly enough to meet v's end-to-end delivery constraints.

After these messages are propagated, each node v knows of all leaders (say L_v) in its 3-hop neighborhood; v also knows of all paths (of length at most 3) from v to each node in L_v. This information can be used to connect the leaders in many different

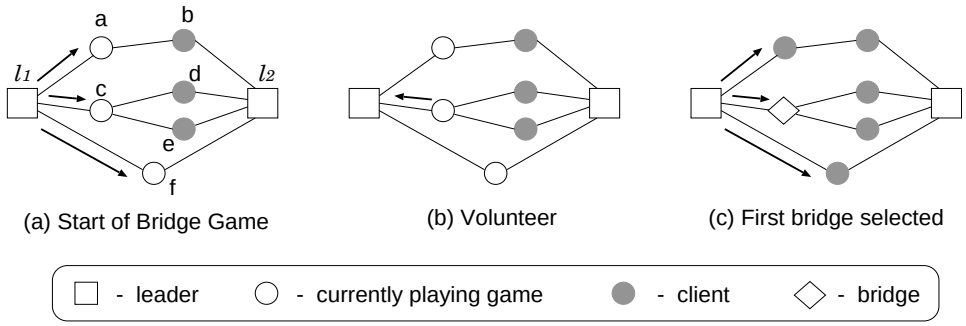


Figure 7.3: First iteration of bridge node selection between two leaders, ℓ_1 and ℓ_2. Nodes are informed to play a bridge game for the first hop on each (at most two hops long) path from ℓ_1 to ℓ_2. The winner of this iteration is node c, who joins the backbone.

ways, such that different metrics (e.g., backbone size [40,42], or minimum cost [103]) are optimized. In this chapter, we consider the simplest case where *each leader connects to* all *other leaders that are in its 3-hop neighborhood.* This will result in larger backbones than are strictly necessary (e.g., $\mathcal{B}_{\text{MST}}$ in Chapter 6). However, as we discuss further in the following sections, some nodes may be punishing one another; these redundant links can be helpful in allowing nodes to re-route packets around punishing areas. Furthermore, Alzoubi et al. [104] show that the hop count increase for any path over such a backbone is bounded by a constant when compared to the path without the backbone. Also, in a unit-disk graph, the resulting set is still a constant-factor approximation of a minimum connected dominating set. Further, it is not immediately clear how to provide incentive for nodes to truthfully report enough connectivity information for nodes to determine which links are redundant. Obtaining backbones that are small yet allow for resilience in the face of punishment is a subject of future work.

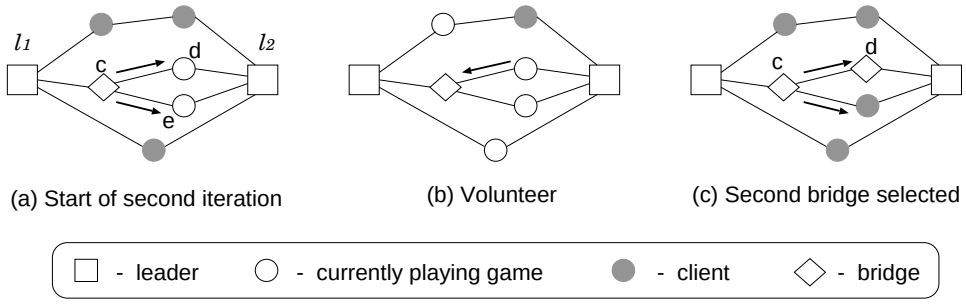


Figure 7.4: Second iteration of bridge node selection. c, the new bridge node after the first iteration is not a neighbor of ℓ_2 and tells d and e to play the original VTD game. In this example, d becomes a new bridge node.

Bridge Node Selection For a leader to connect to other leaders, it must select a set of *bridge nodes* to forward packets between leaders. The key difference in terms of workload between bridge nodes and leaders is that leaders accept clients and buffer their packets (while the clients sleep). When leader ℓ_1 wants to connect to leader $\ell_2 \in \mathcal{N}^3(\ell_1)$, it uses the following bridge node selection process.[2] First, using its knowledge of its 3-hop neighborhood, ℓ_1 determines all of the available paths of length at most 3 from itself to ℓ_2. Let $\mathcal{B}(\ell_1 \rightarrow \ell_2)$ denote the set of one-hop neighbors of ℓ_1 who are on at least one of these paths. In Figure 7.3, $\mathcal{B}(\ell_1 \rightarrow \ell_2) = \{a, c, f\}$.

We provide pseudocode for our bridge selection protocol in Algorithm 3. To summarize, each hop on the path from ℓ_1 to ℓ_2 is obtained by applying VTD. Because the goal of each VTD game is to select a single, high-capacity node, we need not use the GVTD game. Instead, nodes play the standard, complete-graph VTD game, and use the

[2]To break symmetry, only the leader with the smaller ID will initiate the bridge node selection process.

Algorithm 3 Play Bridge Game(ℓ_1, ℓ_2, L)

Called by u when it receives a PLAYBRIDGEGAME message from prev_hop to connect ℓ_1 to ℓ_2. L is the list of potential volunteers.

1: **if** $u \notin L$ **then**
2: return
3: **end if**
4: $t \leftarrow$ Calculate-Complete-VTD-Time$(c_u, |L|)$
5: Wait for time t
6: **if** after waiting, prev_hop has not announced a volunteer **then**
7: Send BRIDGEVOLUNTEER message to prev_hop
8: **if** prev_hop replies with a BRIDGEACK **then**
9: **if** $\ell_2 \notin \mathcal{N}^1(u)$ **then**
10: $L' \leftarrow \{v : v \in \mathcal{N}^1(u) \wedge \ell_2 \in \mathcal{N}^1(v)\}$
11: Broadcast PLAYBRIDGEGAME(ℓ_1, ℓ_2, L')
12: **end if**
13: **end if**
14: **end if**

function derived from Eq. 7.1. The first iteration of the game is played by the nodes in $\mathcal{B}(\ell_1 \rightarrow \ell_2)$. Each node playing the game stops when either it volunteers and informs the previous hop on the path, or when the previous hop broadcasts an acknowledgment to the first volunteer. In Figure 7.3, c is the winner of the game (*i.e.*, the first node to volunteer) and becomes a bridge node. It then either informs ℓ_2 of the path (if c is connected to ℓ_2), or else initiates another VTD game to obtain the next hop. As shown in Figure 7.4, the second iteration of the game is played by the nodes who are both one-hop neighbors of c and ℓ_2 (i.e., d and e). Note that, since ℓ_1 chose nodes who are within two hops of ℓ_2, this

process will require at most two such games.

Incentive for Truthfulness In our protocol, nodes have incentive to truthfully report neighbors' leaders. A leader accumulates this information to learn how to connect to nearby leaders. Non-leader node u must include a leader ID for each of its neighbors; otherwise, u's leader will be able to detect the missing information and punish u (see Section 7.4). Suppose u decides to report wrong information. For instance, u could pretend that all of its neighbors share one leader, making it less likely that u's leader would choose u as a candidate bridge node. However, the wrong information sent by u will be detected by u's other neighbors, and u will be punished. Each node therefore has incentive to include each neighbors' leader ID correctly.

Bridge Selection without VTD The bridge selection algorithm incurs some overhead due to the waiting times and control message exchanges. An alternate scheme is to forgo the bridge nodes' VTD games and to just have ℓ_1 designate bridge nodes. If a designated node refuses the request, then ℓ_1 in turn can refuse to provide the backbone service to the node. With this alternate scheme, the bridge selection is immediate, but the backbone may include high-cost nodes. In our simulations, we experiment with the full VTD-based bridge selection algorithm; in our implementation, leaders are chosen using GVTD, but bridges are assigned by leaders without undertaking the VTD game.

7.4 Incentive-Compatible Forwarding

Another issue to consider after the backbone construction is how to ensure correct end-to-end forwarding when backbone nodes are selfish. In fact, this subject alone constitutes a large yet orthogonal space for interesting research [49, 51, 52]. In this section, we present a high-level approach that we take without elaborating on details.

We employ a punishment scheme by channel jamming (i.e., transmitting continuous signal on the wireless medium). Since we assume that each node wants to achieve high goodput (Section 7.1), and nodes cannot receive messages while the jamming is active, this punishment leads to strict loss of utility for misbehaving nodes. In fact, we can show that this strategy can lead to a desirable equilibrium from a game theoretic perspective, although we do not describe the detailed proof here. This analysis is based on the assumption that the channel jamming is perfect; nodes cannot receive any messages during the punishment. In Section 7.6 we present our experiment results to show the effectiveness of jamming using commodity hardware and software.

7.5 Simulation Results

In this section, we present results from ns-2 simulations of our backbone construction protocol. First, we study the performance when all nodes in the system are rational and none of them deviate from the protocol. Also, we analyze the price of irrationality by allowing some of the nodes in the system to deviate by refusing to be either a leader or a bridge node. Finally, we compare the performance of our backbone algorithm with a cooperative backbone construction scheme. We conduct our studies on three different

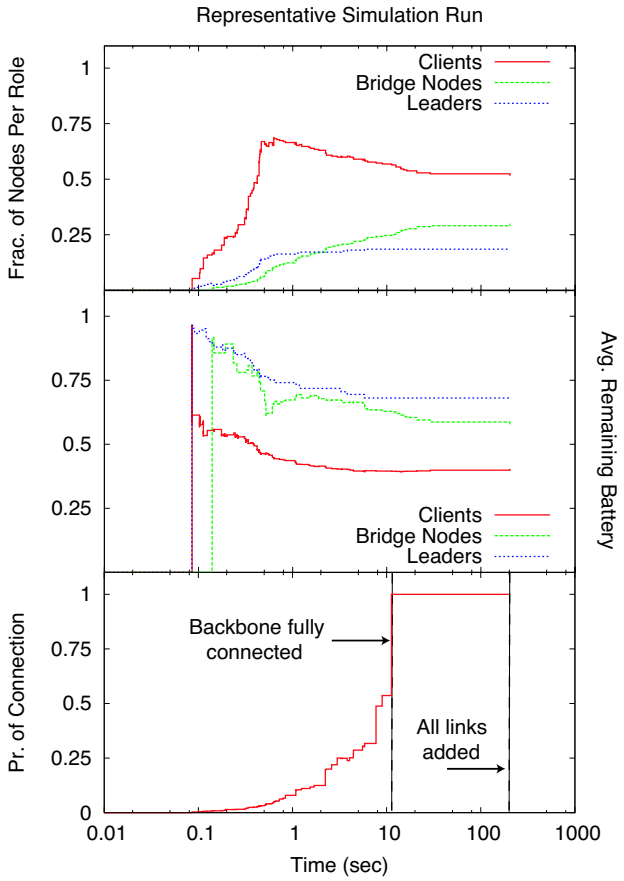


Figure 7.5: Representative backbone properties over time, $N = 227$.

topologies composed of 130, 227, and 306 nodes [105]. Each of them is generated from an urban setting in Portland, OR, modeling transceivers placed along the roadways.

All Peers Rational and Well-Informed In this set of experiments, all peers follow the backbone construction protocol as detailed in Section 7.3 (*i.e.*, they are rational), and each node knows that the distribution of costs is uniform (*i.e.*, they are well-informed). In Figure 7.5, we present a sample run of this scenario on the 227 node topology. We can observe several key properties. First, as shown in the top plot, the majority of nodes are

	Fraction of Nodes			Avg. Fraction of Remaining Battery		
N	Leader	Bridge	Client	Leader	Bridge	Client
130	.191 (.00828)	.323 (.0218)	.486 (.0240)	.746 (.0262)	.545 (.0381)	.388 (.0260)
227	.198 (.00924)	.318 (.0253)	.484 (.0293)	.690 (.0244)	.578 (.0140)	.395 (.0185)
306	.160 (.00662)	.316 (.0171)	.524 (.0191)	.736 (.0388)	.592 (.0181)	.389 (.0226)

Table 7.1: Average backbone size and remaining battery value for each type of nodes. Values in parenthesis are standard deviations. All peers know the correct distribution of costs (uniform). All values are measured once the backbone has been completely constructed.

clients, and hence more than half of the network can enter sleep mode. Also in the top plot, we see that the number of leaders is smallest, while the number of clients is largest. For example, around 20% of nodes are leaders while more than 50% of nodes are clients in sleep mode. In Table 7.1, we present means and standard deviations of 10 runs for different settings. We observe that our scheme leads to small backbones across various scenarios. Since the number of nodes doing each role is inversely proportional to the overhead in performing that role, we can achieve a longer network lifetime in practice.

Network lifetime is further improved by choosing the high-capacity nodes to be backbone nodes. The middle plot of Figure 7.5 shows that our system chooses high-capacity nodes in the backbone. Specifically, the average capacity of leaders (around 0.7) is significantly higher than that of clients (around 0.4). Again, in Table 7.1, we observe this is a general trend across various environments. Our scheme builds a small backbone using high-capacity nodes even in selfish settings, which leads to significant

	Backbone Completion Time	
N	Connected	All Links
130	14.7 (13.0)	231 (230)
227	23.9 (14.4)	231 (202)
306	34.7 (22.1)	379 (234)

Table 7.2: Means and standard deviations of backbone construction time in seconds. All nodes know the correct distribution of volunteering costs (uniform).

increase in network lifetime as demonstrated in Chapter 6. The plot shows that, as time progresses, higher-capacity nodes are chosen first, and lower-capacity nodes are added to the backbone later as necessary. Spikes in bridge node costs occur when there are no better choices for bridge nodes between a pair of leaders.

Finally, in the bottom plot of Figure 7.5, we show the probability of a random source-destination pair communicating over the backbone. As is evident from the plot, it can take a considerable amount of time to include *all* backbone nodes (more than 3 minutes in this case). A connected backbone, however, is formed *quickly* (around 10 seconds in this plot), and every source-destination pair can communicate with each other. This is because many of the links that are added are redundant (see Section 7.3). We show further evidence of this in Table 7.2. For example, with 306-node networks, it takes around 34 seconds to form a connected backbone, but 379 seconds to add all the links. Also important to note is that while many nodes are added for redundancy, few nodes are promoted into the backbone once the backbone is connected.

	Average Remaining Battery			Completion Time (in sec)		Coverage of Maximum
N	Leader	Bridge	Client	Connected	All Links	Connected Component
130	.397	.215	.064	873 (414)	1849 (454)	.986 (.0108)
227	.357	.226	.073	1560 (236)	1970 (390)	.908 (.109)
306	.376	.223	.065	1860 (656)	2170 (495)	.985 (.0170)

Table 7.3: Results when all peers believe that the cost distribution is uniform when it actually is long-tail (majority of nodes have low battery). Note that due to many low-battery nodes, only 25% of the runs resulted in a fully connected component. The completion times shown are from those runs only. Values in parenthesis are standard deviations.

The Effect of Incomplete Information One of the assumptions we make in our systems is that the volunteering cost distribution F of is public knowledge. This is a standard assumption in many game theoretic scenarios, but it is generally untrue in practice. To study the effect of incomplete information on our protocol, we ran experiments where each node in the system assumes that the cost distribution F is uniform, when in reality costs actually follow a different distribution. In practice, it is unclear what a reasonable distribution of battery values is. In our experiments, we use the Pareto distribution because in this distribution, the real costs are substantially different from what is assumed by each node.

We present our results in Table 7.3. For all system sizes, the distribution of costs for each role remains the same as when the nodes are well-informed. In other words, leaders still have the highest remaining capacity, clients have the least, etc. The fundamental difference is in the completion time; the average completion times are orders of magnitude

longer than the case with uniform distributions (see Table 7.1). The coverage of the maximum connected component in Table 7.3 shows that a few nodes do not join the backbone before the simulation has ended. This shows that there are a few nodes whose battery values are so low that they are patient enough to wait as long as it takes until one of their neighbors volunteers. In 75% of our experiments, this exceeded the duration of the simulation (20 minutes).

Besides the long-tail distribution, we also experimented with normal distributions. Although we do not report the results here, our findings were similar: *The distribution of costs according to roles remain independent of the accuracy of the information. The time to have a fully connected backbone, however, increases.*

We can use the following fact to explain this: *the behavior of a node can change drastically depending on how it perceives others' remaining capacities.* When nodes' battery levels are lower than the estimated average, they become increasingly patient, and the backbone takes more time to converge, as evident in Table 7.3. Conversely, when nodes have a battery level higher than the perceived average, they become more willing to volunteer, and the backbone constructions is significantly faster.

The Price of Free-Riding We now study the effect of a node trying to free-ride from the system. By free-riding, we mean that the node refuses to take any role in the backbone. We show that free-riding has an adverse effect on *each* node in the network, including the free-riders, and conclude that free-riding is not a rational strategy.

We experiment with a varying number of nodes acting as free-riders on the 130-node topology. Clearly, if all nodes deviated in such a manner, the network would be

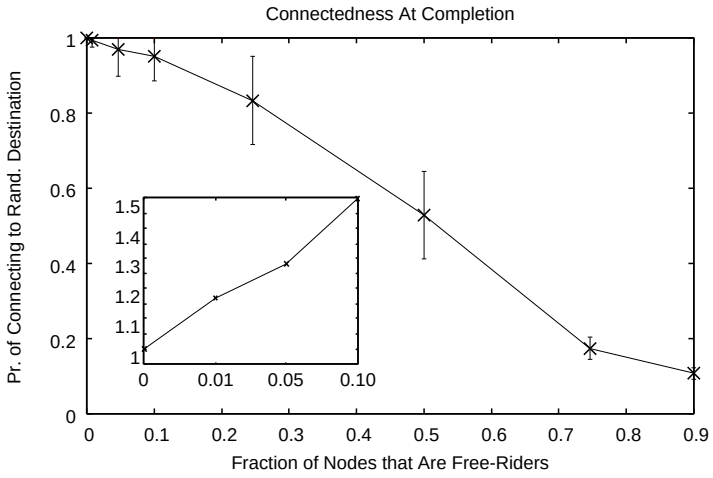


Figure 7.6: Given an increasing number of nodes who refuse to take part in the backbone, the probability of any node (selfish or not) being able to connect to a destination chosen uniformly at random declines. Even when the network forms a connected backbone, the free-riders delay the process (inset). $N = 130$.

completely disconnected and yield no social benefit.

As expected, we see in Figure 7.6 that the connectivity quickly declines with respect to the number of free-riders in the system. In other words, *rampant free-riding causes system collapse*, so utility-maximizing, rational nodes will have no incentive to free-ride on such a large scale. When only a small percentage of nodes refuse to be either leader or bridge nodes, the backbone may be connected, but the probability goes down. In the inset to Figure 7.6, we focus on the regime with few free-riders (between 0% and 10%), and include only the runs that resulted in a connected backbone. On the y-axis, we plot the factor increase in time it takes for a connected backbone to form over the time it takes when all nodes are rational. We observe that *even when it does not fragment the network,*

free-riding negatively affects each *node's utility by delaying the backbone from becoming connected.* Hence, a rational node trying to maximize its network connectivity would not free-ride.

Comparison with a Cooperative Scheme We use SPAN [19] as an example scheme for cooperative backbone construction and compare it with our scheme. We performed experiments with SPAN using the code written by the authors of SPAN. We report results when we use the 306-node network and uniform distribution of battery level, which are representative of other results in different scenarios. We use values right after SPAN finds a connected backbone. On average, SPAN includes 84 of 306 nodes in the backbone, which is 61 fewer than our scheme (Table 7.1). However, our scheme includes more backbone nodes by design so that messages can detour areas under punishment using redundant paths (Section 7.3.2). Interestingly, although our proposed scheme does not assume cooperation, our scheme results in higher average battery level of backbone nodes than SPAN (0.640 vs. 0.543). It is because SPAN uses randomization as well as battery level when selecting backbone nodes and thus includes many low-battery nodes in the backbone. We observe that the cooperative nature of SPAN leads to shorter convergence time of 12.9 seconds than that of our proposed scheme (34.7 seconds). In summary, our results indicate that *our proposed incentives-compatible scheme can achieve similar performance to existing backbone construction schemes that assume cooperation.*

7.6 Implementation Results

In this section, we describe our implementation and present results from a testbed consisting of laptops running GNU/Linux (kernel version 2.6.11), equipped with an 802.11g card with the Atheros chipset. We modify the MadWifi open-source wireless device driver. We use the Click software router package [106] to implement the protocol in user space. We begin with an overview of our implementation, and report our results in Section 7.6.2.

7.6.1 Implementation and Testbed

Driver-level Changes Our protocol requires leaders to buffer packets for sleeping nodes. The 802.11 AP-mode operation natively supports such buffering for *associated* nodes that are asleep, but ad-hoc mode in the current driver does not. Hence, we use AP mode for leader nodes and managed mode (the default for 802.11 clients) for non-leaders. We ensure that non-leader nodes stay associated with one leader by disabling the periodic scanning (which finds better APs).

The primary purpose of the backbone is to allow nodes to conserve energy. This is done by putting nodes to sleep. However, the original MadWifi driver does not support sleep mode; we extended the driver to support full sleep-mode operation in managed mode. We can implement our scheme on top of native ad-hoc networks by extending sleep mode support to ad-hoc mode in the future.

In a multihop network, neighbors (regardless of backbone leader/client relation) ought to be able to communicate directly with each other. In the original 802.11 driver,

however, nodes in managed mode can only communicate with their AP; we have extended the driver to permit unrestricted communication even in managed mode.

Finally, in our testbed, the range of a single node at low transmission rates (e.g., 1 Mbps) was too large, and we could not obtain interesting multihop paths. To reduce the effective range, we pinned the MAC-level transmission rate at 11Mbps for all frames generated by the backbone protocol and application. We would have preferred to reduce the transmit power, but this was not supported in the driver we used.

Backbone and Routing The backbone protocol is implemented entirely in user-space as a Click element and uses our modified MadWifi driver. To route packets over the backbone, we use the Click implementation of the routing protocol proposed by Draves et al. [86].

In addition to user data and routing messages, the backbone layer exchanges periodic control messages. These messages are used to determine the network state; for example, to learn the numbers of one-hop and two-hop neighbors. Additionally, the backbone layer takes decisions about the operating mode. For example, it decides when to volunteer or to go to sleep. Each node uses its locally compiled information about its neighbors and its current remaining battery to determine its volunteering time. We use the battery values reported by a WISE interface call, which in turn uses ACPI (Advanced Configuration and Power Interface) in Linux.

Testbed Our testbed consisted of 12 laptops, laid out according to the floor plan as shown in Figure 7.7. In our experiments, nodes ignore high-error and unidirectional links

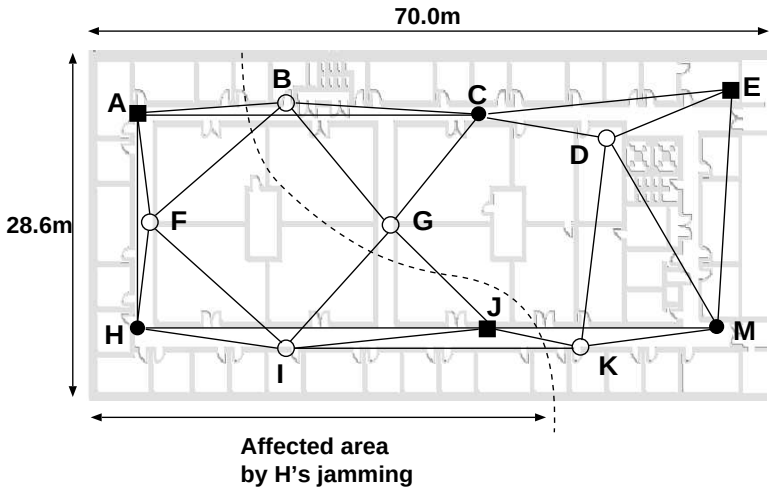


Figure 7.7: Experiment layout. In our backbone experiment, 50% of the nodes (B, D, F, G, I, K) are in sleep mode. Nodes with square (A, E, J) are leaders, nodes in black dots (C, H, M) are bridges.

when selecting leaders and bridges. All packets generated by or destined to non-backbone nodes always go through their leader. Bridge nodes can directly receive packets from other backbone nodes in their vicinity (both leaders and bridge nodes). We measure end-to-end TCP throughput using `netperf` (http://www.netperf.org) and end-to-end latency using `ping`. We observe that the performance of each flow is consistent over multiple runs, and report the average of five different runs.

7.6.2 Experiment Results

In this section, we first investigate the effect of using a backbone on overall network performance. We also use our implementation to quantify the effectiveness and cost of punishment in practical scenarios.

	Backbone		No Backbone	
	Throughput (Mbps)	Latency (ms)	Throughput (Mbps)	Latency (ms)
Flow 1: $E \rightarrow A$	2.36	4.07	2.45	3.64
Flow 2: $I \rightarrow M$	2.42	56.38	2.38	3.93
Flow 3: $E \rightarrow G$	1.28	64.58	2.32	3.62

Table 7.4: Throughput and latency with and without the backbone. Flow 3 follows a longer path over the backbone, resulting in a drop in throughput. With the backbone, Flows 2 and 3 have a sleeping node, which results in larger latency.

7.6.2.1 Effect of the Backbone on Network Performance

In Table 7.4, we compare average throughput and latency of flows with and without a backbone. Flow 1 (using E-C-A) consists only of backbone nodes, and the end-to-end path is the same as the one without the backbone; for Flow 2 (using I-J-M), the path is the same, but now it includes a node (I) that is a client and hence goes to sleep. For Flow 3, the path changes (E-M-J-G with the backbone and E-C-G without it).

For Flow 1, as expected, both end-to-end bandwidth and latency are largely unaffected by the backbone since both the path and node state (awake or asleep) remain unchanged. Flow 2 on the other hand, shows similar throughput results, but experiences higher latency. This increase in latency is because node I is not in the backbone and is in sleep mode (Figure 7.7). Both the throughput and latency for Flow 3 are affected when we use the backbone. The throughput reduces by around 0.80 Mbps because the flow takes one extra hop, and the latency increases by $\sim$60 ms because G is asleep. When J

receives packets destined to G, J buffers the packet until G wakes up for the next beacon (sent every 100 ms), learns about and requests the buffered packet. This additional buffering delay, on average, is about half the beacon period. Although we do not report details here, we also performed experiments with concurrent flows that share common links. Our results show that the throughput using the backbone was comparable to the network without a backbone.

Finally, we report the control overhead and convergence time of our protocol. For the first five minutes of the protocol, there were 61 control messages per node, with the message size being 56 bytes on average. The interval between periodic heartbeat messages is around five seconds, and the vast majority (about 59 out of 61) of control messages are heartbeat messages. In this 12-node network, it took around 7 seconds to complete the backbone formation, which agrees with the results in Table 7.1.

7.6.2.2 Punishment

Recall that we have asserted that jamming is an effective punishment mechanism in Section 7.4; here, we quantify the throughput degradation due to jamming. In our experiment, we assume H detects a misbehaving neighbor. Specifically, the destination A of Flow 1 is under punishment as well as the source I of Flow 2. While the two endpoints of Flow 3 are not sufficiently close to H, the path (E-M-J-G) includes node J which is within H's jamming range. We ran two sets of experiments; (1) the node starts jamming as soon as the flows start and (2) the jamming node takes 10 seconds to detect that the misbehaving neighbor is sending or receiving data. We present our results

	Flow 1	Flow 2	Flow 3
Throughput (with 10-sec delay)	0.69	0.26	0.16

Table 7.5: Throughput (in Mbps) with punishment. Without the initial 10-second delay, the throughput was $\sim$0 Mbps in all cases.

in Table 7.5.

In our experiments, we observed that if H starts jamming immediately, then the source was unable to establish a connection, resulting in no packets being sent or received. On the other hand, when the jamming starts 10 seconds after the connection was established, some packets got through, but the throughput is reduced by up to 88%. Except for a small number of packets during the initial 10 seconds, communication through a node being punished (e.g., J) is nearly impossible while jamming is active. Also, Flow 1 has only one node A within H's jamming range (refer Figure 7.7) and maintains higher throughput than Flow 2 in which two nodes (I and J) are affected by the punishment. Since Flow 3 uses a longer path than Flows 1 and 2, and the destination G is in sleep mode, the achievable throughput is even lower.

In Table 7.5, each flow has at least one node within H's jamming range. To investigate the effect of jamming on distant paths, we used a different backbone to experiment with a different path (E-C-G) for Flow 3. Although there is no link shown in Figure 7.7, G occasionally receives packets from H (*i.e.*, a weak link between H and G exists). In this experiment, the throughput over the alternate path is 1.35 Mbps on average, which is more than 8 times the throughput over the original path (Table 7.5). This experiment demonstrates that the effect of jamming punishment is localized, and in a large network,

communication involving well-behaved nodes in distant parts of the network will not be affected by local punishments.

7.6.2.3 Energy Consumption

A punishing node needs to transmit signals constantly, which may significantly reduce its battery life. To quantify the cost of punishment, we measured the battery consumed during a 5-minute interval using ACPI. We recharged the battery after each run to its full capacity. We also disabled components such as the LCD screen to minimize external effects. On average, a jamming node consumes 815 mWh of energy for five minutes while leader and bridge nodes consume 700 mWh. Although jamming consumes more energy, in practical uses, it typically lasts for a shorter period. Hence, in practice, we expect only a modest increase in the energy consumed during jamming. We also observed that a sleeping node consumes 645 mWh for five minutes, while a laptop without a wireless card consumes 595 mWh. We believe that with improved hardware design and a better software implementation, we can increase the amount of energy saved due to the backbone.

7.7 Summary and Future Work

We have examined how internal incentive mechanisms can enforce cooperation in a multihop wireless network consisting solely of greedy nodes. We generalize the well-known Volunteer's Timing Dilemma, based on which we develop an incentive-compatible scheme that constructs efficient routing backbones. Our simulation and implementation

results demonstrate that the resulting backbone forms quickly and provides paths and battery savings comparable to protocols designed for fully cooperative nodes.

This work describes the first complete set of results for backbone formation and naturally leads to many interesting open questions, including how to handle node collusion, how to model systems with some altruistic and some greedy nodes, and how to form efficient backbones while taking possible punishment into account. Enforcing correct forwarding and routing is another orthogonal area of interesting research. Designing an incentive mechanism to achieve an efficient equilibrium for routing and forwarding will be of immediate use for future wireless networks composed of self-interested devices.

Chapter 8

Conclusions and Future Work

In this dissertation, we have addressed the issues caused by heterogeneity in link quality and node capability in wireless networks. We have proposed the WISE abstraction framework that provides a uniform set of interfaces and enables upper-layer protocols to access lower-level details without knowing the specific mechanisms. We also have presented a number of protocol extensions that address specific issues arising due to heterogeneity in wireless networks. We have used analytical techniques to prove theoretical guarantees on the performance of some of our schemes. We also have evaluated these protocol mechanisms using simulations and real-world experiments and demonstrated significant performance improvement.

In the WISE framework, we have defined a set of uniform interfaces that can be used to access lower-level information in wireless networks such as estimated packet error rate over a wireless link, required transmit power over a link, link latency, and remaining battery for a node. We have focused on packet error rate estimation and presented several estimation techniques such that we can choose the most appropriate one according to network environments. We also have performed experiments on real wireless testbeds to validate our estimation schemes. One of them uses a well-known two-state Markov model, and successfully estimates the error rates for packets of arbitrary size in an efficient manner.

We have proposed a new architecture for multihop wireless LANs. Unlike the current system where only a direct link to AP is used, the new architecture allows a node to use multihop paths through intermediate clients to reach the AP if it leads to better performance. We have performed a measurement study in a currently deployed IEEE 802.11 WLAN, and demonstrated that carefully designed multihop mechanisms can lead to significant performance improvement. We have developed protocol mechanisms that consider MAC-specific information and performed simulations to understand the performance of proposed mechanisms. Our results show that our protocol leads to significant improvement not only for the nodes that implement it, but also for those nodes that are not aware of the protocol extension.

We have designed a protocol extension for efficient geographic routing. We have proposed to use a new link metric called NADV that considers both location and link quality. We prove that NADV finds paths whose cost is close to the optimum. Geographic routing with NADV provides an adaptive routing strategy, which is general and can be used for various link cost types. We have presented how NADV can be used with multiple WISE interfaces. In the simulation experiments, the combination of NADV and WISE cost estimation techniques outperforms the current geographic routing scheme in various settings.

We also have considered the difference in node capability, especially in terms of remaining battery life. We have focused on increasing network lifetime and presented a backbone construction scheme. We have theoretically proved that the resulting backbone is essentially smallest possible and simultaneously maximizes the minimum node capacity among all connected backbones. We have generalized the scheme such that we can

build more resilient backbones that can maintain connectivity even in dynamic networks. We have performed experiments and presented results to demonstrate that the proposed backbone scheme outperforms the best existing scheme in terms of increasing network lifetime and maintaining connectivity without adversely affecting message delivery performance.

Finally, we have considered the backbone construction problem when nodes are selfish. We have used game-theoretic approaches and modeled the backbone construction as providing a public good. We have generalized the Volunteer's Timing Dilemma (VTD) and presented an incentive-compatible backbone construction protocol based on the generalized VTD analysis. We have performed simulation experiments and investigated several performance aspects such as backbone size, remaining battery distribution among backbone nodes, and backbone construction time. Our results show that the performance of our scheme is similar to that of an existing backbone construction scheme in which all nodes are assumed to be cooperative.

We also have implemented the backbone construction scheme on a real testbed composed of 12 laptops. We have modified an open-source device driver and performed the first evaluation study on the network performance when we use a backbone. Our results show that although we may sometimes experience performance degradation due to nodes in sleep mode, we can achieve a similar level of performance with and without a backbone.

This dissertation work naturally leads to a number of interesting future research issues. First, in Chapter 3, we have mentioned several approaches to achieve more efficient

WISE implementations (e.g., more efficient PER estimation using other available information). As wireless communications become more prevalent, efficient use of network resources through careful implementation will become more important. In the current implementation of WISE framework, we have focused on the IEEE 802.11-based networks. Although the basic functionality remains the same, different wireless technologies may bring different aspects in implementing WISE interfaces. For example, TDMA (Time Division Multiple Access) will result in fewer collision-induced errors than CSMA, and the channel usage will be more efficient when the network load is higher. It will be an interesting piece of future work to implement the WISE interfaces on top of various underlying MAC protocols and experiment with them.

In our work, we have addressed two aspects of heterogeneity in wireless networks: link quality and node capability. Specifically, the new architecture proposed for multihop WLANs and the protocol extension for geographic routing address problems due to heterogeneous link quality, while the two backbone construction schemes address problems due to heterogeneous node capability. These schemes focus only on one aspect of heterogeneity, and an ideal scheme should consider both aspects simultaneously. For example, we will be able to achieve network lifetime increase *and* efficient network operation if we use a small backbone composed of high-capacity nodes while all links between backbone nodes are of high quality. As a first step, considering a static network with complete information may give a good insight into this type of research problems.

In Chapter 7, we have considered the backbone construction problem in a network composed of selfish nodes. As wireless communications become more wide-spread among individual users, we need to take selfishness into account and design incentive-

compatible mechanisms. Specifically, routing and forwarding have been the most basic functionality in multihop wireless networks, and simple and efficient incentive-compatible mechanisms for the two functions will be essential to the wide deployment of multihop wireless networks. Although several prior schemes that depend on external entities may be applicable to networks with some infrastructure, in decentralized wireless networks, we need mechanisms that do not depend on central authority, which will be a very interesting issue to pursue. Another related issue arises when nodes are not just selfish but *malicious*. Due to the broadcast property of wireless medium, a few malicious nodes can disrupt the entire communication in wireless networks. Although apparently very challenging, addressing this type of problems will be of increasing importance as decentralized wireless networks become prevalent in the future.

Appendix A

Proofs for Theorems in Chapter 6

We describe a special case of the FKG inequality [107]. Consider an event F that is determined by a vector $\vec{Y} = (Y_1, \cdots, Y_m)$ of independent random variables $Y_i \in \{0, 1\}$. Suppose that whenever F holds for $\vec{a}$, F also holds for any $\vec{b}$ that coordinate-wise dominates $\vec{a}$ (i.e., $\forall i, a_i \leq b_i$). Then, we call F an *increasing* event of $\vec{Y}$. For increasing events $F_1, \cdots, F_l$, the following holds: $Pr(\bigwedge_{i=1}^{l} F_i) \geq \prod_{i=1}^{l} Pr(F_i)$.

A.1 Proof of Theorem 6.1.2

We prove that in any D-regular graph with n nodes, $E[|\mathcal{L}|] \leq c\frac{n}{D}\log(D+1)$ for a constant c that approaches 1 as D becomes large. Consider an indicator variable X_v, where $X_v = 1$ iff $v \in \mathcal{L}$. Then, $|\mathcal{L}| = \sum_v X_v$. Let us denote by P_v the probability that node v is nominated as leader. Then, from the linearity of expectation, $E[|\mathcal{L}|] = E[\sum_v X_v] = \sum_v E[X_v] = \sum_v P_v$. Then, to prove the theorem, it is sufficient to show: $\forall\, v, P_v \leq \frac{c}{D}\log(D+1)$. Since all nodes have exactly D neighbors, P_v is same for all v's.

We define $\mathcal{E}_i = Pr[i\text{-th neighbor of } v \text{ does not nominate } v]$. We also denote $\mathcal{E}_0 = Pr[v \text{ itself nominates other node}]$. For each node u, consider a binary random variable Y_u: Y_u=1 iff $c_u > c_v$; Y_u=0 otherwise. Then, $\mathcal{E}_i\,(0 \leq i \leq D)$ is an increasing event of n

random variable Y_u's, and we can apply the FKG inequality to p_v, similarly to [108]:

$$\begin{aligned} P_v &= 1 - \int_0^1 Pr[\mathcal{E}_0 \wedge \mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_D | c_v = t]\, dt \\ &\leq 1 - \int_0^1 \prod_{i=0}^{D} Pr[\mathcal{E}_i | c_v = t]\, dt = 1 - \int_0^1 (1 - t^D)^{D+1}\, dt \end{aligned} \tag{A.1}$$

Let us define $A = \int_0^1 (1 - t^D)^{D+1}\, dt$. Then, $P_v \leq 1 - A$. We want to find a constant value $c \geq 1$ that satisfies the following:

$$A \geq (\tfrac{1}{D+1})^{\frac{c}{D}} \tag{A.2}$$

Then, since $x \geq 1 + \log x$ for all $x > 0$, we have $A \geq 1 - c \log{(D+1)/D}$, and consequently, $P_v \leq c \log{(D+1)/D}$, which is what we want to show.

Now, it remains to determine c. By taking the natural logarithm of Inequality A.2, c should satisfy:

$$c \geq -\tfrac{D}{\log{(D+1)}} \log A = \tfrac{D}{\log{(D+1)}} \log{(A^{-1})}. \tag{A.3}$$

We note the following facts: $H(n) = \sum_{i=1}^{n} 1/i \leq \log n + 1$, and $1 + x \leq e^x$ for all $x \geq 0$. We denote $\exp(x) = e^x$. Then, we can show that $\log(A^{-1}) = \log(\prod_{i=1}^{D+1} (1 + \frac{1}{Di})) \leq \log(\prod_{i=1}^{D+1} \exp(\frac{1}{Di})) = H(D+1)/D$. (See [70] for details.) Then, we can find an upper bound of the righthand side of Inequality A.3, which we can use as c to satisfy Inequality A.3:

$$c = 1 + \tfrac{1}{\log{(D+1)}}. \tag{A.4}$$

It is easy to see that as D grows large, c approaches 1.

A.2 Proof of Lemma 6.1.3

We denote the righthand side of Equation A.4 as a function of degree d. Note that $c(d)$ is decreasing where $c(1) = 1 + 1/\log 2 \approx 2.44$. Then, using similar steps in Appendix A.1, we can show: $E[|\mathcal{L}|] \leq \sum_{v \in V} \alpha\, c(d_v) \frac{\log(d_v+1)}{d_v} \leq c' \sum_{v \in V} \frac{\log(d_v+1)}{d_v}$, where $c' = \alpha\, c(\delta)$ and δ is the minimum degree in the network.

A.3 Proof of Theorem 6.1.4

Consider the following Integer Program (IP) for a minimum dominating set:

$$\begin{aligned} &\text{minimize} && \textstyle\sum_{v \in V} x_v, \\ &\text{subject to} && \forall v,\ x_v + \textstyle\sum_{u:(u,v) \in E} x_u \geq 1,\ \ x_v \in \{0,1\} \end{aligned}$$

Relaxing the integrality constraints, we get a Linear Program (LP) where $\forall v, x_v \geq 0$. An optimal LP solution is a lower bound of the optimal IP solution. If we consider the dual of LP, by using the property of α-locally-regular graphs, we can show that $\sum_{v \in V} \frac{1}{(1+\alpha)d_v}$ is a feasible dual solution. (See [70] for details.) Since any feasible dual solution is a lower bound of any primal solution, from Lemma 6.1.3, we can show that $E[|\mathcal{L}|] \leq c' \sum_{v \in V} \frac{1}{d_v} \log(\Delta+1) \leq c'(1+\alpha)\log(\Delta+1)\, \textit{OPT}$. This completes the proof of $E[|\mathcal{L}|] = O(\log \Delta)\, \textit{OPT}$.

A.4 Proof of Theorem 6.2.1

We denote by v the minimum-capacity node of the resulting backbone. If $v \in \mathcal{L}$, by Theorem 6.1.1, the backbone is a maximum-capacity connected dominating set. We show the case where v is a part of a virtual edge. Consider the virtual edge that included

v while connecting fragments F_1 and F_2. Let us pick any two leaders each from F_1 and F_2 and call them L_1 and L_2, respectively.

We first prove by contradiction that the following set is not connected: $S = \{u \mid c_u > c_v\}$. Note that $L_1 \in S$ and $L_2 \in S$. Let us assume S is connected. Then, L_1 and L_2 have at least one path P consisting only of nodes in S. In this case, F_1 and F_2 can get merged using possibly multiple virtual edges using the nodes on P. As a result, F_1 and F_2 would not have chosen the virtual edge with v. This contradiction proves that S is not connected. Since S is not connected, no subset of S can be a connected dominating set.

Appendix B

Derivation Sketch for Eq. 7.2

Here we prove the expression for the optimal waiting time given in Section 7.2.2. Let $T_v(c_v)$ denote the function for the optimal waiting time for node v, where c_v is v's cost to volunteer. Assume, as discussed in Section 7.2.2, that each node knows its two-hop neighborhood, $\mathcal{N}^2$. For the ease of exposition, let $\mathcal{N}^1(v)$ include only the neighbors of u that have not opted out of the GVTD game. Further, define the set $\mathcal{N}_v^1(u) \stackrel{\text{def}}{=} \mathcal{N}^1(u) \setminus \mathcal{N}^1(v)$, that is, the one-hop neighbors of u that are not also one-hop neighbors of v. Letting $n_u = |\mathcal{N}_v^1(u)|$, we have:

$$\begin{aligned}
Q_v(c_v) &\stackrel{\text{def}}{=} \Pr[v \text{ will have to volunteer}] \\
&= \Pr[\forall u \in \mathcal{N}^1(v) : u \text{ will not volunteer before } v] \\
&\approx \Pr[\forall u \in \mathcal{N}^1(v) : (c_v < c_u) \vee \\
&\qquad ((c_v \geq c_u) \wedge \exists v \in \mathcal{N}_v^1(u) \text{ s.t. } c_v < c_u)] \\
&\approx \prod_{u \in \mathcal{N}^1(x)} \left[(1 - c_v) + \int_0^{c_v} (1 - (1-y)^{n_u - 1}) \, \mathrm{d}y \right] \\
&= \prod_{u \in \mathcal{N}^1(x)} \left[1 - \frac{1 - (1 - c_v)^{n_u}}{n_u} \right] \qquad \text{(B.1)}
\end{aligned}$$

Bliss and Nalebluff [98] show that the partial derivative of the optimal waiting time, $T_v(c)$, with respect to node v's cost c is

$$\frac{\partial T_v(c)}{\partial c} = -\frac{c}{1-c} \cdot \frac{1}{Q_v(c)} \cdot \frac{\partial Q_v(c)}{\partial c} \qquad \text{(B.2)}$$

Using the above approximation of $Q_v(c)$, we obtain the following.

$$\frac{\partial T_v(c)}{\partial c} \approx \sum_{u \in \mathcal{N}^1(x)} \frac{n_u c(1-c)^{n_u-2}}{n_u - 1 + (1-c)^{n_u}} \tag{B.3}$$

The integral of this expression yields our final expression for $T_v(c)$.

BIBLIOGRAPHY

[1] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes. PGP in constrained wireless devices. In *Proceedings of the 9th USENIX Security Symposium*, pages 247–261, Denver, Colorado, August 2000. USENIX.

[2] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of Mobicom*, pages 134–146. ACM Press, 2003.

[3] Henrik Lundgren, Erik Nordstr, and Christian Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 49–55, 2002.

[4] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 12(11):1122–1133, 2001.

[5] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete algorithms and methods for mobile computing and communications*. ACM Press, 1999.

[6] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th ACM/IEEE MobiCom*, pages 243–254. ACM Press, 2000.

[7] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of the 22nd annual symposium on Principles of distributed computing*, pages 63–72. ACM Press, 2003.

[8] Dragos Niculescu and Badri Nath. Trajectory based forwarding and its applications. In *Proceedings of the 9th ACM/IEEE MobiCom*, pages 260–272. ACM Press, 2003.

[9] Ivan Stojmenovic. Position-based routing in ad hoc networks. *IEEE Communications Magazine*, pages 128–134, July 2002.

[10] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Trans. Netw.*, 5(6):756–769, 1997.

[11] Suman Banerjee and Archan Misra. Minimum energy paths for reliable communication in multi-hop wireless networks. In *Proceedings of the 3rd ACM MobiHoc*, pages 146–156, 2002.

[12] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *IEEE Infocom*, April 2003.

[13] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 24–35. ACM Press, 2002.

[14] Volkan Rodoplu and Teresa H. Meng. Minimum energy mobile wireless networks. *IEEE JSAC*, 17(8):1333–1344, August 1999.

[15] J. Gomez, A. Campbell, M. Naghshineh, and C. Bisdikian. PARO: Supporting transmission power control for routing in wireless ad hoc networks. *ACM/Baltzer Journal on Mobile Networks*, 2002.

[16] Jae-Hwan Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proceedings of IEEE Infocom 2000*, pages 22–31, 2000.

[17] Theodore Rappaport. *Wireless Communications: Principles and Practice (2nd Edition)*. Prentice Hall, 2001.

[18] L. Marie Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of IEEE INFOCOM*, 2001.

[19] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5), 2002.

[20] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proc. of MobiCom*, 2001.

[21] Y.-D. Lin and Y.-C. Hsu. Multihop cellular: A new architecture for wireless communications. In *IEEE Infocom*, March 2000.

[22] Hongyi Wu, Chunming Qiao, S. De, and O. Tonguz. Integrated cellular and ad hoc relaying systems: iCAR. *IEEE JSAC, Vol.19, Iss.10*, October 2001.

[23] O. Dousse, P. Thiran, and M. Hasler. Connectivity in ad-hoc and hybrid networks. In *IEEE Infocom*, June 2002.

[24] Benyuan Liu, Zhen Liu, and Don Towsley. On the capacity of hybrid wireless networks. In *IEEE Infocom*, April 2003.

[25] Ulas C. Kozat and Leandros Tassiulas. Throughput capacity of random ad hoc networks with infrastructure support. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 55–65. ACM Press, 2003.

[26] H.-Y. Hsieh and R. Sivakumar. Performance comparison of cellular and multi-hop wireless networks: a quantitative study. In *ACM Sigmetrics*, June 2001.

[27] William D. List Matthew J. Miller and Nitin H. Vaidya. A hybrid network implementation to extend infrastructure reach. *UIUC Technical Report*, January 2003.

[28] Christian Tschudin. Simple ad hoc routing with LUNAR. In *Proceedings of the 2nd Swedish Workshop on Wireless Ad-hoc Networks*, 2002.

[29] Haiyun Luo, Ramachandran Ramjee, Prasun Sinha, Li (Erran) Li, and Songwu Lu. Ucan: a unified cellular and ad-hoc network architecture. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 353–367. ACM Press, 2003.

[30] N. Ben Salem and M. Jakobsson L. Buttyan, J.P. Hubaux. A charging and rewarding scheme for packet forwarding. In *ACM MobiHoc*, June 2003.

[31] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[32] I. Stojmenovic and X. Lin. Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1023–1032, October 2001.

[33] L. Blazevic, S. Giordano, and J. Y. Le Boudec. Self organized terminode routing. *Journal of Cluster Computing,*, 5(2), April 2002.

[34] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 12(11):1122–1133, 2001.

[35] Karim Seada, Marco Zuniga, Ahmed Helmy, and Bhaskar Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 108–121. ACM Press, 2004.

[36] M. Zorzi and A. Armaroli. Advancement optimization in multihop wireless networks. In *Proceedings of VTC*, October 2003.

[37] D. Johnson and D. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic Publishers, 2001.

[38] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), October 2003. IETF RFC 3626.

[39] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *International Conference on Communications*, June 1997.

[40] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *Proc. of the Fourth Annual European Symposium on Algorithms*. Springer-Verlag, 1996.

[41] Peng-Jun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, pages 1597–1604, June 23–27 2002.

[42] Devdatt Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Arvind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 717–724. Society for Industrial and Applied Mathematics, 2003.

[43] J. Wu, M. Gao, and I. Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *Proceedings of IEEE Int'l Conf. on Parallel Processing*, 2001.

[44] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proceedings of IEEE INFOCOM*, 2004.

[45] Y. Wang, W. Wang, and X. Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *In Proc. of ACM MobiHoc*, 2005.

[46] R. Zheng and R. Kravets. On-demand power management for ad hoc networks. In *Proc. of IEEE Infocom*, April 2003.

[47] Li Li, Joseph Y. Halpern, Paramvir Bahl, Yi-Min Wang, and Roger Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multihop networks. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 264–273. ACM Press, 2001.

[48] N. Li and J. C. Hou. FLSS: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of Mobicom*, 2004.

[49] Luzi Anderegg and Stephan Eidenbenz. Ad Hoc-VCG: A Truthful and Cost-Efficient Routing Protocol for Mobile Ad Hoc Networks with Selfish Agents. In *Proc. of MobiCom*, 2003.

[50] Noam Nisan and Amir Ronen. Algorithmic Mechanism Design. In *ACM Symposium on Theory of Computing*, 1999.

[51] S. Zhong, J. Chen, and Y. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of Infocom*, April 2003.

[52] Levente Buttyan and Jean-Pierre Hubaux. Enforcing service availability in mobile ad-hoc wans. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 87–96. IEEE Press, 2000.

[53] Sheng Zhong, Li (Erran) Li, Yanbin Grace Liu, and Yang (Richard) Yang. On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks: an integrated approach using game theoretical and cryptographic techniques. In *Proc. of ACM Mobicom*, 2005.

[54] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining Cooperation in Multi-hop Wireless Networks. In *NSDI*, 2005.

[55] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. of Mobicom*, 2000.

[56] M. Cagalj, S. Ganeriwal, I. Aad, and J. P. Hubaux. On Selfish Behavior in CSMA/CA Networks. In *Infocom*, 2005.

[57] Pradeep Kyasanur and Nitin Vaidya. Selfish MAC Layer Misbehavior in Wireless Networks. *IEEE Transactions on Mobile Computing*, 4(5), 2005.

[58] Maxim Raya, Jean-Pierre Hubaux, and Imad Aad. DOMINO: a System to Detect Greedy Behavior in IEEE 802.11 Hotspots. In *Proc. of MobiSys*, 2004.

[59] Andreas Kopke, Andreas Willig, and Holger Karl. Chaotic maps as parsimonious bit error models of wireless channels. In *Proceedings of Infocom*, April 2003.

[60] Marco Zuniga and Bhaskar Krishnamachari. Analyzing the transitional region in low power wireless links. In *Proceedings of IEEE SECON*, October 2004.

[61] Abtin Keshavarzin, Elif Uysal-Biyikoglu, Falk Herrmann, and Arati Manjeshwar. Energy-efficient link assessment in wireless sensor networks. In *Proc. of IEEE Infocom*, March 2004.

[62] E. N. Gilbert. Capacity of a burst-noise channel. *Bell Systems Technical Journal*, 39:1253–1265, 1960.

[63] E. O. Elliot. Estimates of error rates for codes on burst-noise channels. *Bell Systems Technical Journal*, 42:1977–1997, 1963.

[64] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM*, 2004.

[65] Baruch Awerbuch, David Holmer, and Herbert Rubens. High throughput route selection in multi-rate ad hoc wireless networks. In *First Working Conference on Wireless On-demand Network Systems (WONS)*, 2004.

[66] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE 802.11 Standard, 1999.

[67] S. Lee, S. Banerjee, and B. Bhattacharjee. The case for multi-hop wireless local area network. In *Proc. of IEEE Infocom*, March 2004.

[68] IEEE 802.11h Standard. Part 11. Amendment 5: Spectrum and transmit power management extensions in the 5GHz band in Europe, 2003.

[69] Rex Min and Anantha Chandrakasan. Top five myths about the energy consumption of wireless communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(1):65–67, 2003.

[70] S. Lee, B. Bhattacharjee, A. Srinivasan, and S Khuller. Efficient and resilient backbones for multihop wireless networks. Technical report, CS-TR 4726, University of Maryland, College Park. Availabe at http://www.cs.umd.edu/users/slee/pubs/tr4726.pdf, June 2005.

[71] Atul Adya, Paramvir Bahl, Ranveer Chandra, and Lili Qiu. Architecture and techniques for diagnosing faults in ieee 802.11 infrastructure networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 30–44. ACM Press, 2004.

[72] C.E. Perkins and E.M. Belding-Royer. Ad hoc on-demand distance vector (AODV) routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[73] V. Park and M.S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *IEEE Infocom*, April 1997.

[74] Z. Haas, M. Pearlman, and P. Samar. The zone routing protocol (ZRP) for ad hoc networks, July 2002. IETF draft, Work in progress.

[75] Jinyang Li, Charles Blake, Douglas S.J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 61–69, New York, NY, USA, 2001. ACM Press.

[76] George Xylomenos and George C. Polyzos. TCP and UDP performance over a wireless LAN. In *IEEE Infocom*, 1999.

[77] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proceedings of First Symposium on Networked Systems Design and Implementation*, 2004.

[78] A. Kamerman and L. Monteban. WaveLAN-II: A high performance wireless LAN for the unlicensed band. *Bell Labs Technical Journal*, 1997.

[79] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proceedings of ACM Mobicom*, 2002.

[80] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proc. of ACM MobiCom*, 2000.

[81] Tommaso Melodia, Dario Pompili, and Ian F. Akyildiz. Optimal local topology knowledge for energy efficient geographical routing in sensor networks. In *Proc. of Infocom*, March 2004.

[82] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 181–190. ACM Press, 1998.

[83] Cisco aironet 350 series client adapters data sheet, June 2003. Cisco Systems Inc. Available at http://www.cisco.com/.

[84] Omprakash Gnawali, Mark Yarvis, John Heidemann, and Ramesh Govindan. Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing. In *Proceedings of the First IEEE Conference on Sensor and Ad hoc Communication and Networks*, Santa Clara, California, USA, October 2004.

[85] Scott Y. Seidel, Theodore S. Rappaport, Sanjiv Jain, Micheal L. Lord, and Rajendra Singh. Path loss, scattering, and multipath delay statistics in four European cities for digital cellular and microcellular raiodtelephone. *IEEE Transactions on Vehicular Technology*, 40(4):721–730, November 1991.

[86] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 114–128. ACM Press, 2004.

[87] M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[88] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proc. of symposium on computational geometry*, 2001.

[89] B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.

[90] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[91] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 2nd Edition*. MIT press, 2001.

[92] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Languages and Systems*, 1983.

[93] R. E. Tarjan. *Data structures and network algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1983.

[94] The VINT Project. The Network Simulator–ns-2. Available at http://www.isi.edu/nsnam/ns.

[95] J. Yoon, M. Liu, and B. D. Noble. Random waypoint considered harmful. In *Proceedings of Infocom*, April 2003.

[96] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of Mobicom*. ACM Press, 2002.

[97] Andreas Diekmann. Volunteer's Dilemma. *Journal of Conflict Resolution*, 29(4), 1985.

[98] C. Bliss and B. Nalebuff. Dragon-Slaying and Ballroom Dancing: The Private Supply of a Public Good. *Journal of Public Economics*, 25, 1984.

[99] Jeroen Weesie. Incomplete Information and Timing in the Volunteer's Dilemma. *Journal of Conflict Resolution*, 38(3), 1994.

[100] Andreu Mas-Colell, Jerry Green, and Michael D. Whinston. *Microeconomic Theory*. Oxford University Press, 1995.

[101] John Douceur. The Sybil Attack. In *Proc. of IPTPS*, 2002.

[102] James Newsome, Elaine Shi, Dawn Song, and Adrin Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proc. of IPSN*, 2004.

[103] Yu Wang, WeiZhao Wang, and Xiang-Yang Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proc. of ACM MobiHoc*, 2005.

[104] Khaled Alzoubi, Xiang-Yang Li, Yu Wang, Peng-Jun Wan, and Ophir Frieder. Geometric spanners for wireless ad hoc networks. *ACM Transactions on Parallel and Distributed Systems*, 14(5), 2003.

[105] Hari Balakrishnan, Christopher L. Barrett, V. S. Anil Kumar, Madhav V. Marathe, and Shripad Thite. The Distance-2 Matching Problem and its Relationship to the MAC-Layer Capacity of Ad Hoc Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1069–1079, August 2004.

[106] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[107] C. M. Fortuin, F. Ginibre, and P. N. Kasteleyn. Correlational inequalities for partially ordered sets. *Communications of Mathematical Physics*, 1971.

[108] H. Shachnai and A. Srinivasan. Finding large independent sets in graphs and hypergraphs. In *ACM Symposium on Parallel Algorithms and Architectures*, 2001.