5-1-2008

Mesh-Mon: a Monitoring and Management System for Wireless Mesh Networks

Soumendra Nanda
Dartmouth College

Dartmouth Computer Science Technical Report TR2008-619

Mesh-Mon: a Monitoring and Management System for Wireless Mesh Networks

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Soumendra Nanda

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 2008

Examining Committee:

David Kotz (chair)

Andrew Campbell

Robert L. Drysdale

Robert S. Gray

Charles K. Barlowe
Dean of Graduate Studies

Abstract

A mesh network is a network of wireless routers that employ multi-hop routing and can be used to provide network access for mobile clients. Mobile mesh networks can be deployed rapidly to provide an alternate communication infrastructure for emergency response operations in areas with limited or damaged infrastructure.

In this dissertation, we present Dart-Mesh: a Linux-based layer-3 dual-radio two-tiered mesh network that provides complete 802.11b coverage in the Sudikoff Lab for Computer Science at Dartmouth College. We faced several challenges in building, testing, monitoring and managing this network. These challenges motivated us to design and implement Mesh-Mon, a network monitoring system to aid system administrators in the management of a mobile mesh network. Mesh-Mon is a scalable, distributed and decentralized management system in which mesh nodes cooperate in a proactive manner to help detect, diagnose and resolve network problems automatically. Mesh-Mon is independent of the routing protocol used by the mesh routing layer and can function even if the routing protocol fails. We demonstrate this feature by running Mesh-Mon on two versions of Dart-Mesh, one running on AODV (a reactive mesh routing protocol) and the second running on OLSR (a proactive mesh routing protocol) in separate experiments.

Mobility can cause links to break, leading to disconnected partitions. We identify critical nodes in the network, whose failure may cause a partition. We introduce two new metrics based on social-network analysis: the Localized Bridging Centrality (LBC) metric and the Localized Load-aware Bridging Centrality (LLBC) metric, that can identify critical nodes efficiently and in a fully distributed manner.

We run a monitoring component on client nodes, called Mesh-Mon-Ami, which also assists Mesh-Mon nodes in the dissemination of management information between physically disconnected partitions, by acting as carriers for management data.

We conclude, from our experimental evaluation on our 16-node Dart-Mesh testbed, that our system solves several management challenges in a scalable manner, and is a useful and effective tool for monitoring and managing real-world mesh networks.

Acknowledgements

First and foremost, I thank my advisor, David Kotz. Without his support, patience and expert guidance, I would not have been able to complete this thesis. By working closely with him, I have become a better researcher and have improved my communication and writing skills.

I thank all my thesis committee members, Andrew Campbell, Scot Drysdale and Bob Gray for their help. I thank Samir Das for his feedback on my initial thesis proposal. I am very grateful to Bob for his guidance during my MS thesis, and to Andrew for his feedback on my journal paper and for his general support.

Wayne Allen and Michael Chen, from Intel Corporation, helped in the initial setup of the mesh. I am grateful for the help and feedback that I received from Tristan Henderson, Kazahiro Minami, and Arnab Paul in the early stages of this thesis. David Bourque and Chris McDonald helped me understand iptables. Chris also helped me several times in resolving bugs in my code and provided the timer code, which is an important part of the Mesh-Mon engine. I thank Ron Peterson, Yong Sheng and Bennet Vance for helping me debug my code. Bennet, in particular has always listened to my problems with a patient ear and helped me on numerous occasions. I thank Tim Tregubov, who has always been there when I needed help with repairing or upgrading my mesh hardare and software. I thank Judith Hertog for helping me to improve my writing skills.

I would not have been able to deploy my mesh without the support of our department chair; Prasad Jayanti, and our amazing system administrators; Wayne Cripps, Tim Tregubov and Colin Brash. I thank my CMC lab-members, department administrators, especially Kelly Clark and all my friends for their help in deploying the mesh network. I thank Devin Balkcom, Andrew Campbell, Sean Smith, and Jamie Ford, for volunteering to host my mesh devices in their own labs. I am very thankful for all the amazing Dartmouth faculty members and colleagues, from whom I have learnt so much in these last 7 years at Dartmouth. I thank all my friends who have made my years in Hanover so special and truly memorable.

I thank my entire family and especially my loving parents, Kamales and Aparna Nanda, who have always believed in me.

Last but not least, I wish to thank my beloved wife, Cindy Torres, for her unwavering support and faith in me. I could never have finished this endeavor without her help. I dedicate this thesis to her.

Finally, I am grateful for support from several sponsors. This research program was a part of the Institute for Security Technology Studies (ISTS), supported by a gift from the Intel Corporation, by Award number 2000-DT-CX-K001 from the U.S. Department of Homeland Security (Science and Technology Directorate) and by Grant number 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance. Points of view in this document are those of the author and do not necessarily represent the official position or policies of any of the sponsors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Imagine that a large-scale disaster takes place, such as a plane crash in a remote forested location, a desert, in rugged terrain, or in a rural area with no pre-existing infrastructure.

First Responders (FRs) is the collective name given to professionals who (as the name suggests) are the first to respond to an emergency situation. Typically an FR team will comprise of firefighters, police officers, paramedics, specialized personnel and possibly military forces.

FRs require an inter-operable and secure means to communicate with each other, and with the local incident command center or the remote central command office. The central command needs to collect all the information about the disaster site and remotely monitor the current status of the personnel active at the site.

Thus, upon arrival in a remote location, the FRs would need to immediately set up a communication infrastructure to coordinate personnel and to begin their relief and rescue efforts. FRs would also have the same requirements in an urban location that is pre-equipped with some form of communication infrastructure intended to support emergency response operations, since there is always a possibility that an unforeseen disaster will damage the pre-existing infrastructure, limiting its potential utility.

In a large-scale disaster, such as the one that occurred in New York on 9/11, it was observed that the traditional means of communications were severely disrupted or failed completely due to a lack of redundancy. Service personnel found that their voice radio systems were overloaded and often faced incompatibilities across different teams. The employed communication infrastructure did not scale well to the needs of the

large number of diverse FR groups at the site. FRs need a rich information flow both to and from, as well as within, their teams.

Given this current state of affairs, there is a serious need to develop an alternative robust communication system that can operate even in unforeseen and unpredictable situations. We believe that these scenarios dictate the need for a robust high-capacity unified wireless network that can be instantly deployed to provide new infrastructure, and to augment a damaged existing system. This new network must be capable of supporting the demanding voice and data requirements for tomorrow's FRs and must be designed to scale with an increasing number of users whenever necessary.

Cellular wireless-communication systems are one option, but they do not provide uniform or guaranteed coverage in all areas, inside buildings or underground. Current generation cellular networks do not support high-speed downloads or uploads. They also require considerable time and infrastructure to set up.

Ad hoc and mesh networks are types of wireless networks that can be rapidly deployed. They present a potential solution for deploying a high-performance data and voice network for First Responders in scenarios where there is limited or no infrastructure available, or the existing infrastructure is destroyed, inoperable, or overwhelmed. These networks are equally well suited for emergency-response operations and battle-field situations and can also provide several value-added digital features that were previously unavailable in traditional voice-only radio networks.

Mesh networks are typically characterized by a high degree of redundancy in the number of available paths for data to travel through. Thus, even if a few individual nodes fail, the network as a whole may still remain operational, and data may be rerouted on the fly through alternate paths.

We assume that a FR mesh is formed by rapid deployment of portable multi-radio Mesh Nodes (MNs). These nodes may be mounted on vehicles such as fire trucks, ambulances and police cars that arrive at the scene. Several MNs form the mesh backbone of the communication infrastructure in an ad hoc manner. Additional MNs may be deployed in strategic locations, say, on different floors of a building by humans or even robots [100]. Mobile Clients (MCs) such as PDAs, laptops or sensors can then associate with the nearest deployed MN to send information in a multi-hop manner.

Several challenges to managing such a dynamic network infrastructure must be addressed to ensure sustainable and reliable operation of the network. We assume that one or more mobile system administrators

(sysadmins) are present at the deployment site and are responsible for managing the network.

The goal of this thesis is to develop a network monitoring and management solution capable of automatically managing, or assisting a team of system administrators in the management of exactly such a rapidly deployed, unplanned and potentially mobile wireless mesh network. To accomplish this goal, we designed and implemented Mesh-Mon, a mesh networking architecture that runs on Linux, and Dart-Mesh, our 16 node two-tier wireless mesh network that we implemented using commodity hardware and deployed in the Sudikoff Laboratory for Computer Science at Dartmouth College.

1.1 Wireless networks

In this section, we briefly explain the differences between wireless ad hoc networks, Wi-Fi networks, and wireless mesh networks.

1.1.1 Ad hoc networks

Ad hoc networks are wireless networks that can be deployed quickly. A mobile ad hoc network (MANET), is a potential solution for deploying a data and voice network for FRs in scenarios with limited or no infrastructure. All nodes in the network act as equal peers (see Figure 1.1). An ad hoc network typically uses a suitable routing protocol such as AODV or DSR to allow multi-hop communication between the nodes. One or more nodes may act as gateways and connect the ad hoc network with the Internet or some other network. The gateways may use a wired or even a a wireless back-haul to exchange data with an external network. Self-contained ad hoc networks may not even have a back-haul. Ad hoc networks generally have only single-radio nodes and can be considered to be single-tier mesh networks.

1.1.2 Infrastructure-based wireless networks

An infrastructure-based wireless local area network (WLAN) is often called a Wi-Fi network. A WLAN is typically made of several Access Points (APs) connected to a central switch through a high-capacity wired backbone (see Figure 1.2). The switch is typically connected to the corporate intranet or the Internet by a high-speed wired connection. For the wireless network at Dartmouth College, the wired backbone is a gigabit Ethernet.

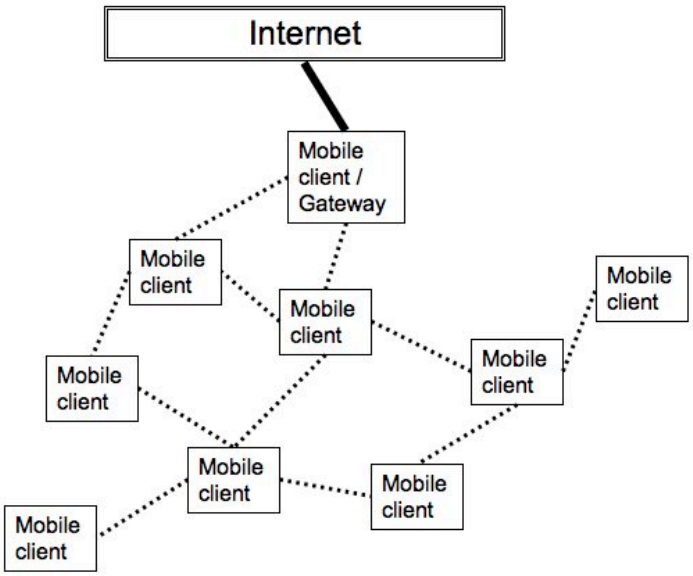

Figure 1.1: Ad hoc network with wired or wireless back-haul

Such networks have gained immense popularity in the last ten years. Wi-Fi networks have become ubiquitous on many college campuses and in many corporations. The APs provide wireless connectivity to mobile clients (typically laptops and PDAs) using IEEE standards based technologies such as 802.11a, 802.11b and 802.11g. These technologies have theoretical peak download speeds that range between 11 Mbps and 56 Mbps. Some devices based on newer standards (IEEE 802.11n) provide even higher theoretical peak download speeds (up to 108 Mbps).

Most 802.11 wireless APs provide coverage for a radius of approximately 100 meters in free space. However, coverage is significantly reduced indoors and depends on the layout of the building, building materials, signal strength, radio frequencies, type of antenna used, physical obstructions and other factors. Actual performance depends on the configuration, network congestion and interference, and is often significantly lower than manufacturer's specifications. Most Wi-Fi equipment vendors provide proprietary mechanisms that allow mobile clients to roam from one AP to another without losing connectivity.

WiMAX (Worldwide Interoperability for Microwave Access) is the IEEE 802.16 standards-based wire-

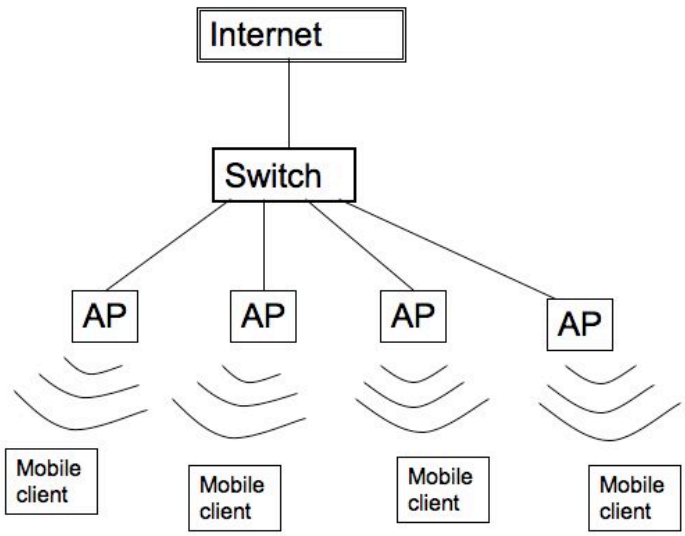

Figure 1.2: Infrastructure-based WLAN with wired back-haul and wired backbone

less technology that provides MAN (Metropolitan Area Network) broadband connectivity. WiMAX is a newer wireless technology standard which may one day replace Wi-Fi. WiMAX supports higher throughput speeds than Wi-Fi and has a longer range and coverage area (about 15 kilometers in ideal conditions) than Wi-Fi. In Korea, Wi-Bro (the Korean equivalent of WiMAX) is already operational and is a popular platform for delivering new pervasive media applications to consumers. Sprint intends to deploy WiMAX networks in several regions of America in 2008.

1.1.3 Mesh networks

There is no universal definition for a wireless mesh network. A mesh network is an ad hoc network that employs one of two connection arrangements: full mesh topology or partial mesh topology. In the full mesh topology, each node is connected directly to each other node. In the partial mesh topology, nodes are connected to only some, but not all the other nodes. Wireless mesh networks thus combine a mesh topology with ad hoc wireless network characteristics.

Mesh networks can be built on a mix of fixed and mobile nodes interconnected via wireless links to form a multi-hop ad hoc network. In a sufficiently dense wireless mesh network, there is rarely a single point of failure and the network is able to recover from the failure of an individual node by routing around it to other nodes in a direct or multi-hop manner.

A mesh network allows nodes to communicate with other nodes without being routed through a central switch point, thus eliminating centralized failure and providing self-healing and self-organization behavior. As shown in Figure 1.3, several mesh routers (MRs) act as APs to the mobile clients. The mesh provides a wireless infrastructure to mobile clients using an ad hoc wireless backbone to route packets to their respective destinations. One or more mesh routers can act as gateways for the back-haul traffic to external networks, and may use a wired or wireless connection.

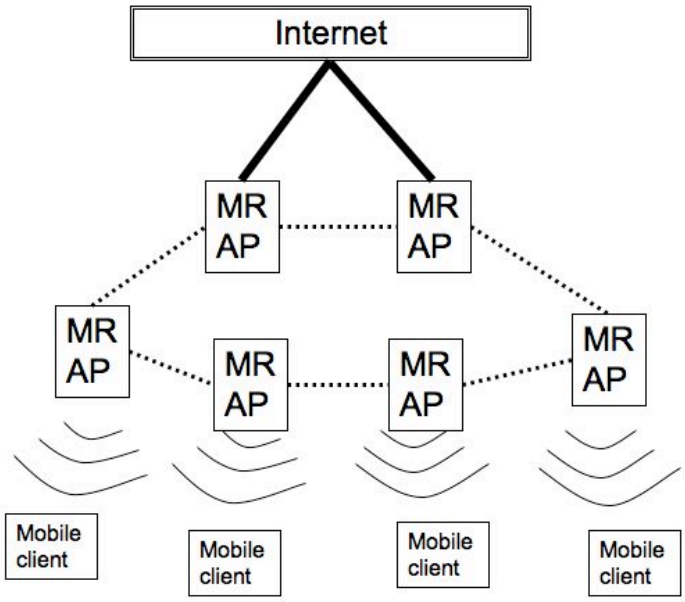

Figure 1.3: Wireless mesh network

An ad hoc wireless network protocol, such as Ad hoc On-demand Distance Vector routing (AODV) [136], Optimized Link State Routing (OLSR) [40], or Dynamic Source Routing (DSR) [92], is typically used for routing packets in the network between mesh routers. There exist over 70 potential protocols that could be

used for routing in a mesh network [4]. Path diversity and redundancy are the key features of a successful mesh network.

Two-tier mesh networks, such as that shown in Figure 1.3, avoid the completely self-organized nature of ad hoc networks. Such networks have two or more radios. Typically one radio is used in the "access tier" as an AP to provide wireless service to mobile clients in the area connected to it (just like in a Wi-Fi access point). The second radio is used in the "mesh tier" by the mesh nodes to form a wireless backbone, using a suitable mesh routing protocol. In this context, the term "mesh routing protocol" is synonymous with the term "ad hoc routing protocol". Thus, each mesh node in a two-tier wireless mesh network provides the functionality of both an access point as well as router.

The back-haul (or mesh side) of the mesh node wirelessly relays the traffic from mesh router to mesh router until it reaches a gateway. The gateway node then connects to the Internet or to another private network (through a wired connection). Another option is to use an alternative technology such as satellite or cellular network links, to wirelessly connect the mesh network with an external network.

We use this two-tier model in Dart-Mesh, our mesh implementation testbed (which is explained in detail in Chapter 2). One benefit of this architecture is that for end users the experience of joining such a network is identical to the process for a client joining a Wi-Fi infrastructure-based WLAN. This process is well supported by all compatible wireless client devices that support the published standards of the Wi-Fi device manufacturers consortium (the Wi-Fi Alliance), which ensures hardware inter-operability [172]. The clients do not need to run any special routing software or configure the radio to run in "ad hoc mode." Indeed, the "ad hoc mode" is not fully supported by many device manufacturers, device drivers and operating systems, and configuring a wireless card to use this mode is not an easy process.

Mesh network products using this two-tier architecture are available from several commercial vendors. Some vendors produce mesh devices with a third radio or even a fourth radio [115] to split the backbone traffic along several orthogonal channels to increase network capacity. Some mesh systems run the routing protocol at layer-3 while others do it at layer-2 or even in an intermediate layer-2.5.

Several cities in the U.S. (such as Philadelphia, Las Vegas and Houston) and around the world (Taipei, Helsinki) have deployed commercial mesh and Wi-Fi networks from several vendors. Some of these deployments use single-radio solutions, while others use multi-radio solutions.

While most mesh deployments are proprietary, there are several research [16] and community-based efforts that use open-source technologies. Locust World is a company that provides a free-to-download bootable Linux CD which converts a laptop into a AODV based "LocustWorld MeshAP" node [107]. The entire LocustWorld distribution is open source. The web-based management tools which they provide, however are proprietary. Meraki Networks is a company that was recently started by former students from the Massachusetts Institute of Technology (MIT), and sells an inexpensive Linux-based mesh-solution [140]. Their management solution too is web-based, centralized in design, and proprietary. Freifunk (which is German for "free radio" network) is a popular community-based OLSR mesh network solution, used by several hundreds of users on a voluntary basis to form community mesh networks [180].

The One Laptop Per Child (OLPC) initiative is a project from MIT that aims to provide low cost laptops to children in developing nations [126]. The OLPC laptop design includes support for mesh networking built into each laptop; the OLPC project exemplifies the hope that low cost laptops and mesh networks can help bridge the "digital class divide".

1.2 Research challenges for a mobile mesh

Several challenges need to be overcome to realize a wireless mesh network suitable for First Responders. The overarching challenge is to engineer and build a wireless mesh network that provides all the desired characteristics in a cost effective manner. We enumerate some of the desirable characteristics of a mobile mesh network and their corresponding challenges:

1. Robustness: The mesh network should be functional even if a few individual nodes fail. For example, in a hazardous scenario a fire may wipe out some sensors and nodes. Thus mesh nodes should be capable of operating in extreme conditions and have sufficient battery life. Mesh nodes which are heat and environment resistant are likely to be expensive to manufacture. Larger and heavier nodes could reduce mobility, making deployment harder.

2. Reliability: The network must maintain connectivity between the majority of critical nodes during the operation in spite of the potential failure of individual nodes. Most client nodes have limited range and limited power, so depending on the scenario we may wish to optimize for a network that functions

in a limited manner for a long time or one in which all nodes run for a shorter time with full power and maximum connectivity. Node mobility can cause links to break. Environmental conditions and physical obstructions can cause temporary or permanent link interruptions.

3. Service quality: The network should provide enough bandwidth to meet the needs of all users or at least high-priority users. The traffic patterns of data in this network will depend on the characteristics of the individual applications that run on each client.

4. Inter-operability: Communication must be secure yet allow different agencies to communicate with each other when operations require interdepartmental cooperation.

5. Configuration: The users would like the network to be quick to set up (almost instantaneous) with preferably zero-on-site configuration. The solution must be easy to deploy. For example, the network manager must ensure that all nodes have the necessary encryption keys and that all nodes are configured correctly with the right settings.

6. Management: We should be able to maintain connectivity and desired network performance. We need to monitor the mesh network to become aware of any problems with the nodes, links or the routing of traffic. We need to identify potential problem areas, nodes that may need assistance or replacement, and help the administrator choose the right corrective action. For example, a node that is about to run out of power should be identified, so someone can manually recharge it or to inform other nodes, to allow them to find alternate routes around the weaker node.

   Monitoring the status of many nodes in a scalable manner is a challenge. Management traffic should not affect the network traffic yet be frequent enough to be meaningful and adaptive to changing network conditions.

7. Location Information: Awareness of the location of each FR and each network device is useful for managing the situation on-site. Location can be determined by Global Positioning Systems (GPS) outdoors and by localization of signal strength of neighboring nodes and wireless routers (indoors). Current GPS technologies do not work well indoors or underground.

8. Security: Depending on the nature of the deployment, privacy, confidentiality and integrity require-

ments of transmitted data must be ensured. Robustness against jamming and attacks or malicious users is desirable. It may be impossible to guarantee resilience in the face of physical layer jamming and other possible attacks.

9. Efficient Routing: Control overhead to route packets in the network should be as low as possible and the routing protocol should provide desired performance. Several mesh routing protocols that use different techniques and optimizations are available (over 70 in fact). It is impossible to qualitatively compare each protocol to decide which is optimal for a given situation or is the best option for all dynamic situations.

10. Updates: It should be possible to update the software on the mobile nodes remotely when required. In a FR scenario updates are likely to occur rarely (if at all) during an event.

    The mechanism for doing a code update can be complicated. The update operation must be fail-safe and should not disrupt the functionality of other nodes in the network or the network traffic. The update mechanism must be designed to work in spite of potential node failures or the network link failures during an update. The mechanism should work even in the case of a mixture of updated and old nodes for a short period of time during the update.

11. Isolation: FRs should not have to share bandwidth with civilians, who may flood the network in an emergency setting. The mesh network should not share the same radio spectrum as civilians, since uncontrolled civilian traffic can flood the shared channel with non-emergency traffic. If FRs use a system isolated from available infrastructure, they may miss out on the potential of using pre-existing infrastructure for routing the network traffic. FR data could be sent at a higher priority or all non-FR users could be disconnected from a pre-existing network (say, a wireless mesh used for community Internet service).

1.2.1 Summary

Although there are many interesting challenges (some technical and some policy-related), resolving all of them is beyond the scope of this thesis and we choose to focus on mesh network management. While we focus our design on the assumption that mesh nodes may be mobile, our solutions are equally applicable to

stationary mesh networks.

1.3 Network management

Network management means different things to different people. Network management is broadly defined as the maintenance and administration of a computer network or telecommunication system. We found this definition of what is network management attributed to Saydam [156] in a popular text book on computer networks [101]:

> "Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

The International Standards Organization (ISO) Telecommunications Management Network model is a widely accepted framework for network management. The ISO model describes the five main conceptual areas of management (commonly abbreviated as FCAPS) as follows: Fault management, Configuration management, Accounting management, Performance management, and Security management. In non-billing organizations Accounting management is sometimes replaced with Administration management.

Fault management

The goal of fault management is to log, detect and respond to fault conditions of the network.

Configuration management

The goal of configuration management is too allow network managers track and modify the hardware and software configurations of devices in the network being managed.

Accounting / Administration management

Accounting of resource usage is traditionally performed in networks for the purpose of billing users. In non billing systems, the goal is to administer the set of authorized users by establishing users, passwords, and permissions, and to administer the operations of the equipment such as by performing software backups and synchronization.

Performance management

The goal of performance management is to quantify, measure, report, analyze, and control the performance of different network components over extended periods of time.

Security management

The goal of security management is to control access to network resources.

Our focus in this thesis is on fault, performance, administration and configuration management for a mobile multi-radio mesh networks. We do not attempt to solve security management issues in this thesis but we discuss relevant security issues in Chapter 6.

1.4 Mesh network management challenges

In general, system administrators for wired networks typically deal with absolute problems. A switch, a hub or a wire is either broken or working in a normal condition. The tools that have been developed to deal with management of such networks implicitly assume the network is static, changes slowly (often in a planned manner) and that devices are stationary. Thus, tools designed and developed for the management of wired networks are unsuitable for mobile mesh networks.

There are many differences between monitoring and managing a wired network or even an infrastructure-based Wireless Local Area Network (WLAN) versus managing an emergency-response mesh network. In an infrastructure-based WLAN, we typically have few constraints on bandwidth because the APs are connected by a wired backbone. Wired links are always more reliable than the wireless links in a wireless mesh backbone. If the network topology is static (as it is in wired networks and Wi-Fi WLANs) and well defined, most faults and problems can be detected, isolated and resolved easily using centralized network mapping and monitoring tools. In a mesh network with dynamic links, however, this is not always the case.

As mentioned earlier, several commercial vendors sell wireless mesh networking products of different types. Most commercial mesh vendors offer proprietary solutions for wireless mesh-network management [39, 131, 141, 179]. These solutions are vendor specific, and little is known about the management

solutions they employ. Furthermore, they are typically designed for a well-planned mesh network with stationary nodes. Some vendors do provide SNMP compatibility in their products to allow generic Network Management Systems (NMS) software to monitor their devices.

Sysadmins for a mesh must manage the network based on the goals of the deployment and the available resources at their disposal. These goals may be related to area of coverage, duration of the operation, and service quality requirements in addition to the characteristics of the deployment area.

In our FR mesh scenario, we assume that links may be unstable and have limited bandwidth. Mobility, node failures or link failures can create partitions. Some regions may have "holes" where no wireless coverage is available for clients. Nodes may malfunction or be incorrectly configured. The MCs may have difficulty connecting to MNs for several reasons. We posit that managing such a network requires a management architecture that is capable of adapting to these dynamic conditions and diverse failures. Conventional management tools were never designed for this purpose.

Network management requires an understanding of the state of the network and this understanding is derived from network monitoring. We believe a proactive approach is best for monitoring a mobile mesh network for several reasons. Problems and faults need to be anticipated before they occur in a mobile wireless mesh network. A single problem itself could lead to a fractured network and thus prevent the network manager from gathering information about the original causes of the very problem that we wish to investigate, diagnose and eventually eliminate. A reactive approach is not viable in such situations and would potentially lead to longer diagnosis times and possibly incorrect diagnosis. Thus, we advocate proactively gathering as much information as possible before an undesirable network incident occurs. We must also try to take into account in our analysis the possibility that the information we currently possess is outdated or inaccurate, since the network or parts of it may be constantly changing due to mobility, failure or reconfiguration.

1.5 Mesh-Mon

In this thesis, we designed and developed Mesh-Mon, a network monitoring and management system. We built our system from the ground up to solve several management challenges for a mobile mesh network. Mesh-Mon consists of many inter-related components that perform several tasks in an integrated manner, such as local information collection and sensing, storage of local and neighborhood information, partition-

aware communication of network information between components, and distributed analysis of the state of the network. Mesh-Mon software runs on each mesh node that is a member of our managed mesh network.

The mesh routing protocol is the most significant software component of any mesh network. During our laboratory experiments, we observed many instances where the mesh routing protocol implementation did not perform correctly or as expected and often made communication between individual mesh nodes unreliable (if not impossible). Thus, any mesh monitoring system that relies solely on the specific underlying routing protocol will be less useful, or even fail to operate, if the mesh routing protocol fails. We thus set out to design a monitoring solution for mesh networks that was independent of any weaknesses of the mesh routing protocol.

In Mesh-Mon, we can replace the underlying routing protocol used in our experimental testbed with any other protocol (proactive, reactive or hybrid) and Mesh-Mon can monitor and manage the network with practically no modification. We designed Mesh-Mon to work independently of the underlying mesh routing protocol and we consider this feature to be one if its strengths, since the mesh routing protocol is the key point of differentiation between different mesh networks.

Mesh-Mon cannot guarantee delivery of management packets if the hardware fails or if there are disconnected partitions. Mesh nodes that are in different partitions self-organize and cooperate to help manage all reachable nodes. We allow mobile clients to assist in the delivery of management packets across disconnected partitions with a tool called Mesh-Mon-Ami that runs on each client of the mesh. Mesh-Mon-Ami is an optional client-side component of the Mesh-Mon system.

Mesh-Mon has several other unique features, such as the use of novel social network analysis-based centrality metrics to solve relative management problems and a fully distributed design, that differentiate it from other existing and proposed mesh network management solutions.

1.6 Contributions

We summarize the contributions of this thesis below:

- In this thesis, we implement Dart-Mesh, a two-tier wireless mesh network that provides reliable wireless broadband access for mobile clients with performance comparable to that of a 802.11b Wi-Fi

network.

- We design and implement Mesh-Mon, a distributed and scalable mesh network monitoring and management architecture that helps system administrators monitor and manage a mobile mesh to ensure its sustainable operation.

- We demonstrate Mesh-Mon's ability to adapt to the management needs of two extremely different mesh networks running different mesh routing protocols, and its potential to manage mesh networks running any routing protocol.

- We evaluate Mesh-Mon and demonstrate our success in resolving several management challenges. Mesh-Mon nodes cooperate and can often can resolve problems autonomously.

- We apply novel techniques derived from social-network analysis to analyze mesh networks to reveal relative information about different nodes that are useful from a management perspective.

- We introduce the localized bridging centrality (LBC) metric. This new centrality metric is an improvement over the bridging centrality metric, since our metric can be calculated in a distributed manner, yet provides equally effective results. We also develop the localized load-aware bridging centrality (LLBC) metric and demonstrate its effectiveness in identifying critical bridging nodes in networks with non-uniform traffic flows.

- All the results we present are from experiments and not from simulations. While simulations are useful for research, they cannot capture the complexity of real-world network management challenges.

1.7 Dissertation outline

The rest of the thesis is organized as follows:

Chapter 2 introduces the Dart-Mesh testbed we built after testing several prototype designs.

Chapter 3 details our design philosophy for Mesh-Mon; our management framework and its implementation details. We provide details of each component and the algorithms involved.

Chapter 4 describes our experimental results that demonstrate the effectiveness of our system and our new centrality metrics.

Chapter 5 compares our system with related work in this domain.

Chapter 6 discusses the strengths and limitations of our system, unresolved challenges, and suggestions for future work.

Finally Chapter 7 presents our conclusion.

Chapter 2

Dart-Mesh

In this chapter, we explain the system-level design and technical details of our mesh network architecture. Our focus in this thesis is on mesh network monitoring and management. Network simulations are a useful technique for evaluating network designs and configurations that may be difficult or prohibitively expensive to actually implement. Simulations, however, are not accurate enough or detailed enough to expose the true management challenges that emerge in a real mesh network. Thus, we were interested in implementing a network monitoring and management architecture on a real working two-tier mesh network.

Initially, we did not intend to build our own mesh network architecture. As part of a generous gift from the Intel corporation, we received 16 nodes from an experimental test-bed and related mesh software. We intended to build a mesh networking management system on top of the received Intel mesh system.

Unfortunately, after we tested the received nodes, we discovered several shortcomings in terms of the performance and the usability of the received hardware and software. We soon realized we needed to build a better platform that was stable and usable.

We faced several challenges in building a stable two-tier mesh network. We experimented with several hardware and software prototypes until we succeeded in building "Dart-Mesh," our stable mesh testbed. We provide details of the original Intel-built mesh system followed by details of our Dart-Mesh architecture and other experimental prototypes.

2.1 Intel experimental mesh testbed

We received 16 Intel mesh nodes in April 2005 (see Figure 2.1). These nodes were built by Intel for a trial project with 31 mesh nodes to provide Internet access to a rural residential community in Lyme, NH (located close to the Dartmouth College campus in Hanover, NH). After their short deployment exercise, the mesh nodes were provided to us on an as-is basis, and we were permitted to change them in any way we saw fit.

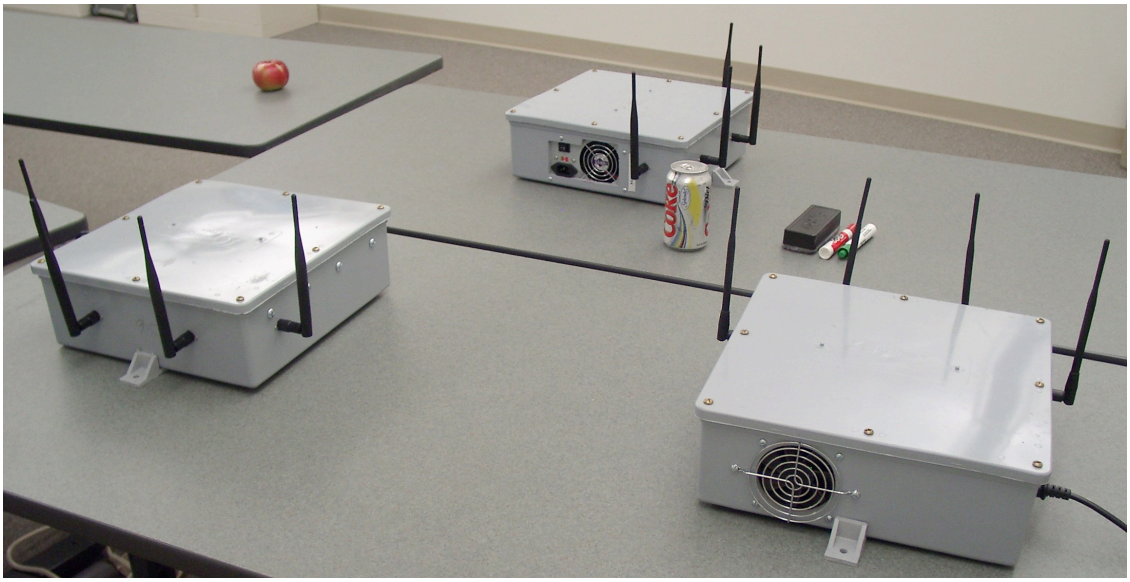

Figure 2.1: Three of the Intel mesh nodes

We were given limited information on how they were previously used or how they actually worked. Through a series of email conversations with the former deployers, analysis of startup scripts, experiments and reverse-engineering, we discovered how the mesh operated. We received more information about the original deployment after the information was published in November 2005 [5]. By then, we had concluded that the supplied configuration performed poorly, and that there were some serious challenges to its usability, which is why we began pursuing alternate designs for our own mesh testbed.

2.1.1 Hardware platform

The Intel mesh devices were originally designed to be mounted indoors in a rural environment and provide Internet access to stationary residents inside an apartment complex. Each Intel mesh node was identical and was mostly composed of off-the-shelf components. Each node had a 1.2 GHz Intel Pentium III processor,

256 MB of Random Access Memory (RAM) and a 80 GB hard disk drive. The system used a small-form-factor Soekris motherboard equipped with three Mini-Peripheral Connect Interface (PCI) slots. Each node was equipped with three Mini-PCI radios with connectors for external antennas that were mounted outside the enclosure. Two of the radios used the Prism2 chipset and supported only 802.11b while the third radio used the Atheros chipset and supported 802.11a/b/g. Each mesh node had an ATX sized power supply and was enclosed in a 12 inch square gray poly vinyl chloride (PVC) plastic case enclosure. The power supply had a thermostatically controlled fan that supposedly operated only when cooling was needed. The case enclosure had an additional fan to help provide cooling and ventilation to the system. The motherboard supported a compact flash expansion slot and had 3 Ethernet ports that were unused in the original deployment.

Once the PVC case was sealed shut using special screws, the only way to configure the device was by logging in through a wireless connection and a secure shell (ssh) client. When the case was physically unsealed and the cover removed, it was possible to connect a keyboard and VGA monitor to the motherboard by using a specialized connector, but we could not connect a mouse. When the mesh box was switched on and powered up, the fans in the system ran constantly while making a loud noise. We were concerned that the noise issue could be a major detriment to the deployment of these nodes inside a person's office.

2.1.2 Software details

The mesh nodes came installed with Redhat Linux version 9. The mesh routing protocol used was kernel AODV v2.1 [96] developed at the National Institute of Standards and Technologies (NIST). We were not provided with any of the management tools or experimental code, such as the ETX [50] extensions for AODV developed for the previous deployment [5]. The Prism2 cards used the HostAP driver [110] and the Atheros card used the MADWiFi driver [103].

We set up a small mesh testbed in our lab with up to 8 nodes. We tested the devices, learnt how to configure and re-program them. The mesh nodes were designed to run autonomously once switched on. A startup script initialized each device on bootup. All required network interfaces on each device were configured at boot-time and essential services such as `dhcpd`, `aodv` and `iptables` were started. Software updates (if any) had to be provided wirelessly through an available wireless interface, unless the

node also had an Ethernet connection.

Programming a fully wireless device is a tricky affair. As long as we did not make any changes to the initialization scripts while experimenting with the mesh nodes, a simple reboot of the node would remove our changes and restore the mesh to its original or previous configuration (which was hopefully stable). However, if we made a critical mistake in the initialization scripts or in the programs called by the initialization scripts, then we were essentially locked out of the device. The only option to repair such mistakes was to physically pry open the box and to log in after connecting a keyboard and monitor with the specialized connector. This manual process was time consuming and laborious.

2.1.3 AODV

We present a brief introduction to the AODV protocol since it is an important component of the Intel Mesh architecture and is also used in experiments on Dart-Mesh. Perkins and Royer introduced the Ad hoc On-demand Distance Vector Routing (AODV) [137] in 1999. The AODV protocol is in the process of being standardized at the IETF and its specifications are published as an experimental Internet Engineering Task Force (IETF) Request For Comments (RFC) document [136].

AODV is a reactive routing protocol and thus builds routes only when nodes require them and not in advance. AODV uses sequence numbers to ensure the freshness of routes. AODV is loop-free, self-starting, and scales to large numbers of mobile nodes. AODV is often used as a benchmark by which other ad hoc routing protocols are measured.

AODV builds routes using a route request and route reply cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a route request (RREQ) packet across the network. When a source needs a route to a destination, it initiates a route discovery process by flooding route request packets throughout the network to search for a path to the destination. Upon receiving this packet, nodes update their routing table information and set up reverse pointers to the source node in their route tables. The RREQ packet can be uniquely identified by a sequence number so that duplicate RREQs can be recognized and discarded. Upon receiving non-duplicate RREQ, an intermediate node records the previous hop and checks whether there is a valid and fresh route to the destination in its own local route table. If this is the case, the node sends back a route reply (RREP) packet to the source; otherwise it rebroadcasts

the RREQ packet. As the RREP packet traverses though the selected route, each node along the path sets up a forward pointer, updates corresponding timeout information and records the latest destination sequence number (for checking the freshness of the route).

For path maintenance, link failures are detected by a mechanism based on the periodic exchange of short "Hello" message packets. When a route failure is detected, a route error (RERR) packet is sent back to all sources to erase route entries using the failed link. A route rediscovery procedure is initiated if the route is still needed.

2.1.4 Network architecture

The Intel mesh nodes used the two 802.11b radios for their primary operation. Each radio interface was assigned a static Internet addressable Internet Protocol (IP) address from a pool of addresses reserved for the mesh deployment. The first radio was set up as the *Mesh-interface* in "pseudo-IBSS ad hoc mode" and formed the mesh backbone by routing packets between mesh nodes using AODV. This interface was used to dynamically form point-to-point links. The second 802.11b radio was setup as the *AP-interface* in "Master mode" using the HostAP driver. This interface allowed clients (such as laptops or PDAs) to associate with the mesh and receive an IP address from the local Dynamic Host Configuration Protocol (DHCP) server. This interface was used to form static multipoint-to-point links. Each mesh node was capable of supporting at most six clients, since the DHCP server had a pool of only six IP addresses. Each mesh node announced a unique ESSID.

The mesh nodes in general did not use the third 802.11a/b/g radio, unless that node was a gateway node. The third 802.11a radio was used as the *Internet-interface* in "Managed" mode to connect to a wired Internet Gateway router, thus providing Internet connectivity to the mesh.

Static routes in the form of `iptables` entries were added to the routing tables on individual mesh nodes to route traffic between interfaces on each mesh node, and between mesh nodes and the fixed gateway.

2.1.5 Initial usage experience

We tested different topologies with single-hop and multi-hop routes in our own lab testbed. We used Linux and Mac laptops as mesh clients. We ran ttcp [118] and Iperf [175] throughput tests between clients con-

nected to the same mesh box or to different mesh boxes. We employed laptops as wireless packet sniffers running tcpdump and ethereal. We used a tool called EtherApe [176] to visualize the traffic flows.

The first tests we tried were simple ping tests to see if each mesh box could ping each other. Similarly, we tested the clients' ability to ping other clients through the mesh. Although the initial ping tests were successful, as we did more tests we started noticing several performance issues. The network would abruptly stop working after a period of seemingly normal operation. It was challenging to discern what was the root cause of any fault (especially for transient faults).

When we ran multi-hop experiments (such as a string topology from client1 to meshbox1 to meshbox2 to client2) and attempted to do a TCP or UDP `Iperf` test from client1 to client2, we often did not observe any data transfer taking place until we manually started a background ping between client1 and client2. As soon as the ping would start we were able to observe throughput as high as 400 kbps between the clients. If we stopped the ping, then the problem would reappear after approximately 30 seconds,

When using a client to connect to the Internet through the mesh, the performance was inconsistent. DNS resolution often failed. Connections were unstable and often timed out unexpectedly.

When we used Prism2, Atheros or Cisco chipset based wireless cards in the Linux clients that connected to the mesh, we observed reasonably high throughput (above 200 Kbps on average) depending on the number of hops in the path. In identical scenarios, when we used Lucent Orinoco chipset based cards, the throughput would drop to as low as 20 Kbps (and sometimes even lower). The same Lucent cards in the same laptops were able to get good throughput numbers when connected to regular Wi-Fi access points. Hence, we were puzzled by this non-uniform behavior.

The mesh used pre-assigned static routes to the Internet gateway. There was no support for multiple gateways or a mechanism to find an alternate gateway when the primary gateway failed.

We investigated deeper and after extensive debugging we discovered that the main cause of most failures pointed to the implementation of AODV we were using, which had several bugs that accounted for some of the erratic behavior we had encountered. We contacted the author of the AODV code from NIST by email, but were unable to resolve all the problems.

AODV, being a reactive protocol, does not store routes to destinations until they are requested. The implementation of the route-discovery process was time consuming and not very robust. Routes that had

not recently been used would timeout unexpectedly.

The poor throughput with the Lucent cards was probably caused by a driver issue in either the mesh node or on our clients. The 802.11a radio in our mesh nodes came with an early version of the Multi-band Atheros Driver for Wi-Fi (MADWiFi) driver that did not support the ad hoc mode well on RedHat 9 or even the newer Fedora Core 4 versions of Linux [103] . Several of the problems that we discovered were eventually confirmed by Intel [5].

2.2 The Dart-Mesh architecture

We realized that one of the limitations of the Intel mesh was the instability of the specific implementation of the AODV protocol we were using. We updated a few nodes to Linux Fedora Core 4 to try newer device drivers, which we hoped would have fewer bugs. We found that our implementation of AODV would no longer compile correctly with the newer Linux kernel (2.6 series). We contacted the maintainer of NIST kernel AODV implementation, but he was unable to resolve the issue. Kernel AODV currently supports only the older 2.4 series of Linux kernels and has not been updated since 2004.

We wanted to deploy a high-performance stable mesh testbed that reused our existing hardware resources. We wanted to keep one interface in master mode to allow clients to easily join the mesh and surf the Internet. We did not wish to modify other protocols or implement our own mesh routing protocol from scratch, since our primary goal was to build a mesh-network management solution.

After trying several implementations of different routing protocols on Linux and Windows (detailed in Section 2.4), we were able to build a stable mesh using a public implementation of the OLSR protocol [178] without making any hardware changes. When we initially started development, we used OLSR version 0.4.10 for the majority of our experiments, but we eventually updated most nodes to OLSR version 0.5.2 (released in July 2007). After switching to OLSR, we managed to overcome most of the problems we faced with the original Intel mesh running AODV. We named our stable mesh architecture “Dart-Mesh,” which is short for “Dartmouth Mesh.”

2.2.1 OLSR

The Optimized Link State Routing (OLSR) protocol is a proactive link-state routing protocol originally proposed by T. Clausen and P. Jacquet from INRIA, France [88]. OLSR, AODV, and the Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) [130] protocol are the three protocols currently under review for standardization by the MANET [86] working group of the IETF. The details of the OLSR protocol are documented in IETF RFC 3626 [40].

As the name suggests, the OLSR protocol is an optimized version of the classical link-state algorithm. The nodes in the mesh elect a subset of nodes to act as Multi-Point Relays (MPRs). MPRs are elected nodes that forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to a classical flooding mechanism, where every node retransmits each message when it receives the first copy of the message. Unlike the classic link-state algorithm, partial link-state information is distributed in the network by these MPR nodes.

The Multi-Point Relaying technique leads to a significant reduction in the number and size of flooded messages. Link-state information is used by each node to calculate routes to each destination using Djikstra's shortest path algorithm (with hopcount as the distance metric). Thus, unlike AODV, each OLSR node continuously calculates routes to all possible destinations in the network in advance, even if they are not actively being used.

We use a public implementation of OLSR (available for download at www.olsr.org) written by Andreas Tønnesen for his MS thesis [177]. The implementation is written in user space and thus does not require any kernel modifications. The code is portable across several platforms. Tønnesen implemented some additional features considered to be optional extensions in the original RFC and added the option of using the Expected Transmission Count (ETX) [50] metric to select routes based on link quality instead of hop count. This last feature is not compliant with the specifications of the OLSR RFC.

Another notable feature of the OLSR implementation is the support for user-developed plugins that can extend OLSR's functionality. The implementation can be easily recompiled to add new custom message types to OLSR, which can be flooded in the network efficiently using the MPR mechanism as opposed to naive flooding. The implementation works well on both Redhat 9 and Fedora Core 4, and the code base is actively maintained.

2.2.2 Dart-Mesh network design

We maintain several features from the original Intel design in Dart-Mesh. The two main changes are the use of the public implementation of OLSR instead of kernel AODV, and the use of private class C addresses instead of Internet-addressable IP addresses. Our mesh uses the two 802.11b radios for its primary operation. We configure the first radio interface *wlan0* as the *Mesh-interface* in "ad hoc mode" and form the mesh backbone by routing packets between mesh nodes using OLSR instead of AODV. We configure this interface to join the ESSID "test_olsr" on 802.11b channel 11 (2.462 GHz). We choose to use private addresses in the $192.168.1.x$ range for *wlan0*, where x is a multiple of 10. This helps us quickly identify an individual mesh node, since the integer value calculated by dividing x by 10 serves as a unique mesh node identifier.

We set the second 802.11b radio interface *wlan1* as the *AP-interface* in "Master mode" using the HostAP driver [110]. This interface is configured to advertise "Dart-Mesh" as its ESSID on channel 11 (2.462 GHz). This interface allows clients such as laptops to associate with the mesh and receive an IP through the dynamic host configuration protocol (DHCP) daemon running on the mesh node (`dhcpd`). The mesh node runs `dhcpd` on *wlan1*. Each mesh node is capable of supporting at most 9 clients at a time as the DHCP server has a pool of 9 IP addresses. The range of addresses given out range from $192.168.2.(x + 1)$ to $192.168.2.(x + 9)$, where x is the same as the last octet of mesh node's IP address on its mesh interface. Thus the last two octets make it easy to identify which client is associated with which mesh box.

We do not use the third radio for regular operations in the mesh. However, a system administrator can login and use the third interface for wireless sniffing (e.g., to capture a packet trace) in monitor mode without disrupting regular traffic flow.

To allow clients in different mesh nodes to communicate with each other, we add static route table entries during initialization (using the `route add` command with the appropriate arguments) on each mesh node to route packets according to the addressing convention described above.

We allow a few gateway nodes to connect the mesh to the Internet through a wired Ethernet connection. Since the original design did not plan for the use of Ethernet cables, we manually drilled holes in the PVC case to allow the cable to pass through.

Our mesh now uses private class C IP addresses, but we wish to allow mesh nodes and clients to communicate with the Internet. We solve this problem by allowing the gateway nodes to perform Network Address

Translation (NAT) for our private addresses. The gateway node that is connected to the Internet through an Ethernet cable acquires an Internet addressable IP address on interface *eth0* (by running the command `dhclient eth0`).

We then add *iptables* entries to masquerade all packets that originate from the 192.168.1.0/24 network and are destined for external Internet addresses as coming from the IP address assigned to interface *eth0*, represented here as the variable $eth0_IP. The command we use is:

```
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j SNAT --to-source $eth0_IP
```

On the non-gateway mesh boxes, we add an iptables entry to masquerade all traffic from the clients originating in the 192.168.2.1–255 range to use the IP address assigned to the *wlan0* mesh interface represented here as the variable $wlan0_IP. We use the following command:

```
iptables -t nat -A POSTROUTING -s 192.168.2.0/24 -o wlan0 -j SNAT --to-source $wlan0_IP
```

Only the gateway mesh nodes, that are a subset of all the mesh nodes, are directly exposed to the Internet. We do not specify explicitly which gateway node any intermediate node must use, as required in the original Intel mesh. OLSR allows gateways to announce connectivity to the Internet through Host Network Announcement (HNA) messages. Mesh nodes add routes locally to the best available path to the Internet. The OLSR_dynamic_gateway plugin allows nodes to check for Internet connectivity periodically and send HNA messages to announce the presence or loss of Internet connectivity.

We configured our mesh to use most of the default settings for OLSR. We found that using the "Link Quality" option (which breaks compatibility with the RFC standard) was preferable to using the default hopcount-based route selection option.

2.2.3 Dart-Mesh usage experience

The deployed Dart-Mesh system was stable and routed packets consistently between mesh nodes and between clients. Most users who tested the system did not realize they were behind a NAT gateway or even connected to a mesh and not a Wi-Fi network. The Internet surfing experience was good and comparable with the experience of using a DSL connection and almost as fast as using our department's 802.11b Wi-Fi network. We managed to overcome all the limitations of the earlier Intel mesh deployment.

To connect to the mesh, users set their laptop client's SSID to "Dart-Mesh" and request an IP address

through DHCP. In Mac OS X and Windows XP, by default the client associates with the mesh node with the strongest signal and requests an IP address automatically. On Linux clients, one way of requesting an IP address is by running the `dhclient` command. The dhcpd server running on the mesh node provides an IP address to the client if its address pool is not exhausted.

Once the client receives an IP address and is connected to the mesh and at least one node in the mesh is a gateway node, the client can surf the Web, watch streaming videos, send and received email, etc. The packets originating in the client travel from the client to the AP interface of the nearest mesh node. If the mesh node is a gateway node then the packets are routed to and from the Internet through the wired connection. If the mesh node is not a gateway node, then the packets travel one or more hops through intermediate mesh nodes until the packets reach the nearest gateway.

Both our Mesh backbone and AP interfaces use 802.11b, which has a maximum theoretical capacity of 11 Mbps. We tried using the 802.11a interface (which has a theoretical capacity of 54 Mbps) for our mesh backbone, but were not able to get stable results with any operating system and driver combination on our hardware platform. In ad hoc mode, the Atheros radio would often lock up and would require a manual reset. The developers of the MADWiFi driver listed this as an unresolved issue even with the latest driver release we tested in March 2008.

2.2.4 Limitations

We enumerate the main limitations of our Dart-Mesh design. Most of the limitations are minor and can be resolved by future software updates, architectural modifications and configuration adjustments. Some of these limitations are imposed by our operating environment while others are inherent limitations of IP routing on OLSR and AODV.

1. Client roaming is not fully supported:

   Since we modeled our Dart-Mesh platform on the original design used for the rural deployment, which was designed for stationary mesh nodes and stationary clients, our mesh does not support clients roaming from one mesh node to another in a seamless manner. After re-association, the client must request a new IP address and will lose its previous TCP connections. Most commercial systems overcome this problem by using proprietary handoff protocols, which involve signaling and commu-

nication between mesh APs and layer-2 based routing protocols, so that packets are always routed to the correct MAC address. Engelstad et al. discuss this problem and a few potential workarounds for the OLSR protocol [58]. Amir et al. at JHU propose a mesh routing protocol designed primarily to support better handoff performance [6]. In theory, Mobile IP can also be used to solve this problem for any routing protocol [138, 135]. In WLANs, a new protocol called Inter-Access Point Protocol (IAPP) can potentially be used to coordinate handoffs [42], but the IAPP specified in IEEE 802.11f is only a recommended optional extension to the IEEE 802.11 standard [43]. Many of these handoff techniques are proprietary and some of them are patented [10].

2. Route switching issues:

   OLSR selects routes based on the quality of the reported link between two nodes. Thus the recorded route in a ping from a source to a destination several hops away may differ from the return path traversed from the destination back to the source.

   In our experience if a node has two separate links (or paths) to two gateway nodes and if the two links have similar link-quality levels, then OLSR keeps oscillating between the two paths every few seconds. This oscillation can interrupt a TCP connection. This problem can be mitigated to some degree by manually tweaking the weight that OLSR gives to certain links, to force OLSR to prefer one link over the other. Ramachandran et al. discuss similar route flapping and routing-stability issues in two static mesh networks, one at UCSB and one at MIT [146].

3. Security:

   Our mesh network allows open access to all clients. We do not encrypt data or limit the access of users or audit users in any way. We revisit this important drawback in Chapter 6.

2.3 Sudikoff deployment

We distributed 16 of our Dart-Mesh boxes across all three floors of Sudikoff Lab, the Computer Science department building at Dartmouth College. Most of the mesh boxes are placed high in the cable trays in the hallways or in the offices and labs of faculty members and graduate students who volunteered to host them.

If the mesh node is a gateway, we provide the mesh node with a wired Ethernet connection to a department switch. Figure 2.2 represents a layout of a typical deployment on three floors of Sudikoff with 14 mesh nodes, 5 of which are gateways.

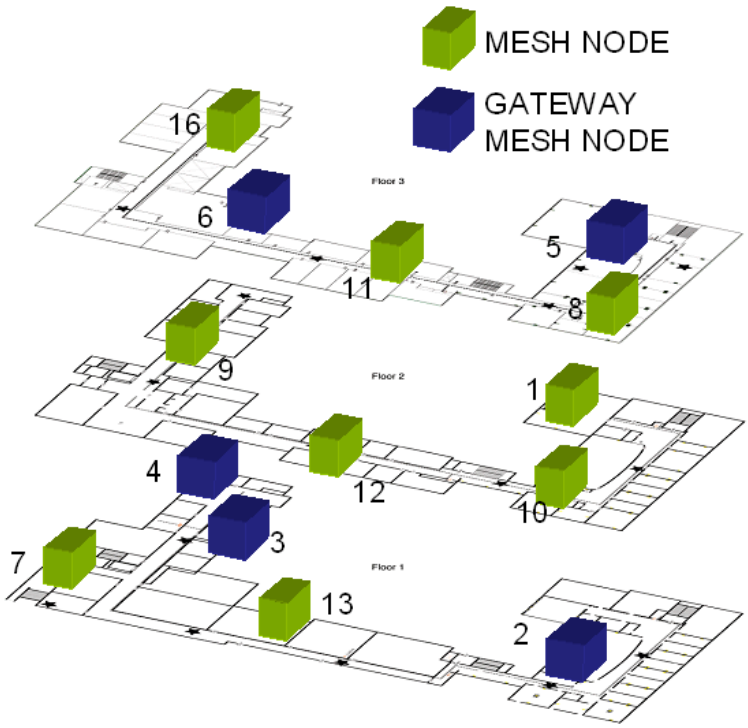

Figure 2.2: Sudikoff deployment with 14 nodes

To mitigate the noise issue, we placed our mesh nodes in locations that already generated a lot of noise, such as near water coolers and printers. Once switched on manually, we had no easy way of stopping and restarting a mesh device remotely. This setup made running certain experiments challenging as we had to physically visit each box and turn them on or off. If we required an immediate reboot or a halt (shutdown without a restart) then we could remotely log in to the mesh device through the mesh interface (if the device was connected to the mesh) or login as root in though the AP interface (if we were within range) and issue a `reboot` or `halt` command as needed.

In addition to the 16 mesh nodes, we used two Linux laptops (running OLSR) and one Apple PowerBook laptop to help in mobile experiments. We also had 6 stationary Linux laptops with wireless cards that we often used as clients for tests and experiments.

Due to the rules regarding channel assignments for wireless experiments in Sudikoff, we were only

allowed to use channel 11 (2.462 GHz) in the 802.11b frequency spectrum range. As a result, both our Mesh and AP interfaces used the same channel and could interfere with each other. We would have observed improved performance if we used separate and orthogonal channels for the two radio interfaces.

2.4 Other prototypes

Before we built our current implementation of Dart-Mesh, we experimented with various alternatives. We tested AODV-UU [128] on a few laptops and a few mesh devices. We were not impressed by the performance and gateway mesh nodes had to be configured manually. We also had some initial problems compiling and running earlier versions of AODV-UU on Fedora Core 4. Fortunately, the latest build of AODV-UU is stable and we used it to demonstrate the capabilities of our management system in handling multiple routing protocols.

We also experimented with the Microsoft Research (MSR) mesh toolkit [149]. We were able to set up a stable mesh network with 4 laptops running Windows XP. The MSR mesh software was GUI-based and thus almost impossible to use on our mesh nodes, since our Intel boxes are headless and did not have mouse support. Secondly, the MSR mesh toolkit supported multiple wireless cards but did not support putting the wireless card into "Master" mode to act as an AP, limiting the client devices that could use the mesh to only those running Windows XP. Dart-Mesh allows clients to connect to the mesh running any operating system and we regularly use Linux, Mac OS and Windows laptops in our experiments and for daily mesh use.

Figure 2.3: Aruba AP-70 node

We built a second hardware platform for mesh networks using modified Aruba AP-70 access points [8] running OpenWrt [154]. OpenWrt is a GNU/Linux-based firmware for embedded wireless devices.

One of our concerns while working with the Intel mesh boxes was the high decibel noise the fans made.

We were afraid that people would complain about the noise and we would be forced to shut down or remove deployed mesh nodes. The Aruba AP-70s (see Figure 2.3) are lightweight, compact (less than an inch thick), passively cooled devices equipped with an embedded ARM processor, two Atheros-based wireless radios, an Ethernet port and 32 MB of flashable RAM. Since these devices lack a fan or any moving parts, they are completely silent.

We experimented with AODV-UU and OLSR on the AP-70 devices. We implemented a mesh system (running OLSR) that was virtually identical in functionality to our Intel-based Dart-Mesh nodes. The AP-70 devices running OLSR were integrated into a mixed mesh network with the Intel-based Dart-Mesh nodes. Unfortunately, the AP-70 devices were not very stable and would randomly lock up after about 24 hours of usage. We did not have enough devices to experiment with and they were fairly expensive. The Aruba devices did have USB ports, which our Intel boxes lack and which would make further expansions possible.

2.5 Summary

Building a stable mesh architecture took several more months than we expected, but we never imagined we would have to do so in the first place. The implementation process allowed us to understand the challenges of designing and building a real system. We learnt to appreciate the complexity of hardware and software interactions present in Dart-Mesh. We gained experience on the nature of failures and performance issues to expect in a real mesh network. Our experience thus prepared us to face the challenge of building a real mesh network management platform: our Mesh-Mon architecture.

Chapter 3

Mesh-Mon

Mesh-Mon is our prototype of a comprehensive distributed multi-radio mesh network monitoring system. We present the design philosophy and architecture of Mesh-Mon in this chapter. We also provide the implementation details of Mesh-Mon's primary components.

3.1 Motivations for Mesh-Mon

We monitor a mesh network to measure its performance and to identify potential problems and faults. We define a fault as an event of negative significance to the overall functioning of the mesh. Some faults may be repairable (e.g., misconfiguration of a radio interface) while others may be permanent (e.g., due to a hardware failure). We aim to resolve faults through manual or automatic intervention. Manual intervention could entail a technician physically moving a node or by remotely reconfiguring it; automatic intervention could involve a node adjusting its power level to achieve a better SNR or to reduce interference with other parts of the network.

As examples of common management challenges, we enumerate below some of the problems we encountered during the development and deployment of our initial mesh testbed.

3.1.1 Configuration issues

We sometimes discovered that nodes were manually configured to the wrong SSID or an incorrect channel. Sometimes, the request made by the sysadmin to change channels would be ignored on a device. At other

times, a device would report locally that it was on channel 6 but receive and send packets to the external world on channel 11 and vice-versa. An incorrectly configured DNS setting on the DHCP server could lead to clients being wrongly configured. Thus, clients would connect to the Internet but were not able to resolve host-names and surf the web.

3.1.2 Performance issues

While running benchmarking tests over a multi-hop connection we observed the throughput observed at the client drop from 4 Mbps to 20 Kbps for a period of 5 minutes. We later found that the cause was due to external interference (one of our colleagues was transferring large files wirelessly by UDP with the "retransmit bit" set to off on the same channel). Interference can even be caused by cordless phones and microwave ovens.

3.1.3 Hardware issues

When using older Lucent Orinoco cards in our clients, we observed worse performance results than when using NICs with the Prism2 or Atheros chipset. We detected the root cause only when we changed wireless hardware on the clients.

3.1.4 Software and routing quirks

While running kernel AODV on our mesh, we would find that routes expired when the connection was idle for short periods. Since this behavior was non-uniform, detecting and debugging this problem required several days. OLSR with default settings does a poor job of filtering transient links and often reported unidirectional routes.

3.1.5 Node/link/connectivity failures

A node that has failed is often difficult to distinguish from a node that was turned off or has moved out of range. Often lab members would accidentally disconnect or reboot nodes that were part of our mesh. In most cases the mesh network recovered gracefully, but a tool like Mesh-Mon could have alerted us to the existence of potential problems and the duration of the outage.

Summary

In general, we observed that mesh networks were not always simple to setup, nor easy to configure correctly. Detecting the existence of a problem, locating the source of the problem and resolving the problem was often challenging and time-consuming.

We designed Mesh-Mon to help detect and resolve network faults automatically. Some of the problems described above are unsolvable without human intervention. If Mesh-Mon can at least automatically detect the extent of a problem and indicate which nodes are at risk, our system can help reduce the sysadmin's workload.

3.2 The system administrator's view

The system administrators are responsible for the management and operation of a mesh network. We initially set out to answer the following list of management questions (asked from a sysadmin's perspective) that are relevant to our mesh deployment scenarios. Our goal is to be able to answer most of these questions using Mesh-Mon in an automated manner with a high degree of success for stationary or mobile team of sysadmins, present at the deployment site or at a remote location.

1. How is each node performing and how is the network doing overall? What are the network characteristics present in the short, medium and long-term history? Are there any trends in the collected metrics or interesting features?

2. What faults are present in my network and where are they located? How frequently do they occur?

3. How does my network look structurally from a local as well as global perspective? How is the structure changing over time? We would like to be able to capture the state of the network through distributed snapshots and be able to compare them.

4. Which nodes should the sysadmin be most concerned about from a robustness point of view? Which nodes are best suited for specific roles in the network?

5. How many nodes can fail before I partition my network? We could apply partition detection and partition prediction algorithms for this problem.

6. By an appropriate definition of importance, which nodes are the most important in my network?
7. Similarly, which nodes are the least important and why?
8. If I could or should add or remove a node to enhance the network, which one should it be?
9. Similarly, if I had to update a subset of nodes and reboot them, in which order should I perform the operation?

We apply techniques from social network analysis to answer the last four questions.

3.3 Design goals

The primary aim of Mesh-Mon is to assist one or more administrators in fault detection and performance management. We assume that the mesh network is deployed in an unplanned manner and do not assume that nodes know their geographic location. We could use geographic information in our analysis, but by default our analysis tools assume only topological information since that is what we have available. We assume that the channel assignment is predetermined and static.

Mesh-Mon is primarily designed to:

1. help sysadmins understand the state of the network,
2. help identify and resolve problems in the mesh network,
3. help users deploy, configure and test a new mesh,
4. help one or more administrators debug known and unanticipated problems, and
5. provide information to one or more sysadmins located anywhere in the network, supporting their efforts to manage the network.

Network monitoring is an inherently distributed and complex process. The topology may be constantly changing in a mobile mesh. The frequency of updates versus the volume of transmitted monitoring data are some of the tradeoffs in the accuracy of the information collected. Monitoring and management data travel on the same network we wish to manage. We thus seek to minimize the impact our monitoring process has

on the same network we wish to monitor, yet design a system that is capable of functioning in the face of severe network quality degradation or even network failures.

3.4 Design principles

A reactive management system would be designed to detect a fault only after it occurs by passively studying the effects of a fault on the network. Reactive management systems are more suitable for static environments, where any unexpected changes in the network topology or non responses to polling attempts are often a good indicator of potential failures. In a mobile mesh network, a single fault may affect several neighboring nodes in a network (and possibly disconnect them), leading to partitions, and could make gathering accurate network information challenging or even impossible, after the occurrence of the fault. Hence we advocate the use of a more proactive and stateful approach to network monitoring.

We designed Mesh-Mon as a proactive management system for use in a dynamic and mobile environment. Mesh-Mon is designed to work in an unplanned mesh, where we do not have complete (or possibly any) and accurate prior information about how the network is deployed, or how it may be modified during its operation. We desire that our solution be scalable to be able to deal with large networks. We do not make any assumptions about how big a mesh network Mesh-Mon must manage, nor how the network may grow and shrink in size over time.

If Mesh-Mon generates a large volume of monitoring traffic, then this traffic can affect normal user traffic on the mesh. Mesh-Mon thus should not add significant overhead to the network, but should provide a high level of accuracy, and also be adaptive to changing network conditions. Mesh-Mon uses a combination of active and passive monitoring techniques in combination with a rule-based diagnostics engine. Passive techniques use less bandwidth than active ones, but introduce more latency.

Mesh-Mon has three main principles of operation for monitoring the mesh network:

1. each mesh node and mesh client must monitor itself,

2. each mesh node must monitor its k-hop neighbors and each node must store locally a detailed representation of the local network and a sparse representation of the global network, and

3. each node must help in forming a hierarchical overlay network for propagation of monitoring information.

The first principle is critical since each mesh node and mesh client must be healthy for the network as a whole to be healthy. The second principle aims for distributed analysis by allowing local nodes to cooperate to detect and analyze local problems. Mesh-Mon nodes are designed to cooperate in a peer-to-peer manner to monitor each other and to diagnose local network problems. The third principle is a common aggregation-based approach to deal with scalability in large distributed systems. The overall structure of Mesh-Mon is designed to facilitate self-management when a node or partition is disconnected from the rest of the network.

The performance of any mesh depends heavily on the nature of the mesh routing protocol. We desire that Mesh-Mon function effectively even if we replace our mesh routing protocol with another routing protocol. We thus designed Mesh-Mon to be independent of the underlying layer-3 routing protocol used.

Our current mesh testbed (described in Chapter 2) uses OLSR, a proactive routing protocol, as the mesh routing protocol. We can alternately use AODV. We monitor information exposed by the underlying routing protocol (when it is available). Although we could have chosen to strongly couple Mesh-Mon with the mesh routing protocol and piggyback monitoring information on routing control packets, we chose to avoid this route since it would require modifications to the underlying mesh routing protocol. Instead, we built Mesh-Mon at the application layer, and provided it with the resources required to peek into the network layer and lower layers of the network stack whenever required and wherever possible, without modifying the operating system, routing protocol, device drivers and other components of the system.

This thesis focuses on the collection, distribution and analysis of network information and the resolution of detected faults. The analysis of all available information to diagnose root causes of all possible problems is beyond the scope of this thesis. Root cause diagnosis is extremely challenging since the actual cause of a detected fault may be the result of a complex sequence of events (some of which may not even be observable) over varying time scales. We do explore a large set of problems and gather information that can help an administrator to deal with unexpected problems. As long as the network element under scrutiny is reachable through an external network, remote management options are a possibility. However, for certain problems, such as hardware failures, remote management is not an option.

Mesh-Mon in its current form, will not replace human system administrators and their abilities to rea-

son about complex problems, but Mesh-Mon can help network managers manage a mesh network more efficiently and effectively.

3.5 Mesh-Mon overview

Mesh-Mon is our distributed management system designed for a mobile two-tier mesh network. Mesh-Mon runs on all mesh nodes and manages the collection, communication and analysis of information gathered in the network. Mesh-Mon-Ami (MMA) is a monitoring component that runs on the clients and assists in communication of management information from disconnected areas of the network.

3.5.1 Mesh-Mon components

Both Mesh-Mon and Mesh-Mon-Ami have three common components (as shown in Figure 3.1):

1. local information collection system,
2. information communication and storage engine, and
3. fault detection and analysis engine.

We provide details of the individual components of Mesh-Mon followed by an overview of Mesh-Mon-Ami.

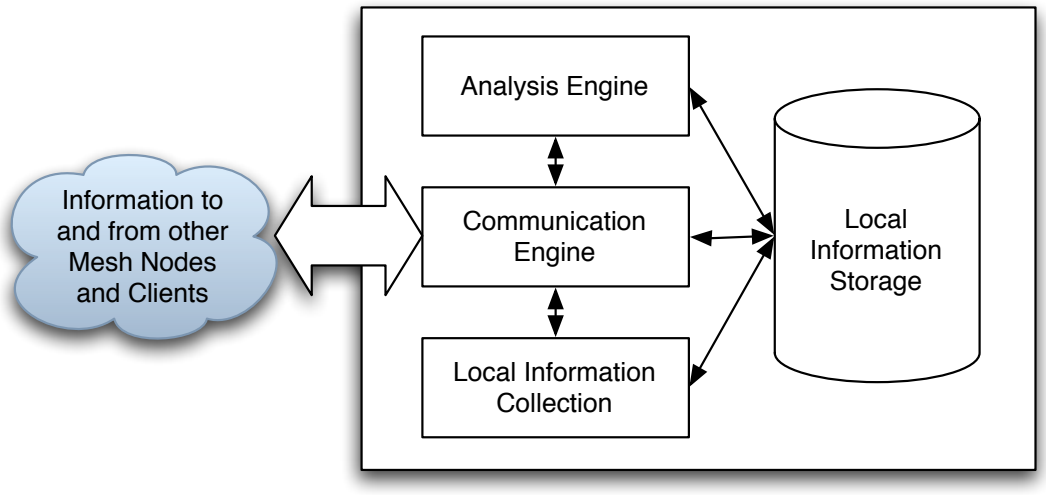

Figure 3.1: Mesh-Mon and Mesh-Mon-Ami common components

3.6 Local information collection system

The goal of this component is to periodically sample measurements and configuration information on each node. We need to measure and collect information about the state of the network at all three lower layers (physical, MAC and network) on both the clients and mesh nodes. The collected information is analyzed for faults or errors by the analysis engine. The summary of the collected information, analysis results and generated alerts are stored and distributed to other nodes by the storage and communication engine.

Most of the information is stored as attribute-value pairs (see Table 3.1 for a few examples). We categorize the information collected as configuration related (e.g., radio channel setting) or measurement related (e.g., battery status). We calculate and maintain statistics (derived by calculations) such as the average number of packets sent per minute.

We record the signal strength of received signals, the signal-to-noise ratio (SNR), battery levels of each node, number of neighbors and clients. We record network statistics of each interface such as throughput, control overhead, total data traffic and current outgoing packet queue length, since we have multiple radio interfaces. Periodically, nodes actively probe each other to measure bandwidth and latency among clients, mesh nodes and external hosts. We provide more details in Section 3.10, describing the details of our network analysis engine.

Our Intel Pentium III CPU-based mesh nodes (with large hard drives for storage) are a powerful platform compared to the embedded systems often used in commercial mesh products, where the goal is to maximize profits by reducing costs. Other less powerful mesh devices may be able to measure only a subset of this data or may only be able to store a limited volume of data locally before running out of memory. For example, our AP-70 based mesh nodes only have 32 MB of RAM and have no additional storage capacity and would thus not be able to store large amounts of local data. Such a configuration does not limit a device from running Mesh-Mon, but only limits the amount of historical data that the device can store.

3.7 Information storage and communication engine

While information storage and information communication are distinct tasks, we present them as one functional component since the two tasks are tightly coupled together in our design and implementation. Each

Table 3.1: Examples of locally collected information

Grouping	Variable	Format	Example
General	CPU type	String	Intel
	OS version	String	2.6.15 FC4
	CPU usage	Float	3.22%
	Memory used	Float	38.5%
Mesh-Mon related	My IP	IP address	192.168.1.50
	My Node Id	Integer	5
	My MeshLeader	Integer	11
	Last heard from MeshLeader	Integer	35 seconds
	Last heard from MeshAdmin	Integer	85 seconds
	Protocol	String	OLSR
	Neighbors	Integer	4
	Partition size	Integer	16
Wireless specific	Noise level	Integer	-95 dBm
	Received signal level	Integer	-71 dBm
	Link quality	String	25/94
	Bit rate	Float	5.4 Mbps
	Channel	Integer	11
	Frequency	Float	2.462 GHz
	BSSID	MAC address	02:02:64:78:51:BE
	Power management	Binary	Off
	Mode	String	Ad-Hoc
Network statistics	Tx packets	Integer	7236
	Tx bytes	Integer	967324 bytes
	Tx errors	Integer	133063
	Rx packets	Integer	49795
	Rx bytes	Integer	474388 bytes
	Rx errors	Integer	0
	Rx dropped	Integer	0
	Fwd packets	Integer	236544
	Fwd bytes	Integer	2802320 bytes

node stores detailed information about itself that it collects locally, information about its k-hop neighbors, and some information about the structure of the entire mesh network. This information stored at each node is a combination of information received from incoming management packets as well as information that the node gathers locally (such as information from the routing protocol). We designed Mesh-Mon to store more information about the local neighborhood and a sparse representation of the global network and distant nodes. This design allows the diagnosis engine to solve local problems locally as well as have adequate information to identify problems of a more global nature, such as network-wide connectivity-related issues, and still be able to scale to large networks.

To aid in development and debugging of Mesh-Mon, we log all information collected or received on each node in a round-robin database in addition to the in-memory storage. This on-disk database allows us to generate simple graphs of observed trends in the recorded variables on each node using `RRDtool` [129]. The use of persistent storage helps in manually debugging nodes that may have crashed or rebooted during an experiment. A round-robin database is well suited for storage of management information since the volume of data stored in it cannot exceed a fixed limit (a useful feature for implementation on memory-constrained devices).

Each node periodically records a conceptual "distributed snapshot" of the network locally by using the most recently received information about the network. Unlike distributed snapshots in the traditional sense, we are more interested in only the state information contained at each node, and not in the messages that may be in transit at any given instant [109]. Studying changes in snapshots could help the sysadmin understand the structural changes in the network over time.

3.7.1 Communication model

Instead of relying on the underlying routing protocol used by the mesh nodes to route normal unicast traffic, Mesh-Mon uses flooding as its protocol for disseminating management traffic. Each Mesh-Mon node uses a flooding broadcast to periodically spread its local information and locally generated alerts in UDP packets.

Flooding has several nice properties that make it attractive for use in Mesh-Mon. Flooding is a simple mechanism, both conceptually and from an implementation perspective, thus making it reliable and practically fail-safe. Flooding allows basic communication between one node and all its one-hop neighbors with

a single packet and is also a fundamental neighborhood discovery mechanism.

Mesh-Mon nodes communicate through flooding even if the routing protocol is disabled. This feature is important for a mesh monitoring system, since in the event that the routing daemon crashes or misbehaves, Mesh-Mon is still able to inform all its neighbors (and ultimately the sysadmin) of this fact. Since our flooding scheme makes Mesh-Mon's communications independent of (and isolated from) the routing protocol that the mesh uses, the mesh could run any layer-3 routing protocol and Mesh-Mon could function without requiring any changes.

At present, in our implementation, each node broadcasts these local-information packets to all nodes in the network via UDP. Each node that receives the broadcast packet locally stores its contents, and rebroadcasts the packet, unless the packet was already broadcast by the same node earlier. This simple flooding protocol, combined with our local "snapshot," has the effect of building a "holographic database" describing the mesh network, in which any single mesh node has an almost complete view of the whole network. Thus, every node in the network stores its own image of the entire network, which the nearest sysadmin can tap into and examine.

Our information dissemination model is simple, satisfies our design goals, and is easy to implement. In the event of a network partition, thanks to our redundant approach to storing information in our holographic database, every node individually knows about the topology of the network and the status of nodes just prior to the partitioning. This information can help automated analysis tools and the sysadmin in detecting when a major partitioning event takes place and deciding how it happened.

We considered other techniques, such as using probabilistic rebroadcasting of management packets, to reduce the overhead of naïve flooding approaches. While flooding can overload a network and should be used sparingly, this is a tested technique that can guarantee data delivery, even in high-mobility scenarios.

3.7.2 Scalability through clustering

The performance of flooding-based routing protocols have been well studied [105, 185, 188, 192]. A broadcast-based flooding approach to disseminate management information cannot scale well to large deployments, since there are many redundant retransmissions, and too many broadcasts can lead to packet storms [127].

The performance of any flooding approach depends on several factors, such as the broadcast frequency, transmission power, degree of each node, and the diameter of the network. The best-known solutions to improve the scalability of flooding protocols include scoped flooding, probabilistic transmissions, use of geographical information, and clustering [13, 82, 90, 97, 119].

We use clustering with scoped flooding as our approach to provide scalability. We call our cluster heads "MeshLeaders." A MeshLeader in Mesh-Mon is similar to an elected cluster head in ANMP [36] or a Multi Point Relay (MPR) in OLSR [40].

For large networks, we limit the flood of information to the k-hop neighborhood around each mesh node, and we build a hierarchical overlay for disseminating information by using a cluster head structure as shown in Figure 3.2. Through a suitable leader-election protocol, each mesh node knows to which cluster it belongs and who is its MeshLeader.

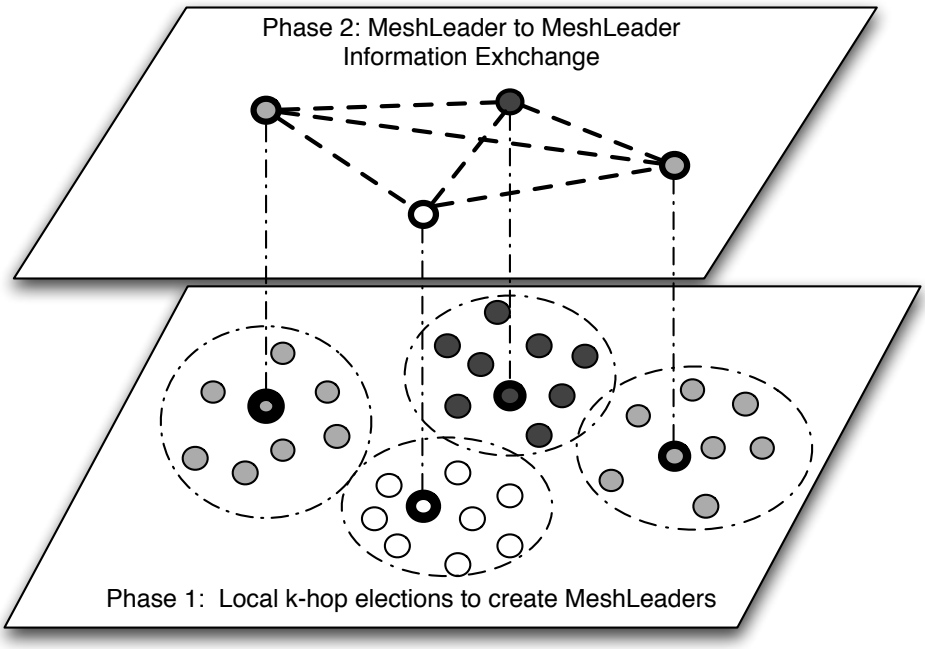

Figure 3.2: Mesh-Leader election and discovery

The MeshLeader uses all the local information collected to diagnose local problems, such as configuration issues, hardware and software faults, and performance issues within its k-hop neighborhood. The MeshLeader also acts as a cache for all alerts generated in its k-hop neighborhood. Thus, a sysadmin could

contact a MeshLeader to learn about the status of nodes in its neighborhood.

Mesh nodes can also communicate directly with other nodes in a simple query-response manner (through unicast packets) even if they are not in the same cluster. This direct end-to-end communication is useful for the analysis engine to initiate tests that verify the freshness and accuracy of received information.

3.7.3 MeshLeader election protocol

A MeshLeader can be selected by nodes in its k-hop neighborhood by using any suitable leader-election protocol. There are several leader-election protocols in the literature [13, 68, 90, 109]. While we did not implement a leader-election protocol in our implementation of Mesh-Mon, we summarize one such protocol [127] that we propose to adapt for Mesh-Mon.

We assume that nodes are aware of the presence of their neighboring nodes and of their respective NodeIDs. Each node has a unique integer NodeID. A *cluster* is a set of nodes formed as follows. A host with a local minimal NodeID (i.e., a node that has the smallest numeric identifier amongst all nodes within its k-hop neighborhood) will elect itself as a cluster head or MeshLeader, and announce this fact to its k-hop neighbors (its cluster) via a flooded message (limited to k hops). In our current implementation we use $k = 2$ but assume that the value of k can be adjusted as the network grows or shrinks.

If there is another node in the same cluster that locally decides that it should be a MeshLeader, it will announce this fact and then one of these two nodes (the one with the higher NodeID) will eventually announce that it is no longer a MeshLeader. If any node finds that there are no MeshLeaders in its k-hop neighborhood, then after a suitable timeout it will announce itself as a MeshLeader, even if its aware of neighboring nodes that have lower nodeIDs than itself. This ensures that all nodes are within reach of at least one MeshLeader.

Clusters are allowed to share boundary nodes and to overlap. Within a cluster, a member node that shares a link with a node from another cluster (see nodes D and E in Figure 3.3), or lies in two or more clusters (see node B in Figure 3.3), is a *boundary node* for the two clusters. To take mobility into account, when two MeshLeaders become direct neighbors, the one with a larger ID gives up its head role, and nodes in its cluster will discover or elect their new MeshLeader as needed. More details and variations on this scheme are available [90, 127]. An alternate election approach could use eigenvector centrality (see Section 4.3) to select the most "central" nodes in the network as MeshLeaders, since nodes with high centrality are known

to spread information quickly and efficiently [30].

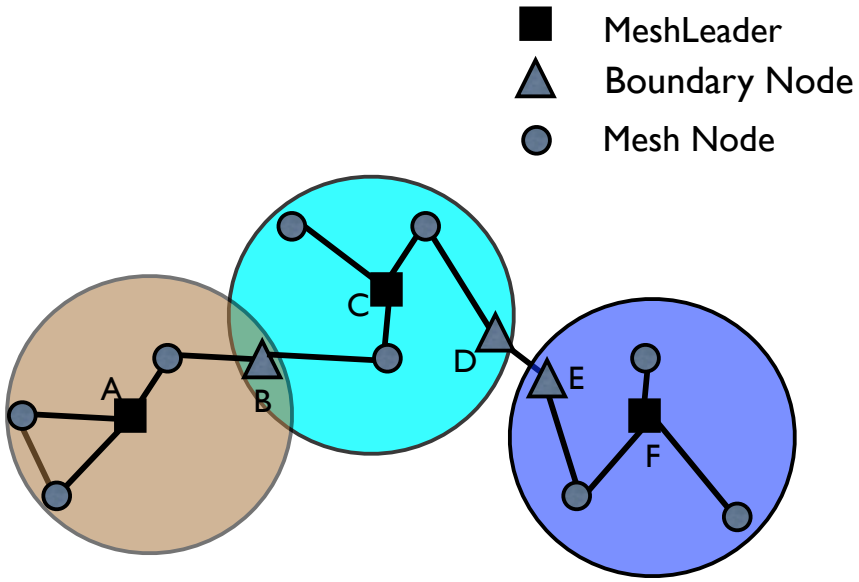

Figure 3.3: A mesh with 3 clusters and $k = 2$; adapted from [127]

To provide additional reliability, it is desirable to have MeshLeaders in adjacent clusters at most $2k$ hops from each other and a small degree of overlap between clusters. This situation ensures that if the unicast mesh routing protocol fails, then some information can be passed along from one MeshLeader to another through the overlapping boundary nodes and our flooding based communication scheme (see node B in Figure 3.3).

In battery-operated systems, the responsibility of being a MeshLeader can be rotated between other nodes to prevent a single MeshLeader's battery from being drained too quickly.

If a node is a MeshLeader, it eventually discovers other MeshLeaders (through an exchange of packets by the nodes that lie on the boundaries between clusters). MeshLeaders exchange aggregated management information with other MeshLeaders directly through unicast packets (using the mesh routing protocol). The authors of the DRAMA management system speculate that this type of hierarchical structure can be repeated recursively (by creating nested tiers of clusters with k-hop MeshLeaders), as many times as necessary to manage networks of arbitrarily large size [94]. This technique of aggregation through nested hierarchies is a widely accepted method to provide scalability in large distributed systems and even in the Internet.

The overhead of sharing network-management information in Mesh-Mon could be further reduced by

aggregation and complex forwarding techniques (such as the Multi Point Relaying technique used in OLSR), but we believe there is value in having nodes share *redundant information* through simple flooding, and of using less aggregation in messages between MeshLeaders in certain scenarios. For instance, there may be benefits to spreading redundant information to all nodes to counteract the effects of multiple simultaneous failures leading to many partitions, some of which may not have MeshLeaders. Eventually the new partitions will elect their own MeshLeaders, but it would help if knowledge the overall network structure (just before the partition) remained for analysis.

3.7.4 Performance

The performance and scalability of flooding and clustering-based approaches has been well studied through analysis and simulation [36, 78, 98, 105, 114, 127, 155, 188]. Since we have not changed the traditional mechanisms used for flooding or selecting cluster heads in Mesh-Mon, we do not focus on these aspects in our evaluation of Mesh-Mon in Chapter 4. In our experimental evaluation results (presented in Section 4.2.3), we study the performance of Mesh-Mon's flooding protocol (with and without MeshLeaders) on networks with up to 16 nodes and found that Mesh-Mon's impact on the overall network was low.

For larger networks, the overhead traffic due to flooding dissemination is limited to the k-hop neighborhood around each node. This principle is also used in the Zone Routing Protocol (ZRP) and the Grid routing protocol, amongst others for the same purpose [79, 87, 104]. Thus, even in larger networks, Mesh-Mon's flooded management traffic within a cluster will not affect the flow of regular data traffic. In addition, pairs of MeshLeaders exchange management traffic consisting of single packets (whose size and frequency are adjustable) and whose overall impact on the network is insignificant. `Iperf` traffic overheads are high but are constant and unaffected by the size of the network. `Iperf` traffic is not essential for Mesh-Mon, and nodes that experience heavy traffic loads can choose to turn off `Iperf` bandwidth tests as needed.

Even though we perform our evaluations on a small testbed, the general performance capabilities of mesh nodes that are more than three hops apart (see Table 4.1) does not bode well for the practical applicability of flat large-scale mesh networks. While a large-scale mesh network with hundreds or even thousands of mesh nodes may be *manageable* using Mesh-Mon with many MeshLeaders, such a network (without multiple gateways) may not be practical for supporting high bandwidth, low-latency real-time applications for clients

who may be many hops apart. A mesh network with many gateway nodes, designed for providing Internet access to its clients, is more feasible. In some scenarios, MeshLeaders may be able to share management traffic through the gateways, e.g., using them as "wormholes" to provide better performance by avoiding long multi-hop wireless routes.

3.7.5 Reliability

As discussed above, we chose a flooding approach to disseminate management information in a reliable manner that is independent of the underlying routing protocol. For a large network with many MeshLeaders, they need to communicate aggregated network information with each other through unicast packets. These unicast packets rely on the mesh routing protocol. This risk is minimized by careful election of MeshLeaders to ensure that clusters have enough boundary nodes that overlap between clusters, to allow for MeshLeaders to communicate through messages piggybacked in Mesh-Mon's flooded packets. For maximum reliability, MeshLeaders can use both unicast messaging and piggybacked messages.

While Mesh-Mon may still be functional in a mesh network in which the mesh routing protocol is unreliable, such a mesh would have limited functionality for its intended users, the mobile clients. In such a situation, Mesh-Mon at least offers the possibility that such a situation can be identified and repaired.

3.7.6 Alert model

As mentioned earlier, Mesh-Mon nodes generate alerts to indicate detected faults or suspected anomalies. For a team of sysadmins, we assume that a multicast communication protocol (e.g., IP multicast) can be used to address all of them simultaneously. In the absence of multicast support in the mesh routing protocol, several unicast packets can be sent.

When an alert is generated, or a situation arises that requires immediate human intervention, a short packet containing just the alert is sent via UDP to the sysadmin immediately. The alert is addressed to a simple process running on the sysadmin's console, called the *Mesh-Admin*, that logs all received alerts. The alerts contain encoded information about the cause of the alert, the severity of the problem, actions taken, and in some cases, a request for action. We provide more details in Section 3.12.

The information in the most recently generated alerts are also piggybacked in the periodic packets broad-

casted to neighbors. Thus, the information conveyed through alerts is cached by MeshLeaders on behalf of their k-hop neighborhood. Any sysadmin can tap into the information stored on a MeshLeader and check the latest alerts from nodes in that neighborhood in the event that the original alert did not reach the sysadmin for any reason.

3.8 Fault detection and analysis engine

Our approach to fault detection follows a two-fold approach. We detect certain faults through a deterministic rule-based system, and we run anomaly detection algorithms on measured and received information in an attempt to detect unusual network behavior.

An anomaly is an observable phenomenon that causes network operations to deviate from normal network behavior or causes a network disruption. Defining "normal" network behavior is not an easy task [26]. Normal traffic behavior depends on the dynamics of the network being studied in terms of traffic volume, the structure of the network, and the nature of applications using the network [174]. Network monitoring and statistical analysis can in theory be used to build models of normal network behavior. Not every detected anomaly is an indication of a fault. Our motivation in detecting anomalies is to detect when performance degradation occurs and to provide additional information to the administrator.

Some network anomalies are associated with performance degradations in the network. Congestion and broadcast storms can be considered to be network performance anomalies. A protocol implementation error can manifest itself as a network anomaly since it can cause an incorrect or undesirable traffic pattern. A denial-of-service attack or a network intrusion is an example of a security-related anomaly that may cause performance degradations.

A fault is a failure in a network device or software entity. Feather et al. classify faults as hard faults or soft faults [61]. Hard faults typically refer to hardware failures and other non-recoverable failures. In the context of networks, Feather et al. define *hard faults* as characterized by the inability to deliver packets, while *soft faults* are characterized by a partial loss of network bandwidth [61]. After a soft fault, the network typically continues to operate in the presence of a failure, albeit at a degraded level. Networks can usually recover from soft faults. For instance, a node on which the routing daemon fails can recover from the failure if the routing daemon can be restarted. Congestion is an example of a soft fault. Some faults may be

unrecoverable without physical intervention. A battery powered node that has depleted its battery will not be recover until its battery is replaced or recharged. Mobility can lead to connectivity related faults such as link breakages (possibly transient in nature) or the creation of new network links or network partitions.

We detect all known faults using a rule-based approach based on our understanding of faults, and their observable effects and characteristics. We detect anomalies by regarding them as "statistically unusual" events by identifying parameters that deviate strongly from their mean observable values. We do not diagnose all observed faults, i.e., we do not claim to correctly correlate all observed faults with their root cause. Accurate diagnosis is extremely challenging since determining the cause and effect behaviors, and the dependencies in a complex system, are non-trivial problems.

Since we assume our mesh is dynamic, we do not wish to rely on a single node to aggregate management information and analyze it. Although a centralized design is best suited for detecting global-scale faults in a fully connected static network, such as for analyzing problems related to connectivity, the central analysis node is both a single point of failure and likely to be a bottleneck, since all management traffic must be forwarded to that single node.

One option to avoid the single point of failure is to replicate the functionality of the centralized management node in all nodes. Now any single node is equally capable of analyzing the network. We would need to provide monitoring information about the global network to every node in the network, which however introduces an even bigger scalability problem (since every node requires information about every other node in the network).

We adopt a decentralized approach to the analysis of network information. Our analysis engine, running on each mesh node, uses all the information it receives from the set of cooperating MCs and MNs in the k-hop neighborhood to detect local faults and anomalies. The analysis engine also uses the network information gathered and shared among MeshLeaders.

Detection of anomalies or faults using our rule-based analysis triggers alerts that are sent immediately to sysadmins, and eventually to neighboring nodes; these alerts implicitly describe a hypothesis about the cause of the problem. The alerts often trigger secondary diagnostic tests to verify the root cause of the problem or to gather additional information for the sysadmin as detailed in Section 3.12. In some cases Mesh-Mon can resolve the issue locally and autonomously.

In the larger network, nodes that detect a problem communicating with another node can communicate directly with that node and attempt to resolve the issue between the two end-points with the cooperation of intermediate nodes and Mesh-Leaders that lie in the path between the two end-points.

Many network problems can be detected locally on a node or by analyzing information received from neighboring nodes. Absence of received information is also a factor in the analysis in some cases.

While we limit our analysis engine to work on information gathered from the k-hop neighborhood, we are still able to diagnose certain global problems (such as detecting partitions) in Dart-Mesh by exploiting information about the global topology exposed by the OLSR routing protocol. For the detection and analysis of problems in large networks that use other routing protocols (such as DSR or AODV), we propose a recursive cooperation mechanism between MeshLeaders, such as the one proposed in DRAMA to allow the diagnosis mechanism to remain scalable and decentralized [37]. We did not implement this mechanism in our implementation, which runs on a network of only 16 nodes.

We provide more details about the implementation of the fault detection engine, its subcomponents and its integration with other Mesh-Mon components in Section 3.12.

Our analysis engine runs on all nodes in the k-hop neighborhood in an almost identical manner. Since several nodes share neighborhood information, multiple nodes may discover the same fault in a neighboring node. We believe that receipt of multiple alerts from several viewpoints can help reinforce a hypothesis and give it more weight. The hypothesis can aid the sysadmin in determining what triggered a fault in the network and provide insight on how to deal with it. We allow the administrator to adjust the rate at which alarms are generated to suit his or her preferences before deployment. In an ideal system, the sysadmin would be able to adjust this rate during continuous operation.

On the other hand, a poorly designed analysis system could lead to multiple contradictory alarms being generated with no means to understand and resolve them. We recognize this is a common challenge faced by many alarm generation systems. We use a deterministic reasoning system that generates alarms only in specific situations to minimize the number of alerts generated and include enough information to decipher the cause of the alarm.

The analysis engine is only as accurate as the information it acts upon. For instance, if a node "mis-reports" information to its neighbors and provides them incorrect information (irrespective of whether the

node is malicious), then the analysis engine will provide erroneous alerts. For example, if a node forwarded 5,000 packets in a minute but, due to an error, reports to the information collection agent that it sent 500 packets, the local analysis would not notice anything wrong. When this information is sent to all its neighbors, they may not notice anything wrong since they do not perform traffic analysis at the granularity of source-destination pairs to realize that their neighbor is lying. The neighbors may notice that there is a sudden increase in the number of received packets on some nodes (which may or may not be anomalous). Another example is a node reporting a set of neighbors or links that do not exist. Since the information is sent to all neighbors and they again spread it to all neighbors, incorrect news can travel fast. Fortunately, the existence or non-existence of certain nodes and links can be probed and tested. The sysadmin alone is responsible for verifying all received alerts.

In a mesh that is operating properly (without any faults), Mesh-Mon can provide useful network statistics that help a sysadmin understand utilization of different areas of the network, observe and study long-term trends, and help find potential bottlenecks or underutilized nodes.

3.8.1 Distributed consensus of alarms

One option we considered (but did not implement) is to allow nodes that detect a problem to inform its neighbors first, before alerting a system administrator. Neighboring nodes that have detected the same problem can decide to suppress their own alerts. Nodes with the same problem are members of a group. Once group membership is decided, a single member of the group can send an alert to the sysadmin stating the problem and the list of the members.

Deciding distributed group membership itself can be a time-consuming process and requires the reliable exchange of messages between group members. Distributed group membership and distributed consensus is a well-studied problem in distributed systems theory and theoretical results indicate that even a single malicious node, or lost message, can lead to incorrect results and may require the algorithm to be restarted from scratch [14, 35, 62]. The benefit of using such a consensus scheme would be a reduced number of alerts being sent to the sysadmin. One disadvantage is delay: since the consensus algorithm requires the coordination of several nodes, there is a greater delay between the occurrence of the event and the alert reaching the sysadmin.

Since Mesh-Mon already collects information from all k-hop neighbors, the MeshLeader can detect groups (if any) in a semi-centralized manner. Discovering that many nodes share a common symptom (fault or anomaly) can be indicative of the severity of a detected anomaly. Its also possible that the consensus is simply coincidental. By running the consensus algorithm and sending an aggregated alert message to the sysadmin, we can reduce the rate of generation of alarms and save bandwidth at the cost of delayed notification being received by the sysadmin. However, a lack of consensus need not be indicative of a lack of problem severity, since a local problem detected by a single node can also be a severe problem.

3.9 Mesh-Mon-Ami

Mesh-Mon-Ami (MMA) is a monitoring component running on a mobile client. In addition to checking for configuration errors and monitoring client performance, a MMA mobile client can help ferry information between disconnected partitions of the mesh network. We use a technique similar to epidemic routing [184] in Delay-Tolerant Networks (DTNs). The client receives management information through the mesh node's AP interface, buffers it and shares it with other MNs that the node associates with (while moving). This MMA relay may be the only means of communication between two disconnected partitions of the physical network without investment in an additional long-range network link. In effect, the MMA client acts as a DTN "mule" for management packets. There is no guarantee of symmetry, as the MMA client moving from partition A to partition B allows B to learn of partition A, but provides A with no knowledge of partition B. However, some node in partition B could trigger an alert to a sysadmin who may deploy additional nodes that could join the partitions.

On receiving a message, the sysadmin sends an acknowledgment, and the MMA can then delete the message from its buffer. Older messages are expired after a preset interval. Messages passed in this manner are encoded so that each MN can quickly validate information received from a MMA, by initiating tests in some cases, to check if the information received is already known. This helps filter older messages and prevents redundant alerts from reaching the sysadmin.

Mobility prediction could also play a role in making MMAs more efficient; for instance, we could send certain messages with clients more likely to move in certain directions. However, at present we do not have good mobility models for the scenarios we envision, nor do we capture location information. We chose

to implement our own DTN message passing mechanism between Mesh-Mon and the MMA to keep this mechanism independent of the underlying routing approach. An alternative approach, such as combining a DTN layer with a routing protocol, could make implementing our MMA communication model easier, but would increase the coupling between management and routing.

3.10 Network analysis

A healthy mesh is a well-connected mesh with high capacity and low end-to-end latency. A well-connected mesh should not be more dense than required to provide sufficient coverage. However, an over-provisioned mesh is wasteful and nodes physically close to each other may interfere with each other. An under-provisioned mesh may not provide sufficient coverage and a few node failures may lead to partitions. Over time, as nodes move around in a mobile mesh, the network may be reconfigured several times and affect the overall performance.

Our primary goals from the sysadmin's perspective are two-fold; we wish to understand the health of the current state of the mesh network, and we wish to study its changes over time. In addition, we aim to detect major faults as soon as they occur and possibly anticipate them before they happen.

3.10.1 Network health monitoring

If the clients and mesh nodes can communicate with each other and with external hosts within acceptable performance levels, it indicates that the end-points of the network are healthy and the network is allowing communication between them. Since each node in Mesh-Mon has an approximate view of the global network, each node can select target nodes and clients, probe them by sending `pings`, measure the round trip times, and log the traversed path to each destination. Depending on the routing algorithm and the stability of links, the route traversed can often change even for consecutive ping packets. Packet loss is often a good indicator of overall link quality and stability and is the basis of several metrics for routing protocols, such as the expected transmission count (ETX) metric [50].

We calculate network-level resource utilization locally on nodes and clients for each network interface; during periods of low utilization, nodes run pairwise throughput measurement tests. In our prototype we use an automated version of `Iperf` to run UDP and TCP throughput tests between nodes, clients and external

hosts [175]. Nodes that are heavily loaded may disable other nodes from probing them temporarily since these tests can involve transferring several megabytes of data that may saturate the wireless medium. Thus, an `Iperf` test could fail, but the `ping` test should still succeed.

To calculate utilization for an interface we calculate the delta (change in value) of the number of transmission bytes over the sampling period. We express utilization as a ratio of the value of [delta(Total bytes sent and received)/(sampling period in seconds)] to the constant value of the interface capacity. For all wireless interfaces we assume the capacity is 11 Mbps, and for wired links such as Ethernet connections at the gateway, the capacity is 100 Mbps. In reality, the actual wireless capacity is much lower and depends on several factors such as the hardware specifications, driver details and the operational environment. In our current system prototype, to detect anomalies, we look for trends and outliers based on the standard deviation from mean values calculated at each sampling interval. On gateway nodes we run a few additional tests to ensure that the connection with the external network is stable.

We calculate local network health indicators from sampled data such as the network utilization, receive discard rate, transmit discard rate, receive error rate, transmit error rate, transmit broadcast rate and the receive broadcast rate [194]. The discard rate is an indication of the number of packets that were discarded by an interface due to resource limitations. A high discard rate is usually an indicator that more buffer space is needed for an interface. A high error rate may be an indicator of a hardware problem. A sudden and sustained jump in the broadcast rate (packets broadcasted per second) can be indicative of a packet storm. A high percentage of failed re-assemblies at the IP layer can be indicative that fragments are being corrupted or discarded at intermediate nodes. A high occurrence of fragmentation could be caused by MTU mismatches.

We record statistics for UDP, TCP and ICMP traffic. Our source for most information collected locally is the $/proc/net$ directory and by parsing the output of system utilities, such as `ifconfig` and `iwconfig`. We do not run per-packet network sniffing tools, although they can be a rich source of information, since we wish to keep CPU overheads to a minimum.

Mesh-Mon keeps track of the volume of monitoring and management traffic each Mesh-Mon node or MMA client generates or forwards. We use this metric to evaluate Mesh-Mon's impact on the network along with other metrics to adapt its behavior to changing network and traffic conditions.

3.10.2 Configuration management

Mesh-Mon aids in local configuration management of each node. Each node has a local set of reference values and acceptable ranges for each configurable component. These values are checked periodically and compared with values and parameters at neighboring nodes. Differences are logged and sent to an administrator to view. In a homogeneous deployment it is expected that all nodes share many configuration variables. The configuration check may fail if an administrator has modified a parameter for debugging or troubleshooting and forgotten to update the reference model. A majority of the parameters are identical for all nodes in the network and the majority of our mesh devices are homogeneous.

All essential services running on each mesh node (such as `olsrd`, `iptables` and `dhcpd`) are periodically monitored to determine their status. If any of these services have crashed, then Mesh-Mon will attempt to restart them with default settings and alert the sysadmin as needed.

3.10.3 Partition prediction and topology analysis

The current network topology defines the connectivity properties of the mesh network. Assuming that the mesh is one large connected component at initialization, partitions occur when specific nodes fail or nodes move away. Nodes in different partitions cannot communicate with each other. By identifying critical nodes and links, Mesh-Mon can anticipate where partitions are likely to occur.

Each node in Mesh-Mon maintains a graph representing the state of links in its k-hop neighborhood. Critical nodes and critical links (equivalent to articulation points and bridges in undirected graphs) are defined as those nodes or links in the network whose removal will partition the induced network subgraph into disconnected components or partitions. We simplify our definition of a critical link to be any link between two critical nodes, or to be a link to any node that has exactly one neighbor.

Tarjan presents a well-known centralized algorithm that uses global knowledge based on Depth First Search (DFS) for this problem [170, 44]. If the root of the DFS search tree has two or more children then it must be a critical node, since the nodes in the two subtrees share only the root in common. Removing the root would thus lead to two isolated partitions.

A localized approach to predict critical nodes

Jorgic et al. proposed a localized approach to predict critical nodes and links by running DFS on just the k-hop neighborhood graph from each node [93]. Thus, each node can quickly check locally if the DFS tree rooted at that node has two or more children. Unfortunately, since graph connectivity is a global property, there is always a possibility any localized algorithm will lead to false positives. False positives, in this case, are nodes that are marked globally critical but in reality are only critical to the local neighborhood and are not a real threat. There is no threat of false negatives with this technique. In their simulation study, the authors observed that about 80% of locally predicted nodes were found to be globally critical in random connected graphs for k=3 [93]. False positives are mainly caused by the presence of long cycles or rings, the detection of which can require global information.

3.10.4 An optimization for OLSR

While we strive to keep Mesh-Mon independent of the underlying routing protocol, we can benefit from information already available without modifications to our software or hardware architecture. Since OLSR is a proactive routing protocol, every node in the network already has some information about the current topology of the entire network. We do not assume that all routing protocol implementations expose internal data structures, so the optimizations we describe below are specific to our OLSR implementation.

In OLSR, the "Topology Change Redundancy" (TCR) setting determines the volume of redundant information propagated in the network. At the minimum TCR setting each node receives information on all nodes and a minimal set of links. When we set TCR to its maximum value (which is "2" as specified in the OLSR RFC [40]) each node actually receives the complete global topology with all links. The number of control packets sent does not change, but the size of each packet increases. Thus the sysadmin can chose to trade routing overhead for critical-node-detection accuracy.

Thus, each connected node can acquire a coarse view of the global topology (from OLSR) and a strong view of its k-hop neighborhood (from Mesh-Mon). We discard any unidirectional links present and combine these two views into a unified graph representing all nodes in the network, but not necessarily all links. We then determine all the critical links and nodes on this graph in $O(V + E)$ time for a graph with V nodes and E links since the algorithm is based on DFS [44]. We also output all nodes in each bi-connected component

in the same pass.

The localized approach proposed by Jorgic et al., uses an identical but easier computation on just the smaller k-hop neighborhood graph. We implemented and use both the centralized and localized approaches in our Mesh-Mon prototype to detect critical nodes. In the centralized case, we output all the articulation points, bridges and compositions of the bi-connected components of the network graph. In a bi-connected component, failure of a single node in that set of nodes will not partition the graph into disconnected components. Stronger definitions of connectivity, such as "tri-connectivity" or "n-connectivity" can also be calculated in a centralized manner.

The localized approach is more scalable, can be used in other mesh networks and has lower computational overhead, but is provably less accurate. The globalized approach we describe for OLSR-based networks is inherently more accurate, but is computationally more intensive for large graphs and requires more network communication overhead. Our solution is to first use the localized approach, and if the localized analysis hints at any critical nodes to then use the globalized approach but, by using an expanding ring search of the topology to verify the presence (or absence) of critical nodes. To confirm a node is indeed critical, a global search is required. Thus, we run DFS on larger topologies only when needed. In either approach, the topology information must be accurate and up-to-date for the results of this analysis to be of practical value.

3.10.5 Partition detection

Closely related to the task of identifying potentially critical nodes is the task of detecting partitions when they occur. Suppose Mesh-Mon detects the presence of a critical node and alerts the sysadmin. The sysadmin could adjust the topology either by moving the node or by adding an additional node in a strategic location. The sysadmin will need to use his or her judgement to maximize use of available resources. If the critical node eventually fails and was a "false positive," then the network is still connected and the network topology monitor can verify the fact (pings will also succeed).

If the node fails and it was a "true positive," then our partition detection algorithm will suspect the existence of two partitions and try to verify the fact by using pings. We detect partitions in Mesh-Mon by studying the change in the reported topology over time and by verifying links are stable over a suitable time

scale.

If a critical link or node fails (or a node moves away), this change will be reflected by the reported network topology. If two nodes can no longer mutually ping each other, but were previously able to do so, each node will independently suspect that a partition has taken place. Each Mesh-Mon node can verify which nodes are still in its partition (through pings and by parsing received Mesh-Mon packets), and guess which nodes are in the other partition. Mesh-Mon will then alert the sysadmin with details about the problem and urge him or her to take corrective action.

In a highly mobile environment partitions can grow and shrink frequently. We found that even in a fully static environment, nodes that have transient links may appear to join a partition and leave it frequently. Transient and temporary link fluctuations, therefore, should be ignored and not be allowed to trigger the partition detection algorithm. A simple solution is not to send an alert unless the suspected partition remains such for a preset period (say 30 seconds). One approach we are investigating is to keep track of the largest stable set of nodes in a partition over a sliding window as the primary components of a partition and listing fringe nodes as those that are not in this set but often join it and leave it over a period of time.

3.11 Social-network analysis

Social-network analysis is normally applied to the study of social networks of actors, usually people and their relationships with other people. In our research domain, we are interested in the positions and roles of individual mesh nodes, and the connectivity relationships between different mesh nodes, which can be characterized in different ways such as direct or indirect, weak or strong. These structures and relationships are worth identifying since they can manifest themselves in certain observable or predictable behaviors of interest. Many social-network analysis techniques and metrics are based on graph theory.

Our goal is to apply social-network-analysis-based techniques to identify properties of individual nodes that can aid a system administrator to manage a mesh network in a more effective manner. While the sysadmin is primarily asked to perform absolute tasks, there may be situations when relative decisions must be made. For example, we posed the following question at the start of this chapter: if the sysadmin had to update a subset of nodes and reboot them, in which order should he or she perform this operation? Since we are interested in relative comparisons between seemingly similar nodes to answer such questions, we need

to develop techniques and metrics to differentiate between nodes and rank them in a distinct manner.

We now detail the techniques borrowed and enhanced from the domain of social-network analysis that can help in providing answers to some of the questions we posed earlier in Section 3.2. We introduce new social-network centrality metrics to study the roles of individual nodes in the network and the relationship of these nodes with their neighbors. We begin by introducing different centrality metrics that are rooted in graph-theory and used often in the analysis of social networks.

Centrality measures can only provide $relative$ measures that can be used to compare nodes against each other at that instant of time for a specific network topology. In the next chapter, we will show how relative ordering proves useful in allowing a sysadmin to decide a prioritized ordering of management tasks on several nodes, such as deciding which nodes should be patched first and in which order.

3.11.1 Degree centrality

One simple way to characterize an individual node in a topological graph is by its degree. The degree of a node in a graph in the mesh context is the number of usable links the node shares with its neighbors, since links are important for routing purposes. A well-connected mesh network is a healthy network. If a node has many neighbors then the failure of a single neighbor should not affect the overall health of the regional network (and thus the global network) adversely. A node with a high degree can be considered well connected and a node with a relatively low degree can be considered weakly connected. The degree of an individual node and the minimum, maximum and average degree of all nodes the entire network are standard characterization metrics in graph theory.

If the global topology is available at a central location, then all the nodes can be quickly ranked according to their degree. However, this degree-based ranking does not convey a good picture of the nature of connectivity in the network since all links are rarely identical. For instance, different links may have varying capacity levels and different latencies. In addition, the existence of neighbor links and their respective qualities fluctuate over time.

3.11.2 Eigenvector centrality

As seen in Figure 3.4, Begnum et al. note that two nodes may have exactly the same degree but need not have similar characteristics [15]. In any network, and especially in an ad hoc or mesh network where nodes must cooperate with each other to route packets, the connectivity of a node depends on the connectivity of its neighbors, not just its own degree. Eigenvector centrality is a centrality measure that is capable of articulating this specific network property.

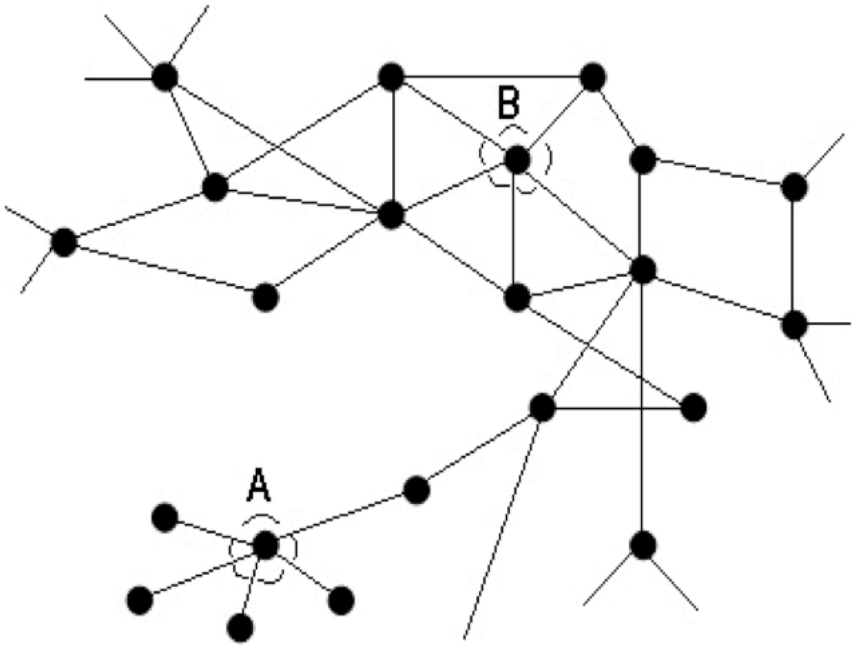

Figure 3.4: Limitations of degree centrality: Note that both nodes A and B have a degree centrality value of 5 (Adapted from [15])

Eigenvector Centrality (EVC) is a concept often used in modern social-network analysis and was first proposed by Bonacich [19, 20]. Eigenvector Centrality is defined in a circular manner. The centrality of a node is proportional to the sum of the centrality values of all its neighboring nodes. In the social-network context, an important node (or person) is characterized by its connectivity to other important nodes (or people). A node with a high centrality value is a well-connected node and has a dominant influence on the surrounding network. Similarly, nodes with low centrality values are less similar to the majority of nodes in the topology and may exhibit similar characteristics and behavior and share common weaknesses. Google uses a similar centrality-based ranking technique (called Pagerank) to rank the relevance of pages in search

results [25].

Our approach is inspired by the work of Bytyci [27], who studied the stability of wired networks using eigenvector centrality calculated on offline traces to rank nodes in the network by using Bonacich's definition of centrality. We use similar techniques as Bytyci (detailed below) in a fully online manner to provide more information about the structure of our mesh network to the system administrator. In particular we focus on changes in the eigenvector centrality over time as indicators of changes in connectivity and potential anomalies. Begnum and Burgess analytically studied centrality with off-line traffic traces on wired networks, but reported limited success in detecting anomalies in different cases [15]. We study centrality measures in an online manner in a mobile mesh network, where changes in the topology may be precursors to impending network faults. Thus, the centrality measure presents an interesting single-dimensional variable that can be analyzed quickly. As an aside, in a recent study of the topology of the structure of the global wired Internet, London was the most central node (based on EVC) on the planet and New York was second [38]. One major drawback of eigenvector centrality is that it can only be calculated accurately in a centralized manner.

Eigenvector centrality is calculated using the adjacency matrix to find central nodes in the network. Let v_i be the i_{th} element of the vector $\vec{v}$ representing the centrality measure, where $N(i)$ is the set of neighbors of node i and let A be the $n \times n$ adjacency matrix of the network. Centrality is defined using the following formulas:

$$v_i \propto \sum_{j \in N(i)} v_j \tag{3.1}$$

which can be rewritten as

$$v_i \propto \sum_{j} A_{ij} v_j \tag{3.2}$$

which can be rewritten in the form

$$A\vec{v} = \lambda\vec{v} \tag{3.3}$$

Since A is an n x n matrix, it has n eigenvectors (one for each node in the network) and n corresponding eigenvalues. One way to compute the eigenvalues of a square matrix is to find the roots of the characteristic polynomial of the matrix. It is important to use symmetric positive real values in the matrix used for calculations [21].

The principle eigenvector is the eigenvector with the highest eigenvalue. The principle eigenvector is

recommended for use in rank calculations [19]. After the principle eigenvector is found, its entries are sorted from highest to lowest values to determine a ranking of nodes. The most central node has the highest rank and most peripheral node has the lowest rank.

This metric is often used in the study of the spread of epidemics in human networks. In the mesh context, a node with a high eigenvector centrality represents a strongly connected node. A worm or virus propagated from the most central node could spread to all reachable nodes in the most efficient manner possible, as opposed to one that was spreading from a node on the extreme periphery. Thus, the central node is a prime target for preventive inoculation or for prioritized software updates.

We calculate three variants of eigenvector centrality for mesh networks. In the first case we use the standard EVC definition calculated with the binary adjacency matrix representing the global topology. This is representative of a mesh network that routes packets solely based on the hop count metric.

In the second case (for OLSR networks only), we use link quality values provided by OLSR to scale the adjacency matrix to obtain the "effective expected adjacency matrix" [27]. We then rank nodes based on their centrality in this new matrix to obtain the "Link Quality (LQ) EVC". Nodes that are connected through more reliable links should get a higher "importance" ranking. Other metrics worth considering as link weights are link capacity and delay measures, but we do not have them accurately available at all times in our online monitoring system. Carreras et al. suggest the use of a T-tolerant adjacency matrix populated with the number of contacts between mobile nodes during an interval of T seconds to study the spread of messages using eigenvector centrality [30].

In our third variant, called the "Gateway (GW) EVC," we consider the Internet as a virtual node in the adjacency matrix and weight the link between any gateway node and the Internet by 10 times the highest weight assigned to other links in the matrix. In this manner, Gateway nodes have the highest centrality and the GW EVC is a simple numerical ranking that reflects connectivity to the Internet. Our hypothesis (validated by our results in Section 4.4.1) is that this metric is a useful tool in understanding network connectivity through gateway nodes and for anomaly detection. For instance, GW EVC can help evaluate how the quality of connectivity to the Internet at each node is affected, when a gateway node fails and subsequently affected clients and other mesh nodes must adjust their routes to connect through an alternate gateway.

In our evaluation, we also considered the case where the adjacency matrix is sparse and may include

disconnected components or isolated nodes, which presents some numerical challenges for online centrality calculation. In such scenarios, the solution is to consider not just the principle eigenvector but the eigenvectors corresponding to the second and third largest eigenvalues. This situation makes automating the EVC calculation process harder, but not impossible.

3.11.3 Betweenness centrality

The position a node occupies in a network can play a role in the node's ability to control or impact the flow of information in the network, as characterized by measures like eigenvector centrality, closeness centrality, graph centrality or betweenness centrality. We focus now on betweenness centrality, which is also called sociocentric betweenness centrality [65]. Betweenness centrality is a key component of the bridging centrality metric detailed below in Section 3.11.5.

The betweenness centrality of a node is defined to be the fraction of shortest paths between all node pairs that pass through the node of interest. So a node with a high betweenness centrality value is more likely to be located on the shortest path between a given node pair in the network, and thus more likely to be important in routing packets. If we assume a uniform distribution of information flow across all node pairs, more information flows through nodes with higher betweenness centrality.

3.11.4 Egocentric betweenness centrality

Although betweenness calculation appears to be computationally intensive, since all pairs of shortest paths must be computed (typically $\theta(n^3)$), a fast (though centralized) technique to compute betweenness centrality that runs in $O(VE)$ time and uses $O(V + E)$ space for undirected unweighted graphs with V nodes and E edges was presented by Brandes in 2001 [24].

An alternative approach is to calculate the betweenness of a node using its egocentric network as opposed to the global network topology. In social networks, egocentric networks are defined as networks of a single actor together with the actors they are directly connected to. Thus, for mesh networks, we calculate egocentric betweenness on the one-hop adjacency matrix of a node. This metric can be calculated in a distributed manner and plays an important role in a new centrality metric for Mesh-Mon that we introduce in Section 3.11.6.

3.11.5 Bridging centrality

Bridging Centrality (BC) is a relatively new centrality metric that was introduced by Hwang et al. in 2006 [85]. Bridging centrality can help discriminate *bridging nodes*, that is, nodes with higher information flow through them (assuming a uniform distribution of flows) and locations between highly connected regions.

The Bridging Centrality of a node is the product of its sociocentric betweenness centrality C_{Soc} (Section 3.11.3) and its bridging coefficient $\beta(v)$. The Bridging Centrality $BC(v)$ for a node v of interest is thus defined as:

$$BC(v) = C_{Soc}(v) \times \beta(v) \tag{3.4}$$

The bridging coefficient of a node describes how well the node is located between high-degree nodes. The bridging coefficient of a node v is thus defined as:

$$\beta(v) = \frac{\frac{1}{d(v)}}{\sum_{i \in N(v)} \frac{1}{d(i)}} \tag{3.5}$$

where $d(v)$ is the degree of node v, and $N(v)$ is the set of neighbors of node v.

According to the authors, betweenness centrality indicates the importance of a given node from an information-flow standpoint, but it does not consider the topological position of the node. On the other hand, the bridging coefficient measures only how well a node is located between highly-connected regions, but does not consider information flow. "Bridging nodes" should be positioned between clusters and also located on important positions from an information-flow standpoint. Thus, the BC metric is an attempt to combine these two distinct metrics by giving equal weight to both factors [85].

Based on their empirical studies, the authors recommend labeling the top 25th percentile of nodes as ranked through BC as "bridging nodes," nodes that are more bridge-like and lie between different connected modules [85]. The authors present results on which nodes are selected by this metric for different networks, and study the impact of removing the highest-ranked bridging nodes from a yeast metabolic network with 359 nodes and 435 edges, as measured by changes in the clustering coefficient, average path length and number of singletons generated.

We note that these bridging nodes are different from the articulation points of a graph that we discover

during topological analysis in Mesh-Mon. Our interest in this metric stems from our discovery that these bridging nodes provide the system administrator with a prioritized set of nodes to monitor from a robustness perspective (more details in Section 4.4.3). The main drawback of BC is that bridging centrality as defined here can only be calculated in a centralized manner with global information, since it depends on sociocentric betweenness centrality. Given the distributed nature of the analysis engine in Mesh-Mon, we require a distributed version of the BC metric that is equivalent in its accuracy and can be calculated in an efficient manner.

3.11.6 Localized bridging centrality

We now present our variant of Bridging Centrality that we call Localized Bridging Centrality (LBC). As the name suggests, we define $LBC(v)$ of a node v using only local information, as the product of egocentric betweenness centrality $C_{Ego}(v)$ and its bridging coefficient $\beta(v)$. The definition of $\beta(v)$ is unchanged from Equation 3.5. LBC is thus defined as:

$$LBC(v) = C_{Ego}(v) \times \beta(v) \tag{3.6}$$

Marsden [111] discovered empirically that egocentric betweenness values have a strong positive correlation to sociocentric betweenness values (calculated on the complete network graph) for many different network examples. Everett and Borgatti [60] also present a similar conclusion that the two metrics are strongly correlated for most networks. The authors also provide a few synthetic examples where egocentric and sociocentric betweenness values do not have a positive correlation.

Daly and Haahr recently applied egocentric betweenness centrality as the basis for a distributed routing protocol in a delay tolerant network [46]. Our approach used to calculate LBC is inspired by the work of Marsden and the recent work by Daly and Haahr.

The key benefit of our approach is that LBC is computationally easier to calculate than BC, and can be calculated in a parallel or fully distributed manner. Secondly, as shown by Marsden [111] and by Everett and Borgatti [60], there is a strong correlation between egocentric betweenness and global betweenness values for most networks, so LBC values should correlate well with BC values. Indeed, our results in Section 4.4.3 show this intuition is correct. While individual nodes can calculate their own LBC metric in a

fully distributed manner, to determine the global rank of each node a central node must aggregate all LBC values or all nodes must use a distributed consensus-based ranking algorithm.

We explore the utility of the LBC metric in our evaluation. As with the Bridging Centrality metric, the LBC metric can help the system administrator identify clusters, their boundaries and the bridging nodes in the mesh network.

A possible alternative definition for the bridging coefficient is to use eigenvector centrality as a substitute for degree centrality in both the numerator and denominator. The disadvantage is the high computational cost of calculating the eigenvector centrality and its centralized computation. We may explore this variant in future work.

3.11.7 Localized load-aware bridging centrality

Our LBC metric is based on the concept of betweenness centrality, which implicitly assumes that all paths between all node-pairs are equally likely to be used. Thus, our LBC metric has the drawback that, like the BC metric, it implicitly assumes that a uniform distribution of traffic flows will exist between all node-pairs in the network. In a real mesh network used to provide last-mile Internet access, however, the distribution of traffic flows will almost certainly be non-uniform and it is likely that Gateway nodes (connected to the Internet) will experience relatively higher traffic loads.

Taking the traffic load into consideration, we introduce the new Localized Load-aware Bridging Centrality (LLBC) metric designed for distributed analysis of bridging nodes in wireless mesh networks. We first compute the traffic load (measured in bytes) in each node locally as the sum of all bytes originating at the node (Out), destined for the node (In), and twice the number of bytes forwarded (Fwd) by that node. We count the forwarded bytes twice in the summation since they are both received and sent by the node.

$$Load(v) = In(v) + Out(v) + 2 \times Fwd(v) \tag{3.7}$$

We use the measured traffic load to calculate the Load Coefficient (β_l) as the ratio of the traffic load of a given node to the sum of the traffic loads of its one-hop neighbors. As the load of a node increases (relative to that of its neighbors' loads), so do the chances of the node becoming a traffic bottleneck, analogous to the reasoning used to define Bridging Coefficient for BC and LBC .

$$\beta_l(v) = \frac{Load(v)}{\sum_{i \in N(v)} Load(i)} \tag{3.8}$$

For homogeneous networks where all radios are identical, using just the total load is sufficient for Load Coefficient calculations. However, it is possible for networks to have many different nodes with different radios, such as the powerful radios used for access points versus the radios used for low-powered sensor nodes. For such heterogeneous networks, we need to normalize the Load values in the numerator and denominators of Equation 3.8. We do so by dividing each Load value by the interface's actual Capacity (or the theoretical capacity of the interface of that node) for that sampling period to obtain the Utilization ratio metric for each node's radio interface. This Utilization metric (see Equation 3.9) for each node provides an additional simple metric that a sysadmin can use to study network traffic usage through visualization (in a graph) or a ranking (in a table).

$$Utilization(v) = \frac{Load(v)}{Capacity(v)} \tag{3.9}$$

To obtain a more general definition of the Load Coefficient (denoted now as β_t) which can be used in any mesh with heterogeneous or homogenous devices, we now substitute the Load metric wherever it appears in Equation 3.8 with the Utilization metric :

$$\beta_t(v) = \frac{Utilization(v)}{\sum_{i \in N(v)} Utilization(i)} \tag{3.10}$$

For homogeneous networks, β_t reduces to β_l since the same Capacity value is a common factor for both the numerator and denominator in Equation 3.10.

Finally, as shown in Equation 3.11, the LLBC value of a node is defined as the product of its Ego-Betweenness (C_{Ego}) and its general Load Coefficient (β_t), using a construction (and reasoning) similar to the one we used earlier for the LBC metric defined in Equation 3.6:

$$LLBC(v) = C_{Ego}(v) \times \beta_t(v) \tag{3.11}$$

The LLBC metric takes into equal account the relative traffic load and the relative positions of neighbor-

ing nodes over the sampling period. Like the LBC metric, the LLBC metric too can be calculated in a fully distributed manner. Over time, the measured traffic load at different nodes will change and nodes that reboot will have their counters reset to zero. Thus, we periodically sample LLBC values and consider the traffic load during the sampling period instead of cumulative values. The LBC metric relies only on the degree of each node, and is thus less useful for real mesh networks where traffic flows are variable and non-uniform.

For two nodes that have equal LLBC values (the nodes may even have the same Ego-Betweenness and Load Coefficient values), the higher Utilization metric of one of the nodes can be used as a tie-breaker to decide ordering. If both the Utilization metrics also happen to be equal, then the node with the higher Load metric wins the tie-breaker.

Although two distant nodes may have identical Load Coefficients (β_t) but greatly different traffic loads, this difference does not affect our LLBC calculation in any negative way. We show that LLBC serves its primary purpose well, which is to identify "bridging nodes," in Section 4.4.4.

Summary

With the introduction of the LBC and LLBC metrics, we conclude the social-network-analysis aspects of Mesh-Mon. Both LBC and LLBC can be calculated efficiently in a distributed manner, thus complementing the distributed design used in Mesh-Mon. We present our results from the evaluation of the LBC and LLBC metrics when applied to mesh networks in the next chapter.

It is important to remember that centrality measures can only provide *relative* measures that can be used to compare nodes against each other at that instant of time and for that specific network topology. This ranking allows a system administrator to prioritize management tasks on several nodes, such as deciding which nodes should be updated (and taken offline for the process) and in which order to do so, while creating minimal disruption to the important parts of the network, and to identify the bridging nodes that are most likely to cause partitions due to failure or mobility.

3.12 Diagnosis engine

In this section, we detail the features of the diagnosis engine. Our diagnosis engine uses all the information it receives from the cooperating MCs and MNs. When the engine detects faults or anomalies, Mesh-Mon

sends alerts to sysadmins and neighboring nodes. These alerts describe a hypothesis about the cause of the problem. The alerts often trigger secondary diagnostic tests to verify the root cause of the problem by gathering additional information. The hypothesis can aid the sysadmin in determining what triggered a fault in the network and provide insight on how to deal with it.

Mesh-Mon's goal is not just to detect a problem, but whenever possible to determine the root cause, resolve it automatically if possible, and to provide timely information and guidance to the administrator. The analysis engine within Mesh-Mon controls and runs the tests described earlier (topology analysis, social analysis, configuration and health monitoring). Based on the detection of any anomaly or negative test result, in some cases, we run a secondary diagnostic tests and generate corresponding alerts. Certain secondary tests are triggered on a mesh node, based on alerts received from a Mesh-Mon-Ami client or other nodes. For example, a node reporting high packet-loss rates between a source and destination triggers additional link-quality checks between all individual links in the multi-hop path. If a mesh node finds it has no neighbors, it will try an increased transmission power-level setting. Some actions taken may have adverse consequences. Increasing the power level can lead to increased interference, while reducing power levels could lead to a loss of some neighbor links.

Our analysis engine is similar to an expert system, since it is programmed using domain knowledge and emulates the analytical process of human experts. To design our analysis engine, we created a state-transition diagram consisting of node states and state transitions based on alerts and results from diagnostic tests. State transitions can include actions such as issuing an alert to a sysadmin, the setting of a parameter, or the execution of another test. The goal is to reach a terminal state that identifies the problem faced by the mesh node, client or network. The engine must also take into account the current state of the node, the results from diagnostic tests periodically run, and messages that are received from other nodes (that describe the current or recent state of those nodes). After reaching a terminal state, the engine returns to the start state and repeats the process.

We faced several challenges in implementing our system such as devising a simple method of expressing the current state, how to deal with analog information vs. discrete information, and which actions should be taken on each transition. We try to group several similar states together to avoid state explosion and design diagnostic tests that result in discrete "Yes" or "No" answers. For example, to test configuration issues, a

diagnostic test will attempt to answer the question "Is the configuration *correct*?". This test may comprise of several subtests, one or more for each configurable component, such as "Is the channel setting correct?" or "Is the Transmission Power within correct limits?". By designing our tests in this manner, we limit the transitions to the two possible cases for each test; either "Yes" or "No".

Our engine generates a hypothesis about the cause of the fault based on the outcome of all the individual tests in a short time window. We present a simplified flowchart that represents the analysis engine (without secondary tests) in Figure 3.5. All diagnostic information, the rules applied and corresponding results are carefully logged and transmitted as an alert to the sysadmin. All generated alerts are sent to the sysadmin and piggybacked on packets Mesh-Mon nodes send to their k-hop neighbors.

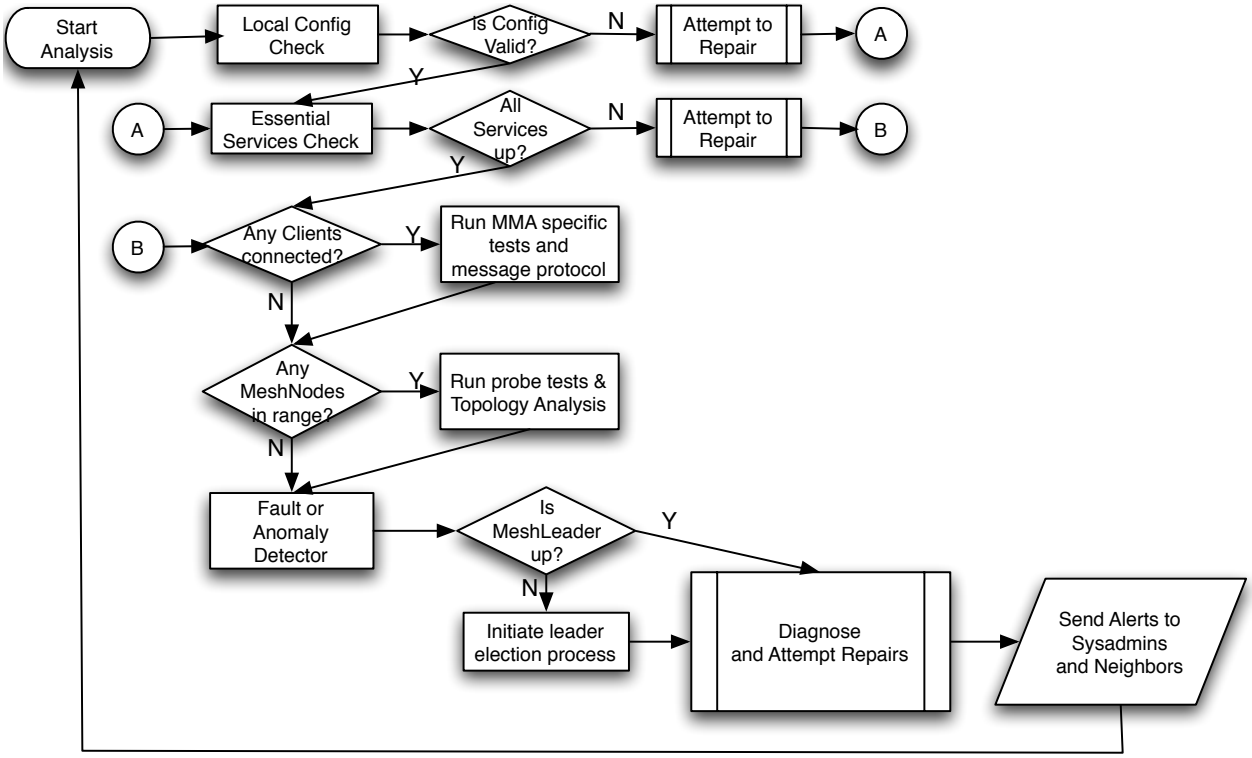

Figure 3.5: Analysis engine flowchart

The sysadmin is responsible for verifying the hypothesis and determining the course of action when Mesh-Mon is unable to automatically resolve the problem.

In our present implementation we only analyze locally generated information and information received from k-hop neighbors. For detecting problems in large networks, we propose that MeshLeaders communi-

cate with each other and cooperatively diagnose problems affecting multiple neighborhoods.

3.12.1 Rule-based diagnosis

Our analysis engine periodically runs through a series of tests on both mesh nodes and MMAs. From the viewpoint of Mesh-Mon running on an individual MN, Mesh-Mon's tests attempt to answer the following questions:

1. Is the MN's configuration valid?

   This test has a series of subtests particular to the individual node, which involve comparisons with information from neighboring nodes. If any of them fail, the engine executes the default action (usually to reset the variable setting to a safe default).

2. Are all essential services (such as `olsrd`, `iptables` and `dhcpd`) running?

   If any essential service is not running, we attempt to restart it. If the restart is unsuccessful, we generate a critical alert and contact the sysadmin.

3. Are any clients currently associated with the MN?

   If so, then the MN communicates with their MMAs to exchange information with them. The MN sends the MMA copies of critical alerts that have not been acknowledged by the sysadmin.

4. Are any other MNs in range?

   If the MN is isolated, it will try boosting the power level to try to contact other MNs. If that does not work, or if the MN is isolated for a long period, the interface is reset and restarted.

5. Based on the best topology information currently available, was there a change in the neighborhood graph since the last analysis run?

   If a change is detected, has a partition been created or healed? Are there new potential critical nodes?

6. Can the MN ping or send and receive packets to other MNs, gateway node, its clients, and external hosts within acceptable performance limits? Based on the results of the previous test, we identify a

Table 3.2: Mesh-Mon alert generation and processing

Scenario / Symptoms	Alert Type	Subtype	Default Local Action	Action On Alert Receipt
Radio is on incorrect channel	Config	Channel	Reset to default channel	Check local channel
`dhcpd` crashed	Services	DHCP	Restart service	Check local services
`iptables` service crashed	Services	iptables	Restart service	Check local services
Broadcast storm suspected	Anomaly	Broadcast	No action	Check broadcast rate
No MCs or MNs are in range	Anomaly	Isolation	Reset radio & boost power	Ping sender
Change in EVC	Anomaly	Centrality	No action	Recalculate EVCs
Drop in avg. throughput	Anomaly	Throughput	No action	Probe links between pair
Jump in dropped frames	Anomaly	Congestion	No action	Check local health metrics
MeshLeader is unreachable	MeshLeader	Re-election	Announce new MeshLeader	Check validity of new leader
New critical node detected	Topology	Criticality	Inform MeshLeader	Run topology analysis
Test site is inaccessible	Fault	External	Test if gateway is reachable	Check test site access

set of critical nodes and randomly selected nodes and clients to probe. The results are analyzed for significant deviation from recent values and long-term trends.

7. Did the network health monitor detect an anomaly?

   We analyze statistics of the health indicators of our local interfaces (such as the mean, standard deviation and moving averages) to detect outliers against both recent information and long-term data.

8. If any alerts were received from a Mesh-Mon-Ami or from other Mesh-Mon nodes or sysadmins, do they correlate with alerts generated locally?

   Certain received alerts may trigger an additional test, the result of which may be sent back to the originator (or more nodes) depending on the alert type and the test result. In Table 3.2 we describe a few scenarios, the corresponding local actions taken, the type of alert that will be generated, and actions that neighbors will execute upon receipt of such alerts.

9. Is the MeshLeader still reachable within k hops?

   If not, it may be time to initiate the protocol to elect a new MeshLeader for the neighborhood.

During each periodic analysis cycle, all the above tests are run in series. The results from all the tests are considered collectively to generate the present "hypothesis" in the "Diagnose" phase shown in Figure 3.5 which determines if any additional tests or corrective actions (if any) must be performed. Finally after

running all the tests in each analysis cycle, every Mesh-Mon node generates its periodic reports and alerts (if any), sends them to the sysadmin and its k-hop neighbors (and to its MeshLeader, which must be within k hops) and attempts local repairs as needed.

3.12.2 Mesh-Mon-Ami analysis engine

The Mesh-Mon-Ami clients run a similar set of tests to diagnose local problems that are specific to the hardware and software configuration of the MMA, as well as running their own analysis of neighborhood and global network health. The MMA assists in the delivery of packets between disconnected partitions whenever possible and does analysis on the packets it receives as well. If the sysadmin is not reachable due to partitioning or other failure, then we use our DTN-like message-passing mechanism in an effort to eventually contact the system administrator through the MMA running on the MC.

We consider the following order of tests running on a Mesh-Mon-Ami client:

1. Is the client's configuration correct?
2. Is the MC associated with a MN?
3. Is `dhcpclient` running and does the client have a valid IP and DNS configuration assigned by its MN?
4. Can the client ping or communicate with its MN, gateway nodes, and external hosts within acceptable performance limits?
5. Did the user signal an alert? (We allow a human user to select from a list of pre-defined alerts, encoded in a manner that Mesh-Mon can comprehend).

3.12.3 Mesh-Mon analysis example

We present a short example that illustrates our analysis approach. Assume that a Mesh-Mon node is unable to access a default external test site, say, www.meshmon.com. The node will run through its local tests indicated in Figure 3.5, and will detect the problem at the "Fault or Anomaly Detector" stage. Assume that all other local checks on the MN have passed correctly. After running through the tests, the engine will reach

the "Diagnose and Attempt Repairs" step. At this stage, Mesh-Mon will check recent alerts from neighbors (if any), run additional secondary tests or attempt local repairs. For this specific fault, the MN will verify that the Internet Gateway is reachable (through `ping`) and that the address is resolvable through the Domain Name System (DNS) nameserver. If the test results recommend a local repair, such as resetting the DNS settings in `/etc/resolv.conf` to a safe reference value, then Mesh-Mon will attempt the repair and check its outcome.

Since our node has not received any alerts from its neighbors (till now) and the local repair has failed, the MN sends an appropriate alert message to the sysadmin and its neighbors. The neighbors parse the received alert and run their own corresponding local diagnosis rules (refer to Table 3.2). In this case the neighbors would also try to access the test site. Since the neighbors do not have difficulty reaching the test site, the problem is probably at the original node. The original node will periodically loop through its analysis routine and if the same test fails again, it will send an alert with a higher priority to the sysadmin. The sysadmin can then remotely login to the MN and manually attempt to repair the problem. Perhaps DNS name resolution was disabled after a remote software update and the sysadmin must roll back the update. Complex problems such as this one require human intervention, while others can be fixed automatically by Mesh-Mon. Problems such as congestion may have no direct solution, but the sysadmin will be made aware of the extent of a problem through multiple alerts from multiple Mesh-Mon nodes.

3.13 Implementation details

It is essential that Mesh-Mon and Mesh-Mon-Ami do not overload the mesh nodes that they are monitoring. Since the mesh could be operating for days or months it is important that there are no memory leaks, race conditions or other bugs that can cause Mesh-Mon crash or use an unreasonable share of local resources.

3.13.1 Scheduler-based design

Both Mesh-Mon and Mesh-Mon-Ami implementations are scheduler driven and written in the C programming language. A task scheduler in the main() function of the Mesh-Mon program cycles through a queue of pending tasks, and (using a timer) checks if a task is due to be executed. The primary tasks are to process local information for analysis and to await the receipt of management packets. New tasks, such as sending

different types of packets to neighbors (or the sysadmin), are added to the queue periodically or as needed. Certain tasks, such as running a bandwidth probe test, require several seconds. Such tasks run in a parallel background thread and a new task is added to the scheduler to process the results of the probe at a future time.

To prevent the CPU from blocking in a busy waiting loop, waiting for packets to arrive, we use the `pselect()` function to poll sockets used for communication of Mesh-Mon packets. This approach allows us to run Mesh-Mon with a low CPU overhead.

3.13.2 MeshLeader

For the leader election algorithm, we considered two schemes. In the first scheme (which is partially implemented), the node with the lowest numeric ID in a neighborhood is selected as the MeshLeader and remains in that role until it fails. In the second scheme, we elect the node with the highest EVC ranking as the MeshLeader. Such a node is likely to be the in the most central position in the neighborhood, which is preferable to selecting a MeshLeader on the periphery of the network.

Our current implementation only supports a single MeshLeader. For larger networks, we would need to add support for the communication protocol and the analysis mechanisms for multiple MeshLeaders.

3.13.3 Analysis engine

One limitation is that our analysis engine cannot be modified at run-time. All the rules that decide which tests to run, which actions to take, and which alerts to send cannot be changed without recompiling Mesh-Mon source code and updating the binary on all nodes running Mesh-Mon. Second, all rules and actions are hardcoded in C code.

A more elegant approach would have been to design all the rules, actions, alerts, and so forth in an external source file with its own language. This approach would also involve building a custom compiler that can translate the custom language to C code or developing a runtime interpreter to execute these externally defined rules.

Our ongoing development of rules and diagnostic tests is based on our experience and expertise in understanding the dependencies between components and the overall design of the mesh network. In the

future, automated dependency-mapping tools may help to improve the methodology used to determine the rules for our rule-based diagnosis engine.

3.13.4 Anomaly Detection

The anomaly detection unit is not fully implemented. The anomaly detection module shown in Figure 3.5 represents a conceptual placeholder for a generic anomaly detection algorithm. We envisioned marking any monitored variables whose mean values were greater than three times their standard deviation as anomalous, but there are many other algorithms possible, such as the ones presented in Section 5.10.

3.13.5 Social network analysis

To calculate EVC, we solve the adjacency matrix for eigenvectors and eigenvalues using a numerical linear algebra package called Octave, which is an open-source alternative to MATLAB [56]. Costenbader and Valente showed empirically that eigenvector centrality calculations are not very meaningful when calculated using incomplete topology data [45]. We thus obtain the most accurate global topology available in an adjacency matrix format on all nodes, as described earlier for Dart-Mesh running OLSR. We do not calculate the LQ and GW EVCs for AODV networks, since the AODV protocol does not provide a link-quality metric. For other networks we propose that a single super MeshLeader run centralized calculations only to calculate EVC (since EVC can only be calculated in a centralized manner)

To calculate the load coefficient for the LLBC metric that we specifically designed for distributed computation in Mesh-Mon, we need to calculate the traffic at each node. This information is partially provided by statistics provided by the wireless interfaces. We collect information from `iptables` using the default filtering and logging rules, to identify how much traffic is flowing within the mesh network, and how much traffic is destined for external networks (such as the Internet).

3.14 Summary

In this chapter we describe Mesh-Mon, our mesh management system and several implementation-specific details. In the next chapter we discuss the evaluation of our system on Dart-Mesh.

Chapter 4

Evaluation

In this chapter, we describe our experimental evaluation of Mesh-Mon running on Dart-Mesh. Since there is no standard methodology or benchmark that is used to evaluate management systems, we present results from experiments on Dart-Mesh that reveal how quickly Mesh-Mon detected a problem, the accuracy of the diagnosis, and on the CPU utilization and bandwidth overheads. These are all factors that are important to evaluate the scalability, accuracy and efficiency of Mesh-Mon. All the other network-state information that our system logs helps to reveal insights into short-term and long-term usage trends in a real mesh testbed.

We do not possess implementations of other management systems or the means to quantitatively compare them directly against Mesh-Mon. We thus present a qualitative comparison of Mesh-Mon against other comparable systems in Chapter 5.

4.1 Experimental setup

We built *Dart-Mesh,* a two-tier mesh testbed using 16 dual-radio Linux devices as described earlier in Chapter 2. Dart-Mesh is a live system deployed on all three floors of our department building (see Figure 4.1). The same hardware was used earlier in a static residential setup [5].

To recap, each Dart-Mesh MN has one 802.11b interface in *Master mode*, creating the access tier, and the second 802.11b interface in *Ad hoc mode,* creating the mesh tier. The access tier appears to mobile clients as a Wi-Fi infrastructure to which they can associate and get network service. The mesh tier routes packets among mesh nodes using either OLSR or AODV.

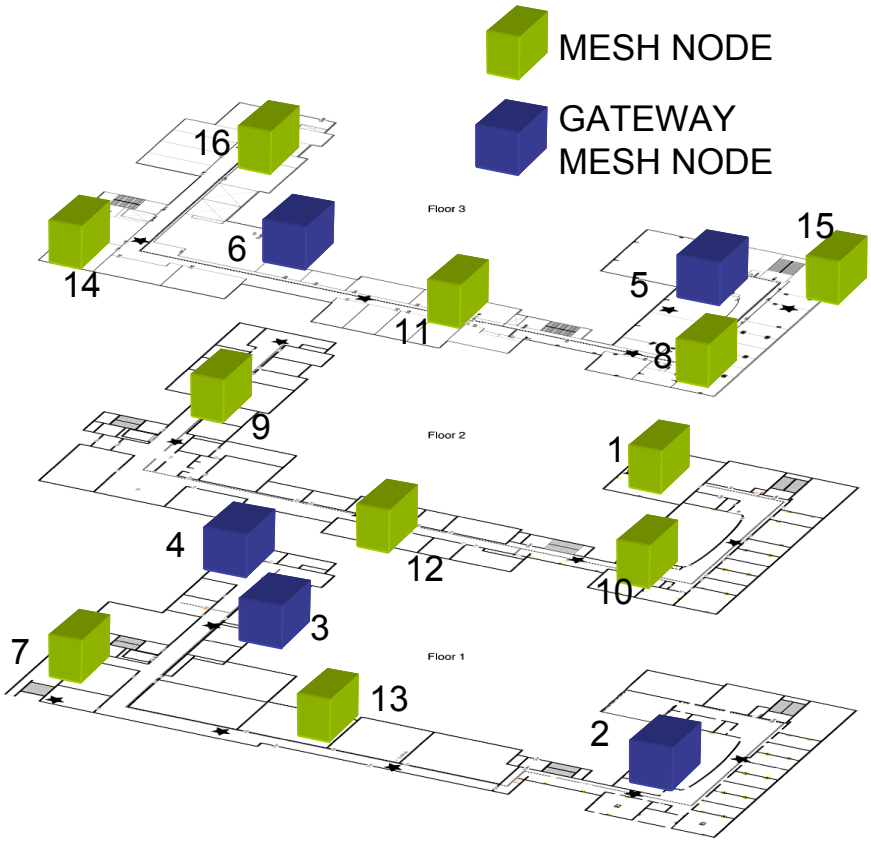

Figure 4.1: Experimental Dart-Mesh deployment with 16 nodes in the Sudikoff Lab for Computer Science

For normal mesh usage, we ran OLSR [178] because OLSR has built-in support for multiple gateways to the Internet, while AODV does not. To demonstrate Mesh-Mon's versatility, we used AODV-UU [128] since it supports Linux 2.6 kernels, while kernel AODV [96] (used earlier with our mesh nodes [5]) only supports the older 2.2 and 2.4 kernels and is no longer maintained.

We arranged all 16 mesh nodes (or a smaller subset of them) in different configurations during our evaluation. Five mesh nodes are connected to the wired Internet and act as Internet gateways, as shown in Figure 4.1. The gateway nodes perform Network Address Translation to help manage connections between the mesh and the Internet. Prior to deployment, all mesh nodes and mesh clients are pre-configured to a fixed common channel respectively.

Overall, the mesh was quite stable and reliable when configured properly. Although we ran our experiments on a real deployment, we introduced failures in controlled experiments to test Mesh-Mon and evaluate our prototype.

In our tests we consider two types of sysadmins: a stationary sysadmin and a mobile sysadmin (running MMA). We collect all alerts destined for the sysadmin's Mesh-Admin process in a time-stamped log on his client laptop. We manually scrutinize the data collected in the logs on each individual node, both during and after a controlled experiment. At present, the output of our system is primarily textual information in the form of logs on each node, alerts and reports on the sysadmin's console, and a few automatically generated graphs.

4.2 Experimental results

During the course of our development, we tested several publicly available routing protocols [96, 128, 178]. To demonstrate Mesh-Mon's ability to support multiple routing protocols, we run most tests in our evaluation sequentially, first with OLSR and then with AODV using the same topology.

4.2.1 Dart-Mesh performance

We present general performance results for both protocols measured on Dart-Mesh (as configured in Figure 4.1) in Table 4.1. Performance decreases rapidly with the increasing number of hops due to the shared nature of wireless media. Thus, a self-contained Dart-Mesh network with 100 nodes may be impractical

Table 4.1: Dart-Mesh performance

Protocol	Category	1 hop	2 hops	3 hops
OLSR	Average TCP throughput	3.36 Mbps	784 Kbps	224 Kbps
	Average ping RTT	8.08 ms	13.74 ms	34.29 ms
AODV	Average TCP throughput	2.87 Mbps	654 Kbps	108 Kbps
	Average ping RTT	6.89 ms	51.54 ms	134.28 ms

Table 4.2: Routing protocol overhead for network of 16 nodes shown in Figure 4.1

Protocol	Average Overhead per node
AODV	2.53 Kbps
OLSR	13.5 Kbps

for most modern traffic workloads. A mesh used for Internet access is certainly feasible with an adequate number of Internet gateways.

When we asked several users of the mesh to provide feedback on their experience when using the mesh, most people stated that they did not realize that it was a mesh network, since the usage experience was identical to that of using an infrastructure Wi-Fi network. Most users of the mesh found the performance to be equal to or superior to that of the existing wireless network in Sudikoff, while a few users found the performance to be slower.

4.2.2 Routing protocol control overhead

Routing-protocol control overheads with AODV are in general much lower than OLSR, because AODV is a reactive protocol. However, AODV routes solely on hop-count and thus often uses inefficient routes with fewer hops of lower link quality. As shown in Table 4.2, the average routing control overhead per node of OLSR was six times that of AODV. The routing overhead for AODV depends most on client usage patterns, whereas for OLSR the overhead depends on the size and stability of the entire network, as well as configuration parameters. AODV periodically forgets unused routes and only stores current one-hop neighbors. A ping test for a multi-hop route could require a fresh route-request cycle that could take a whole second to complete. Thus, the average latency in AODV networks is higher.

4.2.3 Mesh-Mon bandwidth usage

Each Mesh-Mon and MMA packet is 1408 bytes long (to stay within the 1500 standard MTU limit) and may include multiple alerts. Each node generates one Mesh-Mon packet per minute by default. MNs communicate these packets through flooding within the k-hop neighborhood. In our network all 16 nodes were in one k-hop neighborhood. Shorter alert-only packets (100 bytes) are sent to the sysadmin as needed.

The average management overhead purely due to Mesh-Mon, with 16 nodes sharing a single Mesh-Leader was at most 9.8 KBps per node in a network with a diameter of 5 hops. This overhead could be reduced by letting each node rebroadcast packets on a probabilistic basis, by compressing data, or by using multiple MeshLeaders.

In addition, each MN runs `Iperf` once every 2 minutes (to estimate bandwidth), which adds 600 KB of bursty traffic. This overhead can be reduced by decreasing the frequency of this test or by configuring `Iperf` to use smaller payloads. However, we found that using `Iperf` with smaller payloads led to inaccurate bandwidth estimates.

Finally, multiple failures lead to short bursts of additional Mesh-Mon-to-sysadmin alert traffic, but this traffic was never high enough to cause congestion or affect user traffic in our tests with static networks.

The combined traffic overhead of Mesh-Mon with variable network sizes and a single MeshLeader is shown in Figure 4.2 for both AODV and OLSR. The overhead was independent of the routing protocol and depends on the size, topology, network density, and the rate of failure. Mesh-Mon flooded traffic grew rapidly ($O(n^2)$) and dominates as the network size increased, justifying the need for multiple MeshLeaders for larger networks, while the average `Iperf`-induced traffic increased only slightly as the network grew and longer multi-hop routes were created.

Mesh-Mon was able to provide identical management functionality with similar overhead for either AODV or OLSR. Mesh-Mon generated a similar volume of alerts and accuracy for identical tests run on both networks. The only difference between the two protocols, in our implementation, is that for OLSR each MN had local access to the global topology (bypassing MeshLeaders), but in AODV, global topology information was gathered through neighbors and MeshLeaders.

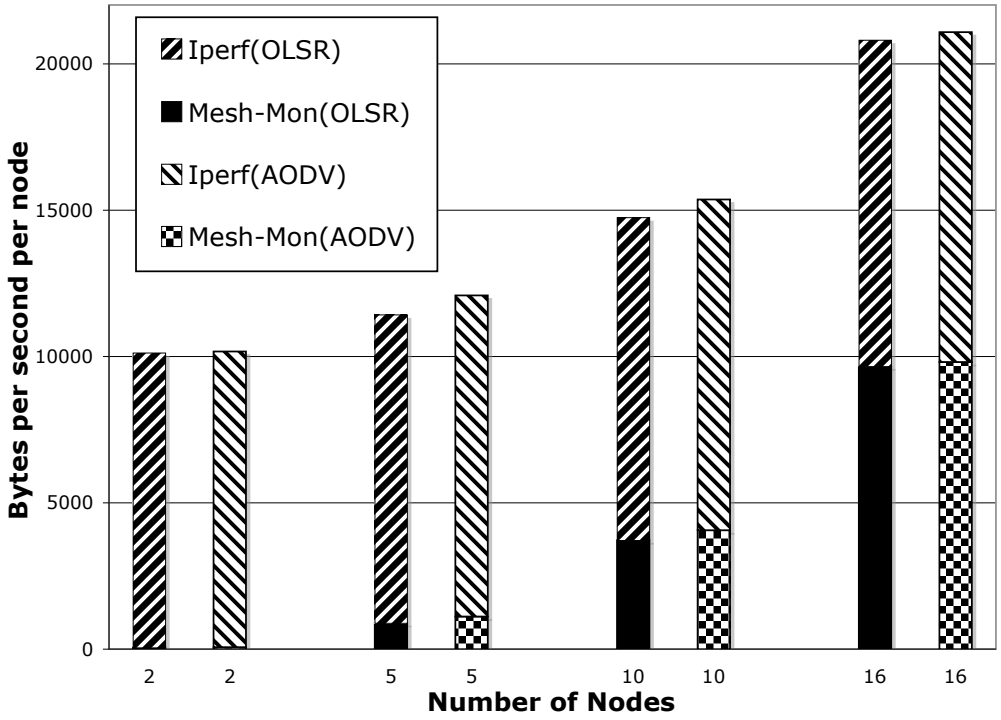

Figure 4.2: Average combined monitoring overhead per Mesh-Mon node

4.2.4 Scalability through multiple MeshLeaders

In Figure 4.3, the rectangles represent mesh nodes labeled by their NodeIDs. Since we did not completely implement our MeshLeader election protocol, to evaluate the scalability impact of using MeshLeaders, we statically assigned two nodes out of the 11 nodes in Figure 4.3, nodes 2 and 11, as MeshLeaders (distinguished by a darker outline). These nodes are great candidates for MeshLeaders where $k = 2$ hops, since any node in each cluster is within 1 hop of the nearest MeshLeader, and the two MeshLeaders are exactly two hops apart from each other. The labels on the links represent instantaneous ETX link quality values for that snapshot and can be ignored.

For this study, we sent out 10 Mesh-Mon packets every minute to allow Mesh-Mon to create a higher load on the network. With $k = infinity$ the Mesh-Mon communication protocol flooded packets throughout the network until every node had received every packet. With $k = 2$, Mesh-Mon nodes did not forward any management packets after two hops, limiting their spread, and thus reducing the total traffic on the network.

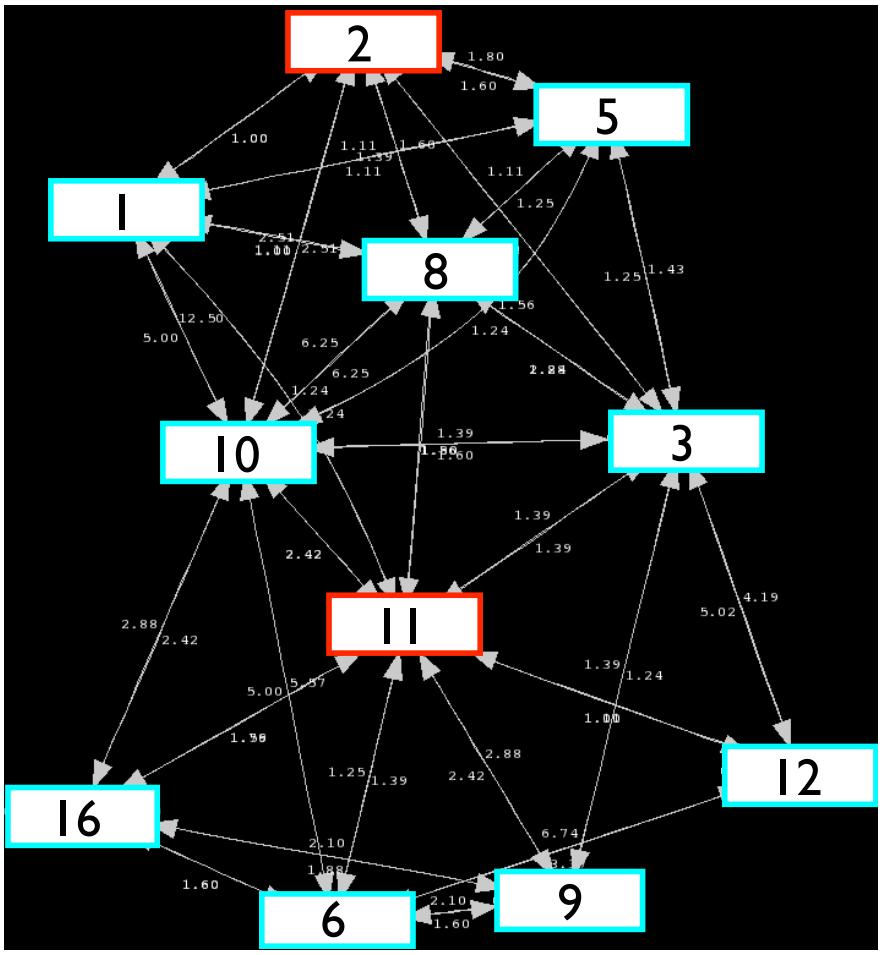

Figure 4.3: Network topology with multiple MeshLeaders

As seen in Figure 4.4, the number of packets sent and thus number of bytes sent per node was almost exactly the same with or without MeshLeaders, but when MeshLeaders are employed, the number of forwarded packets and thus the number of received packets was significantly reduced (36% and 12% lower respectively). For a larger network the savings would be even greater as the diameter of the network increases. Thus, the use of multiple MeshLeaders reduced the communication overheads of Mesh-Mon and added scalability to our system design. Using MeshLeaders had no effect on the overhead due to `Iperf` for estimating available bandwidth.

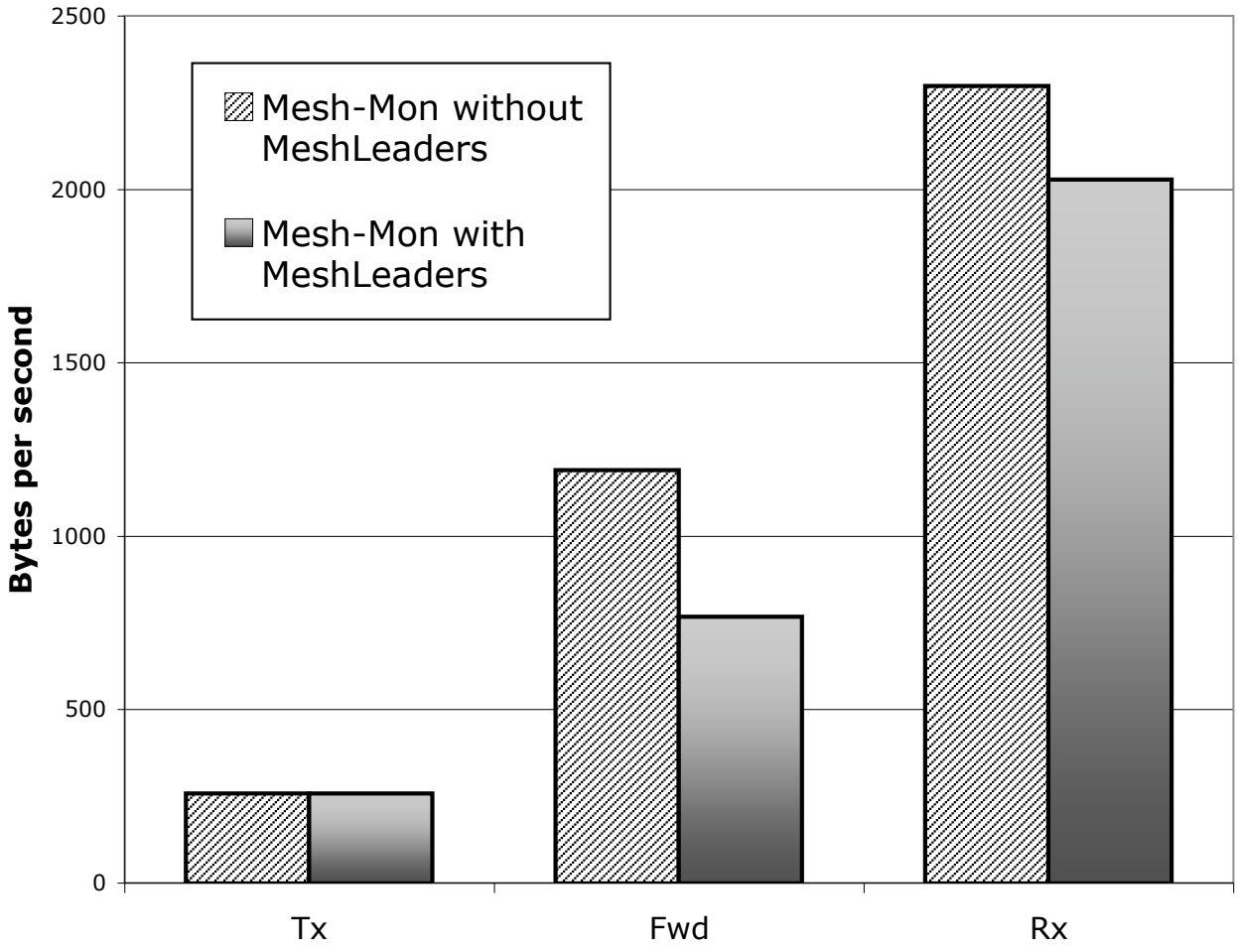

Figure 4.4: Breakdown of the average monitoring overhead per node per second for the network in Figure 4.3 with and without MeshLeaders

4.2.5 Mesh-Mon CPU usage

The CPU overhead of running Mesh-Mon was reported as under 1.5% using the Unix `time` command on our 1.3 GHz Intel Pentium III processor-based nodes. On this hardware, it may be feasible to extend Mesh-Mon's abilities to perform traffic analysis by using a packet-capture program to compute statistics on the composition of individual packets. For our traffic-load analysis, we use aggregate traffic statistics provided by using the built-in `iptables` utility available on all Linux systems.

4.3 Other test cases and qualitative results

We now focus on specific test cases that demonstrated individual features of Mesh-Mon. Our objective in performing the following tests was to verify that the features in the analysis components of our Mesh-Mon worked individually and collectively as intended in our original design.

4.3.1 Node failures

We observed Mesh-Mon's reaction to simulated individual node failures of two types, by rebooting a node and by powering-down a node. We repeated this test on different nodes, including nodes that Mesh-Mon had suspected as being "critical" using our DFS-based topology analysis. We tested both our localized approach as well as our globalized approach for critical node detection and found that both had 100% accuracy, i.e., no false positives or errors. This result was not surprising, since we only had 16 nodes and our testbed configurations were static and did not form large ring-like structures that may cause false positives.

We then proceeded to simulate failures in multiple nodes simultaneously. In later tests we ran the same scenarios in the presence of Mesh-Mon-Ami clients, and depending on how the client was moving during the test, we observed whether the sysadmin got additional information about the state of the network. In all these tests, we evaluated the accuracy of the alerts, the time between the event occurring and the node locally logging the hypothesis and the time delay until the sysadmin received the alert. In post-test analysis we verified that the logs were correct and consistent with the received alerts.

4.3.2 Routing failures

We simulated routing protocol crashes by turning off the routing daemons on select running nodes. We observed that Mesh-Mon was successfully able to continue delivering monitoring information both to and from such nodes through its broadcast communication mechanism. This demonstrated Mesh-Mon's unique ability to function in the situation where the routing daemon on one or more nodes had crashed or was not functioning correctly (possibly due to a poor implementation or a bad software update). Such a situation would potentially render any unicast-communication-based management system ineffective, but did not affect Mesh-Mon.

4.3.3 Hardware failures

We simulated a hardware failure by manually disabling of one or both 802.11b radios (by using `ifconfig`) to emulate a hardware failure on an MN, and checked to see whether Mesh-Mon or the client was able to detect an error. For instance, if a mesh node had its ad hoc interface radio fail and was able to detect it, it may still be able to alert an MMA since its AP interface may still work. Alternately, if the node could not detect the hardware failure, the MMA may notice that it can communicate with the mesh node it is associated with, but not with other mesh nodes. When any radio was turned off, Mesh-Mon was able to detect it as being due to possibly a network driver-related issue or a hardware failure, and would try to reset it using the `ifconfig up` command. After a real hardware failure, resetting the hardware through a software command would probably have no effect. If both radios in the mesh node actually failed, Mesh-Mon would detect the problem and note the fact it in its log, but without any working radio interfaces, it would have no means to communicate this fact to other nodes or clients. Hardware failures are the hardest type of failure to diagnose or resolve. While building our mesh testbed, we detected and resolved the true problem in one of our client laptops only when we replaced a faulty Orinoco-chipset-based wireless card with a substitute card, without changing anything else. In a mesh node with embedded components, spare parts and redundant hardware would make diagnosing hardware failures easier, but the cost of the hardware would rise significantly, potentially reducing their commercial viability.

One unusual issue that we discovered was that Atheros-chipset-based wireless cards would lock up after a random period (between 1 or 2 hours) of usage in ad hoc mode. On further study, we discovered that

this effect was due to an unresolved "ath_mgtstart: discard, no xmit buf" bug (reported by `dmesg`) in the MADWiFi device drivers. This driver issue is still unresolved in the latest driver version available at this writing [103]. What was interesting was that Mesh-Mon was able to automatically resolve and mitigate this issue, because the analysis engine would restart the wireless interface whenever a Mesh-Mon node locally reported a sudden loss of all its neighbors, or was isolated for a long period of time. This specific driver issue was not a scenario that we originally anticipated, but our diagnosis engine was still effective in a positive way.

4.3.4 Configuration

Configuration settings for our mesh nodes are pre-determined and are not supposed to change during operation, unless modified manually by a sysadmin. For our tests, we selectively modified certain configuration parameters on both clients and nodes, to simulate modification of a mesh node by a human adversary, a software entity (such as a worm or a virus), or even an inexperienced user. In our tests we measured the time delay until the problem was resolved by a node taking corrective action or the sysadmin manually resetting the value. We also studied the effect a misconfigured node had on its neighbors and the network. Such information can be useful for future versions of the diagnosis engine. Mesh-Mon was able to successfully resolve all configuration issues within a few milliseconds once they were detected; thus, the latency depends only on how frequently we ran our analysis routines. Our configuration management component uses a deterministic design and can only deal with the problems it was originally designed to handle. Mesh-Mon was not designed to deal with new and unexpected configuration problems, or to help nodes automatically determine optimal configuration settings.

4.3.5 Congestion

We forced various traffic patterns between pairs of clients and managed to saturate specific links. This condition forced the local anomaly detector (monitoring local health metrics) to trigger alerts to the sysadmin and to neighbors whenever a threshold was crossed, and helped to identify individual nodes or regions of congestion.

4.3.6 Multiple simultaneous failure events

After testing the individual cases, we tested for the detection of several of the above faults occurring simultaneously or within a short delay of other events on multiple devices. Mesh-Mon nodes report all problems encountered, ranked by severity, not just the first one. If the faults are unresolved, the alerts are re-sent in the next reporting cycle. Mesh-Mon was again able to detect and send out accurate alerts to the sysadmin, as long as both parties were in the same partition.

4.3.7 Mobility testing and partitions

In our initial setup with 16 nodes, we had a well-connected network. As we introduced several individual node failures and moved individual nodes around, the topology graph became sparse and Mesh-Mon reported the presence of critical nodes. We tested each alert by then turning off the reported critical node and checking whether the network was still connected or partitioned. We had a few false alarms with critical node detection, since a few nodes would run computations with slightly outdated topology data due to mobile nodes and transient links. We found that a simple way to filter these false positives was to wait for at least three identical alerts in a row; such post-processing could be implemented in the Mesh-Admin and its interface with the sysadmin.

After we reduced the network to two distinct partitions, we walked around with a laptop as an MMA, and visited every node in the mesh. The MMA detected the two partitions and sent alerts to all other nodes it connected to (and thus their neighbors and so forth). We repeated this test with three partitions. In both cases, the stationary sysadmin received the MMA's alerts, but the content and timing of the received alert depended on which partition the sysadmin was located in, and the path followed by the MMA. In a real deployment scenario, a robotic or intelligent MMA could be designed to remember where the sysadmin was last contacted and move in an optimized manner to ensure that its alerts reach the sysadmin with a high probability and low latency.

Table 4.3: Ranked average EVC calculations and average degrees

Rank	1	2	3	4	5	6	7	8	9	10	11	12	13
NodeID	1	3	7	2	15	5	8	11	6	13	12	16	9
EVC	0.41	0.38	0.30	0.30	0.29	0.29	0.26	0.24	0.21	0.18	0.17	0.17	0.12
NodeID	1	7	3	15	5	2	13	11	6	16	12	9	8
LQ EVC	0.43	0.40	0.40	0.38	0.37	0.25	0.17	0.15	0.10	0.082	0.081	0.065	0.061
NodeID	5	2	6	7	1	3	15	11	16	8	9	12	13
GW EVC	0.29	0.28	0.27	0.03	0.02	0.02	0.02	0.01	0.006	0.006	0.004	0.003	0.002
NodeID	1	3	2	7	11	15	5	8	6	12	16	13	9
Degree	10.6	9.6	6.6	6.5	6.2	6.01	6.0	5.55	5.4	4.3	4.03	4.0	2.8

4.4 Social Network Analysis

In this section, we present the results from the application of our centrality metrics derived from social network analysis (described in detail earlier in Section 3.11) and designed to aid the sysadmin in the task of network analysis in Mesh-Mon.

4.4.1 EVC calculation results

We present results from the calculation of Eigenvector Centrality (EVC) metrics for a static network with 13 nodes in a single connected component over two hours (the remaining nodes were turned off). Over the duration of measurements, the EVC ranks were stable, the variance in the ranking was low, and the median EVC value was close to the mean. By looking at the three EVC metric calculations (see Table 4.3) and considering the average degree of each node from our experiment, we gained some interesting insights that are detailed below.

The gateway weighted (GW) EVC gave us a ranking of the quality of Internet connectivity a client should expect from a mesh node. From the ranking, we can infer that nodes 2, 5 and 6 were gateways and that that nodes 8, 9, 12 and 13 were not physically close to gateways (or if they were close to any of the gateways, then they have poor-quality links to those gateways). In this example, node 8 was actually just one hop away from gateway node 5.

From the link quality (LQ) EVC, we can see that node 8 had the lowest rank but it had a high average degree, and was centrally ranked in the original EVC. Thus, we concluded that node 8 was surrounded by many neighbors (including a gateway node), but had weak connections with them, and had poor Internet

connectivity. Similarly, we inferred that Node 13 had few neighbors (but had high-quality links)with them, and had poor Internet connectivity. Node 1 was the most well-connected non-gateway node.

We compared our conclusions with the actual topology of the nodes during this experiment (a snapshot from the experiment is shown in Figure 4.5) and Mesh-Mon logs, and we found that they were indeed correct. Please note that Figure 4.5 is an instantaneous snapshot from a long experiment and only depicts 12 of the 13 nodes in the experiment. The three gateway nodes are clearly visible since they had links to the diamond shaped node (labeled zero), which is a virtual node representing the Internet. Also note that node 8, was directly connected to gateway node 5 and that node 9 had only one unidirectional link (to node 16) in Figure 4.5.

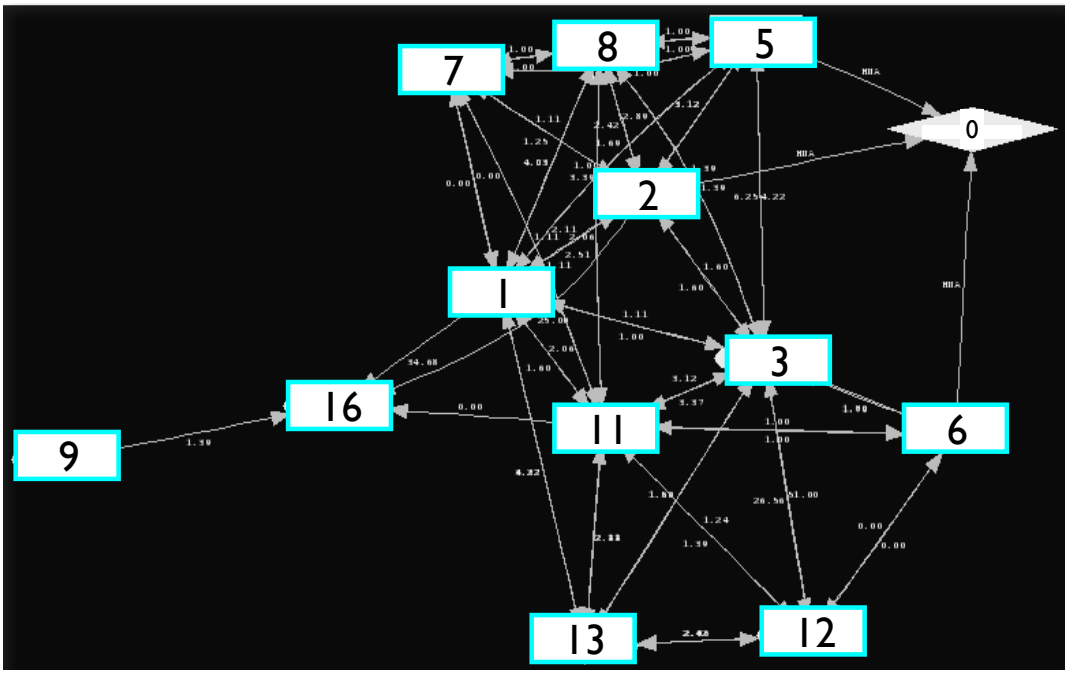

Figure 4.5: Network snapshot from GW EVC experiment

4.4.2 GW EVC changes over time

We now present results on how the EVC value changes after a significant network event. We used the same network shown in Figure 4.5 to study what happens when a gateway node is switched off, when a gateway node loses its Internet connection and when the highest ranked, moderately ranked and lowest ranked nodes are turned off. During the experiment, we powered down a gateway node (GW5) at t=40 minutes to emulate

a node failure, and brought it back online at t=62 minutes, and then we powered it down again at at t=70 minutes. We also removed the Internet connection from another gateway node (GW6) at t=48 minutes by physically removing its Ethernet wire and then reconnecting it 2 minutes later, but we did not power it down.

As seen in Figure 4.6, GW EVC values fluctuate over time in an OLSR network due to the mechanism in which the link quality value is frequently recalculated and the manner in which the topology is distributed. Although GW EVC values fluctuate over time, their moving averages (over a window of 5 samples) are relatively steady as shown for node 11 in Figure 4.7. In spite of the GW EVC fluctuations, we noticed significant changes in the GW EVC that correlated with the triggering events mentioned above. Notice the corresponding abrupt changes in the GW EVC and large variations in its moving average for node 11 in Figure 4.7 slightly after we powered down node GW5 at t=40 minutes, and the large decrease in GW EVC at around t=50 minutes just after node GW6 lost its Ethernet connection. Node 11 was thus probably located close to GW6. This fact can be confirmed from Figure 4.5.

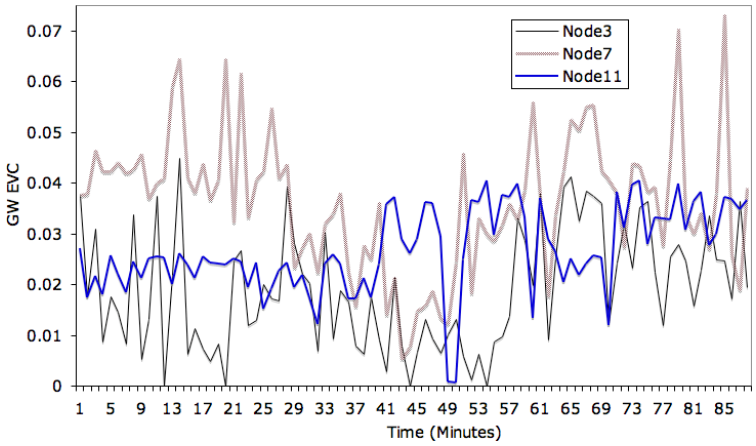

Figure 4.6: GW EVC changes over time for 3 non-gateway nodes

From an analysis perspective, given a report on GW EVC for all the gateway nodes as shown in Figure 4.8, a sysadmin could conclude that if the GW EVC of a gateway node went to zero, then the node had probably crashed (such as when we had turned off gateway node GW5 at t=40 minutes). Notice that subsequently the GW EVCs of other gateway nodes (nodes GW2 and GW6 in Figure 4.8) went up.

If a gateway node lost its Internet connection, but did not crash (GW6 at time t=49 minutes), then its EVC value reduced. When Internet connectivity was restored to node GW6 at t=52 minutes, the GW EVC of that node went up again. At t=62 minutes, GW5 was powered up again and thus its GW EVC rose from

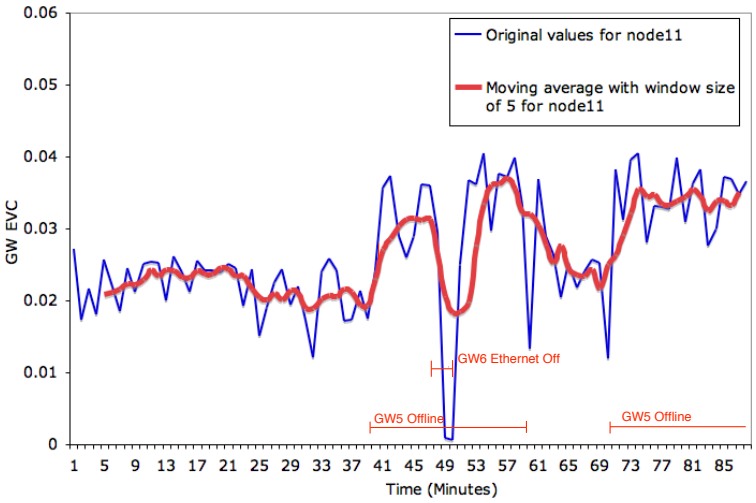

Figure 4.7: GW EVC changes over time for node 11 (not a gateway)

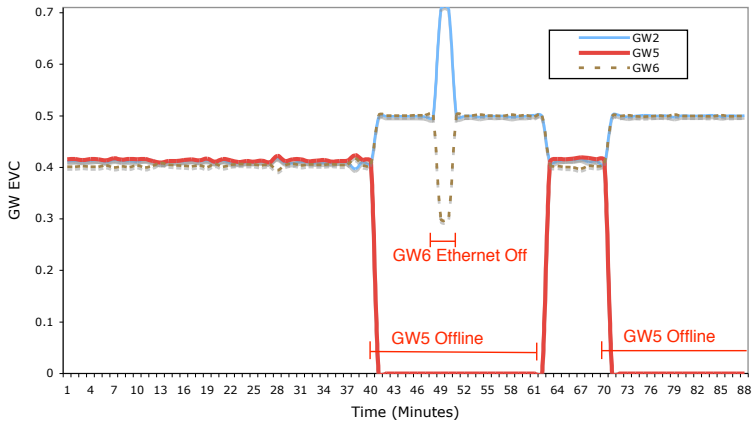

Figure 4.8: GW EVC changes over time for 3 gateway nodes

zero to about 0.4 and then at t=70 minutes it was powered off again and its GW EVC value returns to zero.

Changes in GW EVC for gateway nodes triggered minor changes in EVC values for other nodes based on their topological relationships to the gateway nodes and their link qualities. During our experiment, we also noticed that the GW EVC of node 9 (a low ranked node) was often zero (see Figure 4.9). For node GW5, the GW EVC falling to zero (Figure 4.8) was due to a crash. However, for node 9, we discovered by parsing several logs manually that it had never crashed, but was unable to create symmetric links with its neighbors (another type of anomaly). Note that node 9 has only one unidirectional link to node 16 in Figure 4.5.

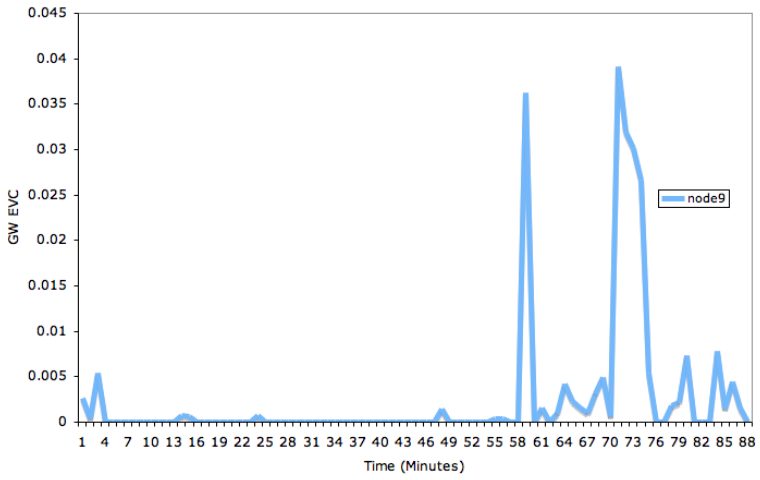

Figure 4.9: GW EVC changes over time for node 9 (not a gateway)

Thus, the EVC rankings and the changes in the moving average of the GW EVC can be used for the detection of network anomalies, with the caveat that the exact cause of the anomaly (such as for node 9) may need more information to identify. The GW EVC is a versatile metric since it combines the adjacency matrix, link-quality and status of gateway nodes into a one-dimensional value for each node. Similarly, the LQ EVC is useful for analyzing connectivity in self-contained mesh networks with no gateways.

4.4.3 LBC vs. BC Results

We now present our results from the application of the Bridging Centrality (BC) and Localized Bridging Centrality (LBC) metrics on three distinct networks. The first example is a synthetic network, the second is a social network example and the third is the topology of our wireless mesh network. All calculations were

Table 4.4: Top six centrality values for the network shown in Figure 4.10, including Sociocentric Betweenness (C_{Soc}), Egocentric Betweenness (C_{Ego}), Bridging Coefficient (β), Bridging Centrality (BC) and Localized Bridging Centrality (LBC)

Node	Degree	C_{Soc}	C_{Ego}	β	BC	LBC	Rank of BC	Rank of LBC
E	2	0.533	1	0.857	0.857	0.457	1	1
B	2	0.155	1	0.857	0.133	0.857	2	1
D	2	0.155	1	0.857	0.133	0.857	2	1
F	3	0.477	3	0.222	0.106	0.666	4	4
A	4	0.655	6	0.100	0.065	0.600	5	5
J	3	0.211	3	0.166	0.035	0.499	6	6

verified using a popular social-network-analysis tool called UCINET [22]. We assigned the same rank to two or more nodes with the same centrality value.

Synthetic network example

We first tested our LBC metric using a synthetic network, presented in Figure 4.10 (this network was also used by Hwang et al. [85]). The rankings produced by Bridging Centrality and Localized Bridging Centrality shown in Table 4.4 are nearly identical, although we note that the BC and LBC values are clearly not identical, nor are the betweenness measures used. Since both BC and LBC are used as a "relative" measure of how nodes differ from each other, the induced ranking is more important than the magnitude of the BC or LBC value and thus in this example our LBC metric is exactly equivalent to the BC metric.

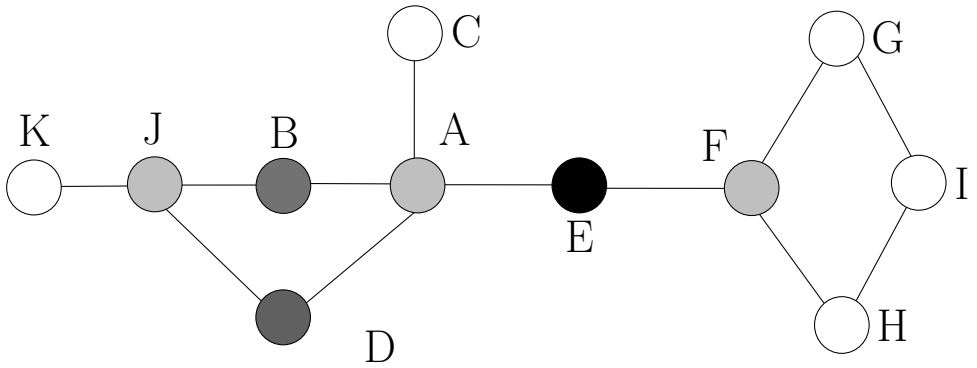

Figure 4.10: A small synthetic network example with its top 6 high bridging score (BC) nodes shaded (Adapted from [85])

Social network example

This example (presented in Figure 4.11) represents game-playing relationships in a bank wiring room and is popular in social-network studies. Marsden [111] presented this example to show how sociocentric and egocentric betweenness measures correlate. Again, the relative ranking of nodes calculated by BC and LBC as shown in Table 4.5 are identical. Visible inspection of Figure 4.11 shows that nodes W5 and W7 are bridging nodes, and the tie between them is a bridge between two connected components.

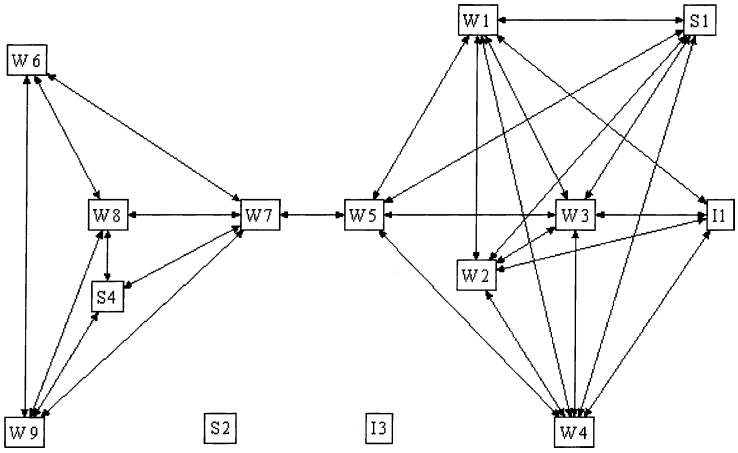

Figure 4.11: Bank wiring room games example (Adapted from [111])

Table 4.5: Centrality values for the network shown in Figure 4.11 sorted by BC values

Node	Degree	C_{Soc}	C_{Ego}	β	BC	LBC	Rank of BC	Rank of LBC
W5	5	30.00	4.00	0.222	6.667	0.889	1	1
W7	5	28.33	4.33	0.179	5.074	0.775	2	2
W1	6	3.75	0.83	0.140	0.528	0.117	3	3
W3	6	3.75	0.83	0.140	0.528	0.117	3	3
W4	6	3.75	0.83	0.140	0.528	0.117	3	3
S1	5	1.50	0.25	0.222	0.333	0.055	6	6
W8	4	0.33	0.33	0.223	0.073	0.073	7	7
W9	4	0.33	0.33	0.223	0.073	0.073	7	7
W2	5	0.25	0.25	0.210	0.052	0.052	9	9
W6	3	0	0	0.476	0	0	10	10
S4	3	0	0	0.476	0	0	10	10
I1	4	0	0	0.357	0	0	10	10
I3	0	0	0	0	0	0	10	10
S2	0	0	0	0	0	0	10	10

Real-world mesh network example

We applied our LBC metric on the topology of Dart-Mesh when it was running OLSR; the topology of the network is shown in Figure 4.12. As mentioned earlier, the rectangles represent mesh nodes identified by their individual IP addresses. The diamond box is a virtual node representing the Internet. Thus nodes 50 and 20 are Internet Gateways since they have links to the diamond shaped box. The virtual node is accounted for in the LBC and BC metric calculations, since we do care if node are acting as "bridges" between the mesh and the Internet. The labels on the links represent instantaneous link quality readings and can be ignored, since we do not consider link-quality values in our metric. The BC and LBC results are presented in Table 4.6 and the nodes are sorted in decreasing order of BC values.

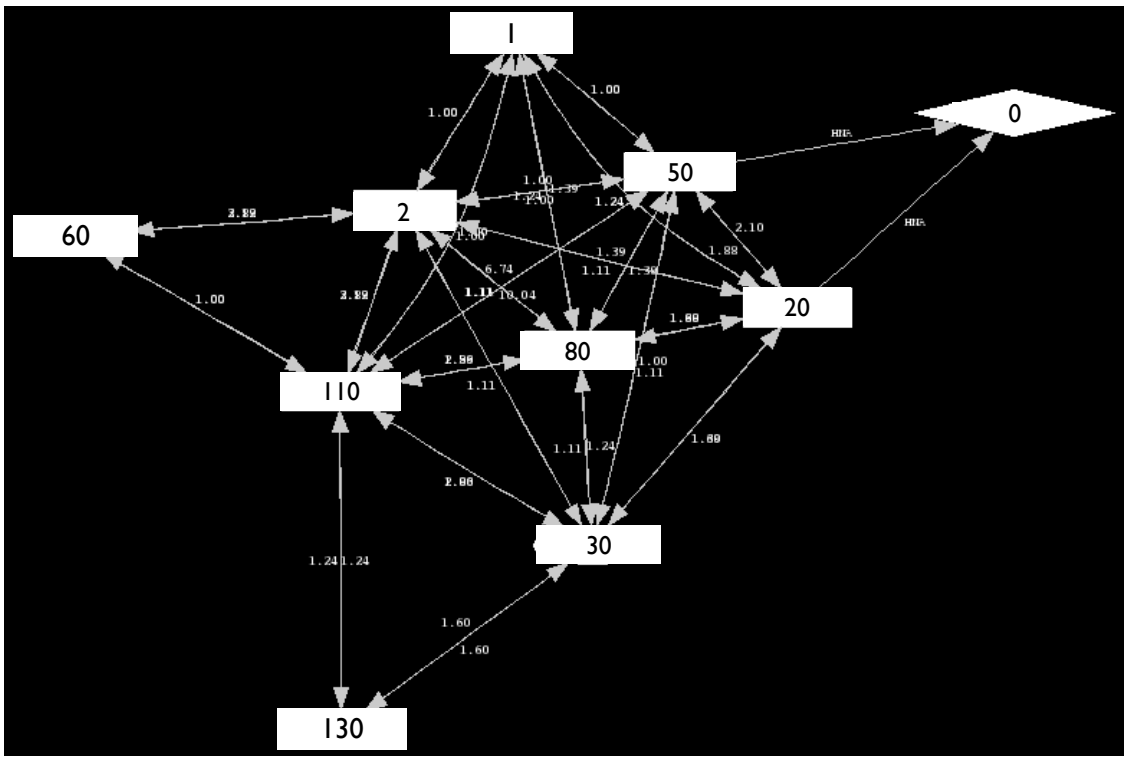

Figure 4.12: A small real-world mesh network

While, in this example, the two rankings produced by the two metrics are not identical, they are quite similar. The top-five ranked nodes are common to both metrics. If we remove any of these bridging nodes, then at least one of the other bridging nodes will become an articulation point. If that node is now removed, the network will be partitioned. For example, if node 110 (the top-ranked-node) fails, then both nodes 30

Table 4.6: Ranked centrality values for Figure 4.12 sorted by BC values

Node	Degree	C_{Soc}	C_{Ego}	β	BC	LBC	Rank of BC	Rank of LBC
110	7	6.367	5.75	0.078	0.496	0.448	1	1
50	7	4.733	3.40	0.096	0.454	0.326	2	3
30	6	3.367	2.75	0.132	0.444	0.363	3	2
2	7	4.067	3.4 0	0.096	0.391	0.326	4	3
20	6	2.867	2.25	0.126	0.361	0.283	5	5
80	6	0.400	0.40	0.173	0.069	0.069	6	6
1	5	0.200	0.25	0.262	0.052	0.065	7	7
60	2	0	0	1.615	0	0	8	8
130	2	0	0	1.650	0	0	8	8
0	2	0	0	1.615	0	0	8	8

and 2 become articulation points. However, if you examine the original network graph in Figure 4.12, you will find that it is bi-connected and has no articulation points. Thus, LBC can help detect nodes that may not be articulation points, but with certain perturbations in the network are the most likely candidates to become articulation points. Our LBC metric allows the system administrator to gather this information in a distributed manner using fewer computational resources than the BC metric.

Although LBC classifies nodes 110, 30 and 2 (co-ranked with 50) as its top three bridging nodes (using the top 25th percentile rule), and BC classifies 110, 50 and 30 as its top three bridging nodes, qualitatively there is little difference between the choices, since all of these nodes lie equally on the boundaries between connected components and removal of any of these nodes would leave some other node as an articulation point. Thus, from the sysadmin's perspective, the top-ranked nodes from BC and LBC are equivalent in functionality.

4.4.4 LLBC vs. LBC results

Real-world mesh network example with one gateway

We applied our Localized Load-aware Bridging Centrality (LLBC) metric on another topology derived from Dart-Mesh. The topology of this network is shown in Figure 4.13. Node 50 was the sole Internet Gateway, providing Internet connectivity to the whole mesh.

We used a different topology from the previous experiment, because this topology helps to highlight the

differences between the two metrics, and also provides a rare mesh example in which ego-centric betweenness and sociocentric betweenness are both zero for many nodes.

The topology of the network did not change during this experiment, which was 10 minutes long. The LLBC and LBC results are presented in Table 4.7 and the nodes are sorted in decreasing order of LLBC values. The Load metric is given here in bytes per 10 minutes.

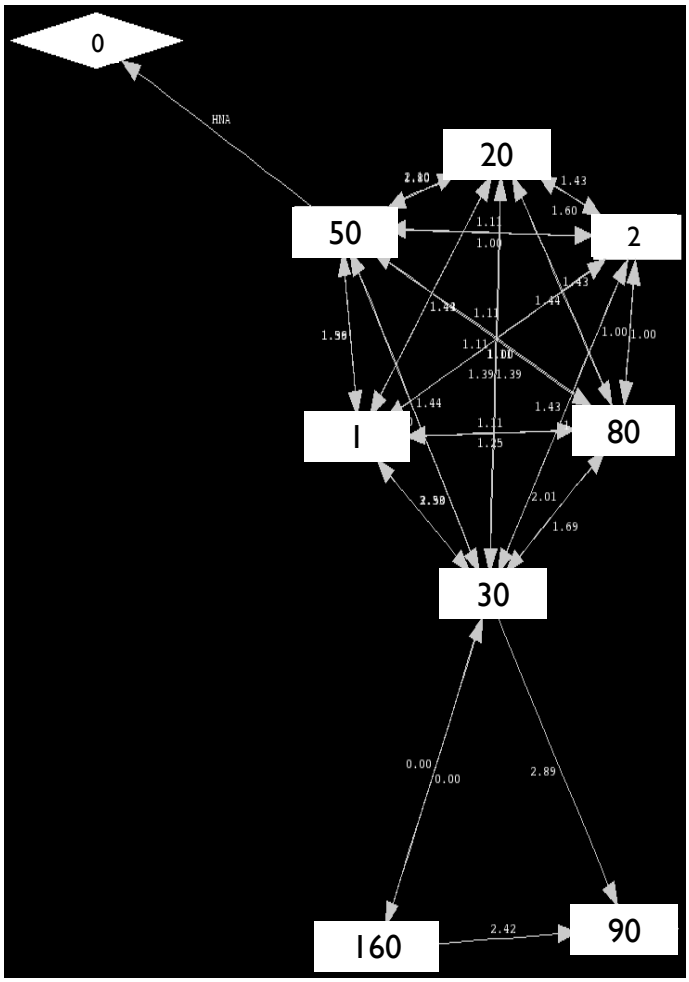

Figure 4.13: A small real-world mesh network with one gateway

During this experiment, node 80 had a high traffic load since we connected one of our clients to the mesh network through it. We then proceeded to download large video files to that client from the Internet using node 50 as our Internet gateway. According to the LBC results, which only consider the topology of the network, node 30 was a more important "bridging node" than node 50. However, our LLBC results

Table 4.7: Ranked centrality values for the network shown in Figure 4.13, sorted by $LLBC$ values

Node	Degree	Load (Bytes)	C_{Ego}	β	β_t	LBC	$LLBC$
50	6	30871080	2	0.1767	0.9747	0.3535	1.9494
30	7	274027	10	0.0726	0.0043	0.7263	0.0438
80	5	30679118	0	0.2198	0.9628	0	0
1	5	262501	0	0.2198	0.0042	0	0
2	5	238071	0	0.2198	0.0038	0	0
20	5	218143	0	0.2198	0.0035	0	0
160	2	94005	0	0.7777	0.2571	0	0
90	2	91602	0	0.7777	0.2488	0	0

Table 4.8: Ranked centrality values for the network shown in Figure 4.14, sorted by $LLBC$ values

Node	Degree	Load (Bytes)	C_{Ego}	β	β_t	LBC	$LLBC$
50	6	32989000	2	0.1182	1.1231	0.2364	2.2462
30	7	305327	10	0.0738	0.0048	0.7389	0.0489
20	6	1125000	2	0.1182	0.0183	0.2364	0.0367
80	5	16208854	0	0.2198	0.3511	0	0
1	5	11011448	0	0.2198	0.2144	0	0
2	5	722022	0	0.2282	0.0117	0	0
90	2	145226	0	0.7777	0.3358	0	0
160	2	127098	0	0.7777	0.2820	0	0

accurately show that node 50 was the most important bridging node by taking into consideration the traffic load on the network (during our monitoring period).

Real-world mesh network example with two gateways

We next applied our LLBC metric on a similar network topology similar to the one used in the last experiment by converting node 20 into an Internet gateway. The topology of this network is shown in Figure 4.14, and now nodes 50 and 20 are the two Internet gateways. The LLBC and LBC results are presented in Table 4.8 and the results are sorted in decreasing order of LLBC values.

Since there were two Internet gateways, traffic flowing to and from the Internet could go through either gateway, depending on what routes were selected by the routing protocol. LBC picked node 30 as its top bridging node. While this node was indeed a critical node, there was little traffic that was flowing through

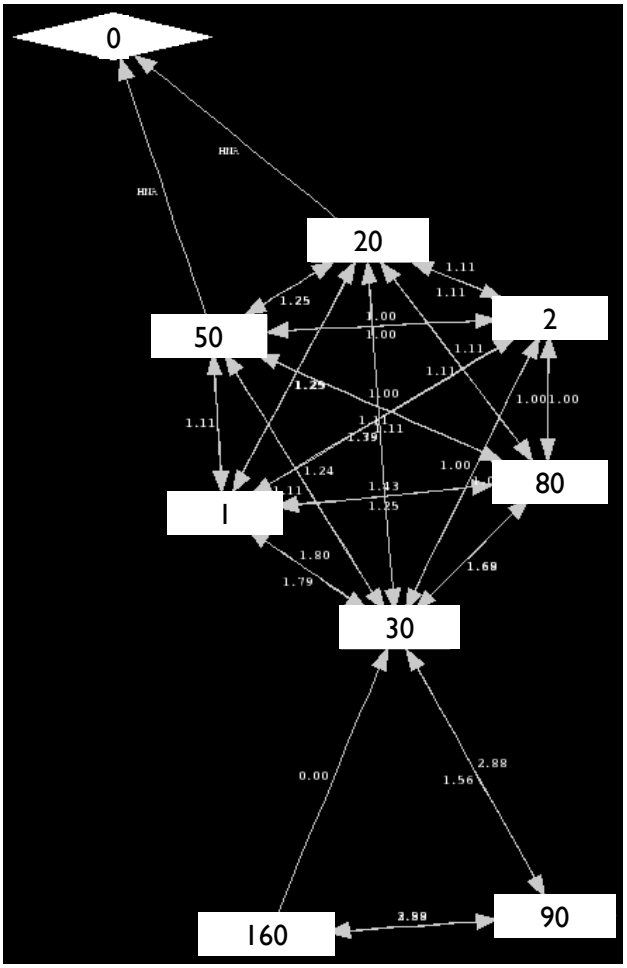

Figure 4.14: A small real-world mesh network with two gateways

this node, so it had little influence on the traffic flowing in the network or on the majority of the nodes, most of which were forwarding Internet-bound traffic through the gateways.

LLBC picked node 50, in fact the most-heavily-used gateway node, as the most important bridging node and indicated that node 30 (a non-gateway node) was a more important bridging node than the gateway node 20, even though node 30 had only one fourth the traffic load of node 20 in absolute terms. The importance ranking generated by LLBC is accurate and insightful; in this scenario, if node 30 failed, then nodes 90 and 160 would be partitioned from the rest of the network. Whereas, if node 20 failed, there was still a potential backup path to the Internet through 50; the LBC rankings were unable to capture this subtle complexity present in this network. The BC ranking was identical to the LBC ranking, and thus not as helpful as the LLBC metric in this scenario (and it required centralized calculations). The distributed manner in which LLBC is calculated complements the distributed analysis engine used in Mesh-Mon.

Our evaluation of the LLBC metric is more qualitative than our evaluation of the LBC metric. The difference between LBC and BC is easier to quantify since both metrics provide identical functionality. We are not aware of any other metric that can be calculated in a distributed manner and can provide similar functionality as the LLBC metric. We also tested LBC on large synthetic mesh topologies with synthetic data traffic patterns and saw similar results. Since these results were based on synthetic models, we do not include them here.

4.4.5 Summary of social-network analysis metrics

We learn several things from our experimental results for the different EVC metrics. We found that the changes in EVC can be potentially used for anomaly detection in mesh networks. We compared the LBC metric to the BC metric in three examples. We presented two scenarios where the LLBC metric was more useful from a management perspective than the LBC metric. Both LBC and LLBC metrics can be calculated efficiently in a distributed manner.

In addition to our new centrality metrics, the sysadmin should also pay attention to existing simple metrics, such as the traffic load at each node, the nodes identified as “heavy-hitters” (source or sink of most traffic), and how much traffic is forwarded by different hosts on behalf of other nodes. We advocate using the right metric for the situation, as dictated by the distribution of flows in the network which may vary from

network to network or by time of day.

4.5 Summary

In this chapter, we present our quantitative results and validation of designed features in Mesh-Mon from experimental evaluations on Dart-Mesh. We also present extensive results from the use of our novel social-network-analysis metrics developed specifically to aid in the task of distributed management of mesh networks.

Chapter 5

Related Work

Several cities (such as Las Vegas, Minneapolis and Taipei) are in the process of deploying mesh networks that will provide 802.11 coverage to the entire city. For example, Taipei is planning a network with 20,000 access points. It will be interesting to observe the management techniques that are deployed to maintain these networks once they are deployed. In the emergency response domain, New York City announced plans in 2006 to build a public safety wireless network of unprecedented scale and scope, including the capacity to provide tens of thousands of mobile users with the ability to send and receive data while traveling at speeds of up to 70 mph citywide. While the future of this ambitious project is uncertain, several smaller mesh networks have been deployed in cities such as Providence and Dallas, for public-safety uses such as video surveillance and connecting first responders [63, 80]. For economic reasons, the Philadelphia mesh network was shut down in May 2008 and the deployment of a city-sponsored mesh network planned for San Francisco is currently on hold.

Sichitiu provides a brief introduction to the primary research challenges for mesh networks [162]. A recent and comprehensive survey on mesh networks by Akyildiz et al. covers most of the state-of-the-art in wireless mesh networking technology, but does not provide any details on mesh network management [4].

Our mesh architecture is similar in structure to mesh networks available from several commercial vendors. From the limited information publicly available, it appears that most vendors have their own proprietary management platforms (some of which are compatible with SNMP). We do not have sufficient information to compare Mesh-Mon directly with their solutions.

5.1 SNMP

Simple Network Management Protocol (SNMP) is the de-facto protocol for management of most networks due to its widespread acceptance [31]. SNMP was originally designed for static wired networks and uses a centralized design. The philosophy of SNMP is simple. Each network device must maintain minimal state on each node in the form of counters and variables. A generic network monitoring solution using SNMP would require an SNMP agent on each mesh node collecting local information as defined in the SNMP Management Information Base (MIB). The SNMP protocol allows for periodic polling of variables as well as "traps," directed events that are triggered by events in the network. A Network management System (NMS) would periodically poll each device from a master node, presenting the situation to an administrator. The administrator could also configure the agents to respond to local conditions through SNMP traps. SNMP serves as a monitoring tool and leaves the task of analysis and management to humans or other software tools.

Many vendors of network devices support SNMP, as do several third-party-software Network Management Systems. SNMP-based systems can have high overheads due to periodic polling of MIBs. The actual overhead of SNMP in a mobile mesh network requires further study, but a far bigger concern is the single point of failure introduced by a centralized management system. To overcome this shortcoming, the idea of a decentralized system for network management, such as that proposed by Mesh-Mon, is not new. Remote Network Monitoring Information Base (RMON) and Script MIB were proposed to decentralize management tasks in an SNMP framework [166]. These tools still depend on the routing protocol, however, while Mesh-Mon does not.

While SNMP is based on a reactive and pull-based philosophy since the NMS polls managed devices for information, in Mesh-Mon we use a more proactive and push-based approach, where Mesh-Mon nodes push out information periodically to each other and to the sysadmins.

We could have built upon RMON or SNMP-based primitives for some of the information-gathering components of Mesh-Mon. However, SNMP support for wireless devices on Linux and device drivers is limited and we would have had to invest a considerable amount of time developing customized MIBs that conform to the SNMP standard and met our specific design needs (some of which are incompatible). By not relying on SNMP, we gained the flexibility to to develop our own framework using a clean-slate design.

5.2 Mesh network management

Commercial mesh systems are bundled with SNMP-based or proprietary centralized management solutions [39, 131, 141, 179]. We are not aware of any commercial decentralized network management systems designed for mobile-multi-radio mesh networks.

The Ad hoc Network Management Protocol (ANMP) is a management solution for ad hoc networks designed to be an extension of SNMPv3 [36]. ANMP is designed to be compatible with SNMP and uses the same structure and protocol data units (PDUs) for data collection through MIBs. Management in ANMP is based on a hierarchical approach where cluster heads poll management information from their cluster members (agents running on each node) in a centralized manner. ANMP supports a three-level hierarchy: the manager, the cluster heads, and the agents in individual nodes. The authors provide details of the proposed ANMP MIB and a simple GUI front-end to allow the manager to browse the MIB, and to "Get" and "Set" MIB PDUs. A complete management solution does not appear to have been implemented, tested or evaluated, since the authors do not provide any quantitative or qualitative results.

The Guerilla Management Architecture (GMA) proposes that nodes in an ad hoc network participate in management functions based on their individual capabilities to perform certain tasks and the current state of the network [158]. The GMA suggests the use of techniques from the mobile-agent community, such as mobile code to implement autonomous management. They propose that nomadic managers (mobile agents) can adapt to changing network conditions autonomously. The nomadic managers can help disconnected partitions manage themselves. The authors present simulation results that show potential patterns in which network hosts may get partitioned due to mobility and how the managers could potentially adapt to the new topology by migrating. Managers decide when and how to migrate based on computation of a utility function [159]. The authors are vague about how management tasks are to be conducted, focusing instead on the potential of a mobile-agent-based approach. The authors assume that nomadic managers are "intelligent" and cooperate to perform management tasks. The GMA solution does not appear to have been implemented or tested in a real network. Neither the ANMP nor the GMA studies provide any experimental results or provide insight into the impact of the management system.

Ramachandran et al. at the University of California at Santa Barbara (UCSB) designed the Distributed Ad hoc Monitoring Network (DAMON), a distributed system for monitoring ad hoc and sensor networks [143,

144]. DAMON uses agents within the network to monitor network behavior and send collected measurements to data repositories or sinks. DAMON is written in Perl, and runs on both Windows and Linux, but is designed specifically for a specific version of AODV. DAMON's architecture supports the monitoring of a range of protocol, device, or network parameters. Other features of DAMON's design include support for multiple repositories, auto-discovery of sinks by the agents, and resiliency of agents to repository failures, but most of these features are not implemented in the publicly-available prototype. Mesh-Mon is designed for scalability and can support any routing protocol (not just AODV) even if it fails or misbehaves.

Lundgren et al. developed another stationary mesh testbed at UCSB that is managed by a set of interconnected components called MeshMan (for management and configuration), MeshMon (for parameter monitoring), and MeshViz (for topology visualization) [108]. Most of the mesh nodes use a wired back-haul for management information and easier control of experiments in the testbed. Our approaches to monitoring share some similarities, but we use in-band communication of all monitored information and have more emphasis on mobility and fault diagnosis.

Jardosh et al. developed SCUBA, an interactive visualization framework designed to assist in the diagnosis of large-scale wireless mesh networks [7]. Various metrics from the network are collected in a central database through a gateway node. SCUBA queries the database to drive its interactive visualization tools. The authors implemented an initial version of SCUBA on a 15-node mesh testbed at UCSB. SCUBA in its present form has limited diagnostic capabilities, but could potentially be used as a complementary tool for Mesh-Mon to help the system administrator visualize the network.

Tuduce and Gross present a prototype of a resource monitoring architecture for mobile ad hoc networks [181]. The goal of their system is to assist individual applications in gaining information about the status of network resources such as the status of nodes in the path from a source to a destination. The clients, in this case the applications, can make queries to the monitoring system and the monitoring system will reply with the requested view of the environment. The prototype uses a clustered hierarchical organization. Only cluster heads are allowed to answer queries. The two types of queries provided are the "Neighborhood Query" and the "Swath Query". The Neighborhood Query provides a graph of the requested diameter describing the network topology surrounding the requesting node. The Swath Query provides a path from a source to a destination and if requested presents some of the neighbors of the nodes along that path. The au-

thors built a working prototype on Linux laptops with 802.11b cards and used AODV (kernel-aodv version 1.2) [96]. The authors measured the query-response delay in a static network. In Mesh-Mon we periodically compute the k-hop neighborhood graph, which is similar to the Neighborhood Query. While we did not implement a Swath Query in Mesh-Mon, we periodically monitor the path between a source and a destination, which varies over time even in a static network (e.g., when using OLSR with link-quality metrics). Alternate routes traversed by packets in OLSR networks thus often use nodes that would not appear in the original results of a Swath Query. Although a Swath Query may be useful in AODV, it would be less useful in our OLSR network, but it may still be a useful mechanism to study interference between nodes in a path and their neighbors.

Badonnel et al. identify basic requirements for a management framework for ad hoc networks [9]. The authors propose (but do not implement) an object-oriented information model for management of ad hoc networks. The authors present a hierarchical management framework similar to ANMP through an extension to the Common Information Model (CIM) standard. The CIM is an object oriented standard developed by the Desktop Management Task Force (DMTF) to define management information primarily for networks of end-host devices [64]. The authors also propose a partial view of an SNMP MIB for the OLSR routing protocol. Nodes distribute management information through SNMP, so the proposed system is similar to ANMP and suffers from the same weaknesses. The authors do not present any evaluation of their management system or provide details of the management capabilities of their design for any purpose beyond basic monitoring.

Riggio et al. propose JANUS, a framework for distributed monitoring of wireless mesh networks [157]. Their proposed approach uses Pastry to make network information that is collected at different layers of the stack, available at all nodes in the system. Pastry is a Distributed Hash Table (DHT) peer-to-peer overlay network [153]. They tested an initial prototype of the JANUS system on 6 Windows nodes. The testbed nodes used the Link Quality Source Routing (LQSR) protocol (based on DSR [92]) on the Microsoft Mesh Connectivity Layer (MCL) [149]. The code for JANUS is implemented in Java running on Windows and is available under the BSD license. Their initial experiments indicate that JANUS is feasible for small networks, but the authors note that the current system design needs significant tuning to reduce the impact of JANUS on user traffic. While DHT-based peer-to-peer file sharing systems can scale to millions of nodes on the Internet, the connectivity properties of wireless mesh networks and the wired Internet have

fundamental differences. It will be interesting to see how their proposed DHT-based architecture scales for larger networks of mobile nodes (with high churn-rates), and how higher-level management functionality can be implemented in such a design (if at all possible).

Multi-radio mesh networks, in which each node has two or more radio interfaces to enable simultaneous transmission and reception, have been developed by researchers at Intel and Intel Research. Papagiannaki et al. deployed such devices in a multi-radio mesh network designed for home use and studied its characteristics [132, 151]. One of their interesting observations was that the slightest adjustment in the position of the nodes or the alignment of the antennas led to significant changes in their performance results. Thus real-time monitoring could give users feedback on how to best configure and deploy a wireless mesh network for better performance.

Qiu et al. from Microsoft Research propose a simulation-based technique to troubleshoot a wireless mesh network [142]. The authors model and simulate a wireless mesh to determine what actions cause what types of behavior and aim to diagnose problems. The input from the monitoring system feeds a simulator that is used to identify the root cause of the observed network behavior. Their approach is interesting, but its feasibility hinges on the quality and accuracy of the simulated model and the simulation tools. The time taken to simulate the system (even if it was perfectly accurate) could be a serious detriment to its implementation for our goal of real-time monitoring. The authors implement several components to demonstrate some of their key concepts but did not build a complete management solution. Recent work by Gray et al. and Liu et al. indicates that simulations and real world results differ greatly and that improving the accuracy of a simulation is a challenging problem [77, 106].

The U.S. Army is considering the use of MANETs as part of the US Army Future Combat Systems (FCS) initiative [182]. Kant et al. propose DRAMA; an adaptive policy-based management architecture for future military networks developed under the CERDEC DRAMA (Dynamic Re-Addressing and Management for the Army) program [94]. The authors evaluate the design of DRAMA v1.0 through simulations in OPNET for scenarios ranging from 12 to 20 nodes and observe that DRAMA incurred low overhead on the simulated OLSR network. Chadha et al. suggest enhancements to DRAMA, such as an adaptive self-forming hierarchy of policy agents that perform role-based management [33]. Chiang et al. study the scalability of DRAMA v2.0 in simulations with scenarios of 120, 240 and 504 nodes [37]. Their simulations show that

Table 5.1: Summary comparison of Mesh-Mon vs. other mesh management systems

System	Structure	Scalable	Partition Aware	Routing Protocol Independent	Implemented
Mesh-Mon	*Hierarchical*	*Yes*	*Yes*	*Yes (Tested with AODV and OLSR)*	*Yes*
ANMP	SNMP-based	Yes	No	No	Simulated
GMA	Mobile agent based	No	Yes	No	No
DRAMA	Hierarchical	Yes	No	No (OLSR)	Simulated
DAMON	Hierarchical	Yes	No	No (AODV)	Yes
MeshMan	Wired control plane	No	No	No (AODV)	Yes
JANUS	DHT-based	No	No	No (LQSR)	Yes

DRAMA generated less network traffic than using SNMP (with a single central collector) since each level of the hierarchy filters information before propagating it. The authors do not provide details on how to specify network policies. DRAMA is a proprietary system and all the published results are simulation-based. The authors note that there are several unresolved challenges, such as the automation of the generation of management policies for dynamic scenarios [37]. Our decentralized management approach shares some similarities with DRAMA, such as using a hierarchical organization to provide scalability. Our management policies are implicit in Mesh-Mon, and at present we do not support more than two levels of management hierarchy. Again our approach is routing-protocol independent and partition aware.

5.2.1 Summary

We summarize the advantages of Mesh-Mon over all comparable systems in Table 5.1. The only drawback of our approach is the relatively high communication overhead due to flooding, but its impact on user traffic was negligible in our evaluations and it enabled Mesh-Mon to be routing-protocol independent and to be capable of monitoring a mesh network even with routing failures. We are not aware of any other mesh-management systems that use social-network-analysis techniques to identify relative management "importance," or to identify critical nodes in the topology to predict future partitions (without using location information).

5.3 Research mesh networks

We present examples of other experimental mesh networks that have been deployed. Many of these networks are managed manually or in an ad hoc manner, since management was not their main focus.

CalMesh is a rapidly-deployable mesh (similar to Dart-Mesh) designed specifically for use in emergency response scenarios [54]. The authors develop and implement a portable mesh solution, but do not provide details on how it is managed. During a short deployment exercise, all nodes relayed information to a single node located at the command center. They later used this data to analyze the characteristics of user traffic.

Bicket et al. developed and deployed the MIT Roofnet, an experimental single-radio 802.11b/g mesh network to provide broadband Internet access to residential users in Cambridge [16]. There were around 40 active nodes on the network in 2007. Roofnet was used to conduct 802.11 measurement experiments to understand the nature of large-scale wireless networks. Their earlier research focuses on studying link-level characteristics of 802.11, such as packet loss, understanding how to find high-throughput routes in the face of lossy links at various bit-rates, and developing new MAC and routing protocols that take advantage of the measured radio properties [3].

The ORBIT (Open Access Research Testbed for Next-Generation Wireless Networks) system is a 400 node two-tier wireless network testbed designed at Rutgers University as part of a collaborative NSF project [148]. The ORBIT radio grid has been available to research users for remotely controlled experiments since October 2005. Each node is always connected through a wired backbone to make experiments easier to monitor and manage, so their management is entirely out-of-band through the wired infrastructure.

Rice University partnered with local non-profit organizations to deploy a mesh network in one of Houstons most economically disadvantaged areas [171]. In January 2007, the network had 18 nodes spanning approximately 3 square kilometers serving over 2,000 users. Camp et al. perform extensive measurements on this network to characterize the propagation environment and correlate received signal strength with application layer throughput [28]. They demonstrate that network performance improves by up to 50 percent with respect to both throughput and reliability, by careful placement of wired and stationary wireless nodes. By contrast, our Dart-Mesh network is unplanned, since our goal was to manage an unplanned and dynamic mesh network. Robinson and Knightly present a measurement-based performance study of deployment factors in general wireless mesh networks using various performance metrics [150]. They found that mul-

tiple back-haul radios per mesh is a cost-effective deployment strategy, but argue that using one radio for client access and one radio for mesh access (the most commonly deployed design) is not the most optimal configuration given a per-user fairness constraint.

The Berlin RoofNet (BRN) is an experimental two-tier mesh network in Berlin, Germany [164]. The BRN uses a DSR-based routing protocol between mesh nodes. To minimize broadcasts used for DHCP, ARP and Internet Gateway selection, the BRN implements distributed services using a DHT mechanism called Falcon (which is similar to Chord [169] but adapted for use in wireless networks). The key benefit is that broadcasts are reduced to unicast packet exchanges. DHCP and other services can be implemented in a decentralized manner. This DHT system can also hold information useful in assisting roaming client nodes during the handoff process. Their simulation results demonstrate the scalability of this approach and the initial implementation results are promising. Since the system is unicast-based, the management system is coupled to the stability of the routing protocol employed.

Several other mesh research testbeds have been implemented, such as the ones at John Hopkins University [91, 6], UCSB [183] and Microsoft Research [59] to name just a few, but we have no additional details on the management solutions that they employ (if any). We were unable to find additional details on how these networks were managed.

5.4 IEEE 802.11s

The IEEE 802.11s standard for inter-operable mesh networks expected to be ratified in early 2008. The 802.11s working group is trying to unify the demands of two distinct proposals called "SeeMesh" and "Wi-Mesh" by competing industrial consortiums. In addition to these two proposals, the group is considering 10 others.

Even without official standards, several mesh networks have been designed and deployed as described above. Goth describes some of the challenges grassroots-community-mesh efforts may face once mesh equipment standards are finally announced [74].

Unfortunately, we do not have access to the draft of the 802.11s standard (the documents are not publicly available) and are not aware of what mechanisms the IEEE will propose for management of 802.11s mesh networks or what usage scenarios are being considered.

5.5 OLSR, AODV and other mesh routing protocols

Since routing protocols play such a key role in defining the performance of a mesh network, we discuss some studies that are related to the routing protocols we discuss or use in this thesis. Most research on mesh networks has traditionally focused on designing a "better" routing protocol or comparing routing protocols through simulations or experiments, while our focus is on building a scalable mesh-monitoring system that can help manage a mesh network running "any" routing protocol.

Clausen et al. simulate OLSR in the ns-2 simulator and studied the impact of selecting a minimal MPR set versus a redundant MPR set, as well as the impact of advertising partial versus full-link state information [41]. They found that the full link-state option yields a 20% higher delivery ratio than regular OLSR, at the expense of a 30% increase in control-traffic overhead.

Hsu et al. simulate OLSR, AODV, DSR, ZRP and OSPF in a realistic battlefield mobility scenario using the QualNet simulator [83]. They use GPS logs from an FCS field exercise and the corresponding traffic patterns for each node in their simulation. For the specific scenario they tested, they observed that AODV and DSR had the best packet delivery ratio and OLSR was not far behind, though DSR had the higher delays in route setup.

Plesse et al. study the performance of a draft version of OLSR specification in a testbed with static and mobile nodes [139]. They observe that OLSR provides good performance overall. They did observe high variability in the performance, however, depending on the nature of the traffic, hop-count, topology and configuration parameters.

Gomez et al. study OLSR performance through both experiments (with 4 laptops) and simulations (on 20 nodes) [73]. They observe that increasing the rate at which OLSR control messages are sent improves the end-to-end connectivity of the network.

Huang et al. study OLSR performance with ns-2 simulations [84]. They conclude that increasing the frequency of Hello messages have a bigger impact on OLSR performance than tuning other control-message types.

Demers and Kant study OLSR performance through simulations in OPNET [51]. They state that the TC and Hello interval timers can be adjusted to tradeoff routing overhead versus route convergence times. The authors propose that a policy-based adaptive network management system may be designed that benefits

from analysis of their simulation results. The authors do not detail how to practically implement such a management system.

Naudts et al. present a packet-pair probing technique for lightweight estimation of link capacity [125]. They test their technique on an OLSR network with only three nodes with no background traffic. They found that their technique provides results comparable to those of `Iperf`, and has lower overheads, but, they observe high variance in the accuracy of their results. They measure channel capacity for one hop links and disseminate this information to other nodes by piggybacking information on OLSR packets.

Das et al. evaluate the performance of AODV in ns-2 simulations [47]. AODV has subsequently been evaluated in simulations by several authors and has been implemented as well [77, 106, 120, 121]. Kernel AODV is a loadable kernel module for Linux developed at NIST by Luke Klein-Berndt [96]. AODV-uu is a user-space implementation for Linux developed at Upsalla University [128].

Mesh-Mon is capable of functioning under any of these routing protocols and numerous others, even if they fail. Detecting when a routing protocol fails, however, is beyond the present abilities of Mesh-Mon; if the behavior of a routing protocol could be modeled as a finite-state-machine, Mesh-Mon could be used to monitor the current state of the routing protocol. We would prefer to allow some other tool manage this task since we believe that a network management system should be independent of the routing system. The output of such a tool could be used as a source of information used by Mesh-Mon.

5.6 WLAN management

We now cover research work in topics that are not directly related to the design and features of Mesh-Mon or wireless mesh networks, but solve similar problems or subproblems in related domains such as infrastructure-based wireless local area networks (WLANs) and wireless sensor networks (WSNs).

Adya et al. present a client-oriented architecture for detecting and diagnosing faults in an IEEE 802.11 infrastructure-based WLAN [2]. Clients that are disconnected from the network or are experiencing problems can send out special beacons and pretend to be access points to gain the attention of special clients that are already connected with the network. The special clients can act as conduits to help connect the disconnected clients with the network. Their system resolves problems in a centralized manner at the access point. Their system also detects rogue access points. Mesh-Mon manages both mesh nodes and mesh clients, but

does not attempt to solve security problems such as rogue access points.

Chandra et al. developed WiFiProfiler, another fault-diagnosis system designed for an infrastructure-based WLAN [34]. A client unable to connect to an access point may still be able to connect to a neighboring client and exchange configuration information using a variant of the communication mechanism described above [2]. Based on the received information, the client can determine if its neighbor faces similar problems or can change its own configuration settings in an automated manner. Their work focuses on clients assisting other clients in a cooperative manner while connected to a stationary access point in a Wi-Fi network; on the other hand, we aim to monitor and help mobile clients and mobile mesh nodes and allow a sysadmin to diagnose a diverse set of problems (a subset of which overlap with theirs).

Mojo is a distributed system for diagnosing physical-layer problems in an 802.11 WLAN [161]. The authors use wireless sniffers collocated with access points, that transmit monitored data by wired Ethernet to a central data repository for analysis. Their work shows an interesting correlation between events detected in the physical layer affecting performance and failures in higher layers. At present, Mesh-Mon has no support for diagnosing physical-layer problems and can potentially benefit from a distributed adaptation of their techniques.

5.7 Sensor network management

Managing a sensor network shares some similarities with managing a mesh network although there are two main differences. Sensor nodes typically route packets in a fixed tree-like pattern from multiple sources to a central sink. Compared to mesh nodes, wireless sensor nodes are severely limited in terms of processing power and memory resources, and are typically battery-powered.

Deb et al. state the Uncertainty Principle for Sensor Networks [49] as follows:

> "In order to measure the state of a network accurately we need to do our measurements at a very fine granularity. Since any probe or measurement will use a finite amount of energy from the system this can change the very state of the system/network. The finer the granularity we measure with or precision we desire, then the more our process will change the very state we are measuring."

This principle is similar in spirit to our feedback argument that in-band measurements used for monitoring the mesh network can affect the very network that we wish to monitor. This issue is unavoidable if we wish to keep our hardware costs low. The same authors describe a distributed parametrized algorithm titled Sensor Topology Retrieval at Multiple Resolutions (STREAM), which lets the user choose a tradeoff between topology detail-level and resources expended to gather the topology [48]. In Mesh-Mon running on an OLSR network, we make a similar tradeoff by adjusting the TC_Redundancy variable [40].

Zhao et al. describe in-network aggregation techniques to help manage the volume of data gathered in a sensor network and to help efficiently inform all nodes about the state of the network [195]. MeshLeaders in Mesh-Mon are well suited to perform in-network aggregation and to share information efficiently with other Mesh-Leaders, but we have not implemented this feature in our current implementation.

Sympathy is a tool for detecting and debugging failures in sensor networks [147]. Once Sympathy detects a failure, it uses an empirically-developed decision tree to determine the most likely cause of packet loss such as a node crash, reboot, or no neighbors present. We use our rule-based diagnosis engine in Mesh-Mon in a similar manner.

Memento is a health monitoring system for sensor networks [152]. Memento has two parts: an energy-efficient bitmap-based protocol to deliver state summaries, and a simple distributed failure detector module. Each node listens to heartbeats from a subset of its neighbors. The failure detector relies on non-receipt of heartbeat messages to declare a node has failed and is adaptive to the rate of incoming packet loss. We monitor changes in the reachable partition in Mesh-Mon, but do not rely on heartbeats since our network is more dynamic than a sensor network. Our communication protocol can be optimized to be more energy-efficient and improve network lifetimes for battery-operated mesh nodes.

NodeMD is a deployment management system that successfully implements lightweight run-time detection, logging, and notification of software faults on wireless mote-class sensor devices [99]. NodeMD introduces a debug mode (integrated with the sensor operating system) that catches a failure before it completely disables a node, and drops the node into a stable state that enables further diagnosis and correction. Mesh-Mon runs on top of the operating system whereas NodeMD is practically an operating system with support for management. Operating systems for mesh nodes are fairly complex compared to those running on sensor nodes, so implementing NodeMD like debugging features for Mesh-Mon would be difficult to

implement and could potentially limit the performance of the mesh node.

In the ATMA framework at UCSB, Ramachandran et al. use a mesh network to provide an out-of-band solution for monitoring a wireless sensor network [145]. Using an out-of-band solution is an expensive proposition, but ensures that the monitoring traffic is isolated and thus cannot disrupt or influence the operation of the sensor network being monitored.

5.8 Distributed network management

Mesh-Mon is a distributed network management system. As mentioned earlier, the idea of distributed management systems is not new. We thus present related work in this area to provide the reader with additional background information and acknowledge key contributions.

Initially all network management systems were centralized or hierarchical. As new technologies evolved and networks became larger and more complex, distributed network management (DNM) systems were introduced. DNM systems overcome the inherent drawbacks of centralized management systems, namely a lack of scalability, flexibility and robustness. Goldszmidt and Yemini are credited with introducing the concept of distributed network management by delegation (MbD) [190, 70, 72]. Their work is often cited as the precursor for the design of several proposed agent-based management systems that subsequently followed [71]. Delegation in the management context refers to the idea of some management entity transferring the responsibility, power and authority for a specific task to another entity [112]. Delegation is normally a one-to-one or one-to-many relationship between a manager and one or more subordinate agents. Mesh-Mon is not a MbD system but designed to be cooperative and peer-based system.

Martin-Flatin and Znaty classify the different basic models of network and system management [112]. Martin-Flatin et al. present a more detailed classification model [113] for distributed enterprise and systems management paradigms based on four criteria: the delegation granularity, the semantic richness of the information model, the degree of specification of a task, and the degree of automation of management.

Goldszmidt and Yemini define a health function as a linear weighted function of managed variables [69]. Health functions can help characterize large heterogeneous distributed systems, but are not always easy to formulate. The definition of a healthy network is configuration and time-dependent and varies from network to network. We formulate our own appropriate health functions for Mesh-Mon.

5.8.1 Mobile agent-based network management

We discuss several proposed mobile-agent based systems for distributed network management. To the best of our knowledge none of these were fully implemented, with the exception of a few prototypes. Interest in mobile-agent research has waned in recent years.

Mobile agents are theoretically more powerful than MbD systems, since their functionality can include all the characteristics of an MbD system but not vice-versa. Baldi et al. and Bieszczad et al. describe how mobile agents have the potential for helping in network management tasks due to their autonomy, flexibility and distributed characteristics [11, 17]. Some of the benefits of mobile agents are the potential for reducing traffic around the management station, independent management capability of autonomous agents and distributed processing load, since operations that would be performed at a central point may be executed on the devices by the mobile agent [11].

Baldi and Picco evaluate the benefits of using mobile agents over a traditional client/server approach using SNMP for a similar management scenario [12]. The performance of a mobile-agent system is better than that of a client-server approach when an agent is capable of semantically compressing data or preprocessing information on a node and transmitting only the results, thus saving bandwidth.

Fugetta et al. present a classification of several mobile code paradigms and focus on the design and performance tradeoffs of using mobile agents for an SNMP data retrieval task [66].

Caripe et al. (at Dartmouth) also consider the idea of mobile agents for management in wireless scenarios [29]. The authors describe an early prototype for their management system using D'Agents [76] as well suited for use in bandwidth-constrained wireless networks.

Zapf et al. describe the design of a mobile agent system, called NetDoctor, that can query local or remote SNMP agents, process the status data, and perform certain corrective actions [193].

Nanda et al. implemented a mobile-agent-based management system called "Administrative-Aglets (AdLets)" using the Java Aglets platform [102] to monitor, configure and manage a heterogeneous wired network (with 30 Linux and Windows PCs) in a unified manner [124].

Bohoris et al. present the architecture and implementation results of a mobile agent system for the network performance management of a static network [18]. The authors compare the mobile agent system with Java Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA).

White et al. suggest that multiple swarms of adaptive mobile agents can be used to locate faults in networks [187]. The authors suggest that agents that visit nodes should act upon or modify the digital equivalent of a chemical trail left behind by ants.

Gasch describes the features of the Multi-Agent System for network Resource Reliability (MASRR), a decentralized architecture of autonomous, collaborative agents that can model normal network operations, detect departures from expected behaviors, and automatically take remedial actions [67]. Their proposed system uses a proprietary machine-learning-based anomaly-detection engine for both intrusion detection and network management.

The GMA proposed the use of mobile-agent techniques for managing wireless ad hoc networks, but the architecture was never implemented [158, 159]. The Object Management Group (OMG) attempted to develop a standard interoperable framework for mobile agents through its Mobile Agent System Interoperability Facility (MASIF) [117]. However, mobile agents research never materialized into practical solutions for real network management issues. Several issues such as security, interoperability, and multi-agent coordination algorithms were never fully solved and integrated correctly into complete solutions.

Mesh-Mon does not have all the capabilities of a generic mobile-agent system. However, our system is designed to facilitate the cooperation between neighbors and the self-management of disconnected nodes in a hierarchical and self-organizing manner, thus meeting the need for scalability and autonomy that mobile agents could theoretically provide.

5.9 Partition detection

We present related work in partition detection since it is an important component of our network analysis engine.

Goyal and Caffery use Depth First Search (DFS) to detect critical links in a mobile ad hoc network [75]. Once a critical link is detected, the authors suggest two ways to avoid or delay its failure: by changing the trajectory of both nodes forming the critical link or by adding more nodes to reinforce. Their approach assumes that locations and speeds of all nodes are known at every node. Mesh-Mon too uses DFS in a similar manner to detect critical nodes in the network.

Hauspie et al. present two metrics that can help in predicting the likelihood of the occurrence of a

partition [81]. The first metric is based on the number of disjoint paths between two nodes (which is expensive to calculate). With more path diversity there is a lower chance of a partition if a single link fails. The second metric is based on the length of a path; the intuition is that longer paths are less robust since a single link failure can be a critical link in the path. Unfortunately their metrics are calculated in a centralized manner.

Milic et al. present a distributed algorithm to detect critical links that uses only local topology knowledge; nodes are required to keep track of position and speed of their one-hop neighbors [116]. The authors predict existence of a critical link by assuming that the probability of a link failure is proportional to the ratio of the distance between two neighbors and the expected transmission range (i.e., nodes further apart are more likely to fail). The scheme requires the distributed construction of the Gabriel graph and uses a closed face-traversal routing technique (used for routing in GPSR) [23, 95] to check for the existence of an alternate path between two nodes that does not use the suspected critical link. This face-traversal routing scheme has been shown to be ineffective in three-dimensional networks [120, 119] and thus is not guaranteed to work in a deployment that has three-dimensional structure (such as our Dart-Mesh deployment where nodes are dispersed on three different floors of a building).

Sheng et al. present a distributed critical-node detection algorithm called the "Detection algorithm based on Midpoint Coverage Circle" (DMCC) [160]. The algorithm requires nodes to know their own geographic locations and those of all their neighbors. There is a step in their detection algorithm that requires finding all possible global paths between two nodes in the neighborhood graph. Thus this scheme is inefficient in practice.

Wang and Li present a technique to predict future partitions based on group mobility patterns [186]. The authors propose a centralized solution and assume that the central server has access to the velocity and position information of all nodes at all instants.

Derhab et al. propose another partition-prediction algorithm based on the partition-detection mechanism used by the Temporally Ordered Routing Algorithm (TORA) [133, 134] and the predicted residual link lifetime of wireless links [52]. The authors do not evaluate the communication costs of their algorithm. For networks that do not use TORA, the cost of implementing such an algorithm are likely to be high. Since signal-strength readings constantly fluctuate, their technique to predict the link lifetime may lead to many

false positives in a practical setting.

In Mesh-Mon we perform all topology-related analysis without using geographic information and without velocity information. DFS can only be calculated in a centralized manner. To improve computational performance, we calculate DFS on the k-hop neighborhood first and use the global topology only if needed, since nodes that are critical in the global topology must also be critical in the k-hop neighborhood.

5.10 Anomaly and Fault Detection

Feather et al. present a statistical technique to determine faults in a wired Ethernet network [61]. They measure performance data parameters and characterize the values into a simple anomaly vector. Using a centralized passive monitoring system, they detect anomalies by identifying vectors that matched any pre-computed fault-feature vector (created from a set of training data).

Thottan and Ji suggest a signal-processing-based mechanism for proactive anomaly detection using intelligent agents [173]. The authors use a Case diagram [32] to determine the relationships of a limited set of handpicked MIB variables that were being monitored. The system was implemented in a centralized manner and was successful in detecting certain faults. It is unclear whether the engine identifies specific faults or generates a generic alarm. The authors claim that their scheme could be extended to work in a distributed system. The same authors present case studies from real network data that demonstrate the power of a signal-processing approach based on abrupt change detection when applied to network anomaly detection [174].

Steinder and Sethi present a fault diagnosis system that allows multiple simultaneous independent faults to be identified [168]. The system uses a fault-localization technique that isolates the most probable set of faults through incremental updates to a symptom-explanation hypothesis. The authors evaluate their system through simulations in a wired network and claim that the system offers close-to-optimal accuracy. The same authors also present a comprehensive survey of fault localization techniques in modern communication systems [167].

Jakobson and Weissman present the design for an alarm-correlation model that was used for three purposes: intelligent alarm filtering, alarm generalization, and fault diagnosis [89]. The authors used a strictly deterministic mode for event correlation and built an expert system for management of a wired network.

Mesh-Mon uses a deterministic rule-based analysis engine that is similar to an expert system, but has a decentralized design. Our design emphasis was on detecting faults as accurately as possible, since anomaly detection methods have high-false positive rates. We can enhance our analysis engine by adopting some of the techniques described above.

Chapter 6

Discussion

In this chapter, we discuss the key features and limitations of our Mesh-Mon solution. We discuss potential avenues for improvement and present suggestions for future work.

6.1 Features

The salient features of Mesh-Mon are as follows.

1. Mesh-Mon uses a localized and distributed approach to both monitoring and management. Thus, our system has no central point of failure. We detect or resolve several management issues automatically with Mesh-Mon, without human intervention.

2. In situations where Mesh-Mon cannot resolve a problem automatically, the information gathered can help the sysadmin evaluate the situation through the automated collection of network state information and diagnostic test results.

3. Our design allows Mesh-Mon to operate even if the routing protocol fails. In reality, the probability of the routing protocol failing is low. We could potentially optimize our design for the common case by running our broadcast-based communication system with lower frequency. However, our broadcast-based communication has the benefit of allowing a node to communicate with all its neighbors simultaneously with just a single transmission. Thus, the benefits of switching to an alternative

unicast-based model would be limited (if any), when used in conjunction with existing radio models, and it would be vulnerable to potential routing protocol errors.

4. Mesh-Mon employs a hierarchical design to provide scalability. Hierarchy is a common technique used to improve scalability for large-scale distributed systems. In general, large self-contained mesh networks have poor performance characteristics and have not been deployed to date. Large-scale mesh networks with multiple wired gateways do exist. Mesh-Mon should scale to networks of any size and Mesh-Mon's performance could be improved by using wired gateways as wormholes between MeshLeaders.

5. Our solution is capable of dealing with partitions (through Mesh-Mon-Ami) and multiple simultaneous failures (as long as they do not affect the operating system or both the wireless radios).

6. The new metrics we introduced, the LBC metric and the LLBC metric, performed successfully in our evaluations. Both metrics can be calculated in a fully distributed manner, thus making them a good match for our distributed analysis engine in Mesh-Mon.

6.2 Limitations

We note the following limitations of our current design and implementation of Mesh-Mon.

1. We did not implement any security mechanisms for Mesh-Mon (or Dart-Mesh). The lack of implemented security mechanisms limit the potential for deployment of our current system outside of an academic setting.

   Siddiqui et al. discuss an overview of security issues related to mesh networks [163]. Through ns-2 simulations, Winjum et al. evaluate the performance impact of applying different protection schemes to OLSR [189]. We could potentially adapt some of these techniques to enhance the security of Mesh-Mon networks running OLSR.

2. Mesh-Mon is not designed to deal with Byzantine failures. We acknowledge that we have no mechanism in place to protect the management system from being compromised or subverted by a determined adversary, and that a single compromised node can potentially bring down an entire network.

3. Our current implementation does not support modification of configuration settings at runtime.

4. Our analysis engine is deterministic and pre-defined. Our engine has no "learning" capability.

5. Mesh-Mon has limited capabilities to help a sysadmin deploy nodes or reconfigure the network layout. Mesh-Mon can tell the sysadmin if a node is critical, but does not suggest a solution (such as where to move the node or where to place a new node). In general, Mesh-Mon could not solve such problems without the availability of accurate location information.

6. We did not implement the MeshLeader election algorithm and the inter-Mesh-Leader communication and analysis protocol.

7. We have implemented limited visualization support for all the data gathered by Mesh-Mon. With the exception of topology maps, all the information generated by Mesh-Mon is textual.

8. We did not fully implement our anomaly detection algorithms in our prototype.

9. Our current implementation may not work correctly in a mixed environment with both Big Endian and Little Endian devices. This problem can be overcome by using a data portability API such as the eXtended Data Representation (XDR) standard [57].

6.3 Future Work

We present a list of ideas that would enhance Mesh-Mon and help overcome most of its limitations.

1. For use in a real-world deployment in a non-academic setting, both Dart-Mesh and Mesh-Mon need additional security features. As a minimal first step, we could encrypt all Mesh-Mon related-traffic on the mesh network and adopt the security "best practices" used in existing commercial networks and their related management systems.

2. Mesh-Mon could be extended to allow sysadmins the ability to remotely configure Mesh-Mon nodes dynamically. At present our system only allows reconfiguration at compile time.

3. Mesh-Mon could be integrated with a customized IDS and thus Mesh-Mon could be extended to provide full Fault, Configuration, Accounting, Performance and Security (FCAPS) management capabilities that we described earlier in Section 1.3. Intrusion detection and fault localization often require monitoring common sets of parameters. Thus, an integrated approach to security and fault management should have lower system-management overheads if they are efficiently combined. We must be careful to ensure that the combined system is not less secure than a system with non-unified components that can provide equivalent functionality.

4. We would like to implement more advanced statistical-analysis techniques borrowed from control-theory, such as 3-sigma and cumulative-sum control charts, to detect "anomalies" [53].

5. We would like to redesign the analysis engine to incorporate a flexible and extensible design. Ideally all rules and actions should be definable in an external format, thus allowing dynamic changes or corrections to incorrect rules previously set.

6. We would like to add advanced-hypothesis tracking and alarm-correlation algorithms, giving our analysis engine more autonomy, and thus the potential to detect unforeseen problems. Research in this area of autonomous computing is in its infancy and self-managing autonomous systems are a new emerging research discipline [55].

7. We could optimize communications between Mesh-Mon nodes through the use of more efficient data structures and compression techniques.

8. We could add location awareness to some of our analysis techniques if accurate and affordable location technologies become available in the future. Such technologies would improve our topology analysis algorithms and allow us to design efficient deployment algorithms to minimize coverage holes.

9. We would like to enhance Mesh-Mon's monitoring capabilities by monitoring wireless interference around each node. Devices such as the "Wi-Spy" are available in the market today [165].

10. We would like to test Mesh-Mon with other routing protocols (such as DSR or LQSR) and other proposed standard protocols that will emerge from the IEEE 802.11s, which may dominate future deployed mesh networks.

11. We would like to test Mesh-Mon on a large-scale deployment, covering a whole campus or even an entire city. Since such a deployment would be expensive, we could first study our system using simulations.

12. While we provide a small set of illustrative examples from a small mesh deployment to study our social-network centrality metrics, we acknowledge that it is important to further evaluate our new metrics on large-scale mesh networks with real traffic patterns, such as the "Freifunk" network in Berlin [1]; unfortunately its traffic patterns are not available. We were not able to acquire both the topology and traffic patterns for any large-scale mesh networks. We encourage mesh operators to make large mesh traces publicly available to the research community [191]. To accurately verify the utility of EVC, LBC and LLBC to sysadmins (in practical management situations) we would need to conduct a user study of Mesh-Mon in which real sysadmins must manage mesh networks for a variety of topologies, densities, sizes, traffic patterns, and deployment scenarios.

Chapter 7

Summary

In this thesis, we studied mobile mesh networks envisioned for rapid deployment and the problem of how to manage such networks. We implemented the Dart-Mesh wireless mesh network at Dartmouth and learned several lessons from the experience. We channelled the lessons learned from our experience to design and develop Mesh-Mon, a monitoring and management system for wireless mesh networks. While we mostly tested Mesh-Mon on a stationary network, we designed it to deal with problems that are likely to occur in a mobile network.

A journal paper describing our Mesh-Mon architecture and initial results was recently published in May 2008 in a Special Issue on Modeling, Testbeds, and Applications in Wireless Mesh Networks of the Computer Communications Journal [123]. A paper describing the LBC metric and results was accepted for publication in the 17th International Conference on Computer Communications and Networking (ICCCN) and will appear in August 2008 [122].

We summarize the **main contributions** of our thesis below.

- We deployed Dart-Mesh; a two-tier wireless mesh network with 16 nodes that provided reliable wireless broadband access for mobile clients. The deployment covered all three floors of Sudikoff Lab, our Computer Science department building at Dartmouth College.

- We designed and implemented Mesh-Mon; a distributed mesh-network management architecture that can help a team of system administrators monitor and manage a mobile mesh network. Our system

has several unique features that are novel and are not present in other mesh-network management systems:

  - The ability to operate on a mesh network running any layer-three routing protocol.
  - The ability to operate even when the routing protocol fails.
  - A distributed analysis engine where nodes cooperate in a distributed manner to identify problems.
  - Monitoring of client nodes as well as mesh nodes and the use of Mesh-Mon-Ami clients to support Mesh-Mon in disconnected environments.

- We showed that our hierarchical structure using Mesh-Leaders helped to add scalability to our design, thus making our system practical for deployment in larger networks. Mesh-Mon nodes add little overhead to the network.

- Mesh-Mon can detect several problems and resolve many of them autonomously and without intervention from the system administrator:

  - Problems related to configuration issues.
  - Problems related to network topology and partitions.
  - Problems related to network performance.

- Through our experiments, we demonstrated Mesh-Mon's ability to adapt to the management needs of mesh networks running different mesh routing protocols: AODV and OLSR. Mesh-Mon is designed manage mesh networks running any layer-three routing protocol.

- We evaluated Mesh-Mon running on Dart-Mesh and demonstrated our success in resolving a number of management challenges.

- We recognized that partitioning can be a significant problem for mobile mesh networks and designed our system to detect partitions and predict potential future partitions. We also proposed the novel idea of using mobile clients running Mesh-Mon-Ami software to act as data-mules that ferry management packets across multiple partitions.

- We applied novel techniques derived from social-network analysis that revealed relative information about different nodes and showed how these techniques are useful from a management perspective.
- We introduced the localized bridging centrality (LBC) metric. This new centrality metric is an improvement over the bridging centrality metric, since our metric can be calculated in a distributed manner, yet provides equally effective results. We also developed the distributed traffic-load-aware bridging centrality (LLBC) metric and demonstrated its benefits.

7.1 Conclusion

In conclusion, we successfully designed and deployed Mesh-Mon, a novel management system for mobile wireless mesh networks. In the future, we would like to deploy our system on a larger scale and continue to improve its capabilities.

Bibliography

[1] Berlin Freifunk Community Mesh. http://berlin.freifunk.net/
Last checked: March 21, 2008.

[2] Atul Adya, Paramvir Bahl, Ranveer Chandra, and Lili Qiu. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 30–44, Philadelphia, PA, USA, 2004. ACM Press.

[3] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 121–132, Portland, Oregon, USA, 2004. ACM Press.

[4] I.F. Akyildiz and Xudong Wang. A survey on wireless mesh networks. *IEEE Communications Magazine*, 43(9), September 2005.

[5] W. Allen, A. Martin, and A. Rangarajan. Designing and deploying a rural ad-hoc community mesh network testbed. In *Proceedings of the 30th Anniversary Edition of the IEEE Conference on Local Computer Networks (LCN)*, November 15–17, 2005.

[6] Y. Amir, C. Danilov, M. Hilsdale, R. Musaloiu-Elefteri, and N. Rivera. Fast Handoff for Seamless Wireless Mesh Networks. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 83–95, Uppsala, Sweden, 2006. ACM Press.

[7] Amit P. Jardosh and Panuakdet Suwannatat and Tobias Höllerer and Elizabeth M. Belding and Kevin C. Almeroth. SCUBA: Focus and Context for Real-Time Mesh Network Health Diagnosis. In Mark Claypool and Steve Uhlig, editor, *Proceedings of the 9th International Conference on Passive and Active Network Measurement (PAM)*, volume 4979 of *Lecture Notes in Computer Science*, pages 162–171. Springer, April 2008.

[8] Aruba Networks Wireless Networking Systems. http://www.arubanetworks.com
Last checked: March 1, 2008.

[9] Remi Badonnel, Radu State, and Olivier Festor. Management of mobile ad hoc networks: information model and probe-based architecture. *International Journal of Network Management*, 15(5):335–347, 2005.

[10] Z. Bajic. Method, system and apparatus for creating a mesh network of wireless switches to support layer 3 roaming in wireless local area networks, November 2, 2006. WO Patent WO/2006/115,828.

[11] M. Baldi, S. Gai, and G.P. Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Proceedings of the First International Workshop on Mobile Agents*, pages 13–26. Springer-Verlag London, UK, 1997.

[12] M. Baldi and G.P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proceedings of the 20th International Conference on Software Engineering*, pages 146–155, 1998.

[13] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 1028–1037. IEEE Press, 2001.

[14] Michael Barborak, Anton Dahbura, and Minoslaw Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, 1993.

[15] K. Begnum and M. Burgess. Principle Components and Importance Ranking of Distributed Anomalies. *Machine Learning*, 58(2):217–230, 2005.

[16] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 31–42, Cologne, Germany, 2005. ACM Press.

[17] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1):2–9, 1998.

[18] C. Bohoris. *Network Performance Management Using Mobile Software Agents*. PhD thesis, University of Surrey, UK, June 2003.

[19] P. Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.

[20] P. Bonacich. Power and Centrality: A Family of Measures. *The American Journal of Sociology*, 92(5):1170–1182, 1987.

[21] P. Bonacich and P. Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks*, 23(3):191–201, 2001.

[22] S.P. Borgatti, M.G. Everett, and L.C. Freeman. UCINET for Windows: Software for Social Network Analysis. *Harvard: Analytic Technologies*, 2002.

[23] Prosenjit Bose, Pat Morin, Ivan Stojmenović, 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 (DIALM)*, pages 48–55. ACM Press, 1999.

[24] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[25] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[26] M. Burgess, H. Haugerud, S. Straumsnes, and T. Reitan. Measuring system normality. *ACM Transactions on Computer Systems (TOCS)*, 20(2):125–160, 2002.

[27] Ilir Bytyci. Monitoring Changes in the Stability of Networks Using Eigenvector Centrality. Master's thesis, Oslo University College, May 22, 2006.

[28] J. Camp, J. Robinson, C. Steger, and E. Knightly. Measurement driven deployment of a two-tier urban mesh access network. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 96–109. ACM Press, 2006.

[29] W. Caripe, G. Cybenko, K. Moizumi, and R.S. Gray. Network awareness and mobile agent systems. *IEEE Communications Magazine*, 36(7):44–49, July 1998.

[30] I. Carreras, D. Miorandi, G.S. Canright, and K. Engo-Monsen. Understanding the Spread of Epidemics in Highly Partitioned Mobile Networks. In *Proceedings of the First Workshop on Bio-Inspired Models of Network, Information and Computing Systems*, pages 1–8, 2006.

[31] J. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[32] J. Case and C. Partridge. Case diagrams: a first approach to diagrammed management information base. *Computer Communication Review*, 19:13–16, January 1989.

[33] R. Chadha, Y. H. Cheng, J. Chiang, G. Levin, S. W. Li, A. Poylisher, L. LaVergne, and S. Newman. Scalable policy management for ad hoc networks. In *Proceedings of the Military Communications Conference (MILCOM)*, pages 2151–2157. IEEE Press, October 2005.

[34] Ranveer Chandra, Venkata N. Padmanabhan, and Ming Zhang. WiFiProfiler: cooperative diagnosis in wireless LANs. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 205–219, Uppsala, Sweden, 2006. ACM Press.

[35] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[36] Wenli Chen, Nitin Jain, and S. Singh. ANMP: ad hoc network management protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1506–1531, August 1999.

[37] C.Y.J. Chiang, S. Demers, P. Gopalakrishnan, L. Kant, A. Poylisher, Y. H. Cheng, R. Chadha, G. Levin, S. Li, Y. Ling, S. Newman, L. LaVergne, and R. Lo. Performance Analysis of DRAMA: A Distributed Policy-Based System for MANET Management. In *Proceedings of the Military Communications Conference (MILCOM)*, pages 1–8, Washington, DC, October 2006. IEEE Press.

[38] Junho H. Choi, George A. Barnett, and Bum-soo Chon. Comparing world city networks: a network analysis of Internet backbone and air transport intercity linkages. *Global Networks*, 6(1):81–99, 2006.

[39] Cisco Mesh Products. http://www.cisco.com/en/US/netsol/ns703/
Last checked: March 1, 2008.

[40] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[41] T.H. Clausen, P. Jacquet, and L. Viennot. Investigating the impact of partial topology in proactive MANET routing protocols. In *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, volume 3, pages 1374–1378, October 27–30, 2002.

[42] IEEE Standards Committee. IEEE Trial-Use Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation. http://standards.ieee.org/getieee802/download/802.11F-2003.pdf
Last checked: March 1, 2008.

[43] IEEE Standards Committee. *IEEE 802.11 Standards for Wireless LANs*. IEEE Press, 2008.

[44] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. Cambridge, Mass.: MIT Press, second edition, 2001.

[45] E. Costenbader and T.W. Valente. The stability of centrality measures when networks are sampled. *Social Networks*, 25(4):283–307, 2003.

[46] Elizabeth M. Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant MANETs. In *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 32–40, Montreal, Quebec, Canada, 2007. ACM Press.

[47] S.R. Das, C.E. Perkins, and E. M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, pages 3–12, Tel Aviv, March 26–30, 2000. IEEE Press.

[48] Budhaditya Deb, Sudeept Bhatnagar, and Badri Nath. Multi-resolution state retrieval in sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 19–29, May 11, 2003.

[49] Budhaditya Deb, Sudeept Bhatnagar, and Badri Nath. A topology discovery algorithm for sensor networks with applications to network management. Technical Report DCS-TR-441, Rutgers University, May 2001.

[50] Douglas S.J. DeCouto, Daniel Aguayo, John Bicket, and Robert Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. *Wireless Networks*, 11(4):419–434, 2005.

[51] S. Demers and L. Kant. MANETs: Performance Analysis and Management. In *Proceedings of the Military Communications Conference (MILCOM)*, pages 1–7. IEEE Press, October 2006.

[52] A. Derhab, N. Badache, and A. Bouabdallah. A partition prediction algorithm for service replication in mobile ad hoc networks. In *Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 236–245, January 2005.

[53] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, sixth edition edition, 2004.

[54] R.B. Dilmaghani, B.S. Manoj, B. Jafarian, and R.R. Rao. Performance Evaluation of RescueMesh: A Metro-scale Hybrid Wireless Network. In *Proceedings of the First IEEE Workshop on Wireless Mesh Networks (held in conjunction with SECON-2005)*, Santa Clara, CA, September 2005.

[55] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomic Adaptive Systems*, 1(2):223–259, 2006.

[56] John W. Eaton and others. GNU Octave. http://www.gnu.org/software/octave/
Last checked: March 1, 2008.

[57] M. Eisler. XDR: External Data Representation Standard. Technical report, STD 67, RFC 4506, 2006.

[58] P.E. Engelstad, A. Tønnesen, A. Hafslund, and G. Egeland. Internet connectivity for multi-homed proactive ad hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 7, pages 4050–4056, June 20–24, 2004.

[59] Jakob Eriksson, Sharad Agarwal, Paramvir Bahl, and Jitendra Padhye. Feasibility study of mesh networks for all-wireless offices. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 69–82, Uppsala, Sweden, 2006. ACM Press.

[60] M. Everett and S.P. Borgatti. Ego network betweenness. *Social Networks*, 27(1):31–38, 2005.

[61] Frank Feather, Dan Siewiorek, and Roy Maxion. Fault detection in an Ethernet network using anomaly signature matching. *SIGCOMM Computer Communications Review*, 23(4):279–288, 1993.

[62] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[63] Glenn Fleishman. Wireless Public Safety Network News Blog. http://publicsafety.wifinetnews.com/
Last checked: March 1, 2008.

[64] Desktop Management Task Force. http://www.dmtf.org/
Last checked: March 1, 2008.

[65] L.C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, 1977.

[66] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[67] L.J. Gasch. Scalable, autonomous monitoring and response for network communications reliability. In *Proceedings of the Military Communications Conference (MILCOM)*, volume 1, pages 423–429. IEEE Press, October 2004.

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

[69] G. Goldszmidt and Y. Yemini. Evaluating Management Decisions via Delegation. In *Proceedings of the IFIP TC6/WG6. 6 in the Third International Symposium on Integrated Network Management*, pages 247–257. North-Holland, 1993.

[70] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 333–340, Vancouver, BC, May 1995.

[71] G. Goldszmidt and Y. Yemini. Delegated agents for network management. *IEEE Communications Magazine*, 36(3):66–70, 1998.

[72] Germán Goldszmidt. On distributed system management. In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 637–647. IBM Press, 1993.

[73] C. Gomez, D. Garcia, and J. Paradells. Improving performance of a real ad-hoc network by tuning OLSR parameters. *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC)*, pages 16–21, June 2005.

[74] Greg Goth. Groups Hope to Avoid Mesh Standard Mess. *IEEE Distributed Systems Online*, 6(9), 2005.

[75] D. Goyal and J. Caffery Jr. Partitioning avoidance in mobile ad hoc networks using network survivability concepts. In *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC)*, pages 553–558, 2002.

[76] Robert S. Gray, M. Diekhans, and M. Roseman. Agent Tcl:A flexible and secure mobile-agent system. *Fourth Annual Tcl/Tk Workshop (TCL 96)*, pages 9–23, 1996.

[77] Robert S. Gray, David Kotz, Calvin Newport, Nikita Dubrovsky, Aaron Fiske, Jason Liu, Christopher Masone, Susan McGrath, and Yougu Yuan. Outdoor experimental comparison of four ad hoc routing

algorithms. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 220–229, October 2004.

[78] Z.J. Haas, J.Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on Networking (TON)*, 14(3):479–491, 2006.

[79] Z.J. Haas and M.R. Pearlman. ZRP: a hybrid framework for routing in Ad Hoc networks. *Ad hoc networking table of contents*, pages 221–253, 2001.

[80] Matt Hamblen. Providence unveils wireless network for public safety. *Computerworld Magazine*, September 5, 2006.

[81] M. Hauspie, D. Simplot, and J. Carle. Partition detection in mobile ad-hoc networks. In *Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networking Workshop (MED-HOC-NET)*, June 2003.

[82] Ting-Chao Hou and Victor Li. Transmission Range Control in Multihop Packet Radio Networks. *IEEE Transactions on Communications*, 34(1):38–44, January 1986.

[83] J. Hsu, S. Bhatia, M. Takai, R. Bagrodia, and M. J. Acriche. Performance of mobile ad hoc networking routing protocols in realistic scenarios. In *Proceedings of the Military Communications Conference (MILCOM)*, volume 2, pages 1268–1273. IEEE Press, October 13–16, 2003.

[84] Yangcheng Huang, S.N. Bhatti, and D. Parker. Tuning OLSR. In *Proceedings of the 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5, Helsinki, September 2006.

[85] W. Hwang, Y. Cho, A. Zhang, and M. Ramanathan. Bridging Centrality: Identifying Bridging Nodes in Scale-free Networks. Technical Report 2006-05, Department of Computer Science and Engineering, University at Buffalo, March 15 2006.

[86] Internet Engineering Task Force (IETF) Workgroup on Mobile Ad-hoc Networks (MANET), 2008. http://www.ietf.org/html.charters/manet-charter.html
Last checked: March 1, 2008.

[87] A. Iwata, C.C. Chiang, G. Pei, M. Gerla, and T.W. Chen. Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications,*, 17(8):1369–1379, 1999.

[88] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proceedings of the IEEE International Multi Topic Conference on Technology for the 21st Century (INMIC)*, pages 62–68, 2001.

[89] G. Jakobson and M. Weissman. Alarm correlation. *IEEE Network*, 7(6):52–59, 1993.

[90] M. Jiang, J. Li, and Y.C. Tay. Cluster Based Routing Protocol (CBRP), draft-ietf-manet-cbrp-spec-01. txt. *IETF Internet Draft*, August 1999.

[91] John Hopkins University (JHU) SMESH Mesh Network. http://www.smesh.org
Last checked: March 1, 2008.

[92] D.B. Johnson, D.A. Maltz, J. Broch, et al. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. *Ad Hoc Networking*, 1:139–172, 2001.

[93] M. Jorgic, I. Stojmenovic, M. Hauspie, and D. Simplot-Ryl. Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. In *Proceedings of the 3rd IFIP Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET 2004)*, Bodrum, Turkey, June 2004.

[94] L. Kant, S. Demers, P. Gopalakrishnan, R. Chadha, L. LaVergne, and S. Newman. Performance modeling and analysis of a mobile ad hoc network management system. In *Proceedings of the Military Communications Conference (MILCOM)*, pages 2816–2822. IEEE Press, October 2005.

[95] Brad Karp and H.T. Kung. GPSR: Greedy Perimeter Stateless Routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom)*, pages 243–254, Boston, Massachusetts, United States, 2000. ACM Press.

[96] Luke Klein-Berndt. NIST kernel AODV, April 2004. http://www.antd.nist.gov/wctg/aodv_kernel/
Last checked: March 1, 2008.

[97] Y.B. Ko and N.H. Vaidya. Location-Aided Routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

[98] P. Krishna, M. Chatterjee, N.H. Vaidya, and D.K. Pradhan. A Cluster-based Approach for Routing in Ad-Hoc Networks. In *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, pages 1–10, 1995.

[99] Veljko Krunic, Eric Trumpler, and Richard Han. NodeMD: diagnosing node-level faults in remote wireless sensor systems. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 43–56, San Juan, Puerto Rico, 2007. ACM Press.

[100] Vijay Kumar, D. Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *IEEE Pervasive Computing*, 3(4):24–33, October/December 2004.

[101] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-down Approach Featuring the Internet*. Addison-Wesley, 2003.

[102] D.B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming Mobile Agents in Java. In *Proceedings of the International Conference on Worldwide Computing and Its Applications: International Conference (WWCA)*, Tsukuba, Japan, March 10-11, 1997. Springer.

[103] Sam Leffler and others. Multiband Atheros Driver for WiFi (MADWiFi), 2008. http://madwifi.org/ Last checked: March 1, 2008.

[104] W.H. Liao, J.P. Sheu, and Y.C. Tseng. GRID: A Fully Location-Aware Routing Protocol for Mobile Ad Hoc Networks. *Telecommunication Systems*, 18(1):37–60, 2001.

[105] H. Lim and C. Kim. Flooding in wireless ad hoc networks. *Computer Communications*, 24(3-4):353–363, 2001.

[106] Jason Liu, Yougu Yuan, David M. Nicol, Robert S. Gray, Calvin C. Newport, David Kotz, and Luiz Felipe Perrone. Simulation validation using direct execution of wireless ad-hoc routing protocols. In *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, pages 7–16, May 2004.

[107] LocustWorld MeshAP Solution. http://www.locustworld.com/ Last checked: March 1, 2008.

[108] H. Lundgren, K. N. Ramachandran, E. Belding-Royer, K. Almeroth, M. Benny, A. Hewatt, A. Touma, and A. Jardosh. Experiences from the design, deployment, and usage of the UCSB MeshNet testbed. *IEEE Wireless Communications*, 13(2):18–29, April 2006.

[109] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1996.

[110] J. Malinen. Hostap: Wireless Driver for Intersil Prism2/2.5/3, 2005. http://hostap.epitest.fi/ Last checked: March 1, 2008.

[111] P.V. Marsden. Egocentric and sociocentric measures of network centrality. *Social Networks*, 24(4):407–422, 2002.

[112] J.P. Martin-Flatin and S. Znaty. Annotated Typology of Distributed Network Management Paradigms. Technical Report T/R SSC/1997/008, École Polytechnique Fédérale de Lausanne, 1997.

[113] J.P. Martin-Flatin, S. Znaty, and J.P. Hubaux. A Survey of Distributed Enterprise Network and Systems Management Paradigms. *Journal of Network and Systems Management*, 7(1):9–26, 1999.

[114] A.B. McDonald and T.F. Znati. A mobility-based framework for adaptive clustering in wireless adhoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1466–1487, 1999.

[115] MeshDynamics Structured Mesh Products. http://www.meshdynamics.com/ Last checked: March 1, 2008.

[116] B. Milic, N. Milanovic, and M. Malek. Prediction of Partitioning in Location-Aware Mobile Ad Hoc Networks. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)*, January 03–06 2005.

[117] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, et al. MASIF: The OMG mobile agent system interoperability facility. *Personal Technologies*, 2(2):117–129, 1998.

[118] Mike Muuss and others. TTCP: Test TCP Utility Program. http://ftp.arl.mil/ftp/pub/ttcp/ Last Checked: March 1, 2008.

[119] Soumendra Nanda. Spatial Multipath Location Aided Routing (MLAR). Master's thesis, Department of Computer Science, Dartmouth College, Hanover, NH, June 2004. Available as a Dartmouth Computer Science Technical Report TR2005-533
http://www.cs.dartmouth.edu/reports/abstracts/TR2005-533/
Last Checked May 1, 2008.

[120] Soumendra Nanda and Robert S. Gray. Multipath Location Aided Routing (MLAR) in 2D and 3D. In *Proceedings of the Sixth IEEE Wireless Communications and Networking Conference (WCNC)*, volume 1, pages 311–317, April 3–6, 2006.

[121] Soumendra Nanda, Zhenhui Jiang, and David Kotz. A combined routing method for ad hoc networks. Technical Report TR2007-588, Department of Computer Science, Dartmouth College, June 2007.

[122] Soumendra Nanda and David Kotz. Localized bridging centrality for distributed network analysis. In *Proceedings of the 17th International Conference on Computer Communications and Networking (ICCCN)*, August 2008. Accepted for publication.

[123] Soumendra Nanda and David Kotz. Mesh-Mon: a multi-radio mesh monitoring and management system. *Computer Communications (COMCOM)*, 31(8):1588–1601, May 2008. Special Issue: Modeling, Testbeds, and Applications in Wireless Mesh Networks
DOI: http://dx.doi.org/10.1016/j.comcom.2008.01.046
Last checked: May 1, 2008.

[124] Soumendra Nanda, Tushaar Sethi, and Shashikant A. Shinde. Mobile Agents for Network Management, May 2001. Undergraduate Senior Project Thesis, Ramrao Adik Institute Of Technology, University of Mumbai, Navi Mumbai, India.

[125] Dries Naudts, Stefan Bouckaert, Johan Bergs, Abram Schoutteet, Chris Blondia, Ingrid Moerman, and Piet Demeester. A wireless mesh monitoring and planning tool for emergency services. In *Proceedings of the Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, pages 1–6, Munich, Germany, May 2007.

[126] N. Negroponte and others. The One Laptop Per Child (OLPC) Project. http://laptop.org
Last checked: March 1, 2008.

[127] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile Computing and Networking (MobiCom)*, pages 151–162. ACM Press, 1999.

[128] Erik Nordström and others. AODV-UU Uppsala University AODV Implementation, 2008. http://core.it.uu.se/core/index.php/AODV-UU
Last checked: March 1, 2008.

[129] Tobias Oetiker and others. RRDtool: Round Robin Database Tool, 2008. http://oss.oetiker.ch/rrdtool/
Last checked: March 1, 2008.

[130] R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684 (Experimental), February 2004.

[131] Order One Networks Mesh Networking Products. http://www.orderonenetworks.com
Last checked: March 1, 2008.

[132] K. Papagiannaki, M. Yarvis, and W. S. Conner. Experimental characterization of home wireless networks and design implications. *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–13, April 2006.

[133] V. Park and S. Corson. Temporally-Ordered Routing Algorithm (TORA) version 1: Functional specification, draft-ietf-manet-toraspec-04. txt, work in progress. *IETF, July*, 2001.

[134] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 3, pages 1405–1413, Kobe, Japan, April 1997. IEEE Press.

[135] C. Perkins. IP Mobility Support for IPv4. RFC 3344 (Proposed Standard), August 2002. Updated by RFC 4721.

[136] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[137] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, New Orleans, LA, February 25–26, 1999. IEEE Press.

[138] C.E. Perkins and K.Y. Wang. Optimized smooth handoffs in Mobile IP. In *Proceedings of the IEEE International Symposium on Computers and Communications (ISCC)*, pages 340–346, 1999.

[139] T. Plesse, J. Lecomte, C. Adjih, M. Badel, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, and A. Plakoo. OLSR performance measurement in a military mobile ad-hoc network. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 704–709, 2004.

[140] Meraki Mesh Networking Products. http://www.meraki.com
Last checked: March 1, 2008.

[141] Motorola Mesh Networking Products. http://www.motorola.com/mesh/
Last checked: March 1, 2008.

[142] Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou. Troubleshooting wireless mesh networks. *SIGCOMM Computer Communications Review*, 36(5):17–28, 2006.

[143] K. N. Ramachandran, E. M. Belding-Royer, and K. C. AImeroth. DAMON: a distributed architecture for monitoring multi-hop mobile networks. In *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, pages 601–609, October 4–7, 2004.

[144] Krishna N. Ramachandran. *Design, Deployment, and Management of High-Capacity Large-scale Wireless Networks*. PhD thesis, University of California Santa Barbara, April 2007.

[145] Krishna N. Ramachandran, Kevin Almeroth, and Elizabeth Belding-Royer. A novel framework for the management of large-scale wireless network testbeds. In *Proceedings of the First Workshop on Wireless Network Measurements (WiNMee)*, Riva del Garda, Trentino, Italy, April 2005.

[146] Krishna N. Ramachandran, Irfan Sheriff, Elizabeth M. Belding, and Kevin C. Almeroth. Routing stability in static wireless mesh networks. In *Proceedings of the 8th International Conference Passive and Active Network Measurement (PAM)*, pages 73–82, Louvain-la-neuve, Belgium, April 2007. Springer.

[147] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 255–267. ACM Press, 2005.

[148] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *Proceedings of the Fifth IEEE Wireless Communications and Networking Conference (WCNC)*, volume 3, pages 1664–1669, March 13–17, 2005.

[149] Microsoft Research. Mesh Networking Academic Resource Toolkit, 2005. http://research.microsoft.com/netres/kit/index.htm
Last checked: March 1, 2008.

[150] J. Robinson and E.W. Knightly. A performance study of deployment factors in wireless mesh networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2054–2062, Anchorage, AK, USA, May 2007. IEEE Press.

[151] J. Robinson, K. Papagiannaki, C. Diot, X. Guo, and L. Krishnamurthy. Experimenting with a multi-radio mesh networking testbed. In *Proceedings of the First Workshop on Wireless Network Measurements (WiNMee)*, Riva del Garda, Italy, April 2005.

[152] S. Rost and H. Balakrishnan. Memento: A Health Monitoring System for Wireless Sensor Networks. In *Proceedings of the 3rd Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, volume 2, pages 575–584. IEEE Press, September 2006.

[153] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 11, pages 329–350, 2001.

[154] G. Rozema and others. OpenWRT Firmware, 2008. http://openwrt.org/
Last checked: March 1, 2008.

[155] Y. Sasson, D. Cavin, and A. Schiper. Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Proceedings of the Wireless Communications and Networking Conference (WCNC)*, volume 2. IEEE Press, 2003.

[156] T. Saydam and T. Magedanz. From networks and network management into service and service management. *Journal of Network and Systems Management*, 4(4):345–348, 1996.

[157] Nicola Scalabrino, Roberto Riggio, Daniele Miorandi, and Imrich Chlamtac. JANUS: A Framework for Distributed Management of Wireless Mesh Networks. In *Proceedings of the 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, May 21–23 2007.

[158] Chien-Chung Shen, C. Jaikaeo, C. Srisathapornphat, and Zhuochuan Huang. The Guerrilla Management Architecture for Ad Hoc Networks. In *Proceedings of the Military Communications Conference (MILCOM)*, volume 1, pages 467–472, October 7–10, 2002.

[159] Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo. An adaptive management architecture for ad hoc networks. *IEEE Communications Magazine*, 41(2):108–115, February 2003.

[160] Min Sheng, Jiandong Li, and Yan Shi. Critical nodes detection in mobile ad hoc network. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications 2006 (AINA)*, volume 2, pages 336–340, April 18–20, 2006.

[161] Anmol Sheth, Christian Doerr, Dirk Grunwald, Richard Han, and Douglas Sicker. MOJO: a distributed physical layer anomaly detection system for 802.11 WLANs. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 191–204, Uppsala, Sweden, 2006. ACM Press.

[162] M.L. Sichitiu. Wireless Mesh Networks: Opportunities and Challenges. *Proceedings of World Wireless Congress*, 2005.

[163] Muhammad Shoaib Siddiqui and Choong Seon Hong. Security issues in wireless mesh networks. In *Proceedings of the International Conference on Multimedia and Ubiquitous Engineering (MUE)*, pages 717–722, Seoul, Korea, April 2007.

[164] R. Sombrutzki, A. Zubow, M. Kurth, and J. P. Redlich. Self-Organization in Community Mesh Networks The Berlin RoofNet. In *Proceedings of the 1st Workshop on Operator-Assisted (Wireless Mesh) Community Networks*, pages 1–11, Berlin, September 2006.

[165] MetaGeek Wi-Spy 2.4 Ghz spectrum analyzer. http://www.metageek.net/Products/Wi-Spy
Last checked: March 1, 2008.

[166] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing, 1998.

[167] M. Steinder and A.S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[168] M. Steinder and A.S. Sethi. Probabilistic fault diagnosis in communication systems through incremental hypothesis updating. *Computer Networks*, 45(4):537–562, 2004.

[169] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[170] Robert. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1:146, 1972.

[171] Technology For All (TFA) Rice Wireless Mesh Network, 2008. http://tfa.rice.edu/
Last checked: March 1, 2008.

[172] The Wi-Fi Alliance. http://www.wi-fi.org/
Last checked: March 1, 2008.

[173] M. Thottan and Chuanyi Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network*, 12(5):21–27, September/October 1998.

[174] M. Thottan and Chuanyi Ji. Anomaly detection in IP networks. *IEEE Transactions on Signal Processing*, 51(8):2191–2204, August 2003.

[175] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf - The TCP/UDP bandwidth measurement tool. http://sourceforge.net/projects/iperf
Last checked: March 1, 2008.

[176] Juan Toledo, Vincent van Adrighem, Riccardo Ghetta, Eran Mann, Frederic Peters, and others. EtherApe: a graphical network monitor. http://etherape.sourceforge.net/
Last checked: March 1, 2008.

[177] Andreas Tønnesen. Implementing and extending the Optimized Link State Routing protocol. Master's thesis, University of Oslo, Norway, 2004.

[178] Andreas Tønnesen. OLSR version 0.4.10, 2006. http://www.olsr.org/
Last checked: March 1, 2008.

[179] Tropos Networks Public Safety Solutions. http://www.tropos.com/public_safety.html
Last checked: March 1, 2008.

[180] Sven-Ola Tücke Sven-Ola Tücke Sven-Ola Tücke and others. Freifunk firmware for mesh routers, 2008. http://www.freifunk.net
Last checked: March 1, 2008.

[181] C. Tuduce and T. Gross. Resource monitoring issues in ad hoc networks. In *Proceedings of the International Workshop on Wireless Ad-Hoc Networks (IWWAN)*, pages 259–264, May 31–June3, 2004.

[182] United States Army. Future Combat Systems Program. http://www.army.mil/fcs/
Last checked: March 1, 2008.

[183] University of California Santa Barbara (UCSB) MeshNet. http://moment.cs.ucsb.edu/meshnet/
Last checked: March 1, 2008.

[184] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-2000-06, Duke University, April 2000.

[185] K. Viswanath and K. Obraczka. Modeling the performance of flooding in wireless multi-hop Ad hoc networks. *Computer Communications*, 29(8):949–956, 2006.

[186] K.H. Wang and Baochun Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 2, pages 1017–1021. IEEE Press, 2002.

[187] T. White, A. Bieszczad, and B. Pagurek. Distributed Fault Location in Networks Using Mobile Agents. In *Proceedings of the Second International Workshop on Agents in Telecommunication Applications (IATA)*, pages 130–141. Springer, 1999.

[188] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 194–205, Lausanne, Switzerland, 2002. ACM Press.

[189] E. Winjum, A. M. Hegland, P. Spilling, and O. Kure. A performance evaluation of security schemes proposed for the OLSR protocol. In *Proceedings of the Military Communications Conference (MILCOM)*, pages 2307–2313. IEEE Press, October 17–20, 2005.

[190] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In *Proceedings of the International Symposium on Integrated Network Management*, pages 95–107, 1991.

[191] J. Yeo, D. Kotz, and T. Henderson. CRAWDAD: a Community Rsource for Archiving Wireless Data At Dartmouth. *SIGCOMM Computer Communications Review*, 36(2):21–22, 2006. http://crawdad.cs.dartmouth.edu/manet.php
Last checked: March 1, 2008.

[192] Y. Yi, M. Gerla, and T.J. Kwon. Efficient flooding in ad hoc networks: a comparative performance study. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 2, 2003.

[193] M. Zapf, K. Herrmann, and K. Geihs. Decentralized SNMP management with mobile agents. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 623–635, 1999.

[194] D. Zeltserman. *A practical guide to SNMPv3 and network management*. Prentice Hall PTR, USA, 1999.

[195] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 139–148, May 2003.

Appendix A

Abbreviations

Abbreviation	Expanded Form
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
BC	Bridging Centrality
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DFS	Depth First Search
DSR	Dynamic Source Routing
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
ESSID	Extended Sevice Set Identifier
EVC	Eigenvector Centrality
ETX	Expected Transmission Count
FR	First Responder
GPS	Global Positioning System
GW	Gateway
HNA	Host Network Announcement

Abbreviation	Expanded Form
IBSS	Infrastructure Basic Service Set
ICMP	Internet Control Message Protocol
IEEE	International Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Standards Organization
LAN	Local Area Network
LBC	Localized Bridging Centrality
LLBC	Localized Load-aware Bridging Centrality
LQ	Link Quality
MAC	Machine Access Code
MANET	Mobile Ad hoc Network
MASIF	Mobile Agent System Interoperability Facility
MBps	Mega Bytes per second
Mbps	Mega bits per second
MC	Mobile Client
MIB	Management Information Base
MN	Mobile Node
MMA	Mesh-Mon-Ami
MPR	Multi-Point Relay
MR	Mesh Router
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NMS	Network Management System

Abbreviation	Expanded Form
OLSR	Optimized Link State Routing
OMG	Object Management Group
PCI	Peripheral Connect Interface
PVC	Poly Vinyl Chloride
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request For Comments document
RMI	Remote Method Invocation
SNMP	Simple Network Management Protocol
SSH	Secure Shell
TBRPF	Topology Dissemination Based on Reverse-Path Forwarding
TCP	Transmission Control Protocol
TCR	Topology Change Redundancy
UDP	User Datagram Protocol
WAN	Wide Area Network
WLAN	Wireless Local Area Network