# Experiences from the Design, Deployment, and Usage of the UCSB MeshNet Testbed

Henrik Lundgren, Krishna Ramachandran, Elizabeth Belding-Royer, Kevin Almeroth,
Michael Benny, Andrew Hewatt, Alexander Touma, Amit Jardosh
Department of Computer Science
University of California, Santa Barbara

***Abstract*— In this paper we report on our effort and experience to design, deploy and use our 30 node wireless mesh testbed, the *UCSB MeshNet*. Compared to simulation, the construction and utilization of a wireless mesh testbed poses many new challenges. We discuss the challenges with distributed testbed management, non-intrusive and distributed monitoring, and node status visualization. These are vital components in a sustainable wireless mesh testbed, but at the same time, non-trivial to design and realize. As a case study, we present the UCSB MeshNet architecture, including its management, monitoring, and visualization systems. We share our lessons learned from this effort and believe that they will be valuable to other researchers who develop experimental wireless mesh networks.**

## I. Introduction

In wireless local area networks (WLANs), the wired Ethernet is replaced by wireless connections through access points (APs) that are deployed in designated areas. Each AP, in turn, must be connected to a wired backbone network. This restricts the deployment of the wireless network and the placement of APs. A *mesh network* is a more flexible solution to provide wireless network access. An infrastructure mesh consists of deployed wireless mesh nodes that have minimal or no mobility, and serve the purpose of a multi-hop infrastructure backbone. Each mesh node has wireless router functionality and forwards data packets on behalf of other mesh nodes. Typically, one or more nodes have external network access and act as gateways on behalf of the rest of the mesh network. End nodes, such as laptops and PDAs, may use the mesh infrastructure, but are not required to implement mesh functionality themselves. This is in contrast to *ad hoc networking*, where even end nodes are expected to implement functionality such as routing and self-configuration. Figure 1 shows an example of a multi-hop mesh network backbone that consists of dedicated mesh nodes.

Mesh network research has so far primarily relied on simulation for evaluation and performance prediction. Simulation is attractive since it enables test repeatability, parameter exploration, and scalability. Real world stochastic factors, such as wireless randomness and node mobility, can be controlled and predicted through the use of models. However, models are often simplified to reduce simulation time, or because of the difficulties in accurately representing certain real world phenomena. Another shortcoming is that most simulators lag with respect to recent protocol improvements and new protocol versions. Research studies have shown that simulation results depend heavily on the simulator and models used [1], [2] and that some (performance degrading) effects may not be visible until the protocols are exposed to real world settings [3].

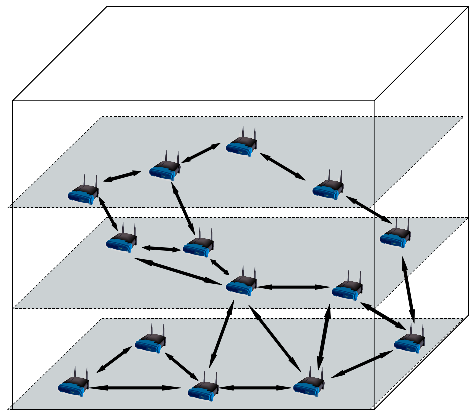
Fig. 1. A wireless multi-hop mesh network backbone inside a building, covering several floors. The black arrows indicate available radio links.

Simulation studies therefore need to be complemented by more realistic testing. This testing is vital to improve the understanding of wireless network systems as it allows researchers to study phenomena that may not be visible in simulator settings. Emulation offers one possibility that represents a middle ground between simulation and real world testing. Typically, wireless randomness and/or node mobility are controllable through emulation [4], [5]. Another advantage of emulation testbeds is that the software under test typically is executed in its final operation system environment. To capture the full effect of the wireless medium and node mobility, however, real testbeds are needed. A few major efforts with ad hoc network testbeds have been used to study various performance effects in wireless and mobile settings [6]–[8]. The primary difference between ad hoc network testbeds and mesh network testbeds is that the former typically consist of mobile devices with limited battery power. This, in turn, means that the lifetime of the ad hoc network is impeded by the battery duration. In contrast, mesh testbeds typically constitute more permanent infrastructures. For example, MIT's Roofnet consists of 30 stationary IEEE 802.11b nodes deployed in apartments on the MIT campus and has been running since 2002 [9]. As another example, a PC-based 23 node IEEE 802.11a indoor mesh testbed was recently built by Microsoft Research [10].

Our testbed, the UCSB MeshNet[1], has been operational since 2004 and is similar to the previous two mesh testbeds with respect to the number of nodes and the fact that the testbed is composed entirely of geographically distributed, physical nodes. The UCSB MeshNet is built from a mix of IEEE 802.11b/g Linksys WRT54G wireless routers and small form-factor PCs, equipped with IEEE 802.11a/b/g PCMCIA wireless network interface cards. The 30 nodes are distributed in offices and research labs throughout the Engineering I building on the UCSB campus. The Linksys WRT54G routers have been upgraded with the OpenWRT[2] Linux distribution so that they can run in ad hoc mode. All nodes have an ad hoc routing protocol installed that establishes and maintains paths between nodes. Currently, the Ad hoc On-demand Distance Vector (AODV) protocol [11] is deployed on the testbed nodes, but any ad hoc routing protocol implementation developed for Linux can be used.

Design, deployment, administration, and operation of a mesh network are challenging tasks. Typically, the mesh network design is dependent on the hardware/software platform and network deployment, as well as administration and operation restrictions/requirements. For mesh network design, there are hardware issues such as the selection of a hardware platform on which the designated testbed software must be able to execute and, at the same time, provide information needed for experiment analysis. Another issue related to mesh network design is the selection of the operational and administrative functions that should be supported, which in turn have an impact on the network architecture. For example, network monitoring should be performed non-intrusively, and network management operations, such as software deployment and remote command execution, should be robust. Both monitoring and management solutions need to handle the distributed nature of the mesh network. Another important factor that affects the mesh network design and deployment is whether a wired backhaul network is available. Wired network connectivity may significantly affect node placement during network deployment, or alternatively, the absence of wired connectivity will affect the monitoring and management software architectures.

In this paper, we report on our effort and experience to design, deploy, administer, and operate our mesh network testbed. As a case study, we describe the UCSB MeshNet architecture, including its support services for *management*, *monitoring*, and *visualization*. We found these services to be vital to achieve efficient testbed operation and usage. The most important lesson learned is that the trade-offs related to platform selection, node deployment, and testbed software design are all dependent on each other and thus need to be jointly evaluated.

The remainder of this paper is organized as follows. In Section II we discuss the challenges of design, deployment and usage of a mesh testbed. We describe our UCSB MeshNet testbed in Section III and summarize the lessons learned in Section IV. Finally, we conclude the paper in Section V.

[1]`http://moment.cs.ucsb.edu/meshnet/`
[2]`http://openwrt.org/`

## II. Challenges and Experiences from Design, Deployment and Usage of the UCSB MeshNet

This section discusses the challenges and issues we experienced during construction and use of our mesh testbed. First, we discuss issues and trade-offs in platform selection and node deployment. Next, we describe the design challenges of the three support services that we found to be vital for efficient testbed operation: *management*, *monitoring* and *visualization*. In a mesh network, the design of these services poses challenges different from those in a wired network, mainly due to the mesh network's distributed, wireless, and multi-hop nature. Furthermore, their design is affected by platform selection and node deployment. Therefore, all trade-offs related to platform selection, node deployment, and testbed software design need to be jointly considered early in the planning of the testbed.

### *A. Platform Selection and Node Deployment*

Two important issues to consider early in the planning of a mesh testbed are hardware/software platform selection and node deployment. Node price typically increases with node capability. The number of nodes needed depends on the desired network topology as well as the characteristics of the testbed area. Since the testbed budget is typically limited, the node price and the number of nodes needed becomes the main trade-off. This section discusses these and other trade-offs.

*a) Selection of the Hardware and Software Platforms.* There are three primary factors that influence the selection of nodes: (i) node capability, (ii) planned number of nodes, and (iii) node price. Typically, there is a trade-off between desired node capability, planned number of nodes, and the testbed budget. Nodes with high performance and flexibility are usually desirable but are typically more expensive than less capable ones. Thus, the testbed designer must weigh the node capability against the planned number of nodes and the testbed budget available. In this section, we focus our discussion on node capability.

Node capability is important and should be part of the initial testbed design. The most important node capabilities to consider are: (i) *hardware characteristics*, (ii) *operating system*, and (iii) *configurable parameters*. The hardware characteristics are dependent on the node type. Typically, mesh nodes can consist of stationary PCs, laptops, and/or dedicated hardware such as ordinary WiFi access points modified to operate as mesh nodes[3]. The advantages of dedicated hardware are that it is cheap and that certain models are equipped with multiple radio technologies. On the other hand, dedicated hardware is hard to expand, e.g., increasing hard disk space or adding new radio hardware may not be possible. Other trade-offs with dedicated hardware are that it typically has less hard disk space and processing power. At a minimum, there must be enough hard disk space for the testbed software, such as the operating system, routing protocol, traffic generators, and testbed support system. By testbed support system, we mean the software to operate, manage, and monitor the testbed. Hard

[3]The Linksys WRT54G can be upgraded to run the OpenWRT operating system, which enables installation of new software and the ability to operate the access point as a mesh node.

disk space is also needed to log monitoring and experiment measurement data. Hard disk space constraints force solutions where measurement data must be transfered to stable storage frequently, even during experiment execution. Processing power limitations affect the design of the testbed support system, which should be designed to not significantly impact the node performance during experiments.

The hardware dictates the operating systems that can be supported. In turn, the operating system dictates the software programs that can be used. For example, a dedicated wireless access point may support installation of a Linux-based operating system. Although such an operating system may be similar to a Linux distribution intended for a PC, the software available for execution may still vary.

The ability to configure as many parameters as possible is desirable. For example, transmission power adjustment allows the modification of network connectivity and thereby easy network topology variation. However, the ability to adjust transmission power and the availability of WiFi information is dependent on the WiFi network interface card (NIC). A related issue is that different WiFi NICs may report information that is not directly comparable between NICs. For example, WiFi NIC vendors interpret and present signal strength differently. Therefore, special care must be taken if the testbed consists of nodes with heterogeneous WiFi hardware. However, testbed settings with apparently homogeneous WiFi hardware can also experience firmware diversity. This is due to the fact that WiFi NIC vendors may upgrade to newer firmware versions as well as completely change the chipset while still maintaining the same WiFi NIC model name. This firmware version diversity becomes an issue if the testbed network is to be expanded at a later point in time or if some nodes need to be replaced. This can be an issue at the initial purchase as well. Since the firmware revision number is only available on the NIC itself, retailers have no way to distinguish between different versions. Even a "bulk purchase" can contain WiFi cards with different firmware versions.

*b) Hardware Deployment.* The hardware deployment can be an additional factor in the node platform selection process. For example, the number of nodes needed can be influenced by requirements to cover a specific geographic area, or to support specific network topologies. Therefore, *a priori* knowledge about the testbed area is important. The planning of the physical placement of testbed hardware is commonly a trade-off between the following three factors: (i) *physical access to testbed nodes*, (ii) *availability of power outlets and network infrastructure*, and (iii) *planned network topology*.

Unless there is a dedicated testbed area available, the physical area is likely to have many constraints. In an indoor testbed, the building construction significantly affects the nodes' transmission range. Another potential problem is that nodes may breakdown, need to be replaced, rebooted, or upgraded. Therefore, while many other operations on testbed nodes can be realized via remote access, easy physical access to the nodes is desirable. Remote access can be realized via the testbed network itself, but is dependent upon the network's operational status. The availability of a pre-existing network infrastructure enables out-of-band remote access. Another important factor to consider is that nodes require access to power outlets.

The requirements described above are likely to put severe limitations on node placement. In addition to these requirements, node placement is further impacted by the desired network topologies. There are important benefits to having redundant network connectivity and topology flexibility in the testbed. For example, in a well-connected testbed, multiple topologies can be created by adjusting the output transmission power or turning on or off nodes' network interfaces.

## B. Testbed Support System Design

A good system to support the operation and management of the testbed is important. We found that efficient and easy management, automatic monitoring, and node status visualization are the most important components of such a system. The main challenge is to design these components such that they are distributed, efficient, and have minimal impact on testbed experiments. In this section, we discuss these three components and their design challenges.

*a) Management.* A good management system is crucial for efficient usage of the testbed. The management system's design is dependent on the properties it needs in order to support the desired management operations. In this section, we discuss the desired operations and the properties they require.

The primary operations that the management system should support are: (i) *node configuration*, (ii) *software deployment*, (iii) *software execution*, and (iv) *fault recovery*. First, all nodes must have the correct configuration prior to each new experiment. This requirement includes configuration of traffic generators and monitoring/logging software, as well as varying transmission power, and activation/deactivation of network interfaces in order to control the network topology. Without an automated tool for configuration, test repeatability is jeopardized by the "human factor". Topology control is an essential part of the configuration. Since wireless links vary over time and nodes may malfunction, a system must be able to retrieve historic data on network connectivity as well as current connectivity status. An automated tool for software deployment reduces the deployment time and ensures that all nodes run the correct software version. Software execution is needed to perform configuration tasks as well as for executing programs, such as traffic generators and monitoring tools. If the testbed consists of heterogeneous nodes, or if nodes should be treated differently, the management tool should support node differentiation. Significant time can be saved if the management system is able to easily recover from node faults and bring nodes back to a "good state". However, some faults cannot be handled remotely, e.g., if a hardware failure occurs or if software causes a system crash. Additionally, a management system can be designed to handle starting/stopping experiment execution. A related issue is node time synchronization, which might be needed for the simultaneous commencement of an experiment on multiple nodes and for analysis of time-sensitive data.

In order to support a management operation, the system should have the following properties: (i) *remote operation*,

(ii) *in-band and out-of-band operation*, (iii) *a scalable architecture*, and (iv) *a flexible, extensible, and reliable communication protocol.* Remote operation enables a testbed user to perform management operations without physical node access. For remote management operation, out-of-band communication is desirable in order to not rely upon the testbed network's status. Furthermore, out-of-band communication minimizes interference with the testbed network. However, this requires the nodes to have an additional network interface and access to an alternate network infrastructure. In-band operation enables performance of the management operation regardless of these requirements, but should be designed to have minimal impact on the testbed network. A scalable management architecture enables testbed growth without significantly increasing operation and communication complexity, or decreasing operation efficiency. The communication protocol has to be flexible to handle a multitude of different management tasks. Since new management tasks may evolve over time, the management protocol should be extensible and allow for new functionality to be added without significant updates or changes to the existing management system. Many management tasks require reliable execution, i.e., the testbed user must know whether or not an operation was successfully completed, and if so, be certain that it was completed on all the requested nodes.

In summary, a management system requires remote access to all testbed nodes to perform its operations. The communication between the management system and the testbed nodes, and thus part of the management system's design, depends on whether out-of-band communication is possible. In turn, this depends on the node platform and node deployment. The trade-offs with in-band management are: greater impact on the testbed network, the need for a routing protocol, and the reliance upon a continuously sufficient degree of connectivity between testbed nodes.

*b) Monitoring.* A monitoring system is essential in a mesh testbed. Its goal is to collect and provide the testbed user with the data needed to analyze node status and test results. In this section, we discuss the essential operations and their impact on the monitoring system's design.

The monitoring system's primary operations are: (i) *(continuous) node status monitoring*, (ii) *experiment measurement data collection*, and (iii) *data transfer to stable storage.* Node status monitoring should include configuration and network status, such as link quality, network topology, and available link bandwidth. The collection of this information also relates to the management system, which needs to know the node status to perform node configuration and node fault recovery. Continuous monitoring over time gives information about parameter variability over both short and long time periods. Parameter variability over time, such as link quality, is important when planning which nodes and network topologies to use for experiments. The monitoring system should also support experiment measurement data collection. The difference between node status monitoring and experiment measurement data collection is that the latter typically includes collecting a large amount of data traffic. Therefore, experiment measurement data collection typically requires a modified or separate monitoring operation. However, leveraging the monitoring system's architecture for data collection is typically beneficial. The purpose of both types of data collection is to be able to analyze node status and events in the network. All collected data, therefore, needs to be transfered from testbed nodes to stable storage. Analysis tools can then access the stored data and process it.

To support the operations discussed above, the monitoring system needs to have the following properties: (i) *minimal impact on the testbed network*, (ii) *in-band and out-of-band data transfers*, and (iii) *a scalable architecture*. Additionally, from a testbed user's perspective, the following property is desirable: (iv) *availability of monitoring data in near real-time*. The impact of the monitoring tools on the testbed network should be kept to a minimum. High CPU processing and bandwidth consumption will impact the testbed network such that results may be misleading. Therefore, monitoring tools should strive toward minimal processing, bandwidth, and storage requirements. An experiment measurement data collection system, however, has different requirements compared to continuous status monitoring. Typically, the amount of data collected is larger and the time-granularity is smaller for experiment data collection. As a result, either a larger amount of node storage space, or a higher frequency of stable storage data transfer, is required. The storage requirements for any experiment of arbitrary length and with arbitrary data traffic intensity typically cannot be guaranteed to be fulfilled. A separate backhaul network for out-of-band data transfer allows for more frequent and larger data transfers with a minimal impact on the testbed network. However, the availability of a backhaul network cannot be guaranteed in a distributed mesh testbed. Therefore, the complementary option of in-band data transfer is essential [12]. A scalable architecture enables the testbed to grow and still support all desired operations, without increasing the impact on the testbed network. Supporting both in-band and out-of-band data transfers is one part of a scalable architecture since it relaxes the need for backhaul network availability to all testbed nodes. However, in-band and out-of-band data transfer solutions must still be scalable *per se*. From a testbed user's perspective, node status information should be available in real-time. Typically, frequent transfers of small amounts of data have little impact on CPU processing and bandwidth consumption. Ideally, experiment measurement data should also be available in near real-time. However, delivering this data may result in higher CPU processing load and bandwidth consumption.

In summary, the trade-offs to consider for monitoring system design clearly relate to platform selection and node deployment. One trade-off is the amount of data collected versus the device storage capacity and the data transfer frequency. In turn, the amount and frequency of data transfer has to be considered against bandwidth consumption and testbed impact. Another trade-off is the impact of CPU processing for the monitoring system versus the processing capacity of the testbed node.

*c) Visualization.* The goal of the visualization system is to help the testbed user analyze node status and events in the testbed by graphically present monitoring data.

The visualization system's primary operations are: (i) *monitoring data retrieval*, (ii) *data processing*, and (iii) *data presentation*. The monitoring system transfers monitoring data to a repository. The visualization system, in turn, accesses the repository and processes the data. Finally, the visualization system graphically presents the processed data to the testbed user.

The primary system properties for the visualization system include: (i) *real-time and historical data support*, (ii) *intuitive presentation of results*, (iii) *scalable and extensible architecture*, and (iv) *minimal impact on the testbed network*. The visualization system should support processing and presentation of both real-time and historical data. Visualization of the current status requires that monitoring data is available in near real-time. The update frequency of status information depends on the monitoring system's frequency of data transfer to the repository. Historical data may be retrieved in a manner similar to real-time data by simply changing the time span for which data is to be retrieved. However, historical data may require additional data processing, e.g., to provide statistics for a range of data points. As a result, the presentation of historical data may differ from that of real-time data. Regardless of the type of data to present, the key property of graphical data presentation is to be intuitive. Ultimately, the visualization tool should be more time-efficient than manually parsing raw monitoring data, and more intuitive and easier to review than a text-based alternative. The design and realization of such a presentation tool is an aesthetic challenge. Another design challenge is to make the system architecture both scalable and extensible. It should be scalable in terms of simultaneous accesses to the system, as well as in terms of the number of nodes and the information displayed by the tool. An extensible architecture enables new display information to be added, e.g., experiment-specific data. The final property is that the system should have minimal impact on the testbed network. Typically, the visualization system operates separately from the testbed since neither data retrieval nor visualization include any testbed nodes. However, the data type and availability requirements may, in turn, influence the monitoring system's impact on the testbed network.

In summary, the visualization system relies on a monitoring system to provide data to be displayed. The monitoring system, in turn, may impact the testbed network, depending on the requirements of the visualization system. Another trade-off is how the data processing is divided between the monitoring system and the visualization system. The monitoring system should not process data such that any (statistical) information is lost. On the other hand, data pre-processing decreases the load on the visualization system.

## III. Case Study: The UCSB MeshNet Testbed

The previous section discussed the design challenges for a mesh network testbed, many of which we experienced during the construction of our own testbed. In this section, we provide a case study about the UCSB MeshNet testbed, including the operation of its support services. Figure 2 shows an overview of the testbed architecture. The three primary components are: (i) the management component (*MeshMan*) for node configuration, software deployment, remote software execution, and testbed access control; (ii) the monitoring component (*MeshMon*) that periodically collects node/network status, assists in experiment data collection during tests, and transfers monitoring and experiment data to stable storage (a database); and (iii) the visualization component (*MeshViz*) that displays current network/node status by using the information collected by the monitoring component. The three components were developed such that they can use, but do not rely on, any pre-existing network infrastructure. We describe the testbed platform and deployment, followed by a description of each of the three primary components.

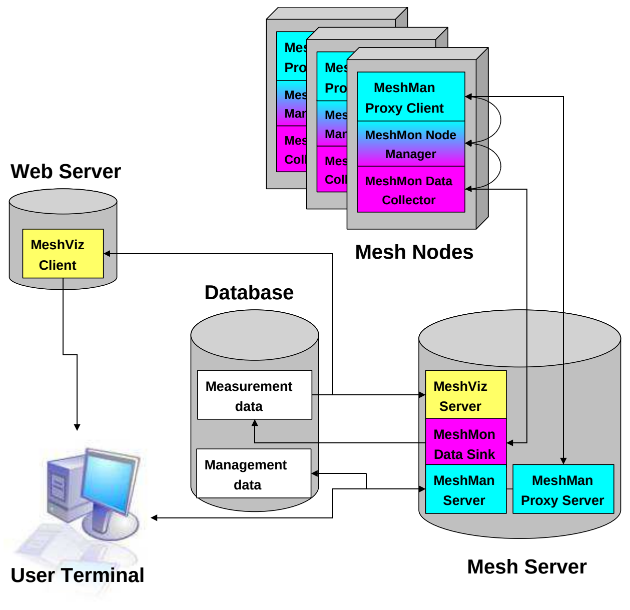

Fig. 2. Overview of the UCSB MeshNet architecture showing the logical placement and the out-of-band communication paths for the MeshMan, MeshMon, and MeshViz components.

### A. Testbed Platform and Deployment

The testbed consists of 30 IEEE 802.11a/b/g nodes. Currently, there are two types of nodes. Half of the nodes are IEEE 802.11b/g Linksys WRT54G wireless routers. The other half of the nodes are small form-factor Linux PCs, equipped with multiple IEEE 802.11a/b/g PCMCIA cards. The Linksys routers have been upgraded to run the Linux-based OpenWRT operating system, which enables them to run in ad hoc mode. Since both types of nodes are running Linux-based operating systems, minimal (or no) software modification is needed for cross-platform sharing. To create and maintain multi-hop paths between nodes, each node runs an ad hoc routing protocol. The use of an ad hoc routing protocol, instead of static routing, enables the network to be easily extended to include new nodes, support changes in node participation, and include mobile nodes. To this end, we have successfully used the Kernel-AODV[4] implementation. However, any Linux-based ad

[4]`http://w3.antd.nist.gov/wctg/aodv_kernel/`

hoc routing protocol implementation can be used on the PCs or cross-compiled for OpenWRT on the Linksys routers.

The trade-off between the Linksys routers and the PCs is that the former have lower purchase cost, but less memory and processing power. The available flash memory on our Linksys routers is only 4MB, but this is still large enough to fit the OpenWRT system and our own testbed software, including megabyte-size log files. Since our testbed software system is not CPU-intensive, the Linksys routers' lower processing power has minimal impact on performance.

The testbed nodes are distributed indoors throughout five floors in the Engineering I building on the UCSB campus. The node density is deliberately high to facilitate multiple possible paths between nodes. The environment consists primarily of offices and research labs. Each node is equipped with a management network interface and is positioned such that it can connect to a pre-existing network infrastructure. The pre-existing network infrastructure acts as a backhaul network that enables communication with the nodes without introducing additional data traffic in the testbed network. The backhaul network is primarily used by the management and monitoring components of our testbed. The backhaul network for a majority of the nodes consists of wired connections to existing subnets. However, some nodes are currently located in areas where wired connectivity is not available. The backhaul network for these nodes consists of a wireless connection to an existing wireless LAN. One advantage for the use of wired connections for the backhaul network is the minimal delay variations between nodes, especially when compared to the wireless links. This is particularly important when running a time synchronization protocol to synchronize the nodes. Time synchronization between nodes is needed to avoid clock skew and for calculation of metrics such as packet transmission delay. We are currently using the Network Time Protocol (NTP) to synchronize our testbed nodes.

Because many nodes are in locations where physical access is restricted, we have installed remote reboot devices to control the power supply to the testbed nodes. In the event that a node hangs or becomes otherwise inaccessible, switching off/on a node results in the node returning to a fully operational state. This component has proven invaluable during new software installation and debugging.

## B. *MeshMan*

The goal of our management system, MeshMan, is to simplify the steps involved in daily testbed operation and experiment execution to the largest extent possible. Figure 3 illustrates the MeshMan architecture. MeshMan consists of two main components. The first is the *MeshMan Server*, which resides on a central server and from which all the management operations are executed. The second is the *MeshMan Proxy*, which resides in part on the MeshMan server (proxy server) and in part on each testbed node (proxy client). The MeshMan Proxy implements all functionality for handling requests from the management server regarding node configuration, software deployment, and software execution. In this section, we discuss these two components and their operation in more detail.

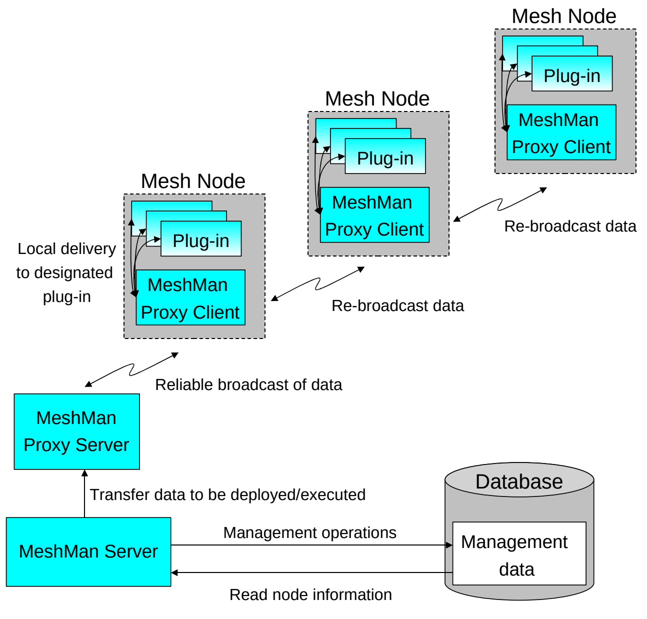

Fig. 3. The MeshMan architecture. As illustrated, the MeshMan Proxy Server communicates with the MeshMan Proxy Client through reliable broadcast in the in-band mode of operation. In the out-of-band operation, the reliable broadcast is replaced with TCP communication over a wired backhaul network.

*a) MeshMan Server.* MeshMan allows network administrators to manage the testbed through a single, simple interface. The primary supported operations are node configuration, software deployment, remote software execution, and group ID assignment. To prevent testbed users from using nodes or performing tests that interfere with any test currently being executed, the management system can force access control for specific nodes or the whole testbed. This access control is a crucial first step toward enabling remote testbed access.

With group ID assignment, nodes can be configured to belong to one or more groups. This enables operations, such as configuration, updates, and access control, on a selection of nodes. For example, if software should only be deployed on specific nodes, placing these nodes into the same group enables the deployment of the software to these nodes in a single operation.

*b) MeshMan Proxy.* The goal of the MeshMan Proxy is to have a transparent and efficient way to transfer data and deploy software to all testbed nodes without knowledge of the current network topology. The MeshMan Proxy can operate both in-band and out-of-band. Figure 3 shows the architecture and data propagation through the testbed network using the in-band solution. However, as we discuss later in this section, out-of-band management is also supported.

The MeshMan Proxy consists of a *proxy server* running on the MeshMan node and *proxy clients* on each testbed node. The proxy client on each node listens for calls from the proxy server. Upon reception of data packets from the proxy server, the proxy client performs two actions. First, if the node has not seen this packet before, it re-broadcasts the packet. Second, the node checks to see whether it belongs to the group of nodes for which this packet was intended. If so, it stores the packet until it has received all the packets that are a part of this specific transmission. Once all packets have been received, the proxy client defragments the received data and transfers it to the local application for which this data was intended (this information is provided in the packet header). If all the data was successfully received and delivered to the local application, the proxy client sends an acknowledgment to the proxy server indicating success. In the out-of-band solution, the MeshMan proxy server communicates with each testbed node individually and directly, through the backhaul network.

For flexibility and extensibility, each testbed node has a range of ports to which applications can listen. The data to be pushed into the network is tagged with information about to which group of nodes this data is intended and to which receiver-side application this data should be delivered. The proxy server then fragments the data and forwards each piece to the proxy clients listening on each node. When in-band mode is used, the data forwarding is based on reliable broadcast. The proxy server waits for acknowledgments from all intended receivers to ensure that they receive the data. If any data delivery fails, the proxy server can be configured to try to deliver the data through any alternative delivery options available. The software deployment can be initiated from a remote machine since the data to be disseminated do not have to reside on the MeshMan proxy server.

The approach of the proxy system handling reliable data transfer and the proxy clients handling data reception, defragmentation, and delivery to the application has several benefits. These benefits include the following: (i) the receiver-side applications can be developed more easily since they do not have to deal with reliable data transfer and fragmentation; (ii) receiver-side applications can be easily incorporated by listening on a specific port; (iii) receiver-side applications can be tailor-made to perform certain actions, such as software installation and command execution; (iv) the reliable broadcast data dissemination allows network communication to occur in mesh networks where no backhaul network is available; and (v) any communication between the proxy and proxy receivers occurs over UDP and thus implements best-effort reliability. Using UDP avoids the TCP congestion control issue in wireless networks. Furthermore, depending on the network topology, it may be more efficient to utilize broadcast communication rather than to set up one data flow per destination, as required by TCP.

## *C. MeshMon*

The goal of MeshMon is to collect node and network status information, and transfer this data to stable storage for analysis. The MeshMon component periodically collects status data from each individual testbed node. The collected data is transfered to a database, which in turn can be queried by the MeshMan and MeshViz components. MeshMon can easily be extended to support collection of experiment measurement data. However, this is currently not implemented in our testbed.

Figure 4 shows the MeshMon architecture and its primary components. The MeshMon architecture is divided into three separate components that interact with each other. The first is the *Data Collector*, which resides on each testbed node and is responsible for aggregating monitoring data and transferring it to the Data Sink. The second is the *Data Sink*, which resides on the central testbed server and is the sink for all aggregated monitoring data that arrives from the testbed nodes. The Data Sink is responsible for the transfer of collected data to stable storage. The third is the *Node Manager*, which resides on each testbed node and is responsible for managing the monitoring tools on each testbed node. The Node Manager is an example of a MeshMan plug-in. Figure 4 illustrates the MeshMon architecture for out-of-band operation using a wired backhaul network. For the in-band operation of MeshMon, a routing protocol is required to communicate to or from the testbed nodes. We now describe each of the three components in more detail.

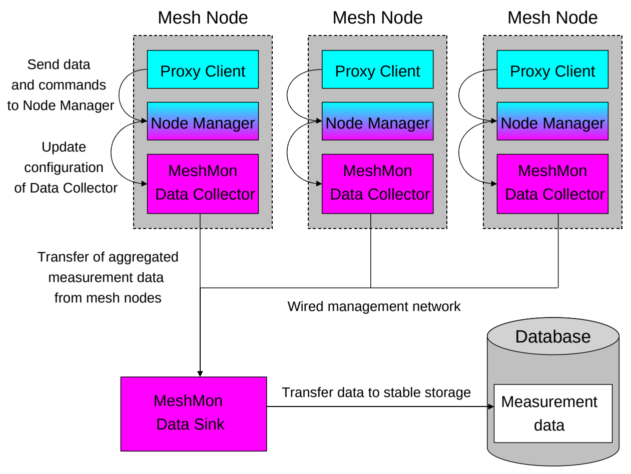
Mesh Node
Mesh Node
Mesh Node
Proxy Client
Node Manager
MeshMon Data Collector
Send data and commands to Node Manager
Update configuration of Data Collector
Transfer of aggregated measurement data from mesh nodes
Wired management network
MeshMon Data Sink
Transfer data to stable storage
Database
Measurement data

Fig. 4. The MeshMon architecture showing the communication between the three components: Node Manager, MeshMon Data Collector, and MeshMon Data Sink. The Proxy Client is the MeshMan component, which here is used to remotely send instructions or data to the Node Manager plug-in.

*a) MeshMon Data Collector.* The Data Collector runs on each testbed node and is responsible for gathering current node and link status as well as measurement data. The Data Collector gathers information at periodic intervals, thereby providing a view of the wireless settings and the node and network status over time. Information about the wireless channel, such as the network BSSID, network ESSID, radio frequency, and noise level is passively collected through existing tools included in the operating system. Node and network status, such as neighbors, routes, and link qualities, are collected through both routing protocol information and active probing of the network. One challenge with active probing is to limit the overhead such that it does not significantly affect overall network performance. The data collection periodicity and the collected metrics can easily be varied through the MeshMan interface. Once the data is aggregated, it is sent to the Data Sink. The data can be transfered either through a management network or through the mesh network itself. In order to not interfere with the mesh network, a separate management network is preferable.

*b) MeshMon Data Sink.* The Data Sink exists on a central server connected to the mesh network. Its purpose is to receive data sent by the Data Collector on each testbed node and transfer the data to stable storage. The Data Sink

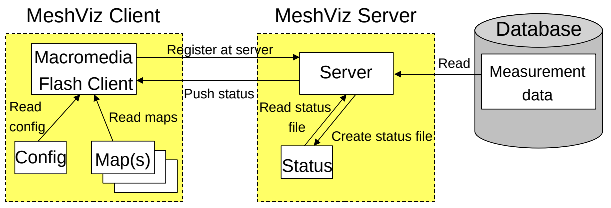

Fig. 5. The MeshViz architecture illustrating the logical division of functionality between the MeshViz Client and the MeshViz Server.

waits for connections from the Data Collector and, once it has received data from all nodes, stores the data in an SQL database. In the case of in-band operation, the Data Sink should preferably be located on one of the mesh nodes that has gateway functionality. Then, the Data Sink has connectivity with both the Data Collector and the database.

*c) Node Manager.* The Node Manager is an application for remote management of monitoring tools and node settings. It is located on each testbed node and is controlled via the MeshMan system. The monitoring tools can be started, stopped, and reconfigured. Additionally, new monitoring tools can be deployed through this interface. The remote management of the monitoring tools is crucial since the ability of a testbed user to physically service each node may not be possible as the number of testbed nodes increases. The Node Manager allows for a testbed user to query for additional node or link status information, e.g., if a particular node is behaving anomalously.

### *D. MeshViz*

The MeshViz Module is a display tool for the mesh network. It enables visualization of the current state of the network. Such a view includes all network metrics collected by MeshMon and allows the investigation of metric variations over time without the need to manually parse collected data. Figure 5 shows the MeshViz architecture. The MeshViz consists of two components: the *MeshViz Client*, which is the display tool, and the *MeshViz Server*, which pushes information updates to MeshViz Clients.

*a) MeshViz Client.* The MeshViz client resides on a web server and consists of a Flash player, an XML-based configuration file, and a collection of JPEG-based maps. The configuration file specifies the initial setup of the flash player and the information to be displayed. The display includes information such as the nodes, their IP addresses, network maps, and node positions. The MeshViz client can alternatively be downloaded to the end user's desktop and run locally. The configuration file and the maps can then be edited to provide a customized view of the network. The MeshViz client periodically connects to the MeshViz server to obtain status updates.

*b) MeshViz Server.* The MeshViz server accepts connections from the MeshViz clients. The database that contains the network status information, collected by MeshMon, is periodically queried for status information. This information is formatted into an XML file, which is then propagated to the MeshViz clients for display. The status information pushed to the MeshViz clients consists of incremental updates. The use of these small updates decreases bandwidth consumption and client processing. Additionally, there is no need to change or rebuild the tool when a new node is installed or a change to an existing node occurs.

## IV. Lessons Learned

In this section we share the most important lessons learned from construction and usage of our testbed. The lessons learned range from unexpected problems to testbed operation efficiency.

*a) Platform selection, node deployment, and testbed software design depend heavily on each other.* The number of planned nodes may depend on the desired testbed area and network topologies. There is also a trade-off between the number of nodes and the node price. In turn, the node price typically increases with node capability. Furthermore, the testbed software design is influenced by the node capability and the availability of a backhaul network for out-of-band communication. In turn, the latter depends on the testbed area and may affect the placement of nodes during deployment. Since all these pieces relate to each other, we recommend the joint evaluation of all trade-offs early in the testbed planning phase.

*b) Initially, 70-80% of the time is spent solving hardware or software related problems.* We have found that in the early stages of the testbed usage a considerable amount of the time is spent investigating various unforeseen problems related to either hardware or software. The best way to reduce this period is through experience sharing and careful planning and selection of the hardware and software platforms.

*c) Documentation of the testbed and its operation is invaluable.* The testbed will evolve over time, so tracking all changes and updates of the testbed, as well as documenting any pending issues and methods of circumvention, is vital. Documentation of the testbed and its operation reduces the need to have the testbed administrator available when someone runs experiments or uses the testbed. We recommend the use of a web-based documentation system, such that all testbed users can continuously both add and acquire testbed information.

*d) A test and measurement phase should follow testbed deployment.* There are likely several unknown environmental and time-dependent factors that impact testbed operation and its performance. Therefore, we recommend testbed deployment to be followed by a “testing phase”, where the testbed operation and performance are closely monitored for an extensive period of time to reveal unexpected changes and instability.

*e) Link quality monitoring is crucial.* Link quality monitoring is important for two main reasons. First, when selecting the nodes and network topologies to be used in an experiment, consideration of both current and historic link information to estimate the quality and stability of the links is important. Second, when analyzing test results, comparison of link conditions between test runs is necessary to determine test repeatability. For efficient acquisition and interpretation of link quality information, we recommend that the testbed include monitoring and visualization systems.

*f) Seemingly uninteresting data may become interesting.* Although we consider link quality to be the primary network parameter to be monitored, we recommend recording all possible information as long as the data collection does not impact the test results. Information that at first may seem irrelevant or uninteresting can be painfully discovered to be crucial in the analysis phase. For example, although an experiment may focus on examining network or transport layer functionality/performance, MAC layer information may become necessary to fully understand and explain the test results.

*g) Optimization and automation of experiment execution will save significant time and improve repeatability.* A considerable percentage of the total time of a set of experiments is commonly related to parameter exploration, dry-runs, and tuning. However, during "production experiments", significant time can be saved through automation and optimization of as many steps as possible. Most time-consuming is typically the operations that require manual involvement. Minimizing manual involvement also improves test repeatability. Hence, we advise the automation of all possible operations in the process of test optimization. This automation involves tasks such as testbed configuration, scheduling of experiments, measurement data collection, and transfer of measurement data to stable storage.

*h) Remote node reboot capability saves significant time.* Testbed nodes may break down or become unaccessible for various reasons such as network interface overload, OS thrashing, and storage space shortage. During early testbed stages and new software development, there is an increased risk of node faults. We recommend the use of a remote node reboot system that does not require physical or network access to the node. Such a system saves considerable time, especially in a testbed where physical node access is restricted.

*i) The "BSSID problem" may partition your network.* If a node loses contact with all other nodes in the network, it will search for other nearby networks. As a result of the search, it may join a nearby network or start its own network. In either case, it will change its BSSID. Joining another network may result in a change of channel; it can also start its own network using the same channel. Because of the BSSID change, the node may not re-join the old network even if it regains contact with it. Hence the network becomes partitioned. Currently, there is no known clean solution to this problem. If the testbed experiences BSSID partitioning, we recommend the operation of WiFi cards in the "pseudo-IBSS" mode, which circumvents the problem by omitting the IEEE 802.11 beacon and BSSID mechanism. However, pseduo-IBSS is a non-standard, proprietary ad hoc mode and is not be available in all WiFi chip-sets and network drivers.

*j) The reported WiFi data rate can be confusing.* If the WiFi data rate is checked using the `iwconfig` Linux utility directly after an explicit data rate change, `iwconfig` may not return the newly selected data rate. We have found that the reported data rate is not updated until there has been at least one packet sent over the WiFi interface. However, we have also observed that the reported data rate may be different from the data rate at which the NIC operates, even though packets have been sent over the interface after the data rate has been updated. Therefore, we recommend performing experiments to verify the operation of the WiFi hardware and driver with respect to data rate reporting and data rate changes.

## V. SUMMARY

Wireless mesh networking is a promising technology for ubiquitous wireless network access. To increase the understanding of real world factors in these networks and to address the limitations of current simulators, the research community has started to conduct more experimental studies. Construction of a wireless mesh testbed is a challenging task. Issues range from platform selection and node deployment, to testbed design for efficient and useful operation. In this paper, we share our experiences gained from the design and construction of the UCSB MeshNet testbed. We discuss design challenges with management, monitoring, and visualization systems in a mesh testbed, and describe how such systems are designed in the UCSB MeshNet. Our lessons learned include unexpected problems, such as network partitioning, as well as testbed operation efficiency improvements, such as remote node reboot. The most important lesson learned is that the choices made regarding either platform selection, node deployment, or testbed software design typically have a significant impact on the requirements and/or the design of the other two parts.

Only a handful of mesh network testbeds currently exist and experimental mesh network research is still in its infancy. We expect to see the deployment of more mesh network testbeds in the near future. We hope that our experiences shared in this paper increase the research community's understanding of mesh testbed construction challenges and lead to the construction of sustainable and more advanced mesh network testbeds in the future.

## ACKNOWLEDGMENTS

The authors wish to thank Rocio Garcia Ruiz for her work on the mesh database, and Ian Chakeres, Stefan Karpinski, Prashanth Aravinda Kumar, Kimaya Sanzgiri, Irfan Sheriff and Yuan Sun for their input to this paper.

Funding for this work is through the NSF Network Research Testbeds (NRT) grant ANI-0335302 as part of the WHYNET project, as well as NSF NeTS Award CNS-0435527 and NSF Career Award CNS-0347886.

## References

[1] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002.

[2] M. Takai, J. Martin, and R. Bagrodia, “Effects of wireless physical layer modeling in mobile ad hoc networks,” in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Long Beach, CA, USA, October 2001.

[3] H. Lundgren, E. Nordström, and C. Tschudin, “Coping with communication gray zones in IEEE 802.11b based ad hoc networks,” in *Proceedings of the 5th ACM International Workshop On Wireless Mobile Multimedia (WoWMoM)*, Atlanta, GA, USA, September 2002.

[4] Y. Zhang and W. Li, “An integrated environment for testing mobile ad-hoc networks,” in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, Switzerland, June 2002.

[5] 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 IEEE Wireless Communications and Networking Conference (WCNC)*, New Orleans, LA, USA, March 2005.

[6] D. Maltz, J. Broch, and D. Johnson, “Quantitative lessons from a full-scale multi-hop ad hoc network testbed,” in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, Chicago, IL, USA, September 2000.

[7] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordström, and C. Tschudin, “A large-scale testbed for reproducible ad hoc protocol evaluations,” in *Proceedings of IEEE Wireless Communications and Networking Conference 2002 (WCNC)*, Orlando, FL, USA, March 2002.

[8] S. Jadhav, T. X. Brown, S. Doshi, D. Henkel, and R. George, “Lessons learned constructing a wireless ad hoc network test bed,” in *Proceedings of 1st Workshop on Wireless Network Measurements (WiNMee)*, Trentino, Italy, April 2005.

[9] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, “Link-level measurements from an 802.11b mesh network,” in *Proceedings of the ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, Portland, OR, USA, August 2004.

[10] R. Draves, J. Padhye, and B. Zill, “Comparison of routing metrics for static multi-hop wireless networks,” in *Proceedings of the ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, Portland, OR, USA, August 2004.

[11] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” IETF RFC 3561, July 2003.

[12] K. Ramachandran, E. Belding-Royer, and K. Almeroth, “DAMON: A distributed architecture for monitoring multi-hop mobile networks,” in *Proceedings of IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, Santa Clara, CA, USA, October 2004.