

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA



Wireless Mesh Network Implementation

By: Karishma Babu
Luis Miguel Cortés-Peña
Prateek Shah
Shivaranjani Sankara Krishnan
ECE8823: Mobile Computing
Fall 2007
Dr. Blough

Abstract

The report details implementation of the infrastructure based wireless mesh networks, certain existing routing strategies and performance analysis of different topologies. The backbone of the network consists of a gateway, an access point, two routers and a client. The bandwidth obtained at these different nodes for different topologies were extensively studied and the phenomenon of bandwidth loss due to multi-hopping was re-validated through experimental verification. Different popular routing schemes like OLSR and BATMAN were implemented and also a static routing scheme was developed for the purpose of experimental evaluation of the mesh network. Certain interesting results were obtained during the course of the project including the effect of placement of nodes of the entire topology and these have been presented in a detailed manner in the report.

1. Introduction

Wireless mesh networks (WMN) are self-organized wireless networks in which component parts (nodes) can all connect to each other via multiple hops. In WMNs, nodes are comprised of mesh routers and mesh clients. Each node operates not only as a host but also as a router, forwarding packets on behalf of other nodes that may not be within direct wireless transmission range of their destinations. Other than the routing capability for gateway/repeater functions as in a conventional wireless router, a wireless mesh router contains additional routing functions to support mesh networking. To further improve the flexibility of mesh networking, a mesh router is usually equipped with multiple wireless interfaces built on either the same or different wireless access technologies. Compared with a conventional wireless router, a wireless mesh router can achieve the same coverage with much lower transmission power through multi-hop communications. Optionally, the medium access control (MAC) protocol in a mesh router is enhanced with better scalability in a multi-hop mesh environment. With the capability of self-organization and self-configuration, WMNs can be deployed incrementally, one node at a time, as needed. As more nodes are installed, the reliability and connectivity for the users increase accordingly. The major applications of these networks include 'last-mile' Internet delivery, public safety, and distributed sensing. The principle here is similar to the way packets travel around the wired Internet, data will hop from one device to another until it reaches a given destination. Dynamic routing capabilities included in each device allow this to happen. To implement such dynamic routing capabilities, each device needs to communicate its routing information to every device it connects with, "almost in real time". Each device then determines what to do with the data it receives — either pass it on to the next device or keep it. The routing algorithm used should attempt to always ensure that the data takes the most appropriate (fastest) route to its destination.

There exist three main types of architectures of WMNs. These include Infrastructure/Backbone WMNs, Client WMNs and Hybrid WMNs. The infrastructure based WMNs is the most commonly used type. This approach provides backbone for conventional clients and enables integration of WMNs with existing wireless networks, through gateway/bridge functionalities in mesh routers. Conventional clients with

Ethernet interface can be connected to mesh routers via Ethernet links. For conventional clients with the same radio technologies as mesh routers, they can directly communicate with mesh routers. If different radio technologies are used, clients must communicate with the base stations that have Ethernet connections to mesh routers. Client meshing provides peer-to-peer networks among client devices. In this type of architecture, client nodes constitute the actual network to perform routing and configuration functionalities as well as providing end user applications to customers. Hence, a mesh router is not required for these types of networks. Hybrid WMN is the combination of infrastructure and client meshing. Mesh clients can access the network through mesh routers as well as directly meshing with other mesh clients.

For our study, we have implemented an infrastructure based wireless mesh network with the backbone consisting of a gateway, an access point and two routers. The two key challenges in an infrastructure based wireless mesh networks that was the primary focus of our study were the choice of appropriate routing algorithm and the choice appropriate topology of the backbone infrastructure. We have implemented and studied existing routing schemes including O.L.S.R and B.A.T.M.A.N. In addition we have also studied the performance of different topologies using static routing scheme.

The organization of our paper is as follows. Section 2 describes our experimental Setup. Section 3 details existing routing protocols that have been implemented by us. The static routing that has been implemented by us is described in Section 4. Results and analysis of those results are presented in Section 5. The conclusions drawn from this study are presented in Section 6.

2. Test Bed Setup

The team decided to use Ubuntu operating system for our test bed because of ease of package installation and dependency resolution. The test bed consists of a gateway (10.0.0.1), two router nodes (10.0.0.2 and 10.0.0.6), and an access point (10.0.0.100) which provide internet and network access to a client. The gateway has internet access through its Ethernet port, and provides internet access through its wireless interface running on Ad-Hoc mode. Figure below shows an example topology.

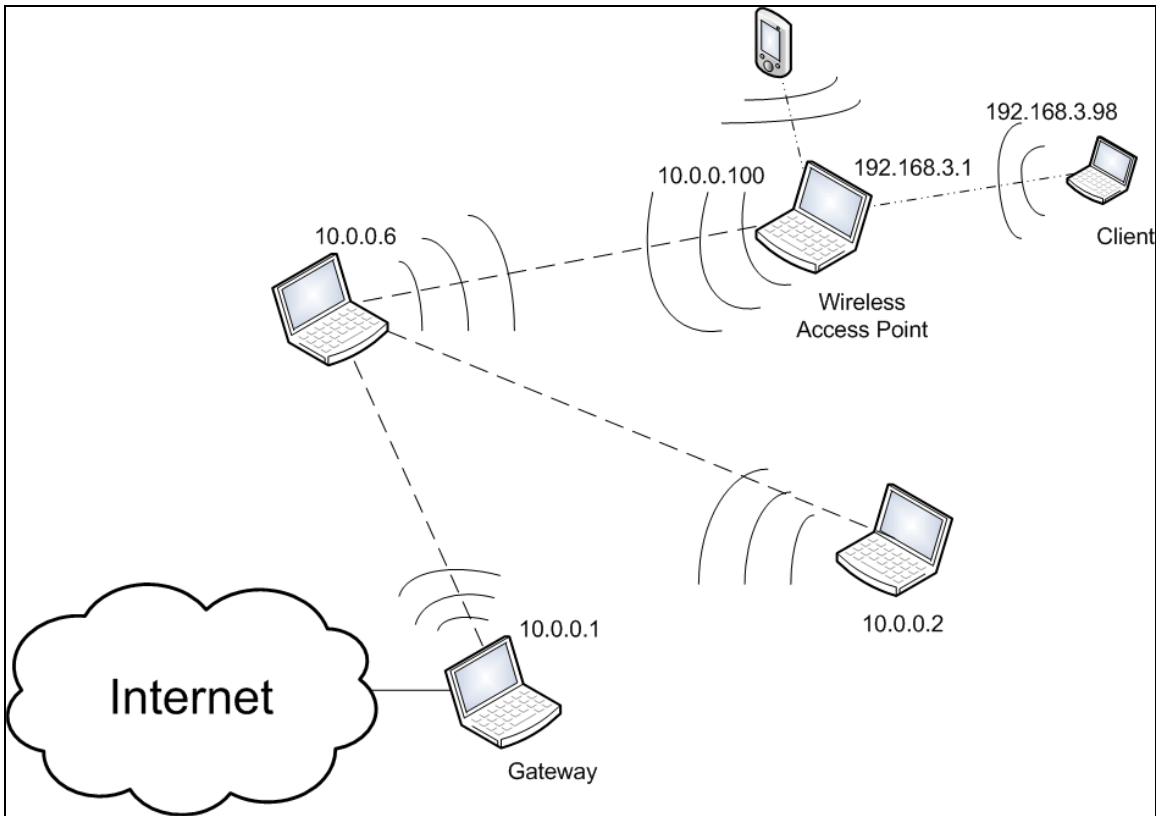


FIGURE 1: Example topology

The backbone of the wireless mesh network used 802.11a to minimize interference from other devices including Georgia Tech's wireless access points GTwireless. Another reason to use 802.11a is that a wireless mesh network implementation would take advantage of 802.11a's regulated frequencies and high maximum speed. The Access Point supports 802.11g for compatibility with most existing clients. The mesh network is transparent to the client. This is shown in the following figure.

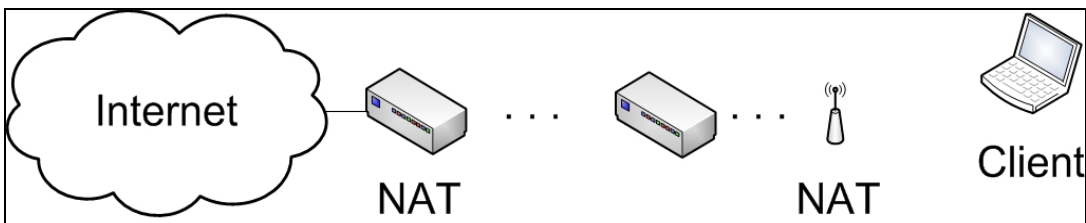


FIGURE 2: Mesh Network

The client connects to the Access Point which is seen as typical wireless access point with Network Address Translation (NAT). The data goes through some hops, then through another NAT, and finally the internet. The test bed can function with any routing

algorithm, as long as the routes are set in the kernel's routing table of each hop. The next sections describe the hardware used, and then proceeds to explain how to setup Ubuntu, the gateway, and the access point.

a. Hardware Used

The hardware used consists of:

- Four Dell Inspiron 1100.
- Five Cisco Systems [AIR-CB21AG-A-K9](#) Wireless CardBus Adapter with 802.11a/b/g support, power control from 10dBm to 20dBm in 6 levels. Most importantly, this wireless card uses Atheros 5212 chipset supported by [madwifi drivers](#) which support Master mode (can be used as an Access Point).
- One Orinoco Gold wireless cardbus with 802.11b support. This wireless card is used by the client.
- One Sony VAIO PCG-FRV28 which has two PCMCIA/CardBus slots. One is used to connect to the mesh backbone in Ad-Hoc mode and the other running in Master mode to provide internet access to the client.

b. Ubuntu Setup

Ubuntu 6.06 LTS Desktop Edition was used as the operating system of the test bed. In order to for the display in the Dell Inspiron 1100 notebooks to operate correctly, its BIOS was updated to version A32. Since the Dell Inspiron 1100 notebooks did not have Windows, a [CD boot version was used](#) for updating the bios. After performing this, the setup was easily done by inserting the setup CD, selecting boot Ubuntu, double clicking on the INSTALL icon on the desktop and filling up the blanks.

After the installation, however, some basic packages required to build applications from source need to be installed. In order to obtain these packages and other packages needed in further steps, some repositories must be enabled. To enable them, uncomment all the lines with URLs from `/etc/apt/sources.list` by:

```
$ sudo vi /etc/apt/sources.list
```

The `sudo` command is Ubuntu's way of running a command in super user mode. After editing the file, an update is performed by performing the following commands:

```
$ sudo apt-get update
```

It is now possible to install the essential packets by;

```
$ sudo apt-get install build-essential
```

Now that the operating system is set, the network interfaces must be configured. The Cisco network card should have been recognized and mapped to `ath0` automatically as the [madwifi](#) drivers are included by default. The remaining is to modify the default configuration so that the wireless card is setup automatically. To accomplish this, the file `/etc/network/interfaces` was modified:

```
$ sudo vi /etc/network/interfaces
```

The configuration to *ath0* was replaced with the following:

```
auto ath0
iface ath0 inet static
wireless-essid Wi-Me
wireless-mode Ad-Hoc
wireless-channel 165
address 10.0.0.1
netmask 255.0.0.0
```

The IP was changed for each computer accordingly. After saving the file, the interfaces must be restarted:

```
$ sudo /etc/init.d/networking restart
```

If the changes did not take effect, the computer was restarted. Now that the OS and interfaces are set, the next step is to setup the gateway.

c. Gateway Setup

To setup the gateway, a [Linux router/gateway HOWTO](#) was used as a guide line. Essentially, IP forwarding was enabled:

```
$ sudo echo 1 > /proc/sys/net/ipv4/ip_forward
```

, and a modified version of *dhclient-exit-hooks* was added so that whenever the computer obtained an IP Address from a DHCP server through its Ethernet port, it sets up iptables with IP Masquerading (NAT) to the inner network. The file can be found on Appendix A. No DNS or DHCP servers were needed.

d. Wireless Access Point Setup

To setup the access point, the */etc/network/interface* file was modified so that it includes the configuration of both *ath0* (the wireless access point interface) and *ath1* (the connection to the backbone of the wireless mesh network). The following is the configuration file:

```
# WiMeAP
auto ath0
iface ath0 inet static
wireless-essid WiMeAP
wireless-mode Master
address 192.168.3.1
netmask 255.255.255.0

#Wi-Me
auto ath1
```

```
iface ath1 inet static
wireless-essid Wi-Me
wireless-mode Ad-Hoc
wireless-channel 165
address 10.0.0.100
netmask 255.0.0.0
```

Since it was no possible to find a way to change the mode from 802.11a (default) to 802.11g in the interface file, the following extra commands must be set every time:

```
$ sudo iwpriv mode 3
$ sudo iwconfig ath0 channel 6
```

It is also needed to forward ip address and to have Masquerade to the client:

```
$ sudo echo 1 > /proc/sys/net/ipv4/ip_forward
$ sudo /sbin/iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -o ath1 -j MASQUERADE
```

To make the wireless access point fully functional for the client, a DNS and DHCP server is needed. For this, the guidelines from the [Linux router/gateway HOWTO](#) are helpful. To setup the DHCP server, simply:

```
$ sudo apt-get install dhcp3-server
```

and, edit the config file located at /etc/dhcp3/dhcpd.conf to something similar to the dhcpd.conf file in the directory named AP in the provided wi-me.tar.gz.

For Ubuntu, there is not chroot by default and, for security reasons, it is recommended for BIND9 (the DNS server). To accomplish this, the [Bind-Chroot-Howto \(Debian\)](#) was used. Setting up both is somewhat tedious, and therefore will be omitted. However, the configuration files used for the DNS server can be found on AP/bind directory in the provided wi-me.tar.gz.

The final step is to setup the 802.11 controller, hostapd. To do so:

```
$ sudo apt-get install hostapd
```

and modify the file /etc/hostapd/hostapd.conf similar to the one located in the AP directory. Now simply start all services:

```
$ sudo /etc/init.d/bind9 start
$ sudo /etc/init.d/dhcp3 start
$ sudo /etc/init.d/hostapd start
```

3. Existing Routing Protocols

Two popular routing protocols were chosen, the first is Optimized Link State Routing (OLSR) and the second is Better Approach To Mobile Ad-hoc Networking (BATMAN). These are discussed in the next sections.

a. Optimized Link State Routing protocol (OLSR)

OLSR is a link-state routing protocol, which along with distance-vector routing protocols comprises the two classes of routing protocols commonly used in computer networks. Link State Routing was invented by John McQuillan in 1978, and was later adapted for use in the Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF) protocols.

The problem with IS-IS and OSPF is that they both depend on the existence of a reliable network, thus rendering them unsuitable for wireless mesh networks. OLSR as a protocol is optimized specifically for ad-hoc networks.

Algorithm

In link-state routing, each node is responsible for calculating the optimal path to its destination for a data packet. This requires each node to maintain the topology of the network in full. To ensure that the topology is available to all the nodes, link-state protocols generally perform topology flooding, in which certain nodes are responsible for transmitting the current state of the network to all other nodes.

In previous protocols, this flooding was done by a designated router on every link. Since physical links do not exist (or are limited) in wireless networks and designated routers are impractical, OLSR instead elects a set of Multi-Point Distribution Relays (MPRs) that act as designated routers. MPRs are picked so that an MPR is at most two hops away from another MPR.

The process of link discovery begins with the transmission of “Hello” messages by each node that include the addresses of any neighboring nodes from whom “Hello” messages had been received previously. If a node receives a “Hello” message that lists it as a neighbor, then it marks a bi-directional link. This process allows each node to discover its one-hop as well as two-hop neighbors. Using this data, the node elects a set of one-hop neighbors with bi-directional links as MPRs.

Once the MPRs have been elected they are responsible for determining the state of the network topology and transmitting it to all the nodes in the network. Each node is thereafter responsible for computing the best route for a packet to its destination. In OLSR the best route is considered as that with the least number of hops. This creates a problem since link quality is ignored, and a single bad hop might be chosen over a two-hop good quality link.

OLSRd

OLSRd is an adaptation of OLSR that considers link quality while electing MPRs and calculating optimal routes. It introduces the concept of Expected Transmission Count (ETX) which is the expected number of transmissions needed to get a packet to reach its destination, and is basically the inverse of link quality for any given link. The ETX over several links is simply taken to be the sum of the ETXs for the individual links. The

optimal path is then not the one with the least number of hops but with the minimum ETX.

Installation

There is an Ubuntu package for OLSR named `olsrd` in the Universe repository. Ubuntu users can download this by using the command

```
$ sudo apt-get install olsrd
```

If the package does not show up, the user may have to enable the Universe repository. This can be done through the Control Panel.

However, since `olsrd` is under development and constantly being updated, users can ensure they have the latest version by downloading it directly from the `olsrd` website: <http://www.olsr.org/index.cgi?action=download>. Download and installation instructions are included.

Evaluation

A brief evaluation of OLSR revealed that much bandwidth is lost as compared to BATMAN because of the control traffic generated by the “Hello” messages and the topology flooding from the MPRs. By increasing the default time between these messages from 2s to 60s, the improvement in bandwidth was noticeable but the detection of new and broken links took proportionately more time. Thus there is a clear trade-off between the throughput and robustness of the network.

b. Better Approach To Mobile Ad-hoc Networking (BATMAN)

Introduction

Classical routing protocols are typically not well suited for wireless ad-hoc networks. This is because such networks are unstructured, dynamically change their topology, and are based on an inherently unreliable medium.

OLSR is currently the most employed protocol for such scenarios. However, due to the constant growth of existing community mesh networks and because of the inherent requirement of a link-state algorithm to recalculate the whole topology-graph, the limits of this algorithm have become a challenge. Recalculating the whole topology graph once in an actual mesh with 450 nodes takes several seconds on a small embedded CPU.

B.A.T.M.A.N. (Better Approach to Mobile Ad-Hoc Networking) is a new routing protocol for multi-hop ad-hoc mesh networks. B.A.T.M.A.N uses a simple and robust algorithm for establishing multi-hop routes in mobile ad-hoc networks. It ensures highly adaptive and loop-free routing while causing only low processing and traffic cost.

The approach of the B.A.T.M.A.N algorithm is to divide the knowledge about the best end- to-end paths between nodes in the mesh to all participating nodes. Each node perceives and maintains only the information about the best next hop towards all other nodes. Thereby the need for a global knowledge about local topology changes becomes unnecessary. Additionally, an event-based but timeless flooding mechanism prevents the accrument of contradicting topology information and limits the amount of topology messages flooding the mesh. The algorithm is designed to deal with networks that are based on unreliable links.

Algorithm

The protocol algorithm of B.A.T.M.A.N can be described as follows:

Each node transmits broadcast messages (originator messages or OGMs) to inform the neighboring nodes about it's existence. These neighbors re-broadcast the OGMs according to specific rules to inform their neighbors about the existence of the original initiator of this message and so on and so forth. Thus the network is flooded with originator messages.

OGMs are small, the typical raw packet size is 52 byte including IP and UDP overhead. OGMs contain at least the address of the originator, the address of the node transmitting the packet, a TTL and a sequence number. OGMs that follow a path where the quality of wireless links is poor or saturated will suffer from packetloss or delay on their way through the mesh. Therefore OGMs that travel on good routes will propagate faster and more reliable.

In order to tell if a OGM has been received once or more than once it contains a sequence number, given by the originator of the OGM. Each node re-broadcasts each received OGM at most once and only those received from the neighbor which has been identified as the currently best next hop (best ranking neighbor) towards the original initiator of the OGM. This way the OGMs are flooded selectively through the mesh and inform the receiving nodes about other node's existence.

A node X will learn about the existence of a node Y in the distance by receiving it's OGMs, when OGMs of node Y are rebroadcasted by it's single hop neighbors. If node X has more than one neighbor, it can tell by the number of originator messages it receives quicker and more reliable via one of its single hop neighbors, which neighbor it has to choose to send data to the distant node. The algorithm then selects this neighbor as the currently best next hop to the originator of the message and configures its routing table respectively.

A packet comes is:

- 1) If is a broadcast of my OGM: Mark Originator as Bi-directional Neighbor and drop it.
- 2) If OGM had Unidirectional Link Flag: Drop it

3) If received via Bidirectional Neighbor and is not a DUPLICATE: Update RANKING of Potential Neighbor to Originator (PNTOG)

4) Rebroadcast packet if

a. is DIRECT_NEIGHBOUR_IF and

i) is Bi-directional neighbor and Best neighbor to OG : Rebroadcast

ii) (is Bi-directional neighbor and not Best Neighbor To Originator) or (is Unidirectional Neighbor): Rebroadcast with Unidirectional Link Flag

b. is not DIRECT_NEIGHBOR_IF and Bi-Directional Neighbor and Best Neighbor To Originator and

i. is not DUPLICATE: Rebroadcast

ii. is duplicate and ttl=best ttl= Rebroadbast

How to install B.A.T.M.A.N on Ubuntu

The pre-requirements are as follows:

- Compile environment and libraries
- gcc
- libc6-dev
- build-essential
- binutils
- makedev
- make
- libpthread

STEP 1:

Download the batman daemon code from the website

<http://open-mesh.net/batman/downloads>

STEP 2:

In order to compile the file:

- untar and make to get the executable file called “batmand”
\$ wget [http://downloads.open-mesh.net/batman/stable/sources/batmand_0.2-current_s
ources.tgz](http://downloads.open-mesh.net/batman/stable/sources/batmand_0.2-current_sources.tgz)
\$ tar xzvf batmand_0.2-current_sources.tgz
\$ cd batmand_0.2-rv451_sources
\$ make

4. Our Static Topology

Wireless mesh networks are typically strategically deployed. The reason is that companies deploying a wireless mesh network want to minimize cost while increasing coverage. Once the network is deployed, there is little need for dynamic routing. For this reason, and the fact that arranging a specific topology with dynamic routing is quite difficult due to non-ideal wireless channels, it was decided to create routes statically.

Algorithm

In order to efficiently setup the whole network's route, a C++ program was written. This program called, not surprisingly, [wime](#), can be setup as a "master", which is capable of reading routes from a file, broadcasting this file, keeping track of routes, adding, and deleting routes in the kernel's routing table. The first step to run this algorithm is to compile it. To do so, change directory to the *src* directory inside the *wi-me.tar.gz* file provided. Then simply make:

```
$ cd src
$ make
```

To run as master, specify master with '-s' and the interface with '-i' (default ath0):

```
$ sudo ./wime -s -i ath0
```

A client first listens for the file. Upon reception of this file, it stores the routes, delete previous routes, and sets the new ones. To run as client, simply:

```
$ sudo ./wime
```

for interface *ath0* or specify the interface with '-i'. This way, setting the whole networks routing can be done from one computer and changing topologies can be done efficiently.

Bandwidth Experiment

The evaluation was conducted using *iperf* which is a tool to measure network throughput. It measures parameters such as bandwidth, delay jitter and datagram loss.

Setup

Iperf can be downloaded and installed on Ubuntu machines simply by using the update manager, since an *iperf* package is available, also in the Universe repository.

```
$ sudo apt-get install iperf
```

To run *iperf*, one node in the network must be chosen as a server. Other nodes can then connect to this node as clients and measure the rate at which they can transmit data. The following commands execute the program:

```
$ iperf -s -p 5000 (Server)
$ iperf -c -p 5000 -t 60 (Client)
```

Where *-p 5000* indicates that port 5000 must be used, and *-t 60* runs the program for 60 seconds. Other options include sending UDP instead of TCP packets, packet length, transmission interval, etc.

5. Results and Analysis

The topologies used for the evaluation included the exhaustive set of topologies possible on a mesh network with a gateway, a client, an access point and two routers. The following figure depicts the set of topologies.

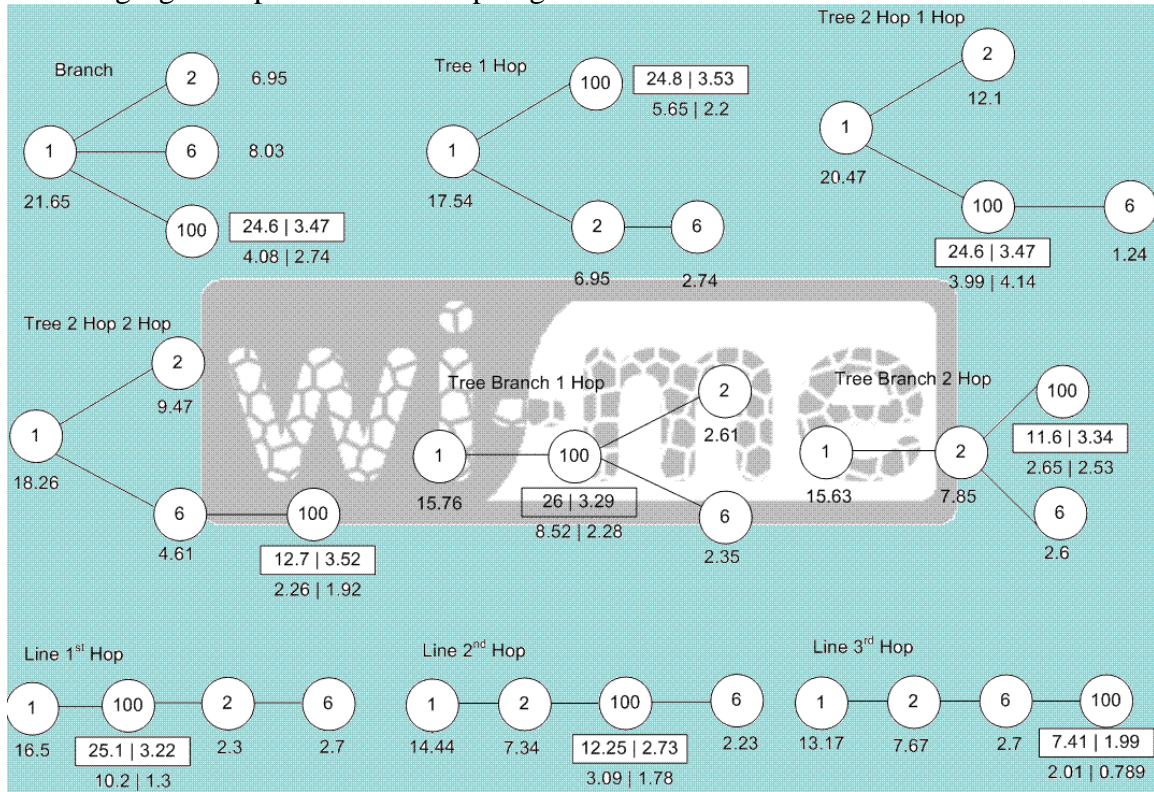


FIGURE 3: Network topologies

Bandwidth at different nodes in the network and also the net throughput of the network for various topologies was obtained using iperf. The results obtained at various different nodes are tabulated in Figure 4.

	Branch	Tree 1 Hop	Two hop first hop	Tree two second hop	Line 1st hop	Line 2nd hop	line 3rd hop	tree branch 1 hop	tree branch 2 hop
Client alone	3.47	3.41	3.53	3.52	3.22	2.73	1.99	3.29	3.34
AP alone	24.6	24.1	24.8	12.7	25.1	12.25	7.41	26	11.6
Client	2.74	2.2	3.14	1.92	1.3	1.78	0.789	2.28	2.53
AP	4.08	5.65	3.99	2.26	10.2	3.09	2.01	8.52	2.65
0.6	8.03	2.74	1.24	4.61	2.7	2.23	2.7	2.35	2.6
0.2	6.8	6.95	12.1	9.47	2.3	7.34	7.67	2.61	7.85

Net throughput	21.65	17.54	20.47	18.26	16.5	14.44	13.169	15.76	15.63
-----------------------	-------	-------	-------	-------	------	-------	--------	-------	-------

FIGURE 4

Based on the results obtained certain intuitive concepts were validated and certain other interested patterns were observed.

The analysis clearly shows that the rate at which bandwidth availability drops over hops is substantial. This phenomenon can be clearly noticed in figure 5 where the bandwidth of client and access points over one two and three hops are plotted for the line topologies. With the Access Point three hops away from the gateway, the client received a meager 800 kilobits per second, or an equivalent 100 kilobytes per second, which while still well above the average modem speeds, is very low as compared to the bandwidth available from the gateway. Also important to consider is that this is without the added control traffic from routing protocols, although such protocols might not be necessary if the backbone of the network is static.

The sharing of the bandwidth was quite fair. We found that a node one hope from the gateway would receive 10Mbps of data when alone, and about 3.3 Mbps when a second node was hopping through it to the gateway. The second node also received about 3.3 Mbps. This implied that the first hop was using two-third of its bandwidth transmitting back and forth the data from the second node, and only one-third of the bandwidth on its own packets. However, once a third hop was added, the second node suffered similar losses and only received about 1.1 Mbps, thus leading to the bandwidth inequities over hops.

It was also noticed that the topologies greatly affect the bandwidth obtained at the client and also at the access point. For instance though *branch* and *tree1hop* topologies both have access points located 1 hop away from the gateway the bandwidth obtained both at the access point and at the client when they are transmitting alone or in tandem with other nodes varies. This variation is visually depicted for all topologies that are one hop away from the gateway in the figure 6 as an illustration of the phenomenon. In general it was observed that branch topology provides maximum throughput while line topology provides the minimum even for the case when the access point is connected to the gateway in 1 hop in both instances.

The problem lies not only with degradation but also with throughput losses. A 10Mbps throughput with one node becomes a net 6.6 Mbps throughput with two, a 34% loss in net throughput despite the same bandwidth and an added potential router. This is because the same amount of data transmitted over two hops uses twice the network bandwidth than it would have when transmitted over one hop. Multi-hop networks are thus a very inefficient use of bandwidth.

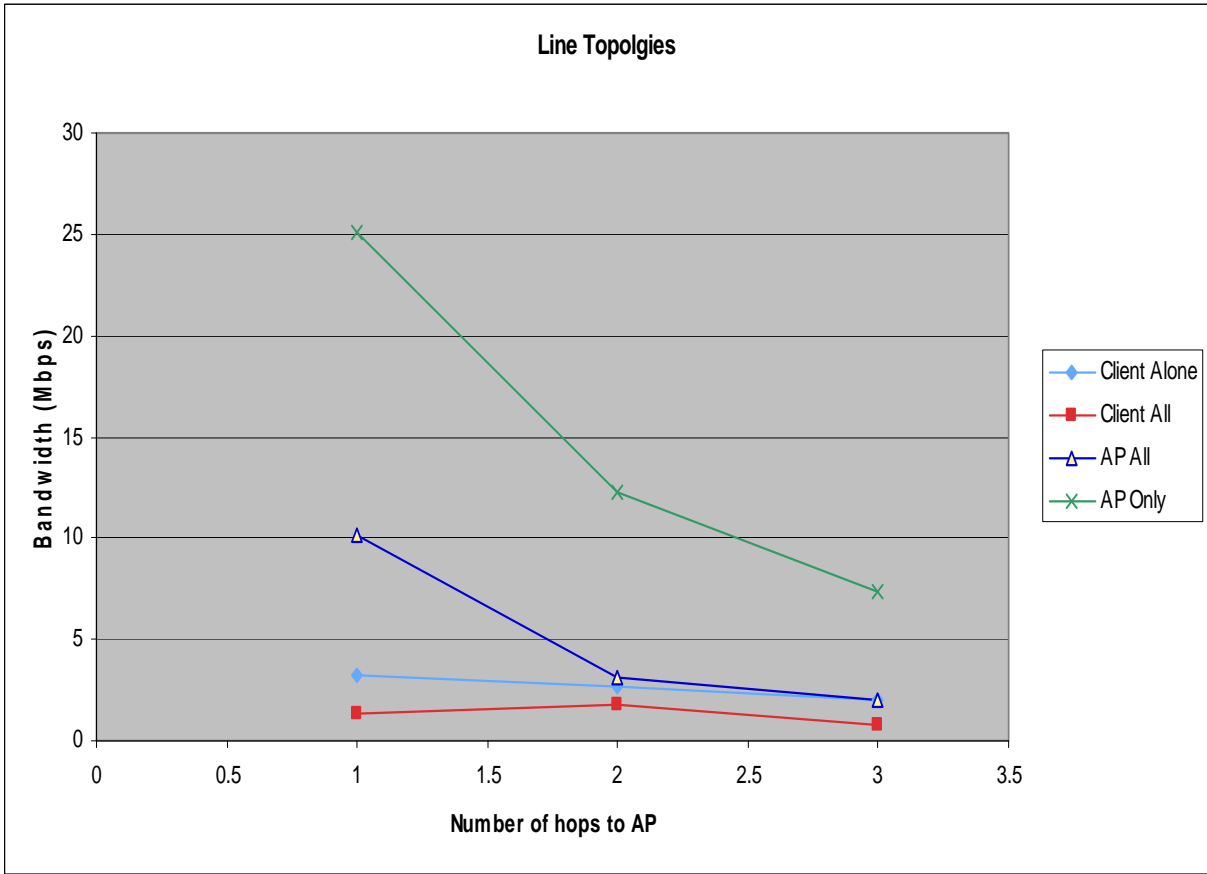


FIGURE 5: Bandwidth of different line topologies

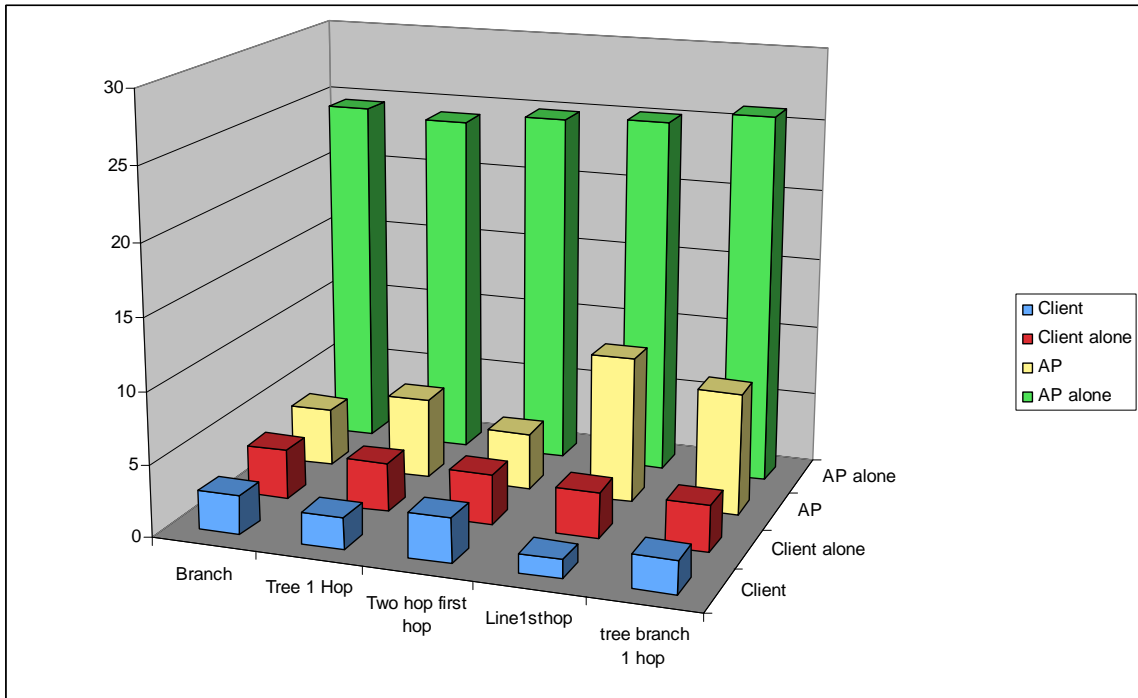


FIGURE 6

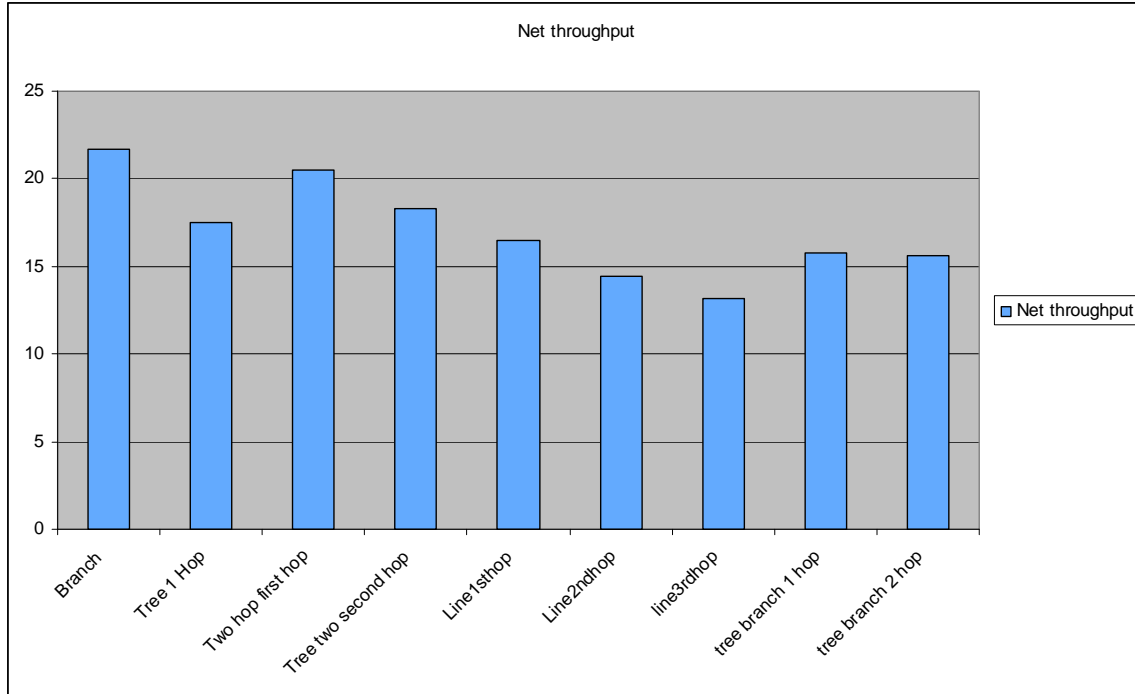


FIGURE 7

6. Conclusion

Performance

It has been observed that there is a loss in bandwidth due to multi hopping in mesh networks. With larger networks, it is easy to predict that the performance degradation over multiple hops, with multiple clients at each hop, would soon degrade performance to unusable levels. It is therefore important that the backbone be reinforced with gateways at regular intervals.

Despite all their disadvantages of bandwidth inequities and throughput losses, wireless mesh networks do offer a promising inexpensive alternative to wired networks. The cost of setting up such a network is nominal, and relative to the reduction in cost the loss in throughput is less than proportionate. While not able to match up to the performance of wired networks, in cases the cost per byte matters more than the number of bytes exchanged wireless networks offer a far better solution.

Topology and Routing

Throughput loss in a mesh network is proportional to the number of hops. An optimal topology for the network is therefore one that has the minimum number of total hops from each client to the gateway. Since there is a proportionality between the number of hops and the maximum distance that a client can be from the gateway, there is also a tradeoff between number of hops and range that must be resolved. Given a fixed set of nodes, however, a routing protocol must aim at minimizing the number of hops used by

data transmissions. This is precisely what a protocol like OLSR aims at doing. However, consideration must also be given to link quality, since beyond a certain point link quality will mitigate bandwidth more than number of hops. OLSRd takes into account link quality by using expected number of transmissions instead of number of hops. BATMAN tries to reduce the bandwidth used by the control traffic of OLSR, while also reducing the number of computations required from each node. BATMAN is so far the most efficient protocol for mesh networks we have found.

From the above argument, it is also obvious that the net throughput is maximum in a branch topology, followed by the tree and line topologies. This is also obvious from the data collected during our evaluation.

Appendix A: Work Distribution

Setting up Ad-Hoc – Prateek, Luis
OLSR implementation – Shivananjani, Prateek
BATMAN implementation – Prateek, Luis
Setup Gateway setup – Luis
Setup Wireless Access Point - Luis
Code – Modify Routing Table – Karishma, Luis
Code – Broadcast routing file – Karishma
Evaluation – Prateek, Shivananjani, Karishma, Luis
Analysis - Shivananjani