

OLSR IN ANDROID OPERATING SYSTEM

By

Nasim Chowdhury

B.Sc., Electrical Engineering (1995)

University of Oklahoma

A Project presented to Ryerson University
in partial fulfillment of the requirements for the degree of

Master of Engineering

Program of Electrical and Computer Engineering

Toronto, Ontario, Canada

©Nasim Chowdhury 2013

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this project. This is a true copy of the project, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my project may be made electronically available to the public.

ABSTRACT

OLSR IN ANDROID OPERATING SYSTEM

Nasim Chowdhury

Program of Electrical and Computer Engineering

Ryerson University

Master of Engineering 2013

Optimized Link State Routing protocol, an ad-hoc routing protocol, has been popular in wireless devices running on Linux operating system for quite some time. In this project we have outlined the process of preparing Android devices for ad-hoc networking, a way to overcome limitations of the OS for continuous UDP communication, ensure all devices communicate on the same wireless Wi-Fi SSID, Cell-ID, subnet and finally implement the Optimized Links State Routing (OLSR) in Android Operating System using Google Nexus 7 devices. Using the code base from ProjectSPAN, an open source project, OLSR protocol has been ported to Android Nexus 7 devices. The core application is divided into two major sections, MANET and OLSR. Mobile Ad-hoc Network portion of the code takes care of setting up the device for ad-hoc mode communication, firewall and peripheral setup while OLSR portion of the code maintains the neighbor tables, MPRs and routing. The project also describes the process by which a device is prepared to run low level custom codes in Android operating system.

The OLSR implementation has been successfully tested with three nodes test bed, demonstrating the multi-hop ad-hoc networking capabilities of this wireless routing protocol. With the aid of the Android's graphical interface the application is able to exhibit the dynamic nature of the OLSR protocol. As nodes and neighbors in the network moves around with respect to time and relative location, OLSR protocol is able to form new neighbors and elect Multipoint Relay in real time.

Contents

1	Introduction	1
1.1	Need for Ad-hoc Networking.....	1
1.2	Challenges of Ad-hoc Networking using Android.....	1
1.3	Main Contribution.....	3
1.4	Chapter Overview	3
2	Communication via Ad-hoc Network	4
2.1	Infrastructure based Wireless Networks	4
2.2	Ad-hoc Wireless Networks	4
2.3	Ad-hoc Routing Protocols.....	5
3	MANET - Usage of Mobile Ad-Hoc Network.....	8
3.1	Disaster Situation and Disaster Relief.....	8
3.2	Military Activities	8
3.3	Instant Infrastructure	9
3.4	Remote Areas.....	9
3.5	Cost Effectiveness.....	9
4	OLSR – Optimized Link State Routing	10
4.1	Node Addressing.....	10
4.2	Information Repositories.....	10
4.3	Timeouts	11
4.4	Control Traffic	12
4.5	OLSR Packet Format	12
4.6	Message types	13
4.7	Multipoint Relay	14
4.8	OLSR Hello Message.....	14
4.9	MPR Section Procedure	15
4.10	OLSR Data Structure	16

4.11	Route Selection using ETX and SPF	16
5	OLSR on Android	18
5.1	Access to Wireless Modem.....	19
5.2	Wireless Extension and Cellular Service Providers	19
5.3	Broad Hardware Spectrum.....	20
5.4	Need for Rooting Android and Installing Custom Kernel.....	20
6	Developing Software for Android.....	22
7	Preparing Android for Custom Application	24
7.1	Process of Installing Custom Kernel.....	24
7.2	Enable USB Debugging	24
7.3	Download Android Root Tool Kit	24
7.4	Install Driver for Android Devices	25
7.5	Unlocking Android Devices.....	25
7.6	Rooting Android Devices.....	26
7.7	Install Custom ROM	27
8	Project SPAN	28
8.1	MANETServiceHelper.java.....	29
8.2	OlsrProtocol.java	29
8.3	Olsrd.....	30
9	Analysis and Evaluation of Results.....	32
9.1	iwconfig	32
9.2	MANET Manger Routing Info.....	34
9.3	OLSR with no MPRs	36
9.4	OLSR in Multi-hop Network	37
9.5	OLSR Debug Information.....	38
9.6	ETX and Delay Comparison for Multi-Hop MANET	39
10	Conclusion and Future Work	42
10.1	Conclusion	42

10.2 Future Work	42
11 Bibliography	43
12 Appendix	44

List of Tables

Table 1: List of Android Ad-hoc Code Evaluated	2
Table 2: Ad-hoc Routing Protocols	7
Table 3: Ad-hoc Routing Protocols for Android	7

List of Figures

Figure 2-1: Infrastructure based Wireless Networks	4
Figure 2-2: Ad-hoc Wireless Network.....	4
Figure 2-3: Multi-hop Wireless Network	5
Figure 2-4: Single-hop Ad-hoc Network	5
Figure 4-1: OLSR Packet Format	12
Figure 4-2: MPR selection Process.....	15
Figure 4-3: OLSR Data Flow.....	16
Figure 5-1: Android Operating System Architecture.....	18
Figure 5-2: Example of Node Distribution in MANET.....	19
Figure 6-1: Application Development using Eclipse.....	22
Figure 6-2: Eclipse Development Environment	22
Figure 7-1: Android Root Tool Kit.....	24
Figure 7-2: Installing Device Drivers and Unlocking Android Devices	25
Figure 7-3: Rooting Android Devices.....	26
Figure 7-4: Verify Rooting Process	26
Figure 9-1: MANET Manger Routing Info	34
Figure 9-2: OLSR in Fully Meshed Network	36
Figure 9-3: OLSR with MPR elected.....	37
Figure 9-4: OLSR Debugging.....	38

Figure 9-5: Distance vs. ETX	39
Figure 9-6: Distance vs. Delay.....	40
Figure 9-7: OLSR 2 hop - Distance vs. ETX.....	40
Figure 9-8: OLSR 2 hop - Distance vs. Delay	41

1 Introduction

Wireless communication has grown in tremendous pace in the last two decades and small handheld devices, such as cell phones and tables, that connect people to communicate anytime, anywhere have become a must have extension of modern society. Most of the communication medium relies on infrastructures to interconnect devices, be it traditional wireless cell phones or the new age smart phones.

1.1 Need for Ad-hoc Networking

There are many situations where users cannot rely on data or telecom infrastructures. The infrastructure may be too expensive or there is no infrastructure at all. Challenge of our times is to utilize the poplar handheld devices and allow them to communicate to each other without any help of infrastructures. In those situations multi-hop mobile ad-hoc networks or MANET may be a great choice for smart devices to communicate in the absence of base stations or access points. Devices in these mobile networks can send data from one host to another remote host by hopping few hosts in between. The characteristics of these mobile devices is that some may act as just hosts, few may act as routers while others can work as routers as well as host. Some routers or hosts may be connected via wires as wireless gateways to join other non-wireless networks or the largest network in the world, Internet.

Android based devices are best platform to develop such application. Android is Linux based open source, free of charge operating system which allows code modification under General Public License guideline. No other wireless operating system currently allows such versatility. In this project Google Nexus 7 devices has been utilized to implement the OLSR - Open Link State Routing protocol – to create MANET – Mobile Ad-hoc Network. OLSR has been implemented on Ethernet networks for quite sometimes [1].

1.2 Challenges of Ad-hoc Networking using Android

Staring with version 4.0, Android allows ad-hoc networking by its built in support for Wi-Fi Direct protocol [2]. Wi-Fi Direct is a Wi-Fi standard that enables devices to connect easily with each other without requiring a wireless access point and to communicate at typical Wi-Fi speeds.

We first investigated the possibility to develop multi-hop capability based on the Wi-Fi Direct. An application was ported to Android that allowed direct communication between two devices for file transfer and other communications. However Wi-Fi Direct does not allow multi-hop communications as setting up nodes in the network requires a manual process for security reasons.

We also investigated an initiative by Serval Mesh project [3]. Serval Mesh is an Android application that allows IP connectivity between mobile phones using Wi-Fi protocol, without requiring a SIM, cellular base stations or Wi-Fi access points or Internet access. However Serval Mesh did not provide multi hop communication capability during the time it was evaluated. Currently however, it is using Mesh routing to provide multi-hop communication.

The OLSR development group has been working on an implementation of OLSR for Android devices as well [4]. However code could not be ported to successfully to newer Android devices such as Nexus 7 with new versions of the OS due to wireless network card issue and the code was out of date and was not maintained.

SPAN project code showed promising results [5]. The Smart Phone Ad-hoc Network initiative is a community supported newer code base. The code required the device has to “rooted” and a custom kernel loaded to have the application manipulate the wireless network card. However the OLSR implantation was not complete to work on Google Nexus 7 devices. The nodes would load the application but could not see each other to form neighbour relationship.

Following table lists the Android software code evaluated for ad-hoc networking:

Android Code	Description
Android Wi-Fi Direct	Peer-to-peer networking over Wi-Fi protocol.
Serval Project	Ad-hoc networking on mesh network
OLSRMeshTether	Multi-hop Ad-hoc NETworking using Wi-Fi protocol and routing using OLSR
MANET Manager	Multi-hop Ad-hoc NETworking using Wi-Fi protocol and routing using OLSR

Table 1: List of Android Ad-hoc Code Evaluated

1.3 Main Contribution

Main contributions of this project are as follows:

- Few researchers have previously attempted to implement OLSR in Android operating system, however most of their endeavours were either obscure implementation, obsolete or still work in progress [1] [4] [3]. As mentioned above we have identified the best code that come close to implementing OLSR and modified it to work successfully under Google Nexus 7 devices with Android's latest operating system (Jelly Bean or 4.2.2) for the first time.
- Ad-hoc networking requires Wi-Fi protocol to act differently than access point based network. We have identified that in Android, the Wi-Fi network card needs to be initialized during start-up of the OLSR application with same Cell-ID to communicate with different devices. The ProjectSPAN code has been modified to take care of this new findings to implement OLSR.
- We have also outlined the process of preparing Android devices by unlocking, rooting and installing custom kernel that is required to run the devices in ad-hoc mode that will facilitate and expedite setting up OLSR for future research projects.
- The OLSR application has been done using Eclipse and Android Development Tools.

1.4 Chapter Overview

Rest of this paper is organized as follows. An introduction to mobile ad-hoc network is given in chapter 2. In Chapter 3, an introduction of Optimized Link State Routing protocol has been presented. Chapter 4 describes requirements of implementing OLSR in Android operating system. This followed by the description of developing software for Android OS in Chapter 5 and Chapter 6 details the preparation required the devices for implementation. Chapter 7 provides the outline of the code used to implement OLSR in Android and Chapter 8 provides the result of the experiments done for this thesis.

2 Communication via Ad-hoc Network

Wireless networks revolutionized the way we communicate. Wireless networks brought the convenience to small handheld devices but poses many challenges in its implementation compared to its wired counterpart. Wireless nodes can communicate in two different ways, infrastructure based or ad-hoc mode.

2.1 Infrastructure based Wireless Networks

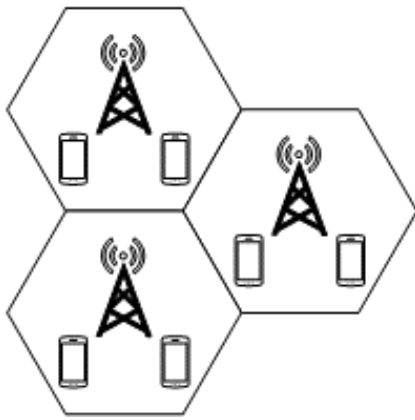


Figure 2-1: Infrastructure based Wireless Networks

Wireless communication that utilizes infrastructure, such as the GSM or UMTS networks or wireless LANs in infrastructure mode, a dedicated base station or access point is setup to communicate between the devices in a hub network situation. All wireless devices in the network must register themselves with the base station or the access point. All data traffic passes through these centralized network infrastructure (Figure 2-1).

2.2 Ad-hoc Wireless Networks

Contrary to infrastructure based wireless networks, the communication in Ad-hoc networks is organized completely decentralized. There is no single base station or access point that controls the flow of network traffic (Figure 2-2). Governed by the physical layer protocol regulations of 802.11a/b/g/n all nodes can be at the same time transmit and receive wireless data as well as forwarding traffic for other nodes.

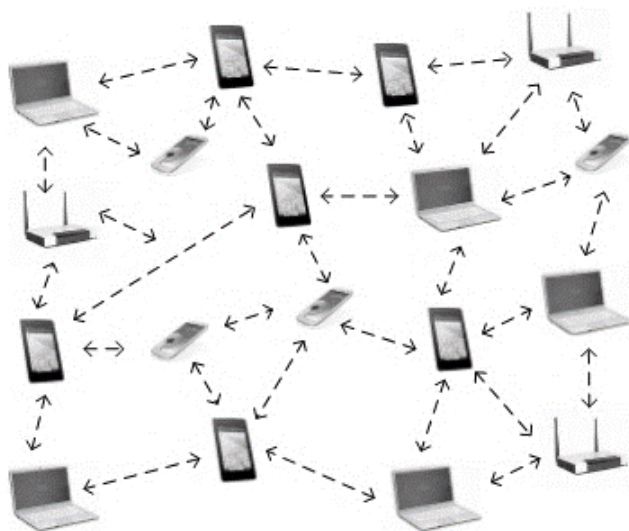


Figure 2-2: Ad-hoc Wireless Network

These nodes can act as node and as router and create a multi-hop wireless network.

Nodes that are within the reach of each other that is within wireless range of a nearby node, can exchange packets between the devices without any the help of any other entity. To transmit packets to nodes that are farther away, nodes that lies in between the source and destination node forward the packets from one to another, like traditional routers, to reach final destination.

The main advantages of ad-hoc networks over the infrastructure based wireless networks are their decentralized self-organizing nature, no requirement to setup any special infrastructure and their flexibility as mobile networks changes with respect to time.

For single hop network, every node is only one hop away from other nodes. As illustrated in Figure 2-4, to reach any node, a device do not need any routing protocol. All devices have to be in the broadcast range of each other.

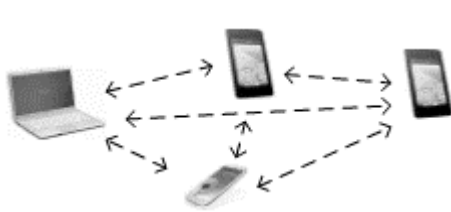


Figure 2-4: Single-hop Ad-hoc Network

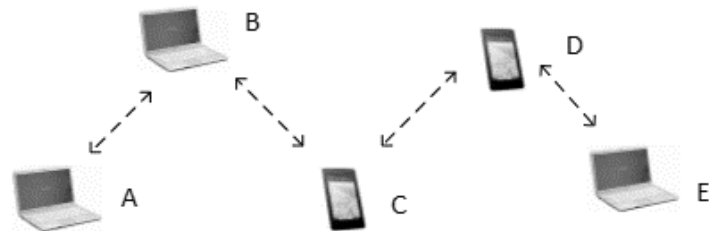


Figure 2-3: Multi-hop Wireless Network

However, when all devices are not within broadcast range, to communicate between remote networks, the devices need a routing protocol. As depicted in Figure 2-3, for node A to communicate with node E, the intermediate nodes B, C and D have to know how to route packets and act as routers for nodes A and E. The nodes in a Multi-hop Network are joined together using special routing protocols. These protocols are designed considering the advantages and limitations of wireless medium, battery longevity, transceiver capabilities and ad-hoc nature of the network.

2.3 Ad-hoc Routing Protocols

There are several different routing protocols developed for ad-hoc networks over the years. These protocols are primarily divided into two categories. There are some protocols that combines the following two types and takes advantage from both categories.

- **Proactive or Table Driven Routing Protocols** - Proactive or table-driven routing protocols maintain several tables containing neighbour and routing information on each node for all the nodes in a network. Each node at predetermined interval sends a small packet and through which keeps record of the neighbors. When there is a network topology change, the nodes propagate the update messages throughout the entire network and to have a consistent and up-to-date topology information of the whole network. Routes are calculated based on the topology.
- **Reactive or On Demand Routing Protocols:** In contrast to Proactive routing protocols, all nodes do not keep up-to-date routes of the entire network. The routes are created as packets are required to be delivered. When a source node wants to communicate with a destination, the routing protocol initiates the route discovery process to find the path to the destination. The route remains valid as new packets are generated for the destination and removed from the routing table if unused after certain timeout period.

Following are some of the well-known ad-hoc routing protocols:

Type of Protocol	Name of Ad-hoc Protocols
Proactive or Table Driven Routing Protocols	Optimized Link State Routing Protocol (OLSR) Destination-Sequenced Distance-Vector (DSDV) The Wireless Routing Protocol (WRP) Global State Routing Fisheye State Routing Hierarchical State Routing Zone-based Hierarchical Link State Routing Protocol Clusterhead Gateway Switch Routing Protocol Better Approach To Mobile Adhoc Networking (BATMAN) Babel Routing Protocol
Reactive or On Demand Routing Protocols	Cluster based Routing Protocols Ad hoc On-demand Distance Vector Routing (AODV) Dynamic Source Routing Protocol Temporally Ordered Routing Algorithm (TORA) Associativity Based Routing (ABR) Signal Stability Routing (SSR) Admission Control enabled On demand Routing (ACOR) Flow State in the Dynamic Source Routing

Table 2: Ad-hoc Routing Protocols

Following is a list of routing protocols that have been implemented or in development for Android operating system.

Ad-hoc Protocol	Implemented / In Progress
OLSR	MANET Manager OLSRMeshTether Serval Project (Proposed)
AODV	MANET Manager (Not yet implemented)
Mesh Routing	Serval Project

Table 3: Ad-hoc Routing Protocols for Android

We are using OLSR as the routing protocols on MANET Manager application. Details of the OLSR protocol will be discussed in Chapter 4.

3 MANET - Usage of Mobile Ad-Hoc Network

A Mobile Ad-hoc Network or MANET is a self-configuring network of mobile devices in an infrastructure-less environment. Nodes in MANET can move arbitrarily and topology of the network can change dynamically with respect to time and relative position of the devices. Need to develop this kind of networks have been grown significantly in the last few years due various requirements. However many aspects of mobile ad-hoc networks are still in research stage. Some of the research topics on the routing protocol layer include among others, support for quality of service for voice and video communication, efficient multicast strategies, protocol optimizations and security related issues.

There can be many application of MANET based networks. Following are few examples where MANET can be very useful.

3.1 Disaster Situation and Disaster Relief

Fukushima, Haiti, Katrina all underline the fact that Infrastructure based telecommunication is not ideal during disaster situation. Infrastructures typically break down in a disaster area. Hurricanes cut phone and power lines, base stations get destroyed by flood, datacenter loses communication, gets destroyed in fire. Emergency responders can only rely on the infrastructures that they setup. Planning for such disasters cannot be done and setup must be fast and reliable but also cost effective.

3.2 Military Activities

Major driving force in ad-hoc networking is defense related activities as many of the research project on ad-hoc network are backed by military institute, especially in the US. War zones and training in adverse environment either cannot depend on existing infrastructure or there is lack of any. As weaponry and communication equipment are getting modernized, communication system that can be setup quickly and move as the army moves becomes an essential tool.

3.3 Instant Infrastructure

In areas where infrastructure is not available, unplanned meetings, spontaneous interpersonal communication etc. becomes tricky. It would take too long to setup infrastructure in short period of time, therefore ad-hoc connectivity makes most sense in such cases.

3.4 Remote Areas

For some remote areas it is sometimes too expensive to set those up in sparsely populated areas. Depending on the communication pattern, ad-hoc network or satellite base infrastructure can provide the effective solution that is required.

3.5 Cost Effectiveness

The service provided by certain infrastructure may not be cost effective for certain application. For example, if there is connection oriented cellular network exist, but a battery operated application sends only a small status information every ten minutes, a cheaper ad-hoc packet oriented network would make much more sense. Registration procedure may take too long and keep-alive may drain out the battery too much, communication overhead may be too high.

4 OLSR – Optimized Link State Routing

As indicated before, OLSR is the chosen protocol for this application of Android Ad-hoc networking. It is a proactive routing protocol for mobile ad-hoc networks. OLSR works best in large and dense mobile networks. The protocol is documented in the experimental Request For Comment (RFC) 3626 [6]. OLSR is table-driven and pro-active and utilizes an optimization called Multipoint Relaying for control traffic flooding. The larger and more dense a network, the more optimization can be achieved as compared to the classic link state algorithm. OLSR uses hop-by-hop routing, i.e., each node uses its local information to route packets. OLSR is well suited for networks, where the traffic is random and sporadic between a larger set of nodes rather than being almost exclusively between a small specific set of nodes. As a proactive protocol, OLSR is also suitable for scenarios where the communicating pairs change over time: no additional control traffic is generated in this situation since routes are maintained for all known destinations at all times. RFC3626 modularizes OLSR into core functionality, which is always required for the protocol to operate, and a set of auxiliary functions. The core functionality specifies, a protocol that is able to provide routing in a stand-alone MANET. Each auxiliary function provides additional functionality, which may be applicable in specific scenarios, e.g., in case a node is providing connectivity between the MANET and another routing domain.

This protocol inherits the stability of a link state algorithm and has the advantage of having routes immediately available when needed due to its proactive nature. OLSR is an optimization over the classical link state protocol, tailored for mobile ad hoc network.

4.1 Node Addressing

OLSR uses an IP address as the unique identifier of nodes in the network. As OLSR is designed to be able to operate on nodes using multiple communication interfaces, every node must choose one IP address that is set to be its main address.

4.2 Information Repositories

OLSR maintains state by keeping a variety of databases of information. These information repositories are updated upon processing received control messages and the information stored is

used when generating such messages. Here follows a brief look at the different information repositories used in core OLSR.

- Multiple Interface Association Information Base - This dataset contains information about nodes using more than one communication interface. All interface addresses of such nodes are stored here.
- Link Set - This repository is maintained to calculate the state of links to neighbors. This is the only database that operates on non-main-addresses as it works on specific interface-to-interface links.
- Neighbor Set - All registered one-hop neighbors are recorded here. The data is dynamically updated based on information in the link set. Both symmetric and asymmetric neighbors are registered.
- 2-hop Neighbor Set - All nodes, not including the local node, that can be reached via a one-hop neighbor is registered here. Note that the two hop neighbor set can contain nodes registered in the neighbor set as well.
- MPR Set - All MPRs selected by the local node is registered in this repository.
- MPR Selector Set - All neighbors that have selected this node as a MPR are recorded in this repository.
- Topology Information Base - This repository contains information of all link-state information received from nodes in the OLSR routing domain.
- Duplicate set - This database contains information about recently processed and forwarded messages.

4.3 Timeouts

Most information kept in these repositories are registered with a timeout. This is a value indicating for how long the registered information is to be considered valid. This value is set according to a validity time fetched from the message from which the data was last updated. The use of such a distributed validity time allows for individual message emission intervals for all nodes in the network. All database entries are removed when no longer valid according to the registered timeout. Such entries are said to be *timed out*.

4.4 Control Traffic

All OLSR control traffic is to be transmitted over UDP on port 698. This port is assigned to OLSR by the Internet Assigned Numbers Authority (IANA). The RFC states that this traffic is to be broadcasted when using IPv4, but no broadcast address is specified. When using IPv6 broadcast addresses does not exist, so even though it is not specified in the RFC, it is implicitly understood that one must use a multicast address in this case.

4.5 OLSR Packet Format

All OLSR control traffic is based upon OLSR packets. An OLSR packet has an OLSR packet header consisting of the packet length and a packet sequence number maintained independently by each interface of the OLSR node. The packet body consists of one or more OLSR messages which are preceded by a message header for each included message. The message header contains the message type, the validity time, the message size, the originator address, a time to live field, the hop count and a message sequence number. The originator address field contains the main address of the node that initially created the message, independent of which interface the message left this node. To avoid establishing routing loops and retransmission of already known data each packet and each message carry a sequence number.

0	7	8	15	16	23	24	31
Packet Length				Packet Sequence Number			
Message Type		Vtime		Message Size			
Originator Address							
Time To Live		Hop Count		Message Sequence Number			
Message							
Message Type		Vtime		Message Size			
Originator Address							
Time To Live		Hop Count		Message Sequence Number			
Message							
...							

Figure 4-1: OLSR Packet Format

- Packet Length - The length in bytes of the entire packet, including the header.

- Packet Sequence Number - A sequence number incremented by one each time a new OLSR message is transmitted by this host. A separate Packet Sequence Number is maintained for each interface so that packets transmitted over an interface are sequentially enumerated.

An OLSR packet body consists of one or more OLSR messages. All OLSR messages must respect this header. The fields in the header are:

- Message type - An integer identifying the type of this message. Message types of 0-127 are reserved by OLSR while the 128-255 space is considered “private” and can be used for custom extensions of the protocol.
- Vtime - This field indicates for how long after reception a node will consider the information contained in the message as valid.
- Message Size - The size of this message, including message header, counted in bytes.
- Originator Address - Main address of the originator of this message.
- Time To Live - The maximum number of hops this message can be forwarded. Using this field one can control the radius of flooding.
- Hop Count - The number of times the message has been forwarded.
- Message Sequence Number - A sequence number incremented by one each time a new OLSR packet is transmitted by this host.

4.6 Message types

- MID message - If a node has more than just one interface it announces these additional interfaces periodically to the other nodes by emitting MID messages. As the nodes main address is already included in the originator address of the message header only the additional interface addresses have to be announced. Based upon this information the Multiple Interface Association Information Base is built in the receiving node.
- HELLO messages - To supply the necessary information for link sensing and (one- and two hop) neighborhood discovery a node periodically emits HELLO messages. Through the exchange of these messages the link set and the information in the Neighbor Information Base is built. These messages are generated and emitted independently for each interface participating in the network. For each different

neighbor and link type combination (link code) a list of addresses with interfaces belonging to this link code is advertised.

- TC messages – OLSR is a host based flat, link state routing protocol. A topology change link state broadcast describes change of links to neighbor nodes. This is done using Topology Control (TC) messages. The message contains a sequence number which is updated every time the advertised neighbor set has changed. TC messages are flooded on regular intervals by using MPR optimization process.

4.7 Multipoint Relay

The idea of multipoint relays is to minimize the overhead of flooding messages in the network by reducing redundant retransmissions in the same region. Each node in the network selects a set of nodes in its symmetric 1-hop neighborhood which may retransmit its messages. This set of selected neighbor nodes is called the "Multipoint Relay" (MPR) set of that node. The neighbors of node N which are *NOT* in its MPR set, receive and process broadcast messages but do not retransmit broadcast messages received from node N.

In an N node neighborhood, MPRs are subset of nodes so that every node in that 2-hop neighborhood must have a symmetric 1-hop link to a MPR and MPRs are elected such that it comprises the minimal set in that neighborhood.

In OLSR MPRs are elected to flood the link state of the network to all nodes. They generate the link state update packets. They only generate the updates for the links they are connected to. As proactive protocol, OLSR periodically sends the link state update message to its neighbors and proactively builds the routing table.

4.8 OLSR Hello Message

Every node sends hello message to all its neighbors and it contains the list of neighbors it already discovered and their state. A neighbor can be in one of the three states:

SYM_NEIGH: A neighbor which has established a symmetric link

MPR_NEIGH: A neighbor with symmetric link has been elected as MPR

NO_NEIGH: A possible neighbor, but a symmetric link has not yet been established.

4.9 MPR Section Procedure

OLSR designates various nodes in a neighbor of N nodes in the following structure:

- N set: set of symmetric 1-hop neighbors
- N2 set: set of symmetric 2-hop neighbors, excluding the node itself and the nodes in N set
- D(y): Degree of the neighbor y that is a member of N excluding the node itself and the nodes in N set

OLSR then calculate MPRs based on the following algorithm:

- Calculate Degree of the neighbor or D(y) where y is a member of N set.
- Designate those nodes in N set as MPRs that are the only nodes to provide a reachability to any node in N2 set.

From the N2 list, remove the nodes that are covered by a node in MPR set.

- If there are any nodes in N2 that are not covered by any node in the MPR set.

- For each node in N calculate the reachability to nodes in

N2 that are not yet covered by a node in MPR set and that are reachable through this node.

- Select a node in N that provides reachability to maximum number of nodes in N2.
- In case of tie, use larger value of D(y) to break the tie.

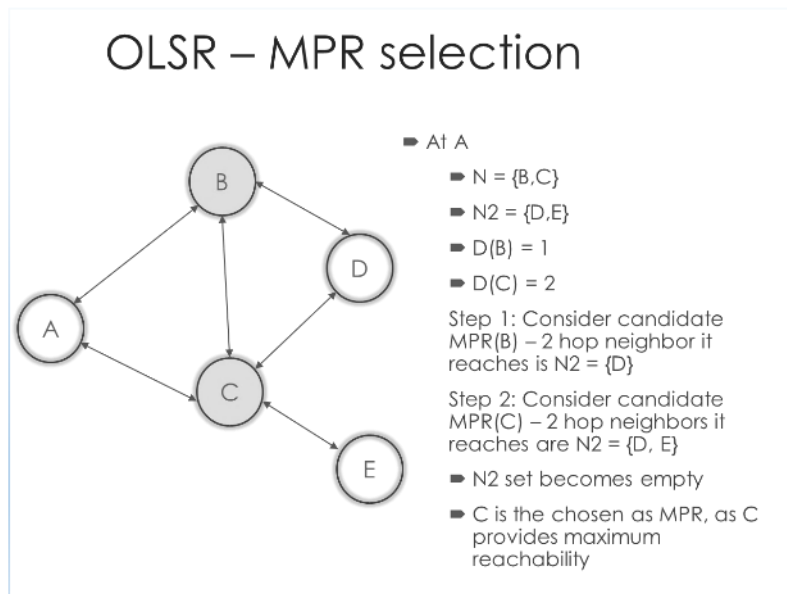


Figure 4-2: MPR selection Process

Figure 4-2 shows an example of how MPR selected in OLSR wireless network.

4.10 OLSR Data Structure

OLSR is table driven protocol. It keeps a number of live tables to keep track of symmetric 1-hop neighbors, 2-hop neighbours, MPRs, Topology Information Base, Duplicate Message Set, Multiple Interface Association Set etc.

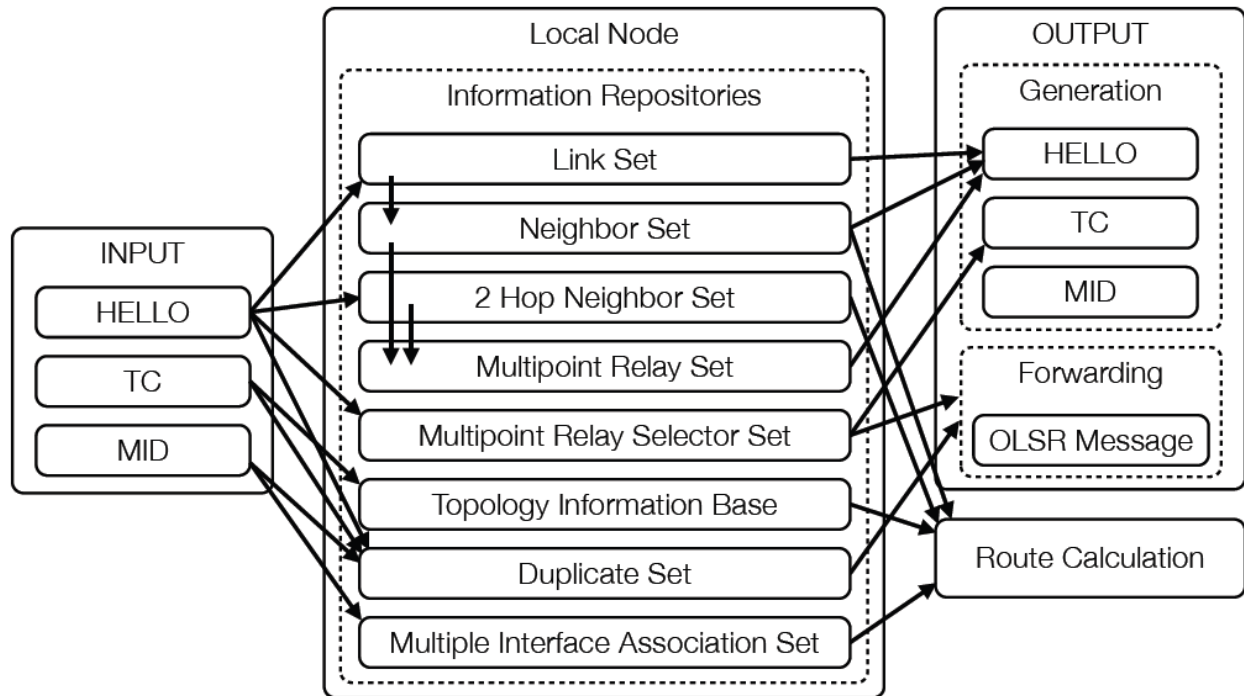


Figure 4-3: OLSR Data Flow

As the Figure 4-3 shows, using the input information of Hello packets, Topology Control packets and information about its Multiple Network Interfaces, each node creates the Link Set, Neighbor Set, 2 Hop Neighbor Set, Multipoint Relay Set, Multipoint Relay Selector Set, Topology Information Base, Duplicate Set and Multiple Interface Association Set tables. Based on these tables each node decides if it would forward any OLSR messages it received to its other neighbours. Based on topology it creates the routes.

4.11 Route Selection using ETX and SPF

Since the release of version 0.4.8, OLSR started to use ETX to measure link quality between two nodes. ETX or Expected Transmission Count metric is a novel method to evaluate metric which finds high-throughput paths on multi-hop wireless networks [7]. ETX metric considers the effects of link loss ratios, asymmetric loss ratio between two directions of each link

and interference on the successive links of a data routing path. On the contrary, minimum hop count metric only chooses the shortest path randomly among various paths with identical hops but does not consider data throughput differences among those paths. ETX metric can help significantly improve throughput by choosing longer but best paths in a wireless mesh network.

The ETX of a link is calculated by using the forward and reverse packet delivery ratios of the wireless link. OLSR send hello messages to its neighbors to create and update neighbor relationships. Within the hello message, OLSR also includes the perceived link quality of its neighbor. Link quality is a simple calculation; if a node send 10 packet and receive acknowledgement for 7 only, link quality or LQ of that neighbor is 7/10 or 0.7. The neighbor also advertises its own perceived link quality of this node as neighbour link quality of NLQ. ETX is of a link to a neighbor is then calculated by the following formula

$$\mathbf{ETX = 1 / (NLQ * LQ)}$$

The MANET Manager in ProjectSPAN utilizes Shortest Path First using a simple implementation of Dijkstra's algorithm to compute the best routing path. Using ETX and SPF, the code is able to achieve best throughput over minimum hop count metric.

5 OLSR on Android



Android is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers. The first Android-powered phone was sold in October 2008. Android is open source and Google releases the code under the Apache License. This open source code and permissive licensing allows the software to be freely modified and distributed by device manufacturers, wireless carriers and enthusiast developers. Additionally, Android has a large community of developers writing applications ("apps") that extend the functionality of devices, written primarily in a customized version of the Java programming language. In February 2013, there were over 800,000 apps available for Android, and the estimated number of applications downloaded from Google Play, Android's primary app store, was 25 billion. These factors have allowed Android to become the world's most widely used smartphone platform.

As Figure 5-1 shows, Android has the following layers: (a) applications (written in java, executing in Dalvik); (b) framework services and libraries (written mostly in java) - applications and most framework code executes in a virtual machine; (c) native libraries, daemons and services (written in C or C++); (d) the Linux kernel, which includes drivers for hardware, networking, file system access and inter-process-communication.

Due to the vast popularity of Android devices, porting OLSR to

Android is the most logical approach to facilitate ad-hoc communication. There is project that is

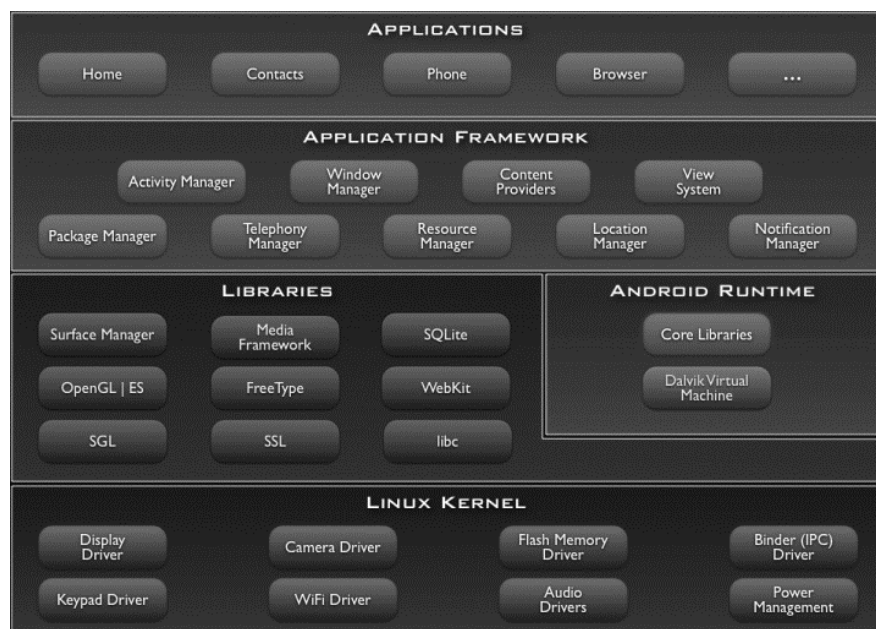


Figure 5-1: Android Operating System Architecture

backed by the Android developer community called ProjectSPAN or ‘Smart Phone Ad-hoc Networking’ that is utilizing OLSR as one of the routing protocol. It was initially funded by MITRE Corporation for Emergency Preparedness and Response situation.

Implementation of MANET on smart devices however has its share of difficulties. There is a wide variety of transceivers that come with Android devices as their Wi-Fi modems. For MANET, it is an obvious choice to use the Wi-Fi network over Bluetooth or NFC. In physical layer, Wi-Fi provides the longest ranges amongst the communication choices and almost all devices are equipped with these modems. Following are some of the issues when implementing MANET/OLSR on smart devices:

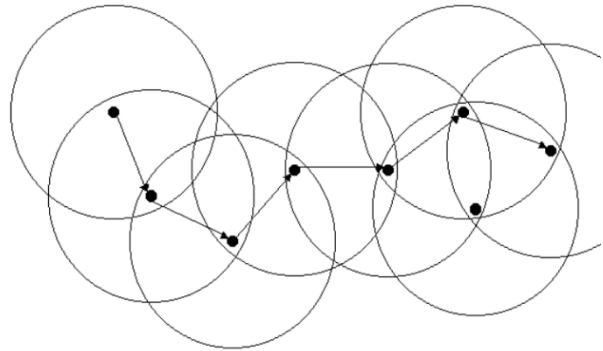


Figure 5-2: Example of Node Distribution in MANET

5.1 Access to Wireless Modem

In the pre-Ice Cream Sandwich or version 4.0, Android framework does not support configuring the built-in wireless modem to operate in any other mode but managed mode. Starting from ICS Android supports Wi-Fi Direct, however the ICS implementation of the Wi-Fi Direct specification does not provide a complete ad-hoc network solution [8]. To implement OLSR on Android devices, the MANET application requires direct access to the modem, so it can scan the network constantly for any data traffic, as well as maintain neighbor table up to date for routing operation. For security reasons and reasons described in the next point, Android operating system allows only limited sets of APIs to control most hardware calls and does not allow any application to have access to its Wi-Fi modem directly to send and receive data.

5.2 Wireless Extension and Cellular Service Providers

To engage the wireless modem in ad-hoc mode the Wi-Fi chip has to be capable of handling an extended command set called Wireless Extension. Wireless Extension is a set of APIs that allows manipulating any wireless networking device in a standard and uniform way [9]. Even

though Android supports these wireless extension command sets, not all Wi-Fi chip has implemented these commands for design and cost constraint.

Furthermore, when this wireless extension is enabled and supported, other software programs can take advantage of this protocol and enable a phone or device to run in tethered mode. In tethered mode, a phone can act as a Wi-Fi access point. Other client devices can use that access point to hop onto cellular provider's data network to reach Internet. Cellular service providers usually disables their phones from allowing it to be used as an access point as they want customers to pay separately for that feature. Therefore to enable a phone to run the wireless extension, you need to run custom code that bypasses the lock cellular providers have put on the phones.

By default all Wi-Fi modems run in infrastructure mode where they connect to access points to reach other networks. If there is a known network that the user has previously joined, it will connect to it as soon as it finds it available. We need to lock the Wi-Fi chip out of this infrastructure or managed mode. To run the chip in ad-hoc mode, it must not switch to infrastructure mode even if a previously known network is within its reach. Since Android does not support ad-hoc mode directly, the only way to access the ad-hoc mode is to use custom Android OS and have root level access to its wireless modem interface.

5.3 Broad Hardware Spectrum

As Android is an open source product and free of charge, a wide variety of vendors adopted this OS to run their cell phones. With diverse products out in the market also creates a challenge for the MANET application to take control of the Wi-Fi chip and enable ad-hoc mode on it. As running a chip in ad-hoc mode is not governed by Android OS, separate code or driver has to be written to run each chip in this mode. However this task is not trivial as Wi-Fi chip manufacturer do not always divulge their information to general public to allow modified drivers to be written.

5.4 Need for Rooting Android and Installing Custom Kernel

Android ignores all UDP packets when the screen is turn off. For OLSR to continue to work while the device in sleep mode, UDP packets must be processed by the device. To provide the above mentioned criteria, low level hardware access is required by the OLSR application. Android rooting is the process of allowing users of smartphones, tablets, and other devices running

the Android mobile operating system to attain privileged control also known as "root access", within Android's subsystem. On Android, rooting can also facilitate the complete removal and replacement of the device's operating system, usually with a custom release of the variation of the Android operating system.

- Need root to modify iptables / routing tables.
- Need root to configure wireless driver and put phone in ad-hoc mode
- Potential security risk for non-technical users

6 Developing Software for Android

Android software development environment has been improving very rapidly. Google now provides a bundled software (Android Developer Tool) where it combines the Android SDK as a plug-in tool with the development tool called Eclipse. Eclipse is an open source software developed under Apache software license agreement. It is a multi-language software development platform with extensible plug-in system. Once properly configured, Android plug-in for Eclipse can create Android Virtual Device where a piece of software or app can be tested within the virtual environment before deploying it to the devices. Using AVD, Eclipse can emulate the display of various android devices of many screen resolutions to ensure the software being created displays on the target devices correctly without actually loading the software on the target devices during debugging stage.

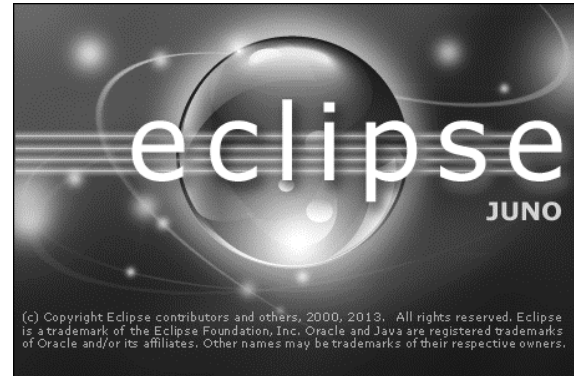


Figure 6-1: Application Development using Eclipse

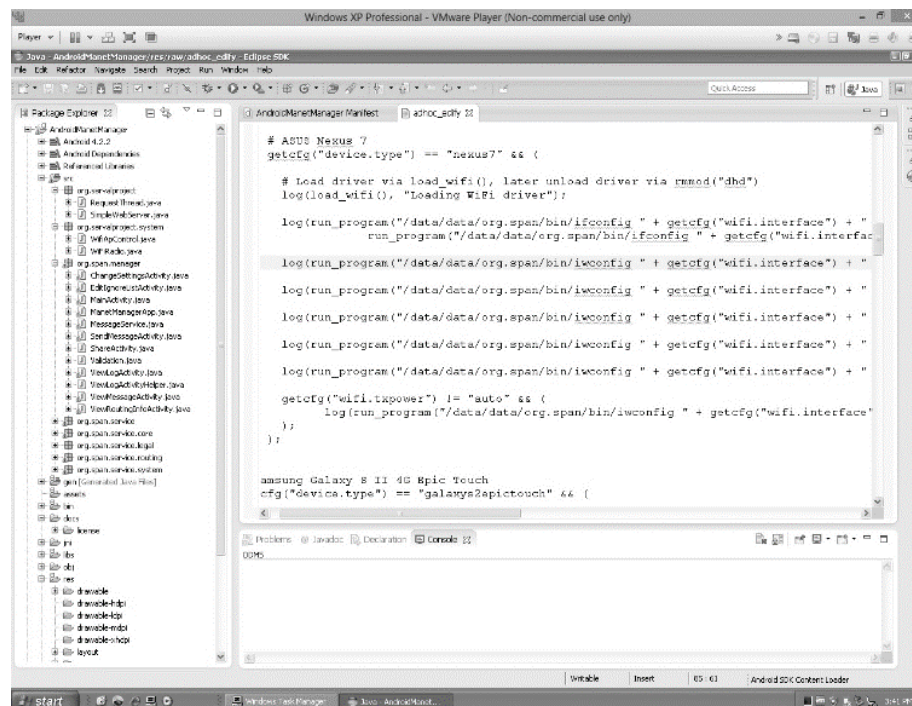


Figure 6-2: Eclipse Development Environment

Even though most of the software can be tested in Android Virtual Device, the OLSR routing protocol cannot be tested properly on virtual devices due to the requirement of communication via wireless modem.

Software developed for Android system can be loaded on the devices in three different ways.

a) Through publishing in Google Play app store where Google manages the application. This is the best way to distribute application as it reaches the masses of Android users.

b) Through sending *.apk file or by using various storage flash drives to upload the application to the devices. This process is only recommended where the application is in testing stage and the received file is from a trusted source for installation.

c) By using the Eclipse software or other software which uses Android Debug Bridge or ADB SHELL access via USB direct connect to a computer.

For this project the process a) is most appropriate as this process allows distribution of the application to millions of end users.

7 Preparing Android for Custom Application

7.1 Process of Installing Custom Kernel

As Android does not support ad-hoc mode operation by default, the Linux kernel needs to be altered. Following procedure details the process of (a) Unlocking, (b) Rooting and (c) installing a custom Linux kernel for Android devices. Since this project used Nexus 7 devices, instructions are specific to that device.

7.2 Enable USB Debugging

- Update Nexus 7 to Android 4.2.2
- Navigate to Settings – About Table
- Tap on “Build Number” for 7 times to enable the Developer Mode in the Android device.
- Go to Settings – Developer options
- Enable USB debugging

7.3 Download Android Root Tool Kit

Nexus Root Tool Kit allows to Unlock and Root a Nexus 7 device. Analogy of unlocking an Android device is like allowing a stranger trusting with your home key. Analogy of rooting an Android device is allowing the stranger to be your head of household who have full access to everything in your house.

- Download Nexus Root Toolkit by WugFresh Development for Windows OS.

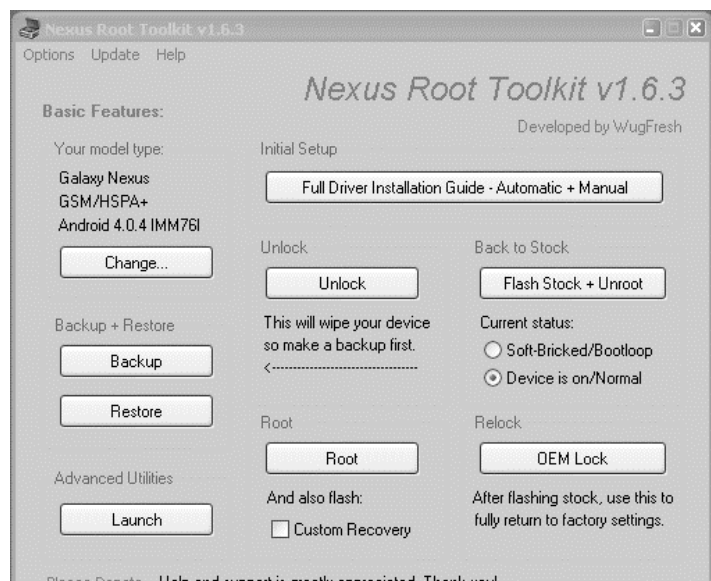


Figure 7-1: Android Root Tool Kit

7.4 Install Driver for Android Devices

- Install the drivers for Nexus 7 using the “Full Driver Installation Guide – Automatic + Manual”. A window such as the Figure 7-2 will open. Perform the three steps below.

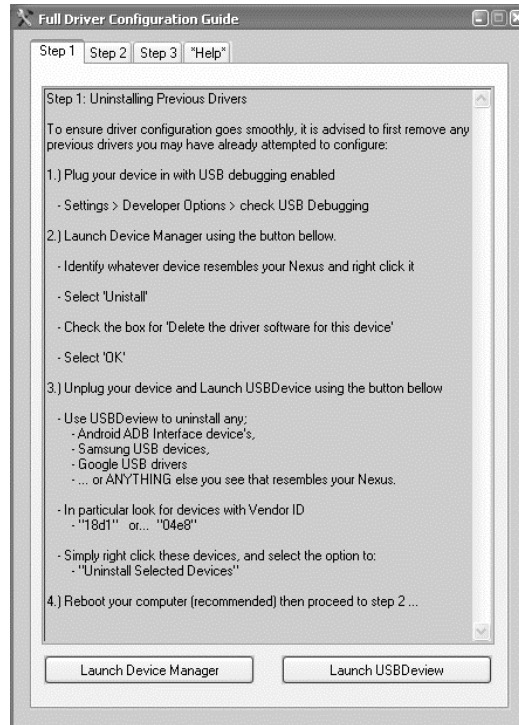


Figure 7-2: Installing Device Drivers and Unlocking Android Devices

- Connect the Nexus 7 to the computer as part of pervious step.
- On Nexus 7 a popup will ask to Allow USB Debugging that the computer is trying to access core Android system. Tap on Allow.
- As part of the test in Step 3, the system will be rebooted into Fast Boot Mode that allows access to boot loader and the Windows OS should be loaded with the proper driver for the next steps. The utility will confirm if the driver loading was successful.

7.5 Unlocking Android Devices

- Next is to “Unlock” the Android device as shown in Figure 7-1. Unlocking is the first process that allows access to the device to eventually gain “root access” and load “custom ROM”. This process wipes the Android device to Factory Default.
- On the Nexus a warning will popup indicating unlocking boot loader may void warranty. Go ahead and tap “Yes”.

- The device will reboot a few times, then follow onscreen instruction. Once it is booted backup, turn on USB Debugging using the steps mentioned above. The NRT software will confirm that the unlocking is complete.

7.6 Rooting Android Devices

Rooting gives necessary permission to any authorized programs to access core system resources that Google Android APIs by default do not support due to security concerns. When a program runs in the context of root user, it can control any device components or access any files bypassing any security Android had implemented. Select Root

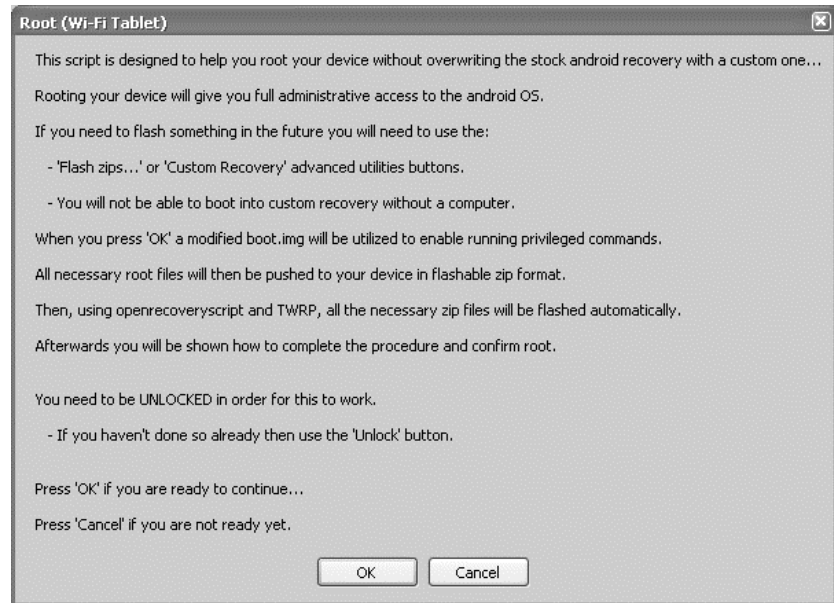


Figure 7-3: Rooting Android Devices

from Figure 7-1 and the following window will pop up. Click OK to start the rooting process. Once the device is rooted, window as Figure 7-4 will ask to confirm if rooting has been successful.

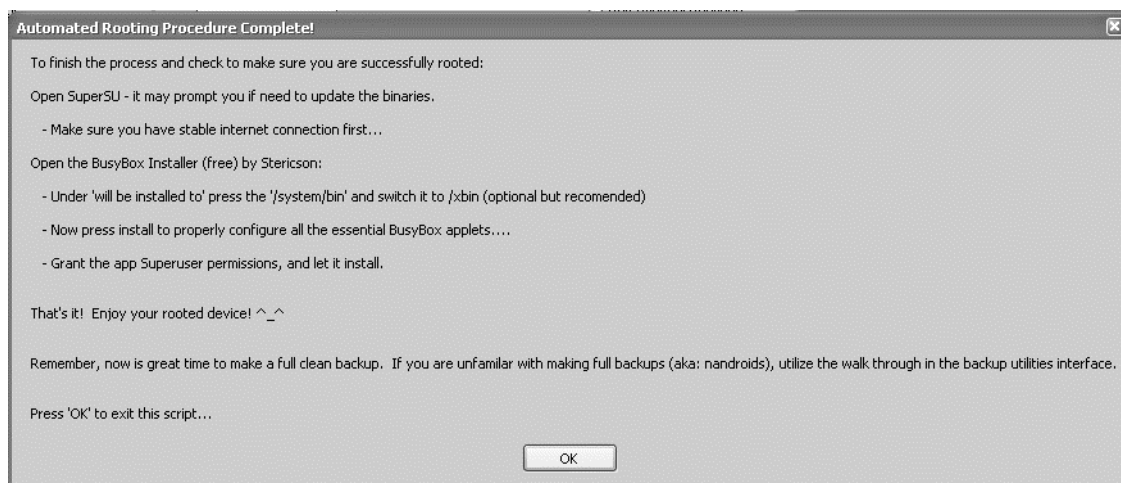


Figure 7-4: Verify Rooting Process

7.7 Install Custom ROM

Download custom kernel/ROM from ProjectSPAN github repository using Chrome. There are device specific custom ROM on the github. For Nexus 7 choose Asus Nexus 7 kernel. The ROM should be downloaded to the folder `/sdcard/download/`.

Download ClockworkMod ROM Manager from the Google Play. Optionally download Android Terminal Emulator.

ROM Manager allows installing custom ROM in Android. To use ROM Manager, you first run the Recovery Setup and follow on screen instructions. If an installation of a custom ROM does not go well, Recovery Setup allows a device to go back to its previous state.

Once Recovery Setup is complete, install the custom kernel that was downloaded previously. In ROM Manager tap Install ROM from SD Card and navigate to Download folder and choose the file `myupdate-nexus7.zip`. Tap on “Reboot And Install”. The device will reboot into ClockworkMod Recovery console and will ask to confirm the installation. Use the volume button to navigate through the menu and use power button to choose the option. Choose “Go Back” to allow the device to reboot.

8 Project SPAN

Smart Phone Ad-hoc Network or SPAN is an open source project that developed the MANET application for Android [5]. The application has two major parts. MANET Manager is the main interface between Android and the user. It allows user to set distinct IP addresses and DNS servers, change default wireless SSID and transmit power of the wireless interface etc. The second piece of the code Olsrd deals with OLSR in particular.

As the main application MANET Manager is launched, the main program starts by invoking `android-MANET-manager-master \ AndroidMANETManager \ src \ org \ span \ manager \ MainActivity.java`. The application then waits for the user to press the Wi-Fi icon to start the ad-hoc service. The application then calls MANETService via MantelServiceHelper code to change the device Wi-Fi mode from infrastructure to ad-hoc mode.

In MANETServiceHelper, `handleStartAdhocCommand()` procedure sets up the device for special power management state. In Android OS, if the screen is turned off, the device ignores all UDP packets. OLSR depends on UDP packets to send and receive data, neighbor hello updates. As the hello packets are periodic and proactive, the Android device has to have the option to receive as well as send UDP packets while the screen is turned off. This is one of the parts where a rooted Android device is necessary to override the default behavior of the OS. At this point `createRoutingProtocol()` start the instance of OLSR for the MANET service.

In `\ android-MANET-manager-master \ AndroidMANETManager \ src \ org \ span \ service \ routing \ OlsrProtocol.java`, the application now checks for the configuration file to start the OLSR protocol against which Wi-Fi interfaces and establishes a gateway to wired connection if there is any. It allows the OLSR daemon to run in the background for continuously polling for neighbors and Multi-Point Relay nodes.

Android allows importing and utilizing some existing codes into Android framework using a toolset called NDK. This NDK toolset allows to implement parts of an app using native-code languages such as C and C++. This is helpful so one can reuse existing code libraries written in these languages, but most apps do not need the Android NDK. Olsrd daemon is not a native

Android application, rather a C++ application ported into Android framework from an open source project under olsr.org foundation. Olsrd code has been separately precompiled under the app named “android-MANET-olsrd”. Basic function of the olsrd daemon is to keep track of 1 and 2 hop neighbors, MPRs and maintaining the routing table by adding or deleting routing entries as neighbors move, enter or leave the OLSR network.

Following are some notable procedure that drives the MANET Manager and OLSR routing protocol in the application.

8.1 **MANETServiceHelper.java**

This program calls the OlsrProtocol procedure to create a new instance of olsr daemon in OlsrProtocol.java program.

getInstance() - Creates a new new MANETServiceHelper instant

8.2 **OlsrProtocol.java**

start(MANETConfig MANETcfg):

- Reads template file conf/olsrd.conf.tpl
- Opens the data file for writing based on the template file
- Starts bin/olsrd based on the data file just created as a background service
- Ignores neighbors in conf/routing_ignore_list.conf
- Waits for 1000 milliseconds for changes to take effect

stop() - Kills the “olsrd” process.

isRunning() - Checks if bin/olsrd process is running

HashSet<Node> getPeers()

- Generates the list of peers
- Uses the routing table as multi-hop peers will not be listed as links or neighbors.
- Calls InfoThread to gather peer info and adds to the peer node table

Class InfoThread

- Run() opens socket to the peer and read their info and writes to the data file. Creates individual thread InfoThread() for each peer and starts to communicate on port 2006
- getError returns error
- getInfo returns info about the peer

8.3 Olsrd

Olsrd is the collection of programs and procedure that is developed as a separate Android program and called from MANET Mangers OlsrProtocol.java as discuss above.

Following are few important functions of Olsrd:

main.c

- Loads OLSR configuration file
- Sets up syslog, initializes message sequence number, various tables, identifies and initialized interfaces, MPR willingness, policy routing settings etc.
- Establishes file lock to prevent multiple olsrd instances
- Creates or opens the socket for routing traffic

tc_set.c

- Generates Topology Change packets and processes incoming TC messages
- Creates, maintains and deletes entries in Topology Information Table
- Detects edge node entries and calculates border by node IPs

duplicate_set.c

- Identifies duplicate messages
- Cleans up broadcast message based on sequence number

neighbor_table.c

- Creates, deletes, expires and maintains 1-hop neighbor table.
- Allows lookup neighbors from the neighbor table.

two_hop_neighbor_table.c

- Creates, delete two hop neighbor table
- Allows lookup of two hop neighbors for MPR calculations

`mpr.c`

- Finds 2 hop neighbors with 1 link
- Finds the neighbor with most 2 hop neighbors with a given willingness.
- Processes the chosen MPRs and updates the counters used in calculations
- Calculates the possible MPR sets.
- Optimizes MPR sets.

`mpr_selector_set.c`

- Calculates and maintains MPR selector sets based on the MPR selected.

`routing_table.c`

- Creates routing entries to be processed by SPF.
- If there is multiple path exist, OLSR breaks ties by selecting the lowest *etx* valued path, then lowest hop count and lastly by IP of the originator.

`olsr.c`

- Pursues OLSR messages from neighbors.
- Evaluates if a message from neighbor should be forwarded, and if so forwards it.
- Updates willingness values to become MPR for the neighbors.

`olsr_spf.c`

- Runs Dijkstra's algorithm to calculate best routes to all nodes in the OLSR network
- Updates the routes for the OS routing table entries.

9 Analysis and Evaluation of Results

In this project we implemented OLSR using three Google Nexus 7 tablets. The devices where unlocked, rooted and custom ROM have been installed before the MANET application was loaded onto the devices.

9.1 **iwconfig**

As the MANET Manager application starts, the application turns off the infrastructure or access point based Wi-Fi interface *wlan0*. Using *iwconfig* utility, the app loads the modem specific driver, assigns device specific IP address and network mask. IP address must be unique and with common network address for a specific ad-hoc WLAN. These settings are read from the config folder of the application. It sets the *wlan0* interface in ad-hoc mode, sets the *ESSID* to common SSID for the Wi-Fi network. It then sets the *Cell ID* of the network to “*ap any*”. The *Cell ID* is a very important concept for the communication between the devices. Even though the SSID is same for all the devices in the network, if Cell IDs are not the same, they will not communicate with each other. Cell ID is a 12 digit hexadecimal number of the access point radio transmitter, similar to MAC address. When a device starts in ad-hoc mode, it scans the network to find other devices with same SSID. If it does not find any other device broadcasting with same SSID, it creates its own Cell ID. Consecutive devices that start to join the same network would look for and will find the same SSID on the network and will join the same Cell ID. Having the same SSID and identical Cell ID allow the devices in the network to listen to and send the messages to each other on the wireless broadcast medium. Next the application also sets the channel and TX power levels as well.

Once the wireless modem driver and necessary parameters are loaded by the application, the devices should now be visible to each other. Using ping utility, it can be verified that if all the devices are within reach of Wi-Fi network, they can be tested to be communicating.

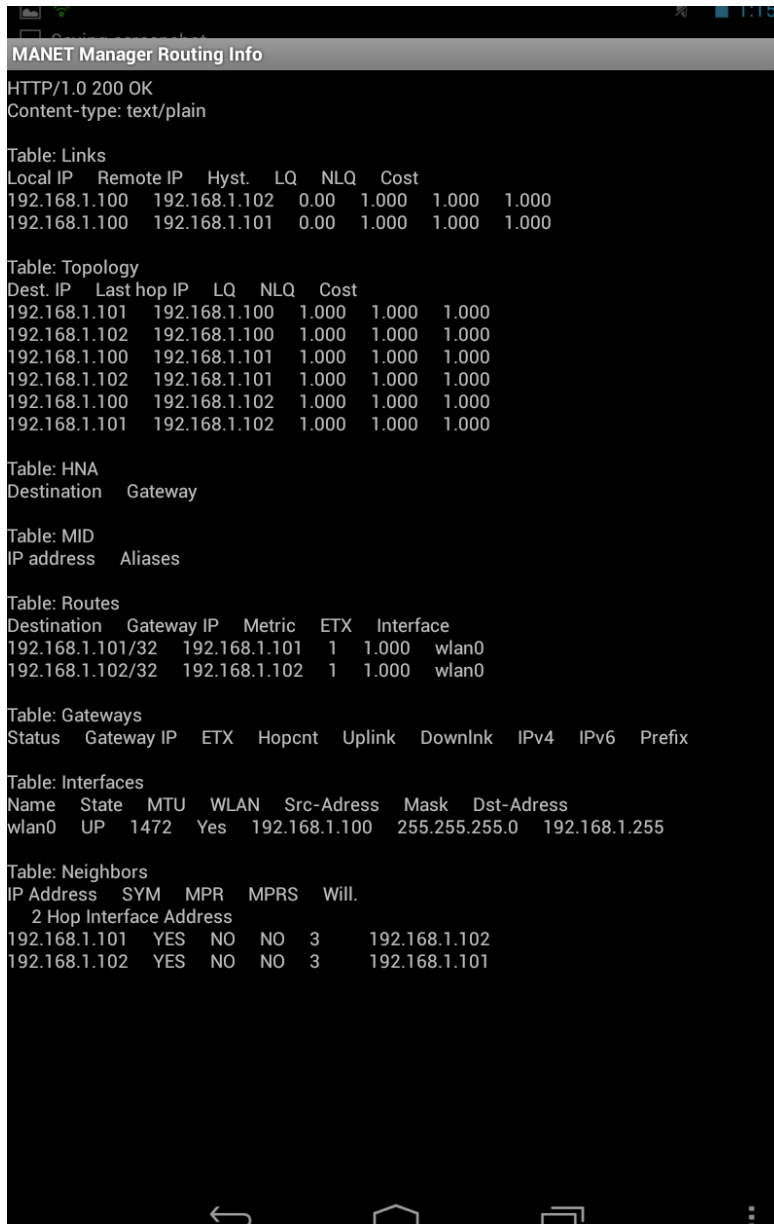
Following diagrams are screenshots from the three Nexus devices which shows SSID, Cell ID and Frequency are identical. By default TX power also has been set to identical value, however that can be changed depending on the requirements.

<p>Device 1</p> <p>IP: 192.168.1.100</p> <p>Sunet Mask: 255.255.255.0</p> <p>SSID: AndroidAdhoc</p> <p>Frequency: 2.412 GHz</p> <p>Cell: 76:09:BF: 6D:09:2A</p> <p>Tx-Power:1496 dBm</p>	 <pre> root@android:/data/data/org.span/bin # root@android:/data/data/org.span/bin # ./iwconfig lo no wireless extensions. dummy0 no wireless extensions. sit0 no wireless extensions. ip6tnl0 no wireless extensions. p2p0 IEEE 802.11bgn ESSID:off/any Mode:Managed Access Point: Not-Associated Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on wlan0 IEEE 802.11bgn ESSID:"AndroidAdhoc" Mode:Ad-Hoc Frequency:2.412 GHz Cell: 76:09:BF:6D:09:2A Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on root@android:/data/data/org.span/bin # </pre>
<p>DEVICE 2</p> <p>IP: 192.168.1.101</p> <p>Sunet Mask: 255.255.255.0</p> <p>SSID: AndroidAdhoc</p> <p>Frequency: 2.412 GHz</p> <p>Cell: 76:09:BF: 6D:09:2A</p> <p>Tx-Power:1496 dBm</p>	 <pre> root@android:/data/data/org.span/bin # root@android:/data/data/org.span/bin # ./iwconfig lo no wireless extensions. dummy0 no wireless extensions. sit0 no wireless extensions. ip6tnl0 no wireless extensions. p2p0 IEEE 802.11bgn ESSID:off/any Mode:Managed Access Point: Not-Associated Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on wlan0 IEEE 802.11bgn ESSID:"AndroidAdhoc" Mode:Ad-Hoc Frequency:2.412 GHz Cell: 76:09:BF:6D:09:2A Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on root@android:/data/data/org.span/bin # </pre>
<p>DEVICE 3</p> <p>IP: 192.168.1.102</p> <p>Sunet Mask: 255.255.255.0</p> <p>SSID: AndroidAdhoc</p> <p>Frequency: 2.412 GHz</p> <p>Cell: 76:09:BF: 6D:09:2A</p> <p>Tx-Power:1496 dBm</p>	 <pre> 127 root@android:/data/data/org.span/bin # 127 root@android:/data/data/org.span/bin # ./iwconfig lo no wireless extensions. dummy0 no wireless extensions. sit0 no wireless extensions. ip6tnl0 no wireless extensions. p2p0 IEEE 802.11bgn ESSID:off/any Mode:Managed Access Point: Not-Associated Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on wlan0 IEEE 802.11bgn ESSID:"AndroidAdhoc" Mode:Ad-Hoc Frequency:2.412 GHz Cell: 76:09:BF:6D:09:2A Tx-Power=1496 dBm Retry long limit:7 RTS thr:off Fragment thr:off Encryption key:off Power Management:on root@android:/data/data/org.span/bin # </pre>

9.2 MANET Manger Routing Info

Once the initial communication is established, MANET Manager then invokes the Olsrd routine. Olsrd sends out hello packets to its neighbors, establishes two-way symmetric link. Following diagrams shows the devices all within 1 hop reach of each other.

Links Table: Olsrd identifies its own node as local IP of 192.168.1.100 and recognizes the two other neighboring nodes 192.168. 1.100 and 192.168.1.102 are both 1 hop away as it populates its link table.



```
MANET Manager Routing Info
HTTP/1.0 200 OK
Content-type: text/plain

Table: Links
Local IP Remote IP Hyst. LQ NLQ Cost
192.168.1.100 192.168.1.102 0.00 1.000 1.000 1.000
192.168.1.100 192.168.1.101 0.00 1.000 1.000 1.000

Table: Topology
Dest. IP Last hop IP LQ NLQ Cost
192.168.1.101 192.168.1.100 1.000 1.000 1.000
192.168.1.102 192.168.1.100 1.000 1.000 1.000
192.168.1.100 192.168.1.101 1.000 1.000 1.000
192.168.1.102 192.168.1.101 1.000 1.000 1.000
192.168.1.100 192.168.1.102 1.000 1.000 1.000
192.168.1.101 192.168.1.102 1.000 1.000 1.000

Table: HNA
Destination Gateway

Table: MID
IP address Aliases

Table: Routes
Destination Gateway IP Metric ETX Interface
192.168.1.101/32 192.168.1.101 1 1.000 wlan0
192.168.1.102/32 192.168.1.102 1 1.000 wlan0

Table: Gateways
Status Gateway IP ETX Hopcnt Uplink Downlnk IPv4 IPv6 Prefix

Table: Interfaces
Name State MTU WLAN Src-Address Mask Dst-Address
wlan0 UP 1472 Yes 192.168.1.100 255.255.255.0 192.168.1.255

Table: Neighbors
IP Address SYM MPR MPRS Will.
2 Hop Interface Address
192.168.1.101 YES NO NO 3 192.168.1.102
192.168.1.102 YES NO NO 3 192.168.1.101
```

Figure 9-1: MANET Manger Routing Info

Topology Table: In its Topology Table, it shows all advertised distances from its neighbors, which in this case all are 1.

Routes Table: In Routes Table, it registers the destination address, the next hop IP address, number of hops to the destination. In MANET, all nodes are in one network, therefore all destination IP will show with /32 subnet mask. The metric changes as the hop count increases to the destination network or node.

Gateways Table: Devices with multiple interfaces can act as gateways. If one of the wireless devices also has a wired interface, that interface can act a gateway to other networks. Gateways table lists the Status, IP, ETX, hop count, uplink and downlink speed, IPv4 and/or IPv6 support and the prefix of the network.

Interface Table: Interface table lists the interfaces that are participating in the OLSR routing protocol. It shows the name of the interface, current status, MTU, if it is a wireless or wired connection, IP address, subnet mask and broadcast address of the cards.

Neighbor Table: Neighbor table lists the IP address of its immediate neighbors, if the link to the neighbor is currently symmetric (SYM), if it is currently acting as multi-point relay (MPR). If it is not an MPR, it can be member of the MPR Selector Set (MPRS), willingness to become MPRs (Will) and finally the 2-hop IP address - the neighbor's neighbor, if any.

9.3 OLSR with no MPRs

Following Figure 9-2 shows the status of three devices where all the devices are 1-hop neighbors of each other. The Neighbors Table shows that the links are symmetric but there are no MPRs elected as all devices are able to directly communicate with the others.

MANET Manager Routing Info
 HTTP/1.0 200 OK
 Content-type: text/plain

Table: Links

Local IP	Remote IP	Hyst.	LQ	NLQ	Cost
192.168.1.101	192.168.1.102	0.00	1.000	1.000	1.000
192.168.1.101	192.168.1.100	0.00	1.000	1.000	1.000

Table: Topology

Dest. IP	Last hop IP	LQ	NLQ	Cost
192.168.1.101	192.168.1.100	1.000	1.000	1.000
192.168.1.102	192.168.1.100	1.000	1.000	1.000
192.168.1.100	192.168.1.101	1.000	1.000	1.000
192.168.1.102	192.168.1.101	1.000	1.000	1.000
192.168.1.100	192.168.1.102	1.000	1.000	1.000
192.168.1.101	192.168.1.102	1.000	1.000	1.000

Table: HNA

Destination	Gateway
192.168.1.100/32	192.168.1.100
192.168.1.102/32	192.168.1.102

Table: MID

IP address	Aliases
192.168.1.100	
192.168.1.102	

Table: Routes

Destination	Gateway IP	Metric	ETX
192.168.1.100/32	192.168.1.100	1	1.0
192.168.1.102/32	192.168.1.102	1	1.0

Table: Gateways

Status	Gateway IP	ETX	Hopcnt	Uplink
1	192.168.1.100	1	1	1
1	192.168.1.102	1	1	1

Table: Interfaces

Name	State	MTU	WLAN	Src-Address
wlan0	UP	1472	Yes	192.168.1.101

Table: Neighbors

IP Address	SYM	MPR	MPRS	Will.
192.168.1.100	YES	NO	NO	3
192.168.1.102	YES	NO	NO	3

MANET Manager Routing Info
 HTTP/1.0 200 OK
 Content-type: text/plain

Table: Links

Local IP	Remote IP	Hyst.	LQ	NLQ	Cost
192.168.1.100	192.168.1.102	0.00	1.000	1.000	1.000
192.168.1.100	192.168.1.101	0.00	1.000	1.000	1.000

Table: Topology

Dest. IP	Last hop IP	LQ	NLQ	Cost
192.168.1.101	192.168.1.100	1.000	1.000	1.000
192.168.1.102	192.168.1.100	1.000	1.000	1.000
192.168.1.100	192.168.1.101	1.000	1.000	1.000
192.168.1.102	192.168.1.101	1.000	1.000	1.000
192.168.1.100	192.168.1.102	1.000	1.000	1.000
192.168.1.101	192.168.1.102	1.000	1.000	1.000

Table: HNA

Destination	Gateway
192.168.1.101/32	192.168.1.101
192.168.1.102/32	192.168.1.102

Table: MID

IP address	Aliases
192.168.1.101	
192.168.1.102	

Table: Routes

Destination	Gateway IP	Metric	ETX	Interface
192.168.1.101/32	192.168.1.101	1	1.000	wlan0
192.168.1.102/32	192.168.1.102	1	1.000	wlan0

Table: Gateways

Status	Gateway IP	ETX	Hopcnt	Uplink	Downlink	IPv4	IPv6	Prefix
1	192.168.1.101	1	1	1	1			
1	192.168.1.102	1	1	1	1			

Table: Interfaces

Name	State	MTU	WLAN	Src-Address	Mask	Dst-Address
wlan0	UP	1472	Yes	192.168.1.100	255.255.255.0	192.168.1.255

Table: Neighbors

IP Address	SYM	MPR	MPRS	Will.
192.168.1.101	YES	NO	NO	3
192.168.1.102	YES	NO	NO	3

Figure 9-2: OLSR in Fully Meshed Network

9.4 OLSR in Multi-hop Network

As OLSR is a self-organizing routing protocol, as the devices are taken further apart, the two edge devices lose direct neighbor relationship with each other. Following Figure 9-3 shows the two devices 192.168.1.101 and 191.168.1.102 are on the edge of the ad-hoc network and 192.168.1.100 is residing in the middle.

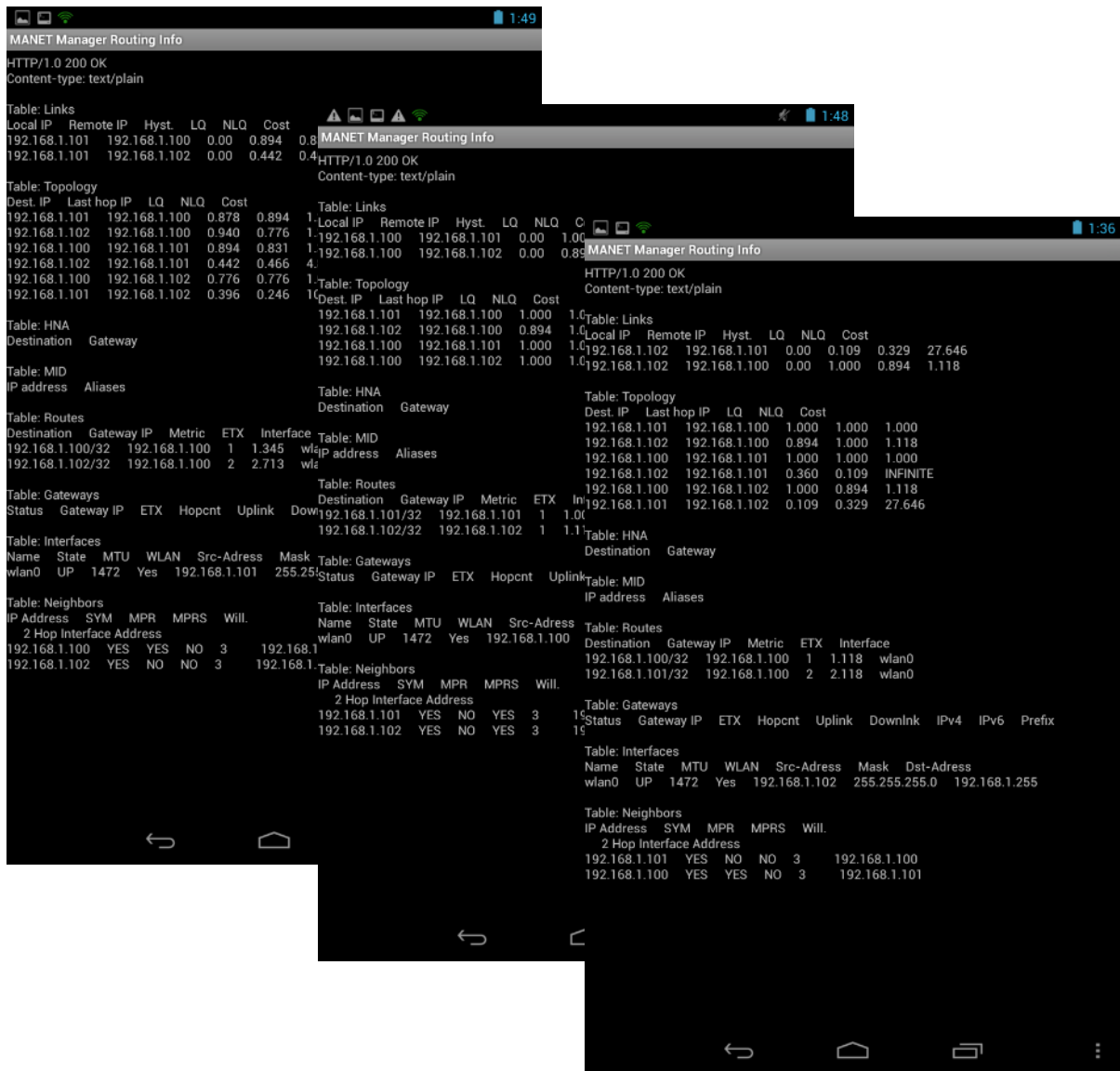


Figure 9-3: OLSR with MPR elected

Device 101 lost neighbor relationship with Device 102. However Device 100 is still neighbor to both devices. Although Device 101 and Device 102 do not have direct relationship with each other, they are still able to communicate by going through Device 100. We can observe

that in the Neighbor Table of Device 101 and 102, Device 100 is listed as an MPR. In the “Routes Table” of 101 and 102 they show the metric to reach each other has changed from 1 to 2. Here Device 100 is converted to work as a router for the two edge devices as it has been elected as MPR. The middle device 19.168.1.100 shows that the two neighbors on the edge have become member of the MPR Selector Set (MPRS).

9.5 OLSR Debug Information

OLSR provides access to debug information via an http portal on port 2006 using “txtinfo” plugin [1]. Following commands are supported in OLSRd.

- Config: `"/config" -> send_what=SIW_CONFIG`
- Gateways: `"/gateway" -> send_what=SIW_GATEWAY`
- HNA: `"/hna" -> send_what=SIW_HNA`
- Interfaces: `"/interface" -> send_what=SIW_INTERFACE`
- Links: `"/link" -> send_what=SIW_LINK`
- MID: `"/mid" -> send_what=SIW_MID`
- Neighbors: `"/neigh" -> send_what=SIW_NEIGH`
- Routes: `"/route" -> send_what=SIW_ROUTE`
- Topology: `"/topo" -> send_what=SIW_TOPO`
- 2-hop neighbors: `"/2hop" -> send_what=SIW_2HOP`

```

u0_a70@android:/ $ wget
u0_a70@android:/ $ wget -O - http://localhost:2006/links
Connecting to localhost:2006 (127.0.0.1:2006)
Table: Links
Local IP      Remote IP    Hyst.  LQ      NLQ      Cost
192.168.1.101 192.168.1.102 0.00   1.000   1.000   1.000
-
100% |*****| 103 0:00:00 ETA
u0_a70@android:/ $ wget -O - http://localhost:2006/neigh
Connecting to localhost:2006 (127.0.0.1:2006)
Table: Neighbors
IP Address    SYM    MPR    MPRS    Will.  2 Hop Neighbors
192.168.1.102 YES    NO     NO      3      0
-
100% |*****| 92 0:00:00 ETA
u0_a70@android:/ $ wget -O - http://localhost:2006/route
Connecting to localhost:2006 (127.0.0.1:2006)
Table: Routes
Destination   Gateway IP   Metric  ETX      Interface
192.168.1.102/32 192.168.1.102 1        1        wlan0
-
100% |*****| 105 0:00:00 ETA
u0_a70@android:/ $

```

Figure 9-4: OLSR Debugging

Using WGET from the command line, the following tables can be accessed and analyzed at any time. As Figure 9-4 shows, various tables can be listed using the command format as below:

wget -O - http://localhost:2006/topo

9.6 ETX and Delay Comparison for Multi-Hop MANET

Using PING utility, following delay statistics have been captured. This is not a comprehensive statistics as many variables will affect the delay in the network, such as, number of nodes, number of hops, TX power, RX sensitivity, temperature, interference etc. Varying the Wi-Fi card's transceiver power can extend the range the following table shows. The power range can be set to automatic or preconfigured from 0 to 1258mW.

These statistics in Figure 9-5 and Figure 9-6 have been gathered where the devices are away from any other Wi-Fi devices in open space. Interference, contention and retransmission is minimal in this environment. When similar test were done in vicinity of other Wi-Fi access points and clients, due to the interference, the distance reachability greatly reduced.

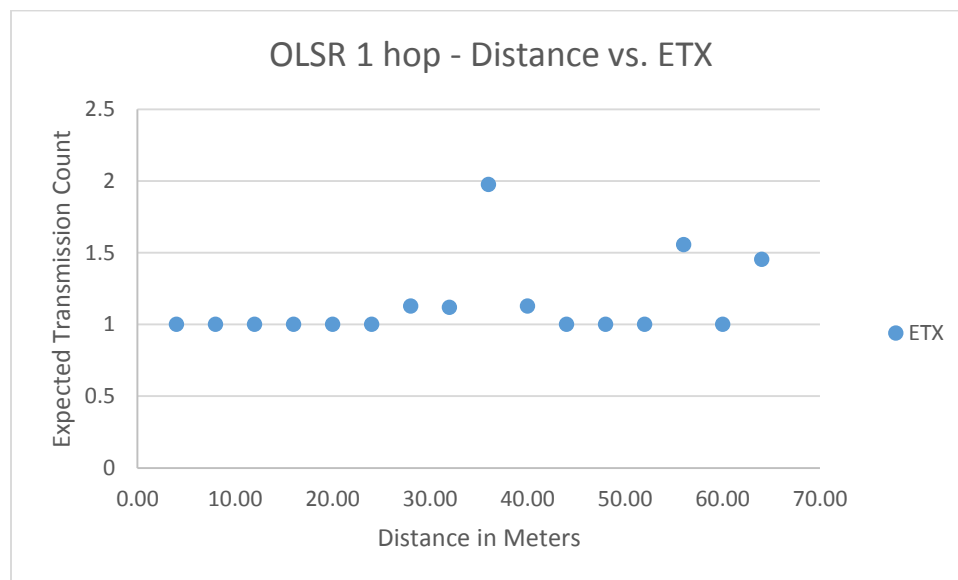


Figure 9-5: Distance vs. ETX

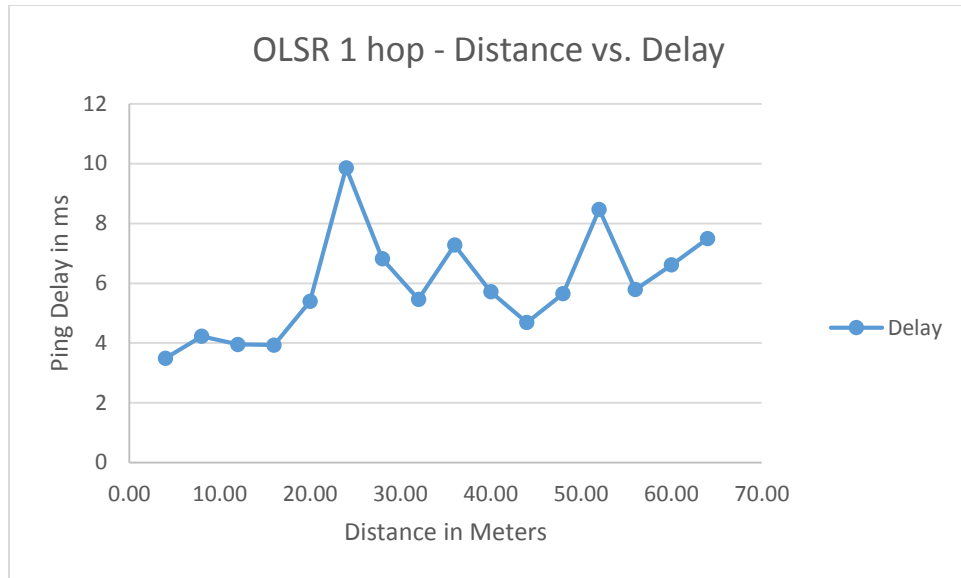


Figure 9-6: Distance vs. Delay

To ensure one of the device acts an MPR we next have placed the three device in an L shaped formation behind concrete building corner where the two furthest devices are reachable by any means other than using the corner device as a router. When OLSR formed neighbor relationship between three devices, the middle device in the corner have been elected as MPR. Following figures show

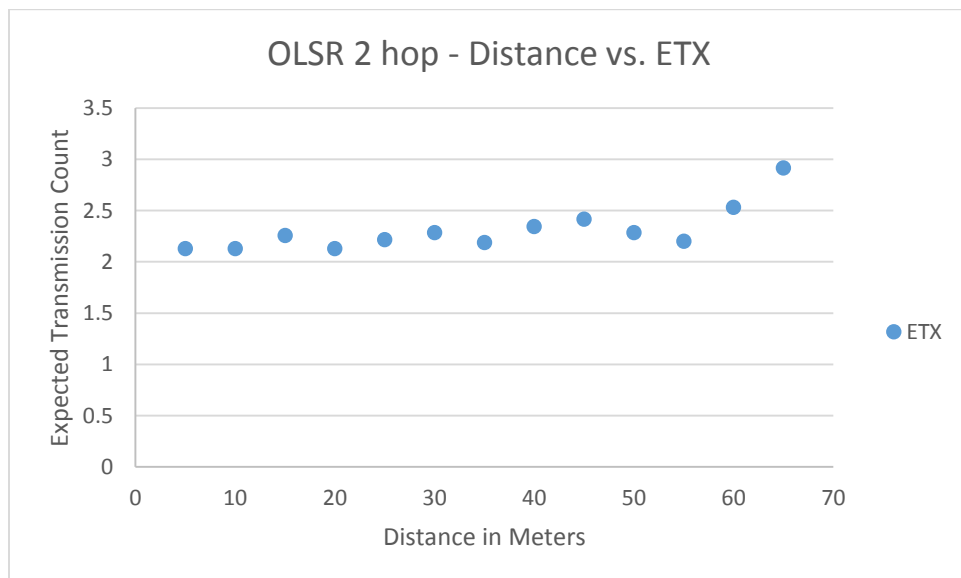


Figure 9-7: OLSR 2 hop - Distance vs. ETX

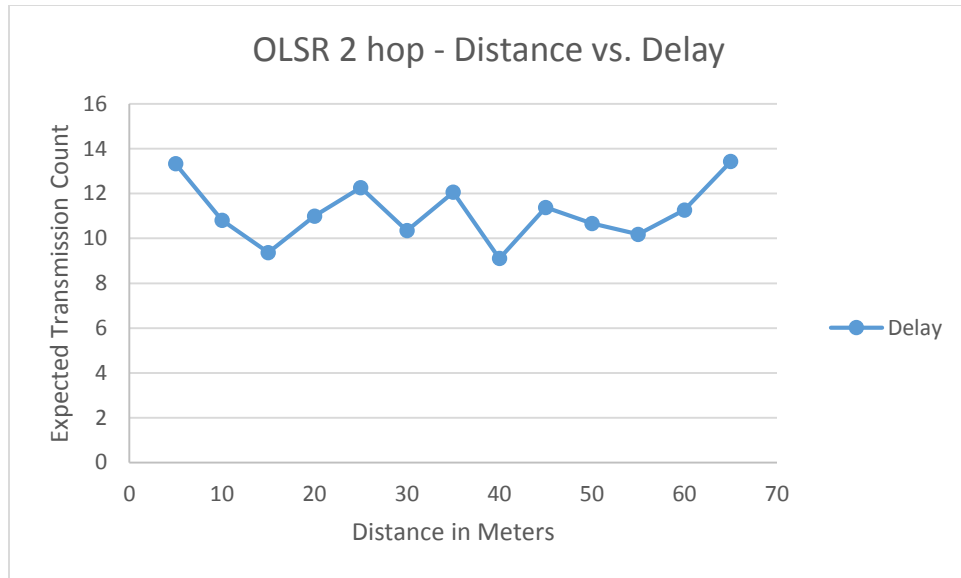


Figure 9-8: OLSR 2 hop - Distance vs. Delay

It has been observed that in open space, the OLSR in MANET Manager is unable to detect high packet loss ratio between distant neighbors and would not form MPR when three devices are kept in equal distance in a straight line. During the tests, packet loss reached up to 45%, however the OLSR would not elect the middle device as MPR.

10 Conclusion and Future Work

10.1 Conclusion

Android Operating System has attracted an increasing number of researchers due to its ubiquitous nature, easy deployment and a wide range of applications. Ad-hoc networks that can provide instant network in absence of infrastructure has always intrigued many. This project describes the process of implementing the Optimized Links State Routing (OLSR), an ad-hoc routing protocol in Android Operating System using Google Nexus 7 devices.

To get some basic understanding of OLSR and Android operating system, we provided a brief introduction to Multi-hop Ad-hoc Network, OLSR protocol and a development tool for Android OS. We have discussed the challenges of implementing ad-hoc wireless communication using Android devices and some procedures to overcome the problems by using various techniques.

We have modeled a test bed of three Android Nexus 7 devices to demonstrate the functionality of the OLSR implementation. Using the Android graphical interface application, we validated dynamic nature of the OLSR protocol. We demonstrated multi-hop wireless communication in ad-hoc mode. Neighbor relationship forms and changes dynamically as the devices are moved around in and out of the wireless range. Depending on link quality, we observed how ETX metric changes and reflects on the routing table.

10.2 Future Work

As future work one can investigate the following:

- Developing the graphical user interface to display debug information for establishing neighbor relationships and MPR election process.
- Port the application to OLSRv2 or B.A.T.M.A.N.
- Port the code to easily implement MANET in iOS, Blackberry and Windows devices.
 - Integrate security in the application with robust authentication, integrity and confidentiality.

11 Bibliography

- [1] "olsrd - an adhoc wireless routing daemon," [Online]. Available: <http://olsr.org/>. [Accessed Jun 2102].
- [2] Google Inc., "API Guide for Wi-Fi Direct," [Online]. Available: <http://developer.android.com/guide/topics/connectivity/wifip2p.html#api>. [Accessed 05 2012].
- [3] "The Serval Project," [Online]. Available: <http://www.servalproject.org/>. [Accessed Jun 2013].
- [4] "OLSR-Mesh-Tether," The Guardian Project, [Online]. Available: <https://github.com/guardianproject/olsr-mesh-tether>. [Accessed Jul 2012].
- [5] J. Robble, "The SPAN Project," 01 09 2012. [Online]. Available: <https://github.com/ProjectSPAN>. [Accessed 01 Sep 2012].
- [6] E. T. Clausen and E. P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," May 2013. [Online]. Available: <http://www.ietf.org/rfc/rfc3626.txt>.
- [7] D. Couto, D. Aguayo, J. Bicket and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *MobiCom*, San Diego, California, 2003.
- [8] J. Thomas, J. Robble and N. Modly, "Off Grid communications with Android," in *2012 IEEE Conference on Technologies for Homeland Security (HST)*, Waltham, MA, 2012.
- [9] J. Tourrilhes, "Wireless Extensions for Linux," 23 Jan 1997. [Online]. Available: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html. [Accessed 20 May 2013].
- [10] A. Tønnesen, "Impementing and extending the Optimized Link State Routing Protocol," Oslo, 2004.
- [11] J. Klein, "Implementation of an ad-hoc routing module for an experimental network," Barcelona and Stuttgart, 2005.
- [12] J. Schiller, *Mobile Communication*, Essex: Pearson Education Limited, 2000.

12 Appendix

OLSR on Android – ProjectSPAN Google Nexus 7 wireless NIC configuration file:

AndroidManetManager/res/raw/adhoc_edify

```
# This script controls actions to be taken when ad-hoc mode is started or stopped.
# -----
# It uses the "edify" language, which is also used for the android OTA update scripts.
# See:
#
http://android.git.kernel.org/?p=platform/bootable/recovery.git;a=tree;f=edify;h=04720f8aaa9a5e0079b79f8be7f11b7f74414162;hb=HEAD
# -----
# NOTE: Must do a project clean and rebuild then completely uninstall app. from device
# and reinstall it for changes made in this file to take effect.

# Actions for starting ad-hoc mode
action() == "start" && (

    # Set "status" property
    setprop("adhoc.status", "running");

    # Remove all iptables rules
    run_program("/data/data/org.span/bin/iptables -t filter -F");
    run_program("/data/data/org.span/bin/iptables -t nat -F");

    # Enable forwarding
    log(run_program("echo 1 > /proc/sys/net/ipv4/ip_forward"), "Enabling kernel packet forwarding");
    run_program("/data/data/org.span/bin/iptables -A FORWARD -j ACCEPT");

    # Wifi mode, do some wifi things...
    getcfg("adhoc.mode") == "wifi" && (

        # Set "status"-Property
        setprop("adhoc.mode", "wifi");

        # Google Nexus 7
        getcfg("device.type") == "nexus7" && (

            # Load driver via load_wifi(), later unload driver via rmmmod("dhd")
            log(load_wifi(), "Loading WiFi driver");
```

```

log(run_program("/data/data/org.span/bin/ifconfig " + getcfg("wifi.interface") + " " +
getcfg("ip.address") + " netmask " + getcfg("ip.netmask")) &&
run_program("/data/data/org.span/bin/ifconfig " + getcfg("wifi.interface") + " up"),
"Activating WiFi interface");

log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " mode ad-
hoc"), "Setting ad-hoc mode");

log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " essid " +
getcfg("wifi.essid")), "Setting essid");

log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " ap any"),
"Setting cell id");

log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " channel "
+ getcfg("wifi.channel")), "Setting channel");

log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " power
all"), "Setting power management");

getcfg("wifi.txpower") != "auto" && (
log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + "
txpower " + getcfg("wifi.txpower")), "Setting transmit power");
);
);

# Gateway
getcfg("gateway.interface") != "none" && (

# log(run_program("/data/data/org.span/bin/iptables -A FORWARD ! --dst " +
getcfg("ip.address") + " -i " + getcfg("wifi.interface") + " -o " + getcfg("gateway.interface") + "
-j ACCEPT") &&

log(run_program("/data/data/org.span/bin/iptables -A FORWARD -i " +
getcfg("wifi.interface") + " -o " + getcfg("gateway.interface") + " -j ACCEPT") &&
run_program("/data/data/org.span/bin/iptables -A FORWARD -i " +
getcfg("gateway.interface") + " -o " + getcfg("wifi.interface") + " -j ACCEPT"),
"Setting gateway forwarding");

log(run_program("/data/data/org.span/bin/iptables -t nat -A POSTROUTING -o " +
getcfg("gateway.interface") + " -j MASQUERADE"), "Enabling NAT");
);

#
# WEP-Encryption
#
getcfg("wifi.encryption.algorithm") == "wep" && (
getcfg("wifi.encryption.setup") == "iwconfig" && (

```

```

    log(run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " key s:"
+ getcfg("wifi.encryption.password") + "")) &&
    run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " key
restricted"), "Activating encryption (iwconfig)");
    run_program("/data/data/org.span/bin/iwconfig " + getcfg("wifi.interface") + " commit");
);
getcfg("wifi.encryption.setup") == "wpa_supplicant" && (
    sleep("2");
    log(run_program("cd /data/local/tmp; mkdir /data/local/tmp/wpa_supplicant;
wpa_supplicant -B -D" +
    getcfg("wifi.driver") + " -i" + getcfg("wifi.interface") +
    " -c/data/data/org.span/conf/wpa_supplicant.conf"), "Activating encryption
(wpa_supplicant)");
);
);

# getcfg("adhoc.mode") == "bt" && (
# #
# # Set "mode"-Property
# #
# setprop("adhoc.mode","bt");
# sleep("3");
# #
# # Bluetooth - start pand
# #
# run_program("/data/data/org.span/bin/pand --listen --role NAP " +
#     "--devup /data/data/org.span/bin/blue-up.sh " +
#     "--devdown /data/data/org.span/bin/blue-down.sh " +
#     "--pidfile /data/data/org.span/var/pand.pid");
# );

# #
# # Remove old rules
# #
# run_program("/data/data/org.span/bin/iptables -N wireless-tether");
# run_program("/data/data/org.span/bin/iptables -F wireless-tether");
# run_program("/data/data/org.span/bin/iptables -t nat -F PREROUTING");
# run_program("/data/data/org.span/bin/iptables -t nat -F POSTROUTING");
# run_program("/data/data/org.span/bin/iptables -t nat -F");

# #
# # Bring up NAT rules
# #
# log(

```

```

#      run_program("/data/data/org.span/bin/iptables -A wireless-tether -m state --state
ESTABLISHED,RELATED -j ACCEPT") &&
#      run_program("/data/data/org.span/bin/iptables -A wireless-tether -s " + getcfg("ip.network")
+ "/24 -j ACCEPT") &&
#      run_program("/data/data/org.span/bin/iptables -A wireless-tether -p 47 -j ACCEPT") &&
##      run_program("/data/data/org.span/bin/iptables -A wireless-tether -j DROP"),
#      run_program("/data/data/org.span/bin/iptables -A wireless-tether -j DROP") &&
#      run_program("/data/data/org.span/bin/iptables -A FORWARD -m state --state INVALID -j
DROP") &&
#      run_program("/data/data/org.span/bin/iptables -A FORWARD -j wireless-tether") &&
#      run_program("/data/data/org.span/bin/iptables -t nat -I POSTROUTING -s " +
#          getcfg("ip.network") + "/24 -j MASQUERADE"),
#      "Enabling NAT rules");

# #
# # IP forwarding
# #
# log(file_write("/proc/sys/net/ipv4/ip_forward", "1"), "Enabling IP forwarding");

# #
# # dnsmasq for wifi tether (bluetooth has pand start it)
# #
# getcfg("adhoc.mode") == "wifi" &&
#      run_program("/data/data/org.span/bin/dnsmasq -i " + getcfg("wifi.interface") + " "+
#          "--resolv-file=/data/data/org.span/conf/resolv.conf " +
#          "--conf-file=/data/data/org.span/conf/dnsmasq.conf");

log("Ad-hoc mode now running");
);
# Actions when stopping ad-hoc mode
action() == "stop" && (

# Set "status" property
setprop("adhoc.status", "stopped");

# Remove all iptables rules
run_program("/data/data/org.span/bin/iptables -t filter -F");
run_program("/data/data/org.span/bin/iptables -t nat -F");

# #
# # Disable forwarding and remove NAT rules.
# #
# log(file_write("/proc/sys/net/ipv4/ip_forward", "0"), "Disabling forwarding");
#
# log(
#      run_program("/data/data/org.span/bin/iptables -D FORWARD -j wireless-tether") &&

```

```

# run_program("/data/data/org.span/bin/iptables -D FORWARD -m state --state INVALID -j DROP") &&
# run_program("/data/data/org.span/bin/iptables -F wireless-tether") &&
# run_program("/data/data/org.span/bin/iptables -X wireless-tether") &&
# run_program("/data/data/org.span/bin/iptables -t nat -F PREROUTING") &&
# run_program("/data/data/org.span/bin/iptables -t nat -F POSTROUTING") &&
# run_program("/data/data/org.span/bin/iptables -t nat -F"),
# "Disabling NAT rules");

# #
# # Bluetooth, kill pand and and dnsmasq processes
# #
# getcfg("adhoc.mode") == "bt" && (
# run_program("/data/data/org.span/bin/pand -K");
# sleep("1");
# kill_process("pand");
# file_unlink("/data/data/org.span/var/pand.pid");
# kill_process("dnsmasq");
# kill_pidfile("/data/data/org.span/var/fixroute.pid");
# file_unlink("/data/data/org.span/var/fixroute.pid");
# );

# Wifi mode, bring interface down, kill dnsmasq/wpa_supplicant, remove module
getcfg("adhoc.mode") == "wifi" && (
run_program("/data/data/org.span/bin/ifconfig " + getcfg("wifi.interface") + " down");
# kill_process("wpa_supplicant");
# kill_process("dnsmasq");

# # place wifi back under framework control
# unload_wifi() && load_wifi();

# module_loaded("bcm4329") && rmmod("bcm4329");
# module_loaded("bcm4325") && rmmod("bcm4325");
# module_loaded("wlan") && rmmod("wlan");
# module_loaded("tiwlan_drv") && rmmod("tiwlan_drv");
# module_loaded("tiap_drv") && rmmod("tiap_drv");
# module_loaded("sdio") && rmmod("sdio");
module_loaded("dhd") && rmmod("dhd");
# module_loaded("wireless") && rmmod("wireless");
# module_loaded("ar6000") && rmmod("ar6000");
);

log("Ad-hoc mode now stopped");
);

```