Master thesis

Implementation of an ad-hoc routing module for an experimental network

by Joachim Klein

Submitted in partial fulfillment of the requirements for the double diploma degree program between the Universität Stuttgart and the Universidat Politècnica de Catalunya.

Main Advisor:

Jorge García Vidal

Universidat Politècnica de Catalunya
Department of Computer Architecture
Office D6-102 Campus Nord
C. Jordi Girona 1-3
08034 Barcelona
Spain

Co-advisor:

Prof. Dr.-Ing. Dr. h. c. mult. Paul J. Kühn

Universität Stuttgart
Institute of Communication Networks and Computer Engineering
Pfaffenwaldring 47
70569 Stuttgart
Germany

Issued: September 2004
Submitted: April 2005

Abstract

This thesis describes the implementation of the ad-hoc routing protocol “Optimized Link State Routing” (OLSR) within the Click Modular Router framework and the design of a wireless test bed based upon this implementation.

The code is divided into modular elements written in C++ combined by the flexible Click configuration language. The router can be executed in user level or kernel level as well as in the network simulator ns.

Performance analysis of the OLSR implementation in Click showed that the single threaded router environment of Click can lead to blocking of the router for packet forwarding while executing route maintenance tasks. This applies especially to large and dense networks with high node mobility.

Based upon this OLSR implementation a test bed of nodes with wireless interfaces was designed. The test bed consists of outdoor nodes installed on the roof of campus buildings as well as ordinary PC equipped with wireless interfaces. Embedded devices within a weatherproof enclosure to be installed on the roof of campus buildings were successfully configured to automatically run the developed OLSR Click code.

Resumen

Este proyecto de fin de carrera describe la implementación del protocolo de routing ad-hoc “Optimized Link State Routing” (OLSR) dentro del Click Modular Router y el desarrollo de una red inalámbrica experimental basada en este código.

El código fuente esta dividido en elementos modulares escritos en c++ que se conectan a través del lenguaje flexible de configuración de Click. El router se puede ejecutar a nivel de usuario, a nivel de núcleo o simulado dentro del simulador de redes ns.

La investigación del tiempo de ejecución del código implementado dentro del Click Modular Router demostraba que la ejecución de solo un thread dentro del Click Modular Router puede resultar en un bloqueo del router mientras este es ocupado con rutinas de mantenimiento de OLSR. Esto especialmente aplica para redes largas y densas con un nivel alto de movilidad.

Basado en este código de OSLR implementado en Click una red experimental inalámbrica fue desarrollado. Esta red experimental consta de nodos instalados en los tejados de los edificios del campo universitario que están equipados con una tarjeta inalámbrica. Estos nodos son pequeños embedded devices dentro de una caja resistente al agua configurados de tal manera que ejecutan automáticamente el código de encaminamiento OLSR.

Zusammenfassung

Diese Diplomarbeit beschreibt die Implementierung des Ad-hoc Routingprotokolls “Optimized Link State Routing” (OLSR) innerhalb des Click Modular Router Framework sowie den Entwurf eines drahtlosen experimentellen Netzwerkes der diese Implementieren verwendet.

Der Quellcode ist in modulare Elemente aufgeteilt die in C++ geschrieben sind und durch die flexible Click Konfigurationssprache verbunden werden. Der Router kann auf Benutzerebene oder innerhalb des Linux Kerns ausgeführt werden, sowie im Netzwerk Simulator ns simuliert werden.

Untersuchung des Verarbeitungsgeschwindigkeit der OLSR Implementierung in Click haben gezeigt, dass die auf nur einem Thread basierende Click Router Umgebung dazu führen kann, dass der Router für Paketweiterleitung blockiert ist, während er OLSR spezifische Routinen ausführt. Dies trifft insbesondere auf große und dichte Netwerke mit hoher Mobilität der Teilnehmer zu.

Auf dieser in Click erstellten OLSR Implementierung basierend wurde ein drahtloses experimentelles Netzwerk entworfen. Dieses experimentelle Netzwerk besteht aus Netzwerkknoten die auf den Dächern der Campusgebäude installiert sind oder aus normalen PC’s die mit einer WLAN Schnittstellenkarte ausgerüstet sind. Embedded devices innerhalb wasserfester Gehäuse zur Installation auf den Campusdächern wurden erfolgreich eingerichtet so dass sie automatisch die entwickelte OLSR Implementierung ausführen.

Content list

List of pictures

List of acronyms

OLSR	Optimized Links State Routing
PC	Personal Computer
MANET	Mobile Ad-hoc Network
GSM	Global System for Mobile communications
UMTS	Universal Mobile Telecommunications System
LAN	Local Areas Network
WLAN	Wireless Local Area Network
DSR	Dynamic Source Routing
IETF	Internet Engineering Taskforce
RFC	Request for Comments
AODV	Ad-hoc On demand Distance Vector
TBRPF	Topology Broadcast based on Reverse Path Forwarding
MPR	Multi Point Relay
IANA	Internet Assigned Numbers Authority
UDP	User Datagram Protocol
IP	Internet Protocol
FPGA	Field Programmable Gate Array
TCP	Transmission Control Protocol
MID	Multiple Interface Declaration
TC	Topology Content
TTL	Time To Live

ARP	Address Resolution Protocol
MAC	Medium Access Control
IEEE	Institute of Electrical and Electronics Engineers
DSDV	Destination Sequenced Distance Vector
TPC	Transmit Power Control
DFS	Dynamic Frequency Selection
POE	Power Over Ethernet
ETSI	European Telecommunications Standards Institute
ISO	International Organization for Standardization
DHCP	Dynamic Host Configuration Protocol
PXE	Preboot eXecution Environment
TFTP	Trivial File Transfer Protocol
NFS	Network File System
MIT	Massachusetts Institute of Technology

1 Introduction

In the last years, mobile communication has grown rapidly. Users benefit from smaller devices, longer lasting batteries and higher bandwidths are available from new communication standards. The interest in self organizing Ad-hoc networks has grown strongly as well.

A MANET is a multi-hop ad-hoc wireless network in which nodes can move arbitrary in the topology. The network does not require any given infrastructure and can be set up quickly in any environment. The deployment of this kind of network has grown rapidly in the last years as well. Still many aspects of mobile ad-hoc networks are being researched. On the routing protocol layer the research topics include among others support for quality of service, efficient multicast strategies, protocol optimizations and security related issues.

The Implementation of an ad-hoc routing protocol should be quickly extendible to ease the introduction of new ideas from ongoing research activities. A modular approach that can easily be altered combined with the existence of a framework of basic router functions like they are provided by the Click Modular Router is one possibility for such an implementation.

As usually new ideas are tested in a network simulator with a certain set of parameters and only for some specific scenarios experiences in real life can differ significantly. New ideas should therefore be also tested and evaluated in an experimental network. This experimental network running the ad-hoc routing protocol implementation should be easy to deploy and update in order to test and evaluate the new ideas from research.

1.1 Implementation work

In this master thesis the OLSR protocol was implemented in the Click Modular Router, based upon existing standard Click elements from the Click Modular Router framework and newly created OLSR specific elements, also designed in a modular way.

To enable real life measurements of the OLSR protocol and further future enhancements an experimental network was designed and implemented. This network consists of PCs and small embedded devices both equipped with wireless radio interfaces and connected to antennas.

1.2 Chapter overview

An introduction to mobile communication with the focus on ad-hoc networks is given in Chapter 2. Chapter 3 introduces the Click Modular Router which was used for the implementation. Chapter 4 describes in more detail the OLSR protocol. This is followed by the description of how this protocol is implemented within the Click Modular Router in this work in Chapter 5. Chapter 6 outlines the concept of the experimental network and describes its implementation. Chapter 7 makes some suggestions for future work to continue the project that was founded by this master thesis.

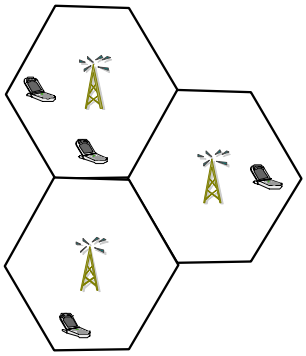
2 Ad-hoc Networks

2.1 Wireless communication

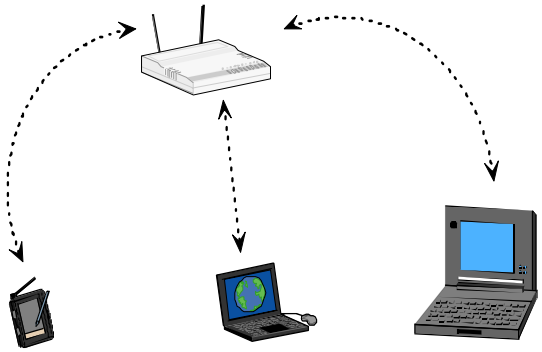
While in wired communication networks some infrastructure has to be setup, wireless communication can occur with or without connecting to special infrastructure.

2.1.1 Wireless communication with infrastructure

In wireless communication based upon special infrastructure, such as the GSM or UMTS networks or wireless LANs in infrastructure mode a special dedicated base station or access point is necessary. The nodes participating in the network have to register themselves at this base station or access point. All communication then is realized through these central coordination nodes. A terminal node and a base station or access point are very distinct in their behavior, as illustrated in Picture 1 and Picture 2.



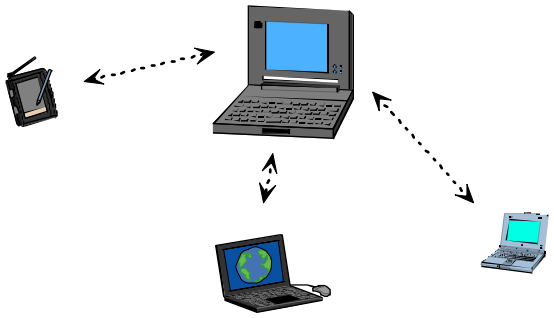
Picture 1: Cellular mobile communication with infrastructure



Picture 2: WLAN in infrastructure mode

2.1.2 Wireless communication without infrastructure

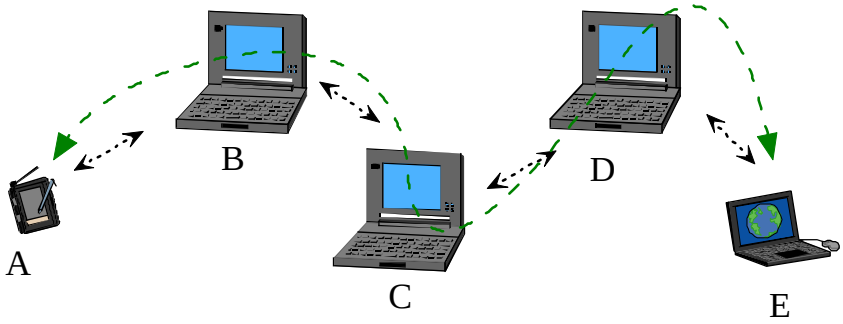
In contrast to the communication with infrastructure, the communication in Ad-hoc networks is organized completely decentralized. There is no central entity regulating or controlling network traffic. All nodes can be at the same time nodes originating and receiving network traffic as well as forwarding traffic for other nodes. Simultaneously they can act as terminal node and as router.



All nodes residing sufficiently close to each other to be within radio range can exchange packets without any further measures, as illustrated in Picture 3.

Picture 3: Direct communication of wireless nodes

To exchange packets with nodes farther away, nodes in between have to forward the packets. In Picture 4 node A can establish a communication with node E as node B,C,D are forwarding the packets exchanged from A to E and vice versa.



Picture 4: Exchange of packets by forwarding

Every node thus simultaneously acts as communication endpoint and as a router for other nodes. Movement of the nodes can lead to changing network topology. The used routing protocol has to be able to react to such changes.

The advantages of ad-hoc networks are especially based upon their decentralized self organizing nature not needing to setup any special infrastructure and to their flexibility with regards to changes. Typical scenarios for the use of ad-hoc networks therefore are mobile units within military operations [1], search and rescue operations in public safety, emergency and disaster applications as for example in the WIDENS project [2], vehicular based applications for example FleetNet [3] and Cartalk2000 [4] and wireless community networks, as for example Freifunk in Berlin [5] using the ad-hoc routing protocol OLSR for their community network.

2.2 Ad-hoc routing protocols

The self organizing decentralized control of the ad-hoc networks as well as the potentially changing topology in mobile networks set the requirements for ad-hoc routing protocols.

Though communication with direct neighbors is rather trivial, communication with distant nodes requires some knowledge about the network topology if a pure flooding mechanism in which all nodes retransmit every packet they receive as data traffic is to be avoided, as it makes very bad use of the available bandwidth and leads to a very high number of collisions.

The ad-hoc routing protocols generally are differentiated in two categories, the reactive protocols and the proactive protocols.

2.2.1 Reactive protocols

Reactive protocols determine the route to a destination on demand. If a communication is about to be set up and no route to the destination is already known, a route discovery is initialized. A route request packet is usually flooded through the network. When this packet either reaches a node with a route to the destination or the destination itself, a route reply is sent back to the source node either by link reversal or through flooding of the route reply packet. Routing occurs either in form of source routing or hop by hop.

Source routing

In source routing each data packet contains the entire route form source to destination. Intermediate nodes do not have to maintain up to date routing information for each active route but rather forward the packets based upon the information stored in the header of the packet. An example for this type of routing protocol is the DSR [6] protocol which has become an internet draft of the mobile ad-hoc network (MANET) working group within the internet engineering taskforce (IETF).

Hop by hop routing

In case of hop by hop routing, each data packet only carries the destination address and the next hop address. Each intermediate node in the path to the destination forwards the packet towards its destination based upon a routing table which it has to maintain for each active route. The AODV protocol as described in RFC 3561 [7] belongs to this category.

2.2.2 Proactive routing

Proactive Routing is based upon a table driven approach. Each node has to maintain routing information to the other nodes in the network. This information is usually stored in a number of different tables which are updated periodically and/or upon the detection of changes within the network. Which information is kept and how it is exchanged varies from the used routing protocols. The two from the MANET working group officially acknowledged routing protocols out of this category are Topology Based Reverse Path Forwarding (TBRPF) [8] and the Optimized Link State Routing (OLSR) [9].

TBRPF (RFC 3684)

TBRPF is a link state routing protocol providing hop by hop routing along minimum hop path to each destination. Each node running TBRPF computes a source tree based upon partial topology information stored in its topology table using a modification of Dijkstra's algorithm. To minimize overhead each node only reports part of its source tree to neighbors. TBRPF uses a combination of periodic and differential updates to keep all neighbors informed of the reportable part of its source tree. TBRPF performs neighbor discovery using differential HELLO messages which report only changes in the status of neighbors, resulting in HELLO messages a lot smaller than those of other routing protocols.

OLSR (RFC 3626)

OLSR is also based upon the traditional link state algorithm. Each node maintains topology information about the network by periodically exchanging link state messages. The optimization introduced by OLSR is that it minimizes the size of each control message and the number of nodes re-broadcasting a message by employing the multipoint relay strategy. The local one hop and two hop neighborhood is discovered through periodic exchange of HELLO messages. Thereafter each node selects some one hop neighbors to be its multi point relay in a way that all two hop neighbors can be reached through at least one of the selected members of the MPR set. Nodes that are not MPRs can receive and process each control packet but do not retransmit them and do not announce network topology to other nodes in the network. Nodes that are MPRs of at least one node forward packets for the nodes that selected them as MPRs and announce all nodes that selected them as MPR by topology content packets to the entire network. Based on its one hop and two hop neighborhood and the

topology information each node calculates an optimal route (with regard to hop count) to every known destination in the networks and stores it in its routing table.

After taking into consideration these different types of ad-hoc routing protocols the proactive category was chosen and the OLSR protocol was used in the implementation part of this thesis and is therefore described in more detail in chapter 4.

3 Click Modular Router

The Click Modular Router [10] is a software architecture for building flexible and configurable routers. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling and interfacing with the network devices. A router configuration is a directed graph with elements at the vertices. Packets flow along the edges of the graph. Configurations are written in a declarative language that supports user defined abstractions.

3.1 Click configuration

In Click configurations the used elements are instantiated and configured with the appropriate parameters. The connections from element outputs to other element inputs are defined. A sample configuration file can be found in the appendix in 9.1.

3.2 Click elements

Inside a running router each element is a C++ object and connections between elements result in virtual function calls with a pointer to a packet structure as parameter.

3.2.1 Properties of elements

The most important properties of an element are:

Element class: like objects in an object oriented program, each element has a class that determines its behavior.

Input and output ports: Ports are the endpoints of connections between elements. An element can have any number of input or output ports, which can have different semantic meanings.

Configuration string: Some element classes support additional arguments used to initialize per-element state and fine-tune element behavior. The configuration string contains these arguments

3.2.2 Packets in Click

Packets in Click are pointers that point to a header structure which holds pointers to the actual data and a set of packet annotations.

Packet annotations

Packet annotations are extra information attached to a packet that is not contained in the packet data. Often used is the paint annotation to mark packets with a certain number for example the number of the incoming interface. Other annotations used are for example timestamp annotation to carry the exact arrival time of the packet, the destination address annotation for the IP Address of the next hop or the cycle count annotation for performance evaluation based on cycle counts.

Packet processing

Click provides two kinds of connections between elements, push and pull. In a push connection, the upstream element hands a packet to the downstream element; in a pull connection, the downstream element asks the upstream element to return a packet. Apart from implementing push or pull methods an element can be implemented with agnostic ports, meaning it can work as either push or pull depending on its context in the router.
Elements not participating in packet processing but rather storing and maintaining data are referred to as Information elements.

3.2.3 Element scheduling

Besides being activated upon a push or pull request, an element can be put on the scheduling queue of the Click router by registering a task or scheduling a timer at a precise time.

3.2.4 Handlers

Click elements can add the possibility to interact with them once they are running through so called handlers. This can for example be done to change the configuration of the element, to

reset or change values or to add or remove entries from Information elements. Handlers can either be read or write handlers, used to retrieve information and return it or to change settings inside the router, if necessary with passing of parameters.

3.3 Runtime environments

A Click configuration can either run in kernel level, in user level or in the simulator. Configurations for the three different environments differ slightly, mostly with respect to packet sources and sinks.

3.3.1 Kernel level

To run in kernel level the kernel has to be patched in order to allow Click elements written in C++ to be compiled and executed inside a kernel module. Thereafter the compiled module includes the Click elements written in C++ and can be installed by the Linux “insmod” command or more comfortable by “click-install”, which not only installs the kernel module but also mounts the Click file system and can receive as a parameter the configuration file of the router. The Click file system is a proc like file system used to communicate with the running kernel module. To change the router configuration once the kernel module is loaded this Click proc like file system can be used. This file system is also used to communicate with the running router through the installed handlers.

For writing elements all usual kernel level limitations apply, especially no use of floating point numbers and of user level libraries is permitted.

In kernel level the Click module replaces the Linux networking stack. Packets have to be explicitly handed to the system or packet sniffers. If not, these packets will only be seen by the Click code running inside kernel level.

3.3.2 User level

Code Compiled for user level Click can be easily executed without patching the kernel. A configuration file is passed as parameter when loading Click. Click gets the packets from the Linux system and other registered packet sniffers or user level programs can see the packets as well, even if the packets are discarded inside the Click configuration.

Interaction with handlers can take place over a socket by using a *ControlSocket* element which establishes a socket listening on the specified port and calling the desired handler inside Click when a matching petition is received.

3.3.3 Simulator

To run the code in the network simulator “ns-2” this simulator has to be installed and the nsclick patch applied [11]. Thereafter elements compiled for simulation can be used inside a Click configuration file that is loaded to the corresponding nodes inside the .tcl script of the simulator.

3.3.4 SMP Click

While standard Click runs on a single processor systems in a single threaded environment, Click SMP [12] can run on multiprocessor systems with multiple threads in parallel.

3.3.5 XORP forwarding plane

Click is especially designed to be an easy to configure flexible forwarding plane. The XORP Router project [13] supports using Click as custom forwarding chain while implementing all other router functionality like route discovery and maintenance in a multithreaded user level environment.

3.3.6 Fast forwarding plane

Packet processing with special network processors or custom built FPGA can achieve higher packet forwarding rates than a Linux software router. One ongoing project is aiming at automatic generation of code for the network processor Intel IXP1200 based upon the Click Modular Router [14]. Another project works on generating custom FPGA based forwarding chains based upon a language similar to the Click configuration language for standard elements [15].

3.4 Related work

Click is used by more and more universities in their research activities. While one typical application of Click consists of gigabit routing test beds also other MANET routing protocol implementations of Click exist:

MIT GRID [16]: implementation of DSDV [17] and DSR as well as geographic forwarding
University of Colorado at Boulder: AODV, DSR, DSDV, ZRP [18], TORA [19]

4 The OLSR protocol

As introduced in chapter 2.2.2 the OLSR protocol is a proactive, table driven routing protocol. It was developed at the “Institut National de Recherche en Informatique et en Automatique” (INRIA) [20], France and is specified in IETF RFC 3626. This specification divides the protocol in two parts: core functionality and auxiliary functions. Each node participating in an OLSR MANET has to at least fulfill the core functionality while it may implement further enhancements from the auxiliary functions without influencing compatibility with core functionality. Messages corresponding to functions not implemented by this node have to be forwarded in such a way that other nodes that do implement these functions can receive these packets. In this thesis only core functionality has been covered and implemented.

The RFC specifies OLSR as a pure route maintenance protocol which is responsible for determining and maintaining routes but not for actually forwarding data packets. This is supposed to be done by some underlying mechanism.

In OLSR each node is identified by a “main address”. Though a node can have multiple interfaces it is identified by just one of these which is chosen to be the nodes main address.

4.1 Information Repositories

As mentioned in chapter 2.2.2 a proactive Routing protocol usually maintains a number of tables with information about the network and its topology. In case of OLSR a variety of Information Repositories are specified.

4.1.1 Multiple Interface Association Information Base

For nodes participating in the network equipped with more than one interface the Multiple Interface Association Information Base stores the mapping of these additional interface addresses to the main address of this node, as the main address is the identifier used to refer to a node.

4.1.2 Link set

The link set contains pairs of interface addresses (not main addresses) and the corresponding times until when this link can be considered symmetric, asymmetric or when it has to be completely removed. These times have to be updated upon reception of a HELLO message from the corresponding node.

4.1.3 Neighbor Information Base

The Neighbor Information Base stores information about four distinct sets that are all related to neighbor discovery.

-Neighbor set:

The neighbor set keeps the list of neighbors of the node. Based upon the link connecting the two nodes a neighbor is either classified as a symmetric or as a non symmetric neighbor.

-Two hop set

The two hop set describes the two hop neighborhood of a node. It stores a list of node pairs describing which two hop neighbors can be reached through which symmetric one hop neighbor.

-MPR set

The MPR set maintains the set of nodes which were elected MPRs by this node.

-MPR selector set

The MPR selector set includes all neighbors that have selected this node as MPR.

4.1.4 Topology Information Base

The Topology Information Base contains information on all link state information received from nodes participating in the OLSR network.

4.1.5 Duplicate set

The Duplicate set keeps track of all messages recently processed and forwarded to avoid reprocessing or re-forwarding of the same message multiple times.

4.1.6 Time-out mechanism

To prevent maintaining stale data information base entries are equipped with a time limit. Only until this time information is being considered valid and used for further processing. When the time has expired, the data is not considered valid or relevant anymore and should be removed from the Information base. The timer for a certain information base entry is (re-)initialized when a message containing the corresponding information is received. The validity time given in the received message specifies until when the entry will be valid.

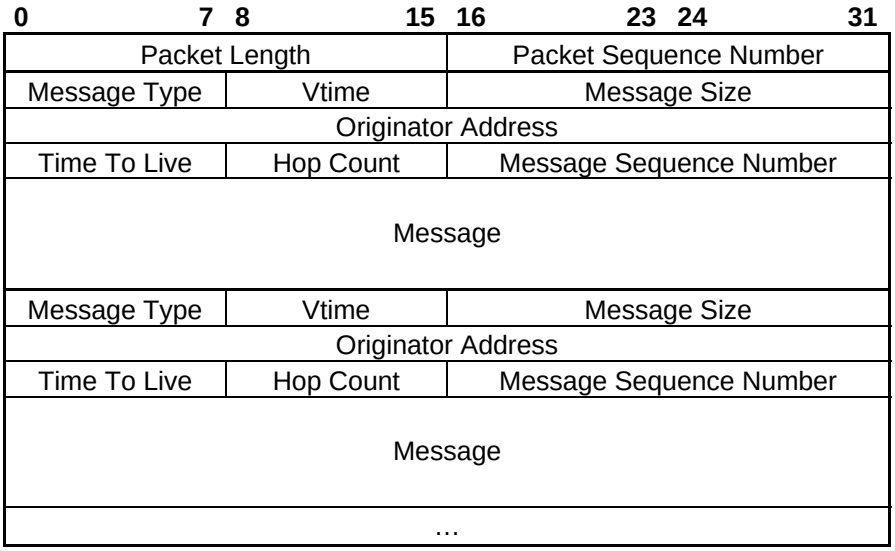
4.2 OLSR control messages

To establish and maintain the Information of these Information Repositories a number of different OLSR messages are defined and exchanged periodically by the nodes participating in the network. Together they form the OLSR control traffic. The Internet Assigned Numbers Authority (IANA) has assigned UDP Port 698 to be used for the broadcast of OLSR control messages. To implement core functionality three different message types have to be supported: HELLO messages, TC messages and MID messages.

4.2.1 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.

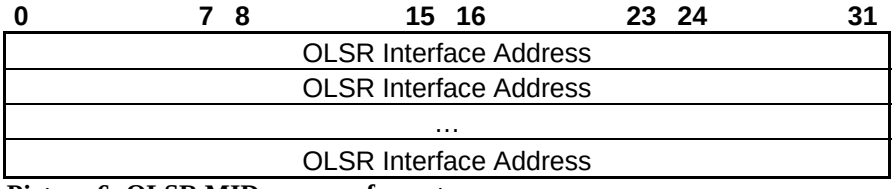
The packet body consists of one or more OLSR messages which are preceded by a message header for each included message. The message header contains the message type, the validity time, the message size, the originator address, a time to live field, the hop count and a message sequence number. The originator address field contains the main address of the node that initially created the message, independently on 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. The full OLSR packet format is shown in Picture 5.



Picture 5: OLSR packet format

4.2.2 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 (see Picture 6). 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.

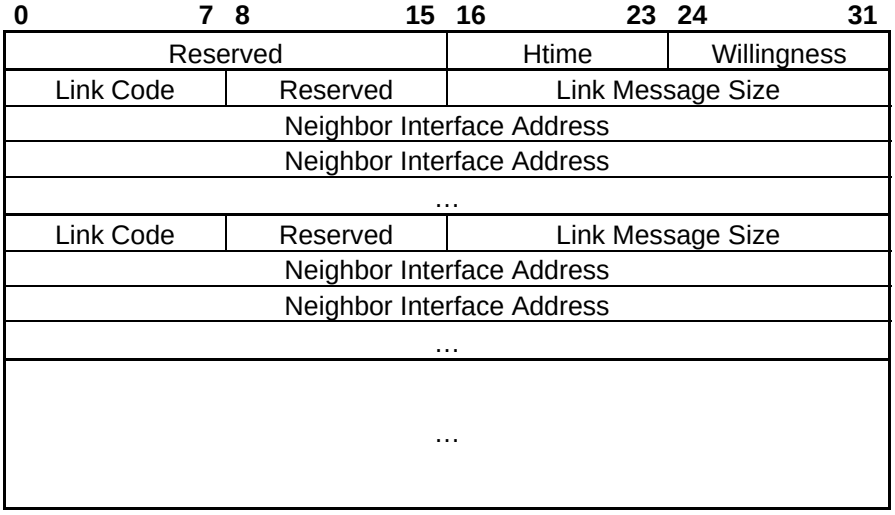


Picture 6: OLSR MID message format

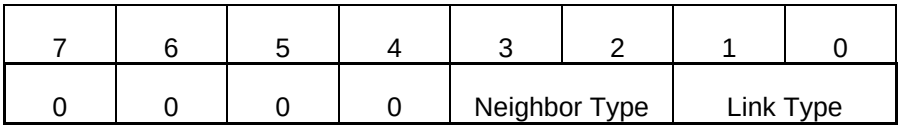
4.2.3 HELLO messages

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 (see Picture 7 and Picture 8).



Picture 7: OLSR HELLO message format

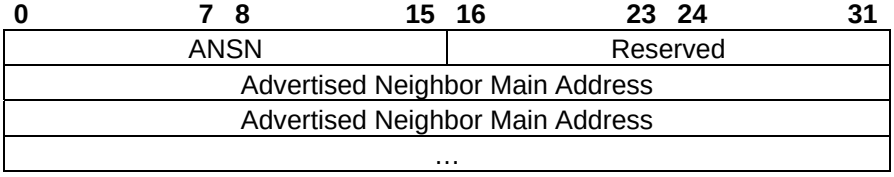


Picture 8: OLSR link code

A link can be unspecified for this interface while the neighbor is a symmetric or MPR neighbor based upon a symmetric link on another interface.

4.2.4 TC messages

A node selected as MPR advertises his advertised neighbor set which in core functionality is the MPR selector set (the nodes that selected it as MPR) to the network by emitting TC messages. The message contains a sequence number which is updated every time the advertised neighbor set has changed. Thereafter follows a list of the advertised neighbors' main addresses as illustrated in Picture 9.

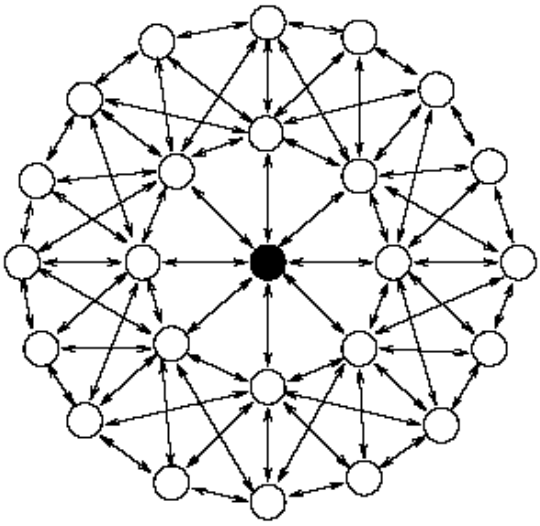


Picture 9: OLSR TC message format

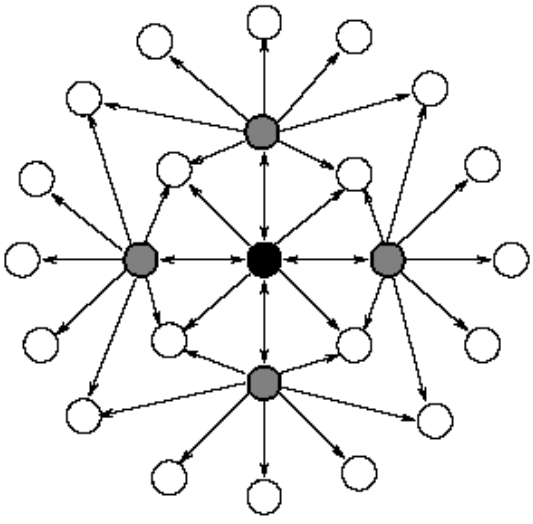
4.3 Multipoint Relaying

A key concept in the OLSR protocol is the Multipoint Relay. The control messages that are not just used for local neighbor discovery (HELLO messages) have to be flooded through the entire network. In its simplest form, flooding means retransmission of every packet at every node. In large and dense networks this leads to a high number of unnecessary retransmissions resulting in big control message overhead and losses of control packets due to collisions of packets.

OLSR uses the multipoint relay concept as an optimization to standard flooding (compare Picture 10 and Picture 11). Based upon its one hop and two hop neighborhoods a node selects one hop neighbors as Multipoint relays in such a way that each two hop neighbor can be reached by at least one member of the MPR set. The neighbor elected as MPR is marked MPR neighbor in the next HELLO message that is sent. The node being elected MPR upon receiving this HELLO message adds the node that selected it as MPR into its MPR selector set (see for more details chapter 4.5). A node only forwards traffic of nodes that are included in its MPR selector set.



Picture 10: Standard flooding with all nodes retransmitting the message



Picture 11: MPR flooding – only MPR nodes retransmit the message

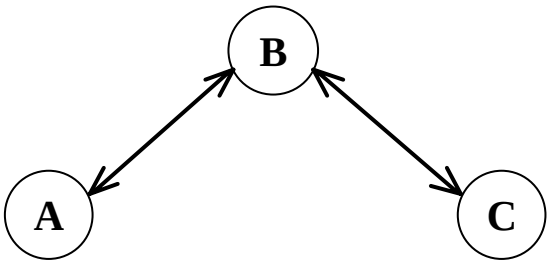
The second optimization that comes inherent with this is the minimization of nodes to announce for spreading the topology of the network. If only nodes elected as MPR announce those nodes that elected them as MPR (the members of its MPR selector set) to the entire network, the information received is sufficient to compute the routing table based upon this topology information and the one- and two hop neighborhood in every node of the network.

4.4 Message Forwarding

Though for known message types there may exist special rules, the RFC defines a default forwarding rule which also applies to not known message types.
This algorithm specifies when message have to be discarded, recorded in the duplicate set and after decreasing the TTL field have to be forwarded on all interfaces in case the sender was in the MPR selector set. As HELLO messages are only used for local neighbor detection they are never forwarded. To avoid packet collisions in the network the forwarded control messages have to be delayed an arbitrary time interval (jitter). Furthermore the period of message emission for generated messages (HELLO, TC, MID) should be varied randomly in small borders around the desired values.

4.5 Neighbor Discovery

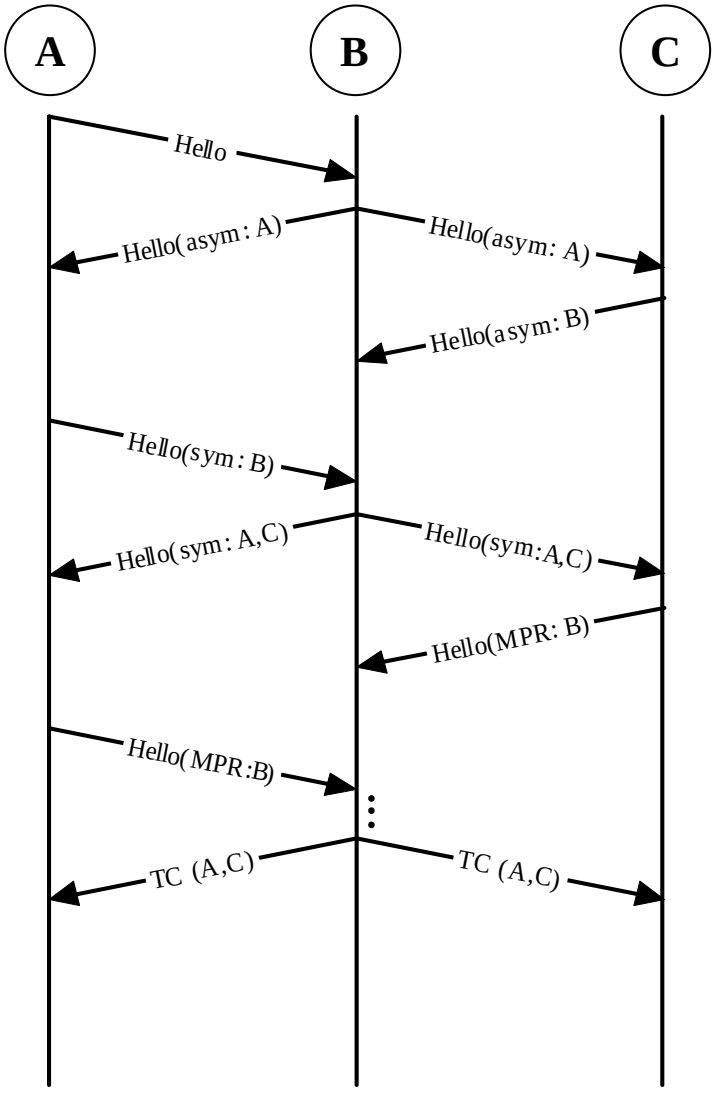
By exchanging HELLO messages the link set, the one hop and the two hop neighbor set are built. Based upon these the MPR set and MPR selector set are determined. The process can be demonstrated well by a small example. In this example node B is situated between node A and node C, where node A and C are not in communication range of each other, while node A and node C are within the range of node B as shown in Picture 12. Picture 13 explains the related packet flow.



Picture 12: Example of neighbor detection in OLSR

At first a node that has no knowledge about its neighborhood emits an empty HELLO message (in this example this is node A). Receiving this empty HELLO message node B knows there is at least an asymmetric link from the sender of this message (node A) to him and announces this in its next HELLO message which is received by node A and node C. A node receiving a HELLO message with its own address in the neighbor list (node A), either as asymmetric or symmetric neighbor, knows there is a symmetric link to this neighbor. Receiving the HELLO message from node B, node C knows there is at least an asymmetric link to node B which it announces in its next HELLO message. Receiving this message node

B knows there is a symmetric link to node C. When node A emits a new HELLO message with node B marked as symmetric neighbor, node B knows upon reception of this message that a symmetric link to node A exists and announces the symmetric links to node A and node C in its next HELLO message. Receiving this message, nodes A and C both know they have a symmetric link to node B and that they can reach each other through node B. Hence node C is a two hop member of node A only reachable through node B and vice versa. Therefore node A and node C select node B as MPR which they announce in their next HELLO message. Receiving these HELLO messages, node B includes node A and node C in its MPR selector set and will from know on forward OLSR messages received from these nodes (except HELLO messages). Furthermore node B starts emitting TC messages with the addresses of node A and C included. In contrast to HELLO messages that are only used for local neighbor detection these TC messages as well as MID messages are forwarded by all MPR nodes.



Picture 13: OLSR Link sensing with HELLO messages

4.6 Topology Determination

The TC messages are used to spread the topology of the network. Upon reception of a TC message the receiving node updates its TC repository either adding a new entry or updating an old entry if the advertised sequence number of the received TC message is more recent than the last stored one.

4.7 Route calculation

While one and two hop routes can easily be determined from the one and two hop set, more distant routes are calculated by a shortest path algorithm (with regard to hop count) using the topology data stored in the topology set.

4.8 Existing OLSR Implementation

A variety of OLSR Implementations do already exist for different systems

4.8.1 INRIA

The INRIA has an implementation of OLSR as described in RFC 3626 v3 in C, and implemented the current RFC 3626 v11 as a prototype in python (pyOLSR) and a new rewritten implementation in C++ (OOLSR) designed for Linux and the simulator ns-2. All versions can be found on the INRIA OLSR webpage [19].

4.8.2 Unik (University Graduate Center)

At the UniK - University Graduate Center the olsrd code [21] was developed which implements full RFC 3626 v11 compliancy, including auxiliary functions, a plug-in interface for further enhancements and from version 0.48 on an alternative metric for route computation, the expected transmission count metric ETX [22].

4.8.3 LRI (Laboratoire de Recherche en Informatique)

The LRI focuses on implementing Quality of Service within the OLSR protocol as part of the QOLSR project [23]. The code written in C++ is fully RFC compliant and there also is a version running in the OPNET simulator.

4.8.4 NRL (Naval Research Laboratory)

The NRL implemented a first ns-simulator based version (nrlolsr-ns) and a newer implementation in C++ for Linux called nrlolsrd [24], nearly RFC compliant without support for multiple interfaces.

4.8.5 GRC (University of Valencia Depart. of Computer Engineering)

The GRC has ported the INRIA code to windows 2000 and pocket PC [25].

5 The OLSR Click implementation

The non-standard elements of this Click OLSR implementation are partially based upon a first intent of OLSR in Click from Tor Einar Dørum [26]. After initial tests of the existing code it turned out that almost all code had to be changed or entirely redone to achieve correct route maintenance, MPR election, multiple interface support and stable execution without terminating in segmentation faults after a very short time. Nevertheless parts of the structure and especially some header files as well as small parts of the code are still based on this first version.

5.1 Elements

The Click OLSR implementation consists of standard Click elements as distributed with the Click code as well as self implemented elements written in C++. These elements can be differentiated in pure information bases, message generators, message processing elements and other elements.

5.1.1 Information Elements

Information elements contain the in chapter 4.1 mentioned tables used by OLSR and provide the necessary functions to add, delete, update and print the contents as well as query them by appropriate find functions. If necessary they include rescheduling the element with use of timers in order to delete entries which validity time has expired. The provided functions are usually called by elements participating in the packet processing to either update the information stored in the Information Element or to query it.

OLSRInterfaceInfoBase:

The OLSRInterfaceInfoBase element maintains the Multiple Interface Association Information Base as described in 4.1.1. containing the mapping of interfaces of other nodes and the corresponding main addresses of that node

OLSRLocalIfInfoBase:

The OLSRLocalIfInfoBase Element contains the addresses of the local interfaces as they are announced by MID messages if more than one main address is present in the node.

OLSRDuplicateSet

The OLSRDuplicateSet element maintains the Duplicate set as described in 4.1.5 and the packet sequence numbers from known neighbor interfaces.

OLSRNeighborInfoBase

The OLSRNeighborInfoBase element contains the Neighbor Information Base as described in 4.1.3, consisting of the neighbor set, the two hop neighbor set, the MPR set and the MPR selector set. It also implements the algorithm for the computation of the MPR set.

OLSRTopologyInfoBase

The OLSRTopologyInfoBase element implements the Topology Information Base as described in 4.1.4.

OLSRLinkInfoBase

The OLSRLinkInfoBase element maintains the Link Information Base as described in 4.1.2. When a link expires the timeout routine takes care of also removing the corresponding neighbor set entries and if the node on the other side of the link was included in it, the corresponding MPR set entry.

OLSRRoutingTable

The OLSRRoutingTable maintains the routing table to use in Click, providing functions to get the next hop and output interface based upon the desired destination node as well as implementing the route calculation algorithm as described in the RFC.

5.1.2 OLSR Message Generators

All three packet generators generate the corresponding packet as described in the RFC. They schedule events for themselves on the Click scheduler by using timers in order to periodically emit packets. To avoid network synchronization the emission period is shortened randomly corresponding to the RFC. A parameter specifying the maximum value by which the period should be shortened is passed to the element.

OLSRHelloGenerator

The OLSRHelloGenerator generates HELLO messages as described in 4.2.3. The link and neighbor types of the neighbors to be announced are determined and put in the newly created packet. HELLO messages have to be generated separately for each interface.

OLSRTCGenerator

The OLSRTCGenerator generates TC messages as described in 4.2.4. if the node is elected as MPR by any other node.

OLSRMIDGenerator

The OLSRMIDGenerator generates MID messages as described in 4.2.2 containing the information stored in the OLSRLocalIfInfoBase if more than one interface is present in the node.

5.1.3 OLSR message processing elements

All OLSR message processing elements process the content of the incoming packet as described in the RFC.

OLSRProcessHello

The OLSRProcessHello element implements the link sensing by processing HELLO packets. If necessary the link set, the neighbor set, the two hop set and the MPR selector set are updated or new entries are added. If relevant changes have occurred a recalculation of the MPR set and the routing table is initiated based on the updated information repositories.

OLSRProcessTC

The OLSRProcessTC element processes the TC messages by extracting the information included in it and updating the Topology Information Base. If relevant changes have occurred a recalculation of the routing table based on the new information in the Topology Information Base is initiated.

OLSRProcessMID

The OLSRProcessMID element processes MID messages. It extracts the information about the interface addresses from the incoming MID message and updates the Multiple Interface Association Information Base accordingly. If changes have occurred, a recalculation of the routing table is initiated.

5.1.4 Other elements of the OLSR implementation

OLSRAddPacketSeq

This element maintains the packet sequence number for an interface. When a packet arrives at this element the packet sequence number is incremented and put in the packet sequence number field of the OSLR packet.

OLSRCheckpacketheader

This element checks the packet header for correct minimum length and whether the packet sequence number is not outdated.

OLSRClassifier

Upon receiving a packet this element separates the possibly multiple messages contained in it (message piggybacking) and classifies them according to the RFC, emitting each message type as well as the not implemented message types and the messages that have to be discarded on a separate output.

OLSRForward

This element implements the forwarding rules as described in the RFC. It adds the OLSR message header and maintains message sequence numbers for all messages originated from this node.

OLSRGetnexthop

This element queries the routing table for the next hop of an IP data packet and sets the dst_ip_annotation and paint annotation of the packet to the next hop IP Address and the number of the interface on which the packet has to leave the node.

JitterUnqueue

Dequeues packets a random time after their arrival at the corresponding queue, while the maximum value of this introduced delay is limited by a parameter. The element provides a special notifier function which has to be called by the queue whenever a packet arrived. Through maintaining an ordered list of departure times and always pulling the first packet from the queue, packets streams stay in order and the maximum delay does not get exceeded.

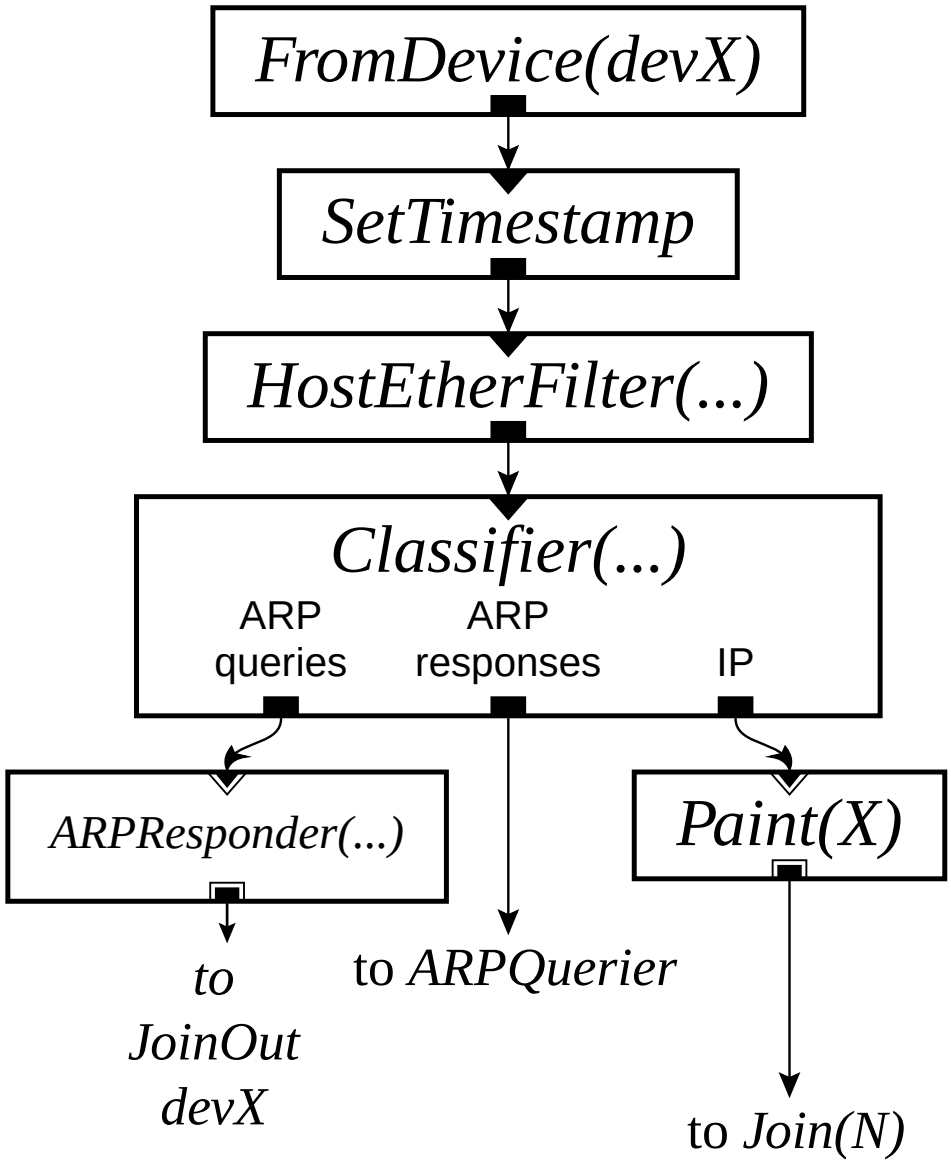
OLSRQueue

This is the queuing element used in conjunction with the JitterUnqueue element. It is basically a standard Click queue modified to call the notifier function in the JitterUnqueue element which has to be passed as parameter.

5.2 Router Configuration

A sample configuration of a multi interface configuration for use in user level will be schematically discussed here. A corresponding Click configuration file for two interfaces can be found in the appendix (see 9.1). To maintain better viewable pictures all discard elements are not displayed. In the text configuration file it can be seen that most elements do have an output where they emit the packets that have to be discarded. These outputs are connected to a discard element but it could be added for example a counter to count the discarded packets or a print element to display their content on the screen, or in user level and in the simulator a *ToDump* element, which writes the packets into a packet dump file which can be analyzed later on.

5.2.1 The Input part of the configuration

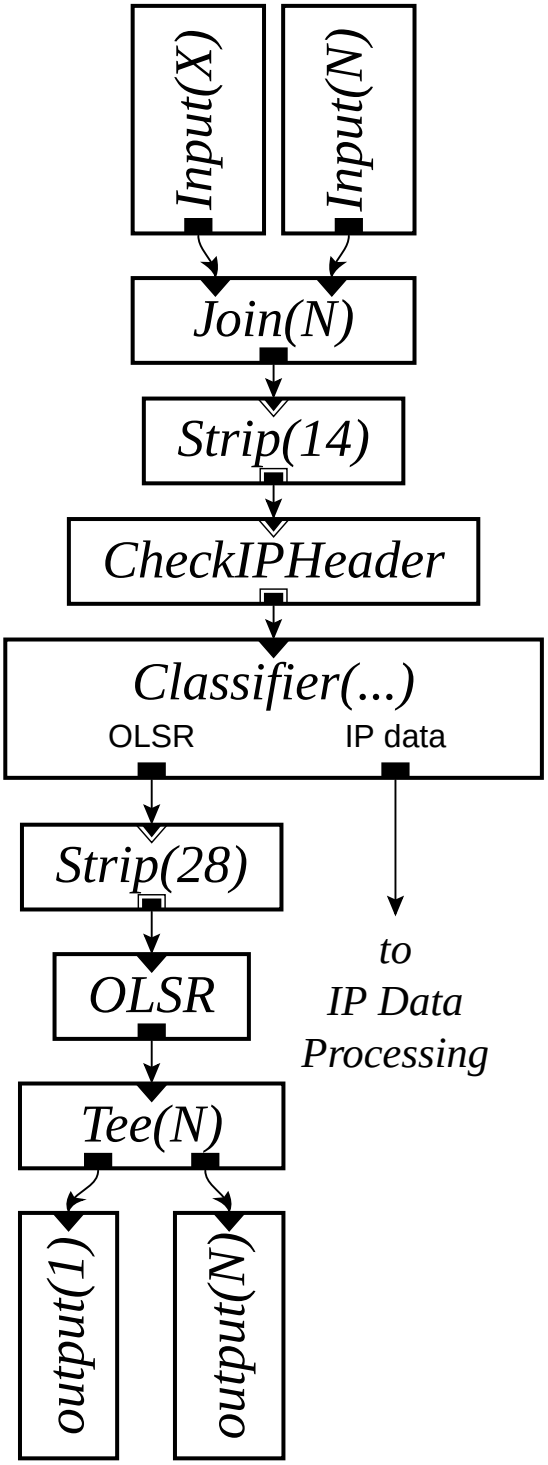


Picture 14: Input part of the configuration graph

The input part, existing separately for each interface, consists of a *FromDevice* element receiving the packets from the system or simulator or device driver (user level / simulator / kernel level). Thereafter *SetTimestamp* assures that timestamp annotations of the packet are set correctly and *Hostetherfilter* filters the incoming packets. It only allows packets that are either broadcast packets or if unicast packets only those that have the IP of the associated network device as destination address. Packets originated by the node itself are also dropped by this element. The following *Classifier* separates packets depending on their Ethertype protocol type whether they are IP packets or ARP queries or ARP responses. ARP queries are handed over to the *ARPResponder*, ARP responses to the *ARPQuerier* of this interface. IP Packets pass the *Paint(x)* element marking the “color” annotation of the packet with the

number of the incoming interface. Once marked, packets proceed to the unified processing of packets for all interfaces while above described input part exists for each interface of the node separately.

5.2.2 General packet processing part

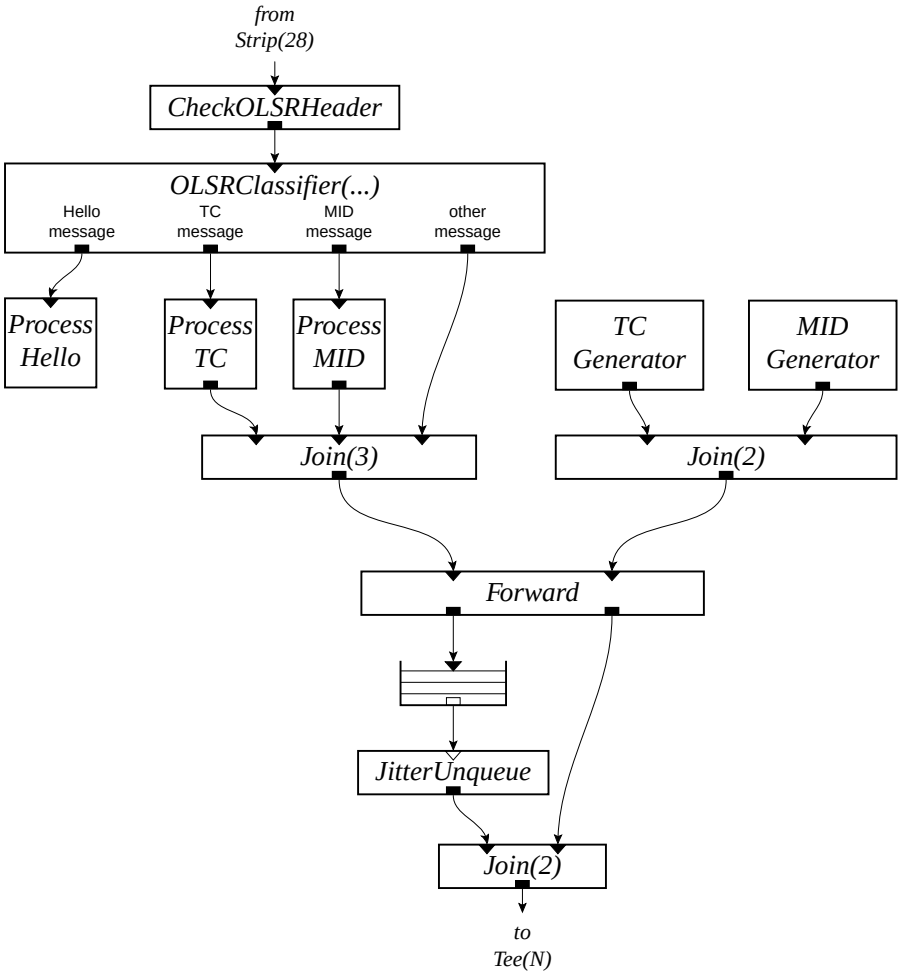


Picture 15: General packet processing part of the configuration

The Input (x) part as described in chapter 5.2.1 exists separately for each interface. From the Join(n) Element on packets are processed by the same elements independently on which interface they arrived on. The Ethernet header is removed by the *Strip(14)* element removing the first 14 bytes of the packet and *CheckIPHeader* filters packets with invalid IP Headers and sets the IP related packet annotations of the packet.

The following *Classifier* differentiates OLSR control packets (using UDP Port 698) and other Data packets. From the OLSR packets the UDP and IP header is removed by the *Strip(28)* Element and they are handed to the OLSR specific part of the configuration which is described in 5.2.3, while IP Packets are passed to the forwarding chain elements described in 5.2.5. Packets processed by the OLSR part that have to be forwarded by broadcasting them on all interfaces are handed to the corresponding output part of the interfaces by the *Tee(N)* element. The output part of the interfaces is described in 5.2.4.

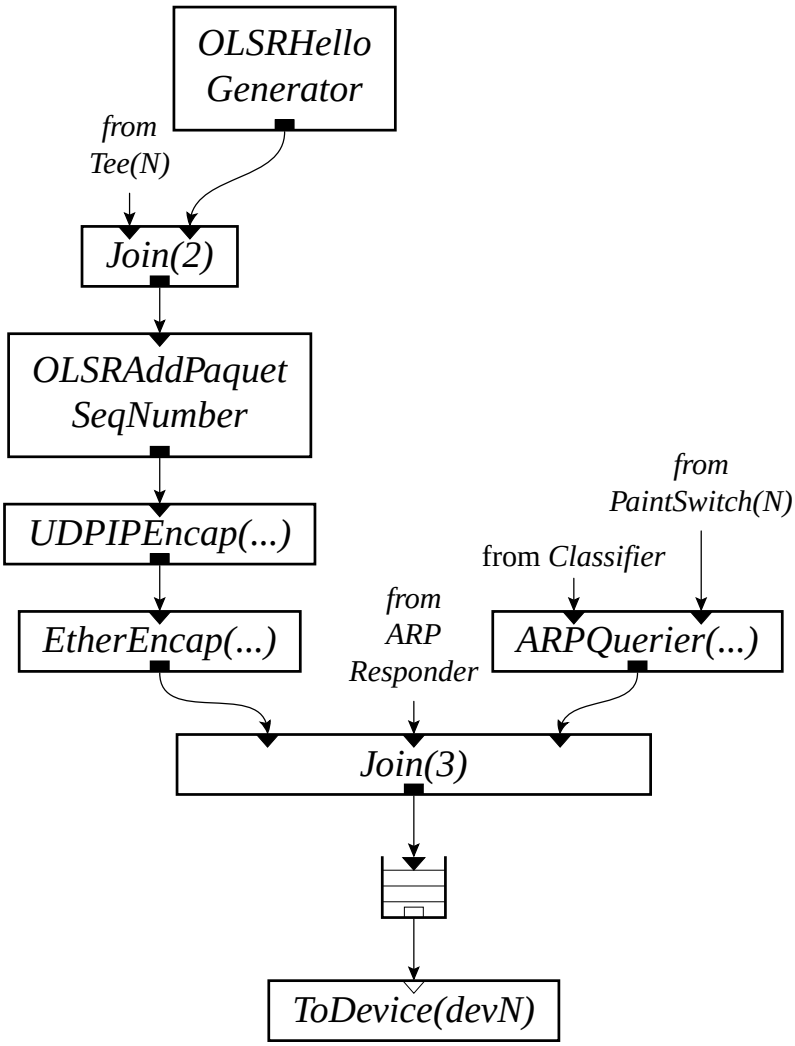
5.2.3 The OLSR specific part of the implementation



Picture 16: OLSR specific part of the implementation

OLSR packets are checked for a correct OLSR header and recent packet sequence number in *OLSRCheckheader*. *OLSRClassifier* determines whether a packet has to be discarded or whether it implements either a known message type (HELLO / TC / MID) or an unknown message type. Known message types are passed to the corresponding message processing elements, *OLSRProcessHello*, *OLSRProcessTC*, *and OLSRProcessMID*. While HELLO messages always have to be discarded after processing, TC and MID messages are passed to the first entrance of the *OLSRForward* element which implements the forwarding rules as described in the RFC. If these messages have to be forwarded by the node they are emitted on the first output, which stores the packets in a special *OLSRQueue* which is connected to a *JitterUnqueue* element introducing the Jitter in message forwarding to avoid packet collisions due to network synchronization. The packet generators whose packets have to be emitted on all interfaces, *OLSRTCGenerator* and *OLSRMIDGenerator*, are connected to the second input port of the forwarding element and leave again on the second output port without being jittered thereafter. Network desynchronization in this case is achieved by altering the packet generation period within a small margin randomly. The jittered messages and the packets from the local packet generators are then emitted on all interfaces *Tee(N)* which is connected to the corresponding output parts of each interface as described in 5.2.2.

5.2.4 The output part of the configuration

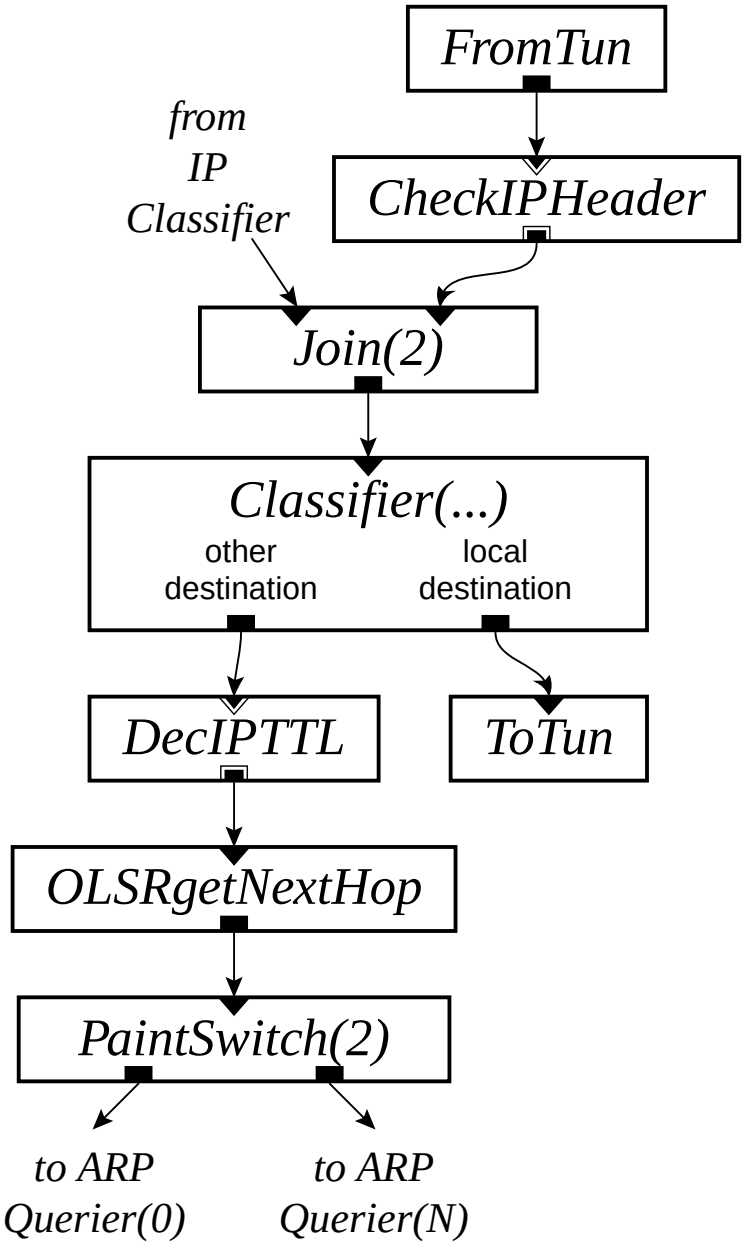


Picture 17: Output part of the configuration graph

The *OLSRHelloGenerator* is located in the output part as it has to generate independent HELLO messages for each interface of the node. Both the generated HELLO message and the other generated or forwarded packets that were described in 5.2.3 then get their packet sequence number added in *OLSRAddPacketSeqNumber*. This packet sequence number has to be maintained for each interface independently. Thereafter packets are encapsulated in UDP and IP headers in *UDPIPEncap* and in Ethernet headers in *EtherEncap.* Then packets are placed into the final queue before the *ToDevice* element from which the packet is pulled by the network interface. Also packets generated by the *ARPResponder* in the input part for this device (see chapter 5.2.1) are put into this queue as well as packets generated by or passed through the *ARPQuerier*. The *ARPQuerier* exists separately for each interface and is part of the output chain of the corresponding interface. This *ARPQuerier* receives on one input ARP responses from the classifier of the input part of the device and on the other input IP packets

to be sent out to the network (see chapter 5.2.5). Received IP packets are encapsulated in an Ethernet address if the Ethernet address of the next hop identified by the dst_ip_address annotation is known. If it is not known the packet is stored and an ARP query is generated. Once a reply to this query including the requested Ethernet address has been received the stored packet is encapsulated in the Ethernet header and placed into the output queue.

5.2.5 IP Data packet processing



Picture 18: IP Data packet processing part of the configuration

The entering IP packets are passed to the IP forwarding part of the configuration by the Classifier as mentioned above (see 5.2.2). Packets generated from the local system (received

from the *Tun* Element) are checked for correct IP headers. Thereafter both types of packets are passed to another Classifier which determines whether the destination address is that of a local interface or of another node. In case of a local interface the packet is passed to the local system by the same *Tun* element. In the other case the TTL field is decreased and if cero the packet is discarded (*DecIPTTL*). *OLSRgetnexthop* determines the next hop of the packet by consulting the routing table. It sets the corresponding next hop destination address and the interface on which the packet has to leave the node by using the corresponding packet annotations. The following *PaintSwitch* then separates the packets based upon their paint packet annotation and hands the packet to the *ARPQuerier* of the corresponding output (see 5.2.4).

5.3 Runtime environments

The code can run either in the simulator, in kernel level or in user level. To be able to run in kernel level as well, no use was made of floating point calculations and of user level libraries. The router configuration file differs slightly in each environment, especially the parts related to packet sources and sinks (either the network devices or the interaction with the local system).

5.4 Configuration generation tool

To make the switches between the environments easier and to add flexibility to the concrete node setup with regard to the number of network interfaces and their names and addresses, a Perl script was written that dynamically generates the router configuration file. It receives a few parameters specifying the setup and further parameters specifying the OLSR related behavior like packet generation periods, jitter values and more alike. For non specified parameters the default values as suggested in the RFC are used (the usage description can be found in the appendix 9.2).

5.5 Tests

A variety of tests were executed to verify correct behavior of the OLSR Click implementation. Special attention was paid to MPR election, routing table computation and packet forwarding.

5.5.1 Tests in the Simulator

To test the algorithms for MPR election and route computation in rather big networks as well as packet forwarding in large and dynamic networks the simulator was used. Furthermore the simulator was used to obtain packet dump files from simulations of such networks. These then were used to verify the corresponding behavior of the code in user level and kernel level, too.

Verifying correct MPR election and route calculation in static scenarios

Using static scenarios the MPR election and route calculation for a small network consisting of eight nodes was confirmed step by step. For larger and still static networks as for example a chess board like 8x8 nodes network the results of MPR election and route computation were verified. MPR election and route calculation proofed to be correct.
Furthermore static simulations with multiple interfaces were set up to verify the correctness of the code when using multiple interfaces. The interfaces of different nodes were interconnected through separate channels. MPR computation and route discovery for the multiple interfaces was confirmed to operate correctly.

Verifying traffic forwarding and number of known routes in dynamic scenarios

The simulator was also used for tests with larger networks scenarios. These scenarios also were dynamic, using the random waypoint model for node movement. A data traffic source was attached to one node and the corresponding sink to another node, thus establishing an UDP connection between these two nodes. Furthermore upon each routing table recalculation, the number of routing table entries was printed out. Analyzing the results showed that based on the temporary network situation due to the nodes movement the number of known destinations in the routing table varied but usually recovered to include all nodes rapidly if no network partition occurred. This behavior could also be observed in the bandwidth of the traffic stream, which differed slightly and even could drop down to zero but recovered rapidly thereafter.

Obtaining packet dump files

A large variety of simulations for scenarios ranging from 2 to 100 nodes were executed. For each number of nodes five different theoretical node densities (nodes per square meter, or medium number of nodes within reception range) were used thus varying the area of the network for simulation with the same number of nodes. During these simulations all incoming

packets of a node were written to a separate packet dump file using the *ToDump* element of the standard Click package. Later on these packets were replayed in a networks only consisting of two nodes, but pretending to be part of the network as it was in the simulations. The node running the code to be evaluated was configured with the same parameters and especially the same IP Address as the node from the simulation which entering packets were used. The node replaying the packets did so by using the *tcpreplay* tool, rewriting the destination MAC Address of the unicast packets to the real life MAC address of the node receiving the packets.

5.5.2 Tests in user level

Two different kinds of tests were carried out in user level. The first consisted in interconnecting two computers running the OLSR Click code in user level and to verify whether they recognized each other and communication did occur. In a second test, the code running in user level was fed with packets from different packet dump files, again observing correct routing table entries for the node.

5.5.3 Test in kernel level

The same tests as described in 5.5.2 for user level were carried out in kernel level.
Furthermore some severe performance evaluation of the code was executed in kernel level as described in chapter 5.6. Further tests within the specially designed test bed were executed and are explained in chapter 6.3.

5.5.4 Interoperability test

Two kinds of interoperability test were executed. In a first one, a node running OLSR Click in user level was connected to a node running the UniK olsrd daemon. It was confirmed that the two nodes saw each other and could communicate.

A second type of interoperability test consisted in feeding a node running the UniK olsrd daemon with the packets obtained from the simulations with the Click OLSR code. The node running olsrd reported the routes the same way as did the node running in the simulation.

5.6 Performance evaluation of OLSR in Click

To evaluate the implementation of OLSR in Click and to examine the suitability of a single threaded router system for such a project a number of measurements were done. First measurements were taken in Click running in user level by using the normal "*get_time_of_day*" routine. These measurements already showed a great processor demand for the OLSR implementation in Click. To obtain more accurate results, cycle count measures were used running Click in kernel space.

5.6.1 Setup

The packet dump files obtained from simulations as described in 5.5.1 were used for the performance evaluation in the same way as for testing the code. Nodes acting as MPR were chosen as sources for the packets as this implies that these nodes have to forward OLSR control traffic. For the measurements regarding IP data packet forwarding nodes were chosen that were along the route from the data packet source and its corresponding sink, at least for a long period of time (as simulations included movement, the routes were subject to changes).

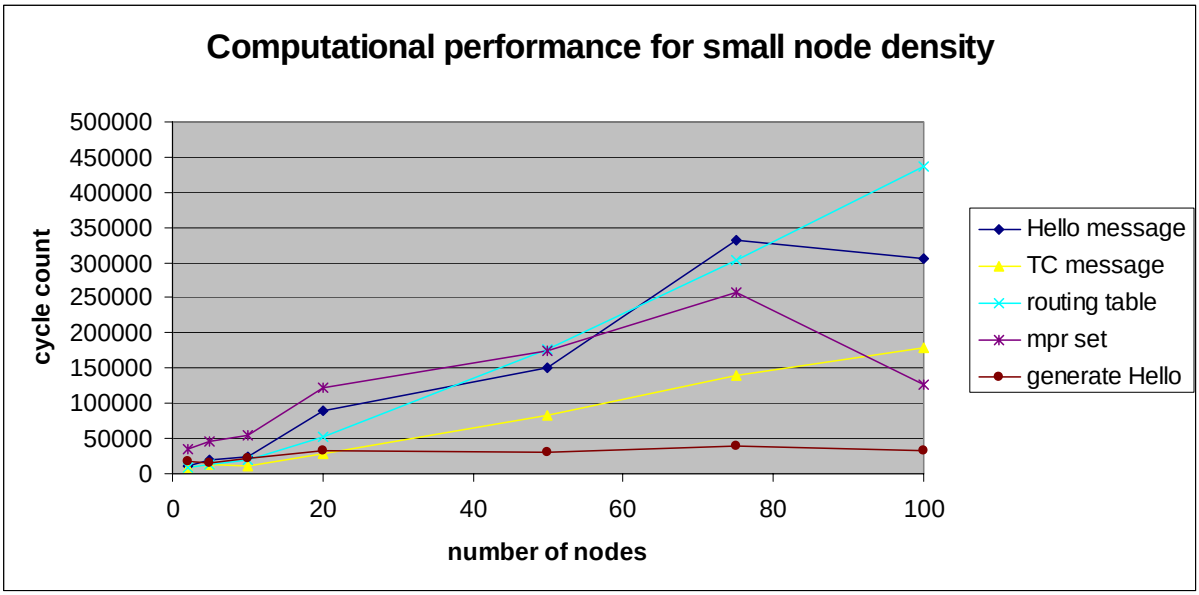
From analyzing the code it appeared that the algorithms for MPR election and route calculation would be the most performance critic ones. Therefore cycle count measurements were added to these routines as well as the corresponding elements handlers to obtain the results of the measurements. Furthermore measurements for packet processing were obtained by the use of the standard Click elements *SetCycleCount* and *CycleCountAccum* (both only available in kernel level). The *CycleCountAccum* element also provides the results of the measurements to the user through handlers.

Tests were run for network scenarios of 3, 5, 10, 20, 50, 75 and 100 nodes. For each number of nodes, 5 different scenarios were used, differing in the size of the area for the nodes. The five areas were chosen to maintain a constant ratio of nodes per square meter and thus a theoretical number of nodes within the radio range of the nodes. The five scenarios for each number of nodes corresponded to a medium number of 10, 20, 30, 40 and 50 nodes within radio range.

5.6.2 Results

The results complied with the assumption that MPR election would scale bad with node density and thus the number of neighbor nodes and two hop neighbor nodes while routing table calculation would scale bad with general network size.
So far the code was not optimized very intensively for execution speed therefore especially the MPR computation algorithm might still be improvable.

Small node density

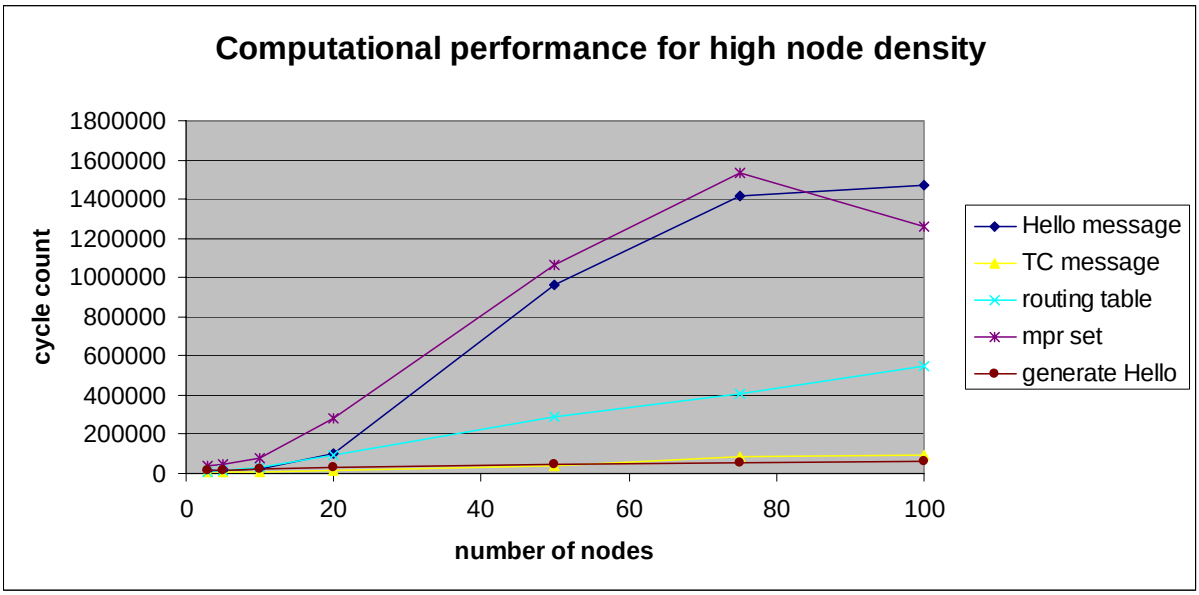


Picture 19: Computational performance for small node density

For small node density (medium of 10 nodes theoretically within communication range) the results of the measurements showed that MPR calculation is more time consuming than route computation for small number of nodes, but as network size grows, route calculation scales even worse.
While HELLO messages can result in MPR set recalculation and route recalculation at the same time, they not always include new information forcing such recalculations. TC messages can only initiate route recalculation if they contain new topology information. Therefore the average processing time of a TC message depends on the time to execute the routing table recalculation but as a lot of TC messages do not result in initiating this recalculation the average time is lower. Processing time of a HELLO message depends on the sum of MPR and route calculation but also a lot of HELLO messages do not require any action to be taken. The generation of a new HELLO message takes comparatively little time. As the time for recomputing the MPR set mostly depends on the size of the local neighborhood but not on the

general network size the drop down in MPR calculation time for the 100 node case is not a strange result.

High node density

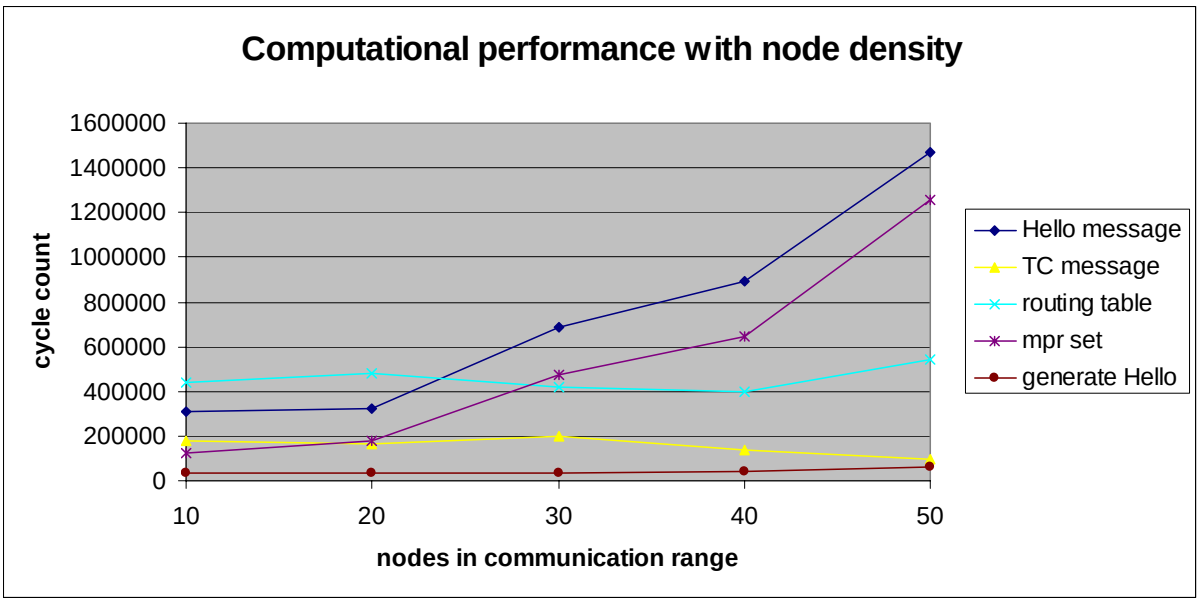


Picture 20: Computational performance for high node density

In the denser network (medium of 50 nodes theoretically within communication range), it can be observed that MPR election and therefore HELLO message processing needs a lot more time than in the network with small node density. Routing table calculation and TC messages processing behave more or less the same as before.

Computational performance depending on node density

Looking at five different density cases for a 100 nodes network the result is even more obvious.

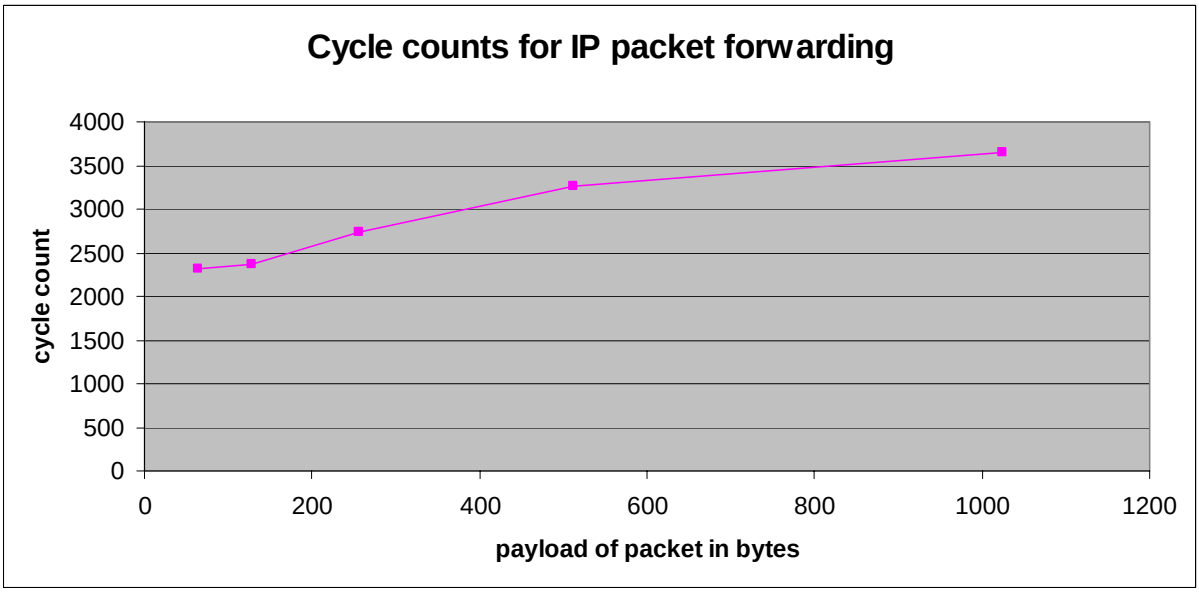


Picture 21: Computational performance with node density

While the calculation of the routing table stays more or less constant and therefore TC message processing as well, MPR set computation and HELLO message processing increase a strongly with increasing node density.

IP Data Packet forwarding

While OLSR calculation algorithms are time consuming especially for large and dense networks, pure IP forwarding with the OLSR Click code proved to be independent of network size and fast. As from element to element only pointers to the packet data structure are passed the results differ only little with the packet size of the packet to be forwarded.



Picture 22: Cycle counts for IP packet forwarding

To obtain these results the cycle count was reset after successfully doing MAC Address resolution by the ARP request and ARP reply mechanism as the *ARPQuerier* enqueues the first incoming packet of an unknown destination until receiving the corresponding ARPreply, leading to too high cycle count measurements for CPU usage due to packet processing.

The values for IP processing neglect the CPU cycles used by the device driver for incoming and outgoing packet, the final enqueuing and dequeing and the *FromDevice* and *ToDevice* elements. But the order of magnitude of this value suggests that pure IP forwarding would even on a rather slow device with only 133 MHz be possible with typical wireless link speeds of 10 Mbps.

5.6.3 Conclusion

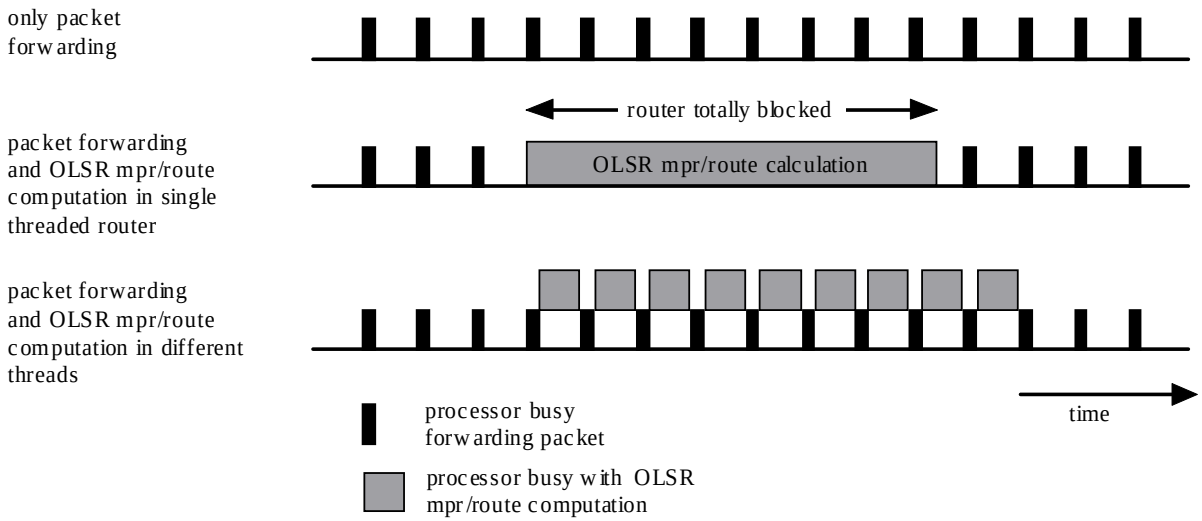
Using a single threaded environment like Click for packet forwarding and more complex route maintenance routines at the same time totally blocks the router during the execution of these maintenance routines. In the OLSR protocol route maintenance tasks can occur after packet reception or after data from the information bases has timed out, especially after links have expired. The eventually following updates of the MPR set and the routing table block the router completely until the execution of these routines has terminated.

One possibility not to block the router too many times would be not initiating a new MPR or route calculation as a result of packet processing but rather scheduling the execution of an update for some time later, thus establishing a maximum number of recalculations per time interval. In large networks this could lead to updating the changes included in more than one received packet with one recalculation rather than a complete recalculation each time. For high number of participating nodes not only the computation of the occurred changes need more time but also more changes take place that require updates to the MPR set or the routing table. On the other hand, some changes should be processed immediately if for example the link to a next hop route has expired to which data has to be forwarded by this node. Especially when introducing quality of service reaction to changes should be done quickly.

Another possibility would be to only process changes to the local neighborhood right away, updating the MPR set and the routing table when changes in the one or two hop neighborhood occur. Upon reception of a TC message with new information about the more distant network topology this information is only updated in the topology information base but no immediate

recalculation of the routing table is executed. Instead the recalculation is scheduled for some time later then probably including more changes published by other arrived TC messages as well or a route table update was necessary due to local neighborhood changes anyhow.

Looking at the processing time needed for packet processing and computation of algorithms also a multi threaded environment seems more appropriate for a routing protocol like OLSR (see Picture 23). Though Click SMP is able to run on double processor systems in two threads, it will still take some time until double/multiple core processors are introduced into small, energy saving mobile devices. Another solution would be the use of the more flexible forwarding plane in Click but doing time consuming route maintenance in a multithreaded environment. This can be done by communicating with the Click based forwarding plane through the handlers provided by the by Click elements. There already exists a project called XORP that aim is a universal software router platform that runs as a multithreaded user level application, using either a normal Linux system as forwarding plane or a Click based forwarding plane.



Picture 23: Single vs. multi threaded router environment

6 Installation of a test bed

As results obtained from simulation usually differ from those experienced in real life a test bed for ad-hoc networks running the developed OLSR Click code was designed and implemented. The in chapter 5.6 discussed drawbacks of a single threaded environment do not effect the routing in the designed network much, as it consists of fixed non mobile nodes, in which updates of the MPR set or routing table occur only a few times.

6.1 Design

At the MIT two test beds are implemented that run their DSR or DSDV grid implementation in Click. While one of these test beds is a test bed within closed buildings, the other test bed, the “Roofnet” [27] is an outdoor roof top network. To not start entirely from scratch but rather benefit from the already invested time and solved problems from the MIT research group the test bed to run the Click OLSR code is based on great parts of the work done at MIT especially their outdoor Roofnet test bed.

The design of a rooftop network on the university campus had to take into consideration the necessary permissions of university administration with regard to frequency use and places of installation.

6.1.1 Frequency use

One mayor issue obtaining permission for the installation of the devices transmitting radio waves on the campus side is non interference with existing installations. To avoid interference with existing wireless networks the first planned steps of the installation use directional antennas in order to limit the area affected by the wireless network.

Furthermore most existing networks on the university campus use the frequency band of 2,4 GHz using devices as standardized in IEEE 802.11b or 802.11g. As the use of 802.11a used to be prohibited in Spain, the frequency band at 5 GHz as specified in 802.11a is so far only used in very few cases. Adding the greater number of available channels compared to 802.11b/g and the high bandwidth of 802.11a, 802.11a was chosen as the wireless standard to be used.

The legislative aspect of 802.11a in Spain changed in June 2003 with the publication of the „Boletín Oficial del Estado" (BOE) number 175 [28] which modified the "Cuadro Nacional de Atribución de Frecuencias" (CNAF) [29], including the new "nota de utilización nacional" UN-128 [30] which regulates the use of the 5 GHz spectrum used in 802.11a

UN-128

In the UN-128 the use of the frequency band 5150 - 5350 MHz is permitted in closed areas when complying the following maximum transmit power limitations as shown in Table 1.

Table 1: UN-128 transmit power

	Maximum transmit power		
Frequency band (MHz)	Systems without TPC	Systems with TPC	Systems with TPC and DFS
5150-5250	30 mW	120 mW	200 mW
5250-5350	60 mW with DFS	200 mW with DFS	200 mW

The permitted transmit power depends on two features the wireless interface card should be equipped with. These are transmit power control (TPC) and dynamic frequency selection (DFS).

The frequency band 5470 – 5725 MHz can be used inside or outside of closed areas with maximum transmit power up to 1 W but the systems using the frequency band should be equipped with TPC and DFS.

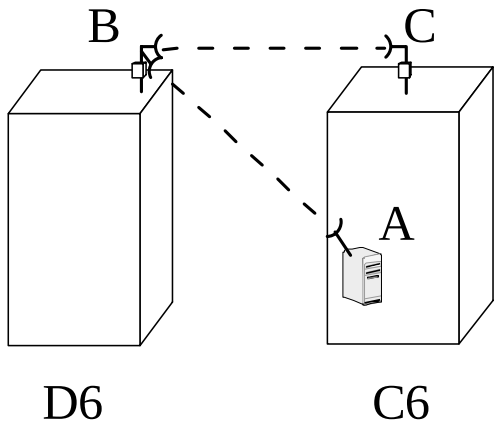
To avoid the restrictions of the frequency band from 5150 – 5350 MHz with respect to the use inside of closed areas, the 5470 – 5725 MHz band was chosen.

6.1.2 Nodes location

To start with a setup that would receive permission from university administration quicker the design was split up in different steps. The aim was to start with a small network to make first tests and prove functionality but still would be easy to extend later on once necessary permission for further installations have been obtained.

Step One

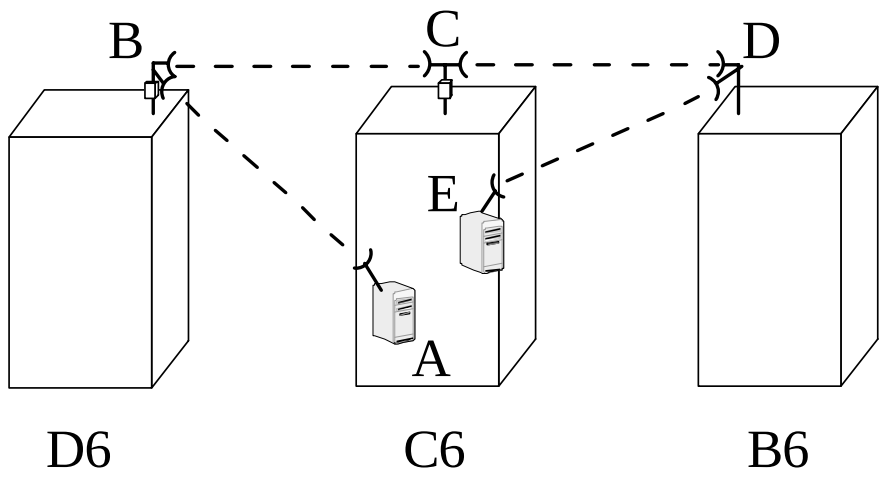
In the first step of the installation the network (see Picture 24) consists of one node with two directional antennas installed on the roof of the building D6 (node B) and two nodes installed inside (node A) and on top (node C) of the building C6, both with just one antenna. This scenario already permits test of packet forwarding at the intermediate node B and analysis of wireless outdoor routes over two hops.



Picture 24: Initial setup of three nodes

Step two

In a second step (see Picture 25) a further node will be installed on top of the building B6 (node D) and another one inside the building C6 (node E). The node on top of the building C6 (node C) will be equipped with a second wireless interface and a second antenna. The node D on top of B6 also uses two directional antennas while the new node E inside of C6 is equipped with only one antenna. This scenario provides four wireless links and routes over up to 4 hops.



Picture 25: Follow up setup with 5 nodes

Future enhancements

This basic setup can be made more sophisticated to develop a more complex test bed in future once the first initial version has been fully installed.

Incrementing the number of nodes

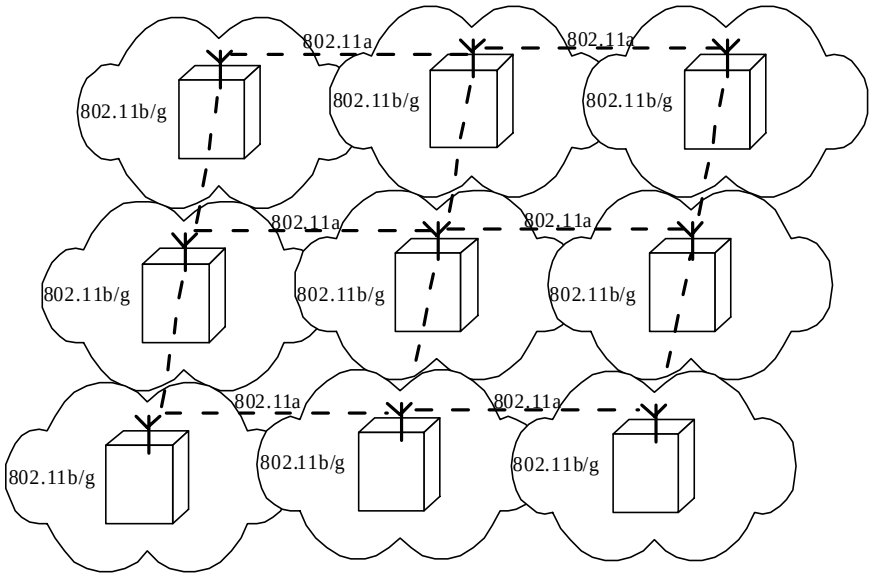
As one of the features of ad-hoc routing protocols is the ease of adding further nodes to the network, the test network can be easily extended by configuring a new node with the Click OLSR code and connecting it at the new site of installation once university administration approved the installation of a further node.

Using omni directional antennas

Using omni directional antennas rather than directional point to point links would allow making use of the multipoint relay feature of the OLSR protocol, which using only point to point links is not taken advantage of. Though frequencies used with omni directional antennas are spread wider using 801.22a should avoid possible interferences on the campus ground. The omni directional antennas would allow building real meshed networks in which a node has a lot of links to other nodes leading to a more interesting network structure. Furthermore the wider area of transmitted signal would allow interaction with mobile devices for real life tests including mobile nodes.

Acting as access point

Furthermore the network can be enhanced to provide access point functionality to users on the campus side. This could be done with the same nodes, offering access point functionality for example in the 802.11b/g frequency band to mobile users while establishing an 802.11a based OLSR network as a backbone.



Picture 26: Mesh backbone in 802.11a with access points in 802.11b/g

6.2 Installation of the nodes

The exact installation of the nodes depends upon whether it is an indoor node (Picture 25: nodes A, E) or one used on the rooftop (nodes B, C, D). The differences are within the used hardware, used antennas and related to the used hardware the software setup.

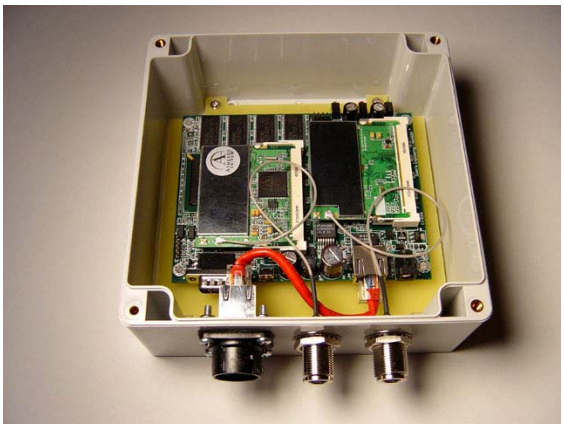
6.2.1 Hardware

Generally a node participating in the network consists of a device running the OLSR Click code. This device is equipped with wireless interface cards and connected to an antenna. The concrete Hardware setup depends upon where the node is located, on the roof top or within a building. While nodes on the rooftop are small embedded devices enclosed in a weather proof box, mounted to an antenna mast and connected to the antenna(s), nodes inside the building are ordinary PCs, also equipped with a wireless interface card and connected to an antenna.

Rooftop nodes

The rooftop nodes consist of a small box called Mark II [31] bundling an embedded device based upon the Soekris 4526 system board [32] equipped with one or two wireless mini PCI

interfaces. These wireless interface cards are connected to the outside antenna connectors of the box through small pigtail cables. The Ethernet port as well is connected to a water proof sealed outdoor connector. Using power over Ethernet (POE) as specified in IEEE 802.3af, this Ethernet connector is also used as power supply for the outdoor installations.

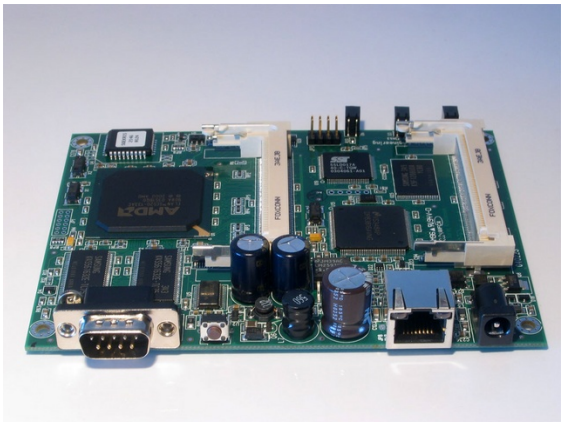


Picture 27: Mark II open

Picture 28: Mark II closed

Soekris 4526

The Soekris 4526 System board is a general purpose low power consuming computer. Its main features related to this thesis are the following:



Picture 29: Soekris 4526 System board

133 MHz AMD ElanSC520 processor
64 MByte SD-RAM
64 MByte Flash Storage
One 10/100 Mbit RJ45 Ethernet port
Two mini PCI type II slots
Serial port
Power over Ethernet support according to 802.3af standard
Low power consumption

Wireless interface card

The used wireless interface card supporting 802.11a and 802.11b/g is based upon the Atheros chip set family. For 802.11a technology Atheros based wireless interfaces are the most commonly used and provide decent open source driver support for Linux operating systems. There is also a special driver “madwifi stripped” [33] which allows close interaction of the driver functionality with Click elements.

Picture 30: Mini PCI Wireless interface card

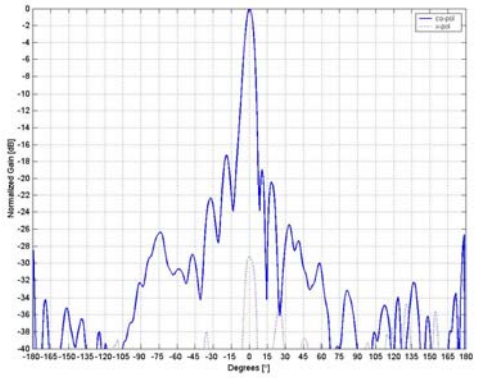
Features include 6 Mbps to 108 Mbps in 802.11a and 802.11g as well as 1 Mbps to 11 Mbps in 802.11b. Furthermore the card offers TPC support, providing flexibility to adjust RF output power as well as DFS support choosing best frequencies to use.

The outdoor antenna

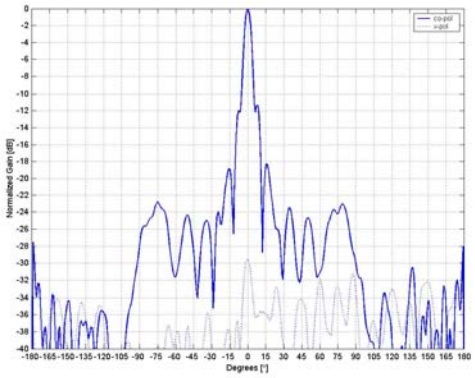
For the outdoor nodes directional antennas of the type parabolic grid were chosen. These provide a good radiation pattern while reducing weight and effects of weather conditions. To limit the area influenced by the emitted frequencies antennas with high directivity were elected. The antenna used, an Equinox SA5224 [34], has the following important characteristics:

Gain (dbi): 25
Front to Back Ratio (dBc): 27
VSWR (typical): 1.3
Beamwidth (dB): 10° horizontal, 11° vertical
Frequency spectrum: 5250 – 5850

Picture 31: Antenna parabolic grid



Picture 32: Radiation pattern E-plane



Picture 33: Radiation pattern E-plane

The antenna is connected to the standard female N panel mount connector of the Mark II boxes.

Indoor nodes

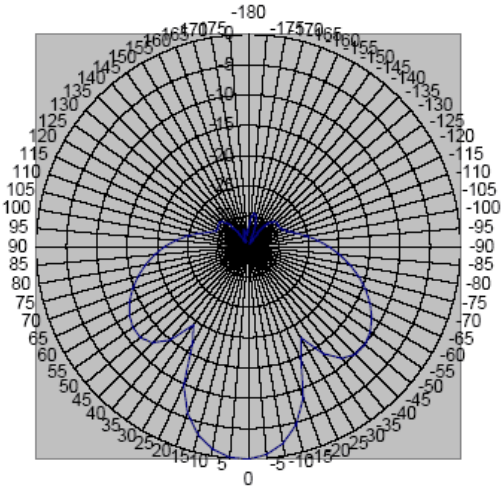
The indoor nodes consist of an ordinary PC equipped with an Atheros wireless LAN 802.11a/b/g adaptor, with the same features as the one described above. This wireless interface card is connected through a pigtail to an antenna cable leading to the antenna.

The indoor antenna

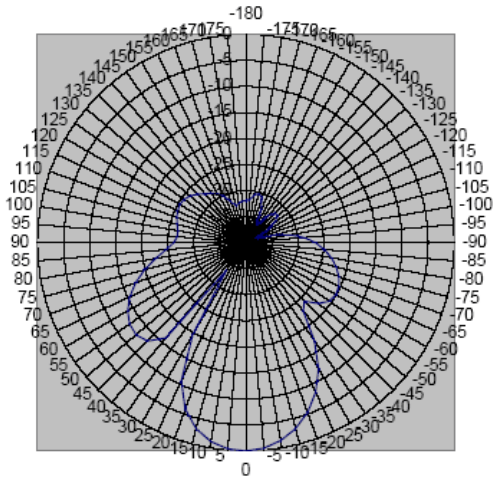
As indoor antenna a planar directional antenna Stella Doradus 564040 [35] was chosen. As the antenna is placed inside the window of the building, between the glass and the shades, a parabolic antenna would need too much space. Therefore a smaller, less directional planar antenna was chosen, which still provides sufficiently good radiation characteristic:

Gain: 15 dBi
3dB beam Pattern: 25° x 30°
VSWR 1.4:1
Front to Back Ratio: 32 dBi
Cross Pol. Discrimination: 20 dBi
Frequency: 5.4-5.7 GHz
Size: 24cm x 23cm x 3,5 cm

Picture 34: Planar antenna



Picture 35: Radiation pattern planar antenna E-plane



Picture 36: Radiation pattern planar antenna H-plane

6.2.2 Software

The software of the nodes should run automatically all necessary configurations upon booting or rebooting the node. To make deployment of nodes easier these configuration scripts are node independent. The parameters that are individual for each node, like the IP Address or whether to run the one or two interface Click script are deduced automatically.

Software installation generally differs for the small boxes based on the Soekris board and full PCs with respect to the installed operating system. While a full PC can basically run any Linux distribution that works with Click, the small boxes have some strict constrains with respect to the size of the Linux distribution due to their limited size of storage on the flash. Generally only a Linux distribution with Click in kernel level, the madwifi stripped driver for the Atheros wireless interface and the Click configuration is required.

Addressing scheme

For the Click OLSR roof top network the private class A network addresses in the range from 5.0.0.0 to 5.255.255.255 are used. The concrete IP Address assigned to a nodes interface is deduced from the MAC address of this interface. As the wireless interfaces are all from the same vendor this guarantees unique IP Addresses.

Frequency settings

Frequency use is regulated differently by each regulatory authority and the concrete country in which the device is used. The regulatory domain and the country code of the wireless card therefore have to be set correctly. From this selection of regulatory domain and country code depend the available channels and frequencies that can be used. For use in Spain the corresponding regulatory domain is the European Telecommunications Standards Institute (ETSI) [36] corresponding to a regdomain value of 0x30 hexadecimal or 48 decimal. The country code of Spain is 724 [37].

Node startup

After loading the Linux operating system a script (*/etc/init.d/roofnet*) calls the configuration and initialization routine (*/home/roofnet/scripts/start_roofnet.pl*) which sets up the system to run as an OLSR routing node. In this Perl script, the necessary kernel modules (*/home/roofnet/kernel/*) are loaded. This includes the needed device drivers for the wireless

interfaces to run the madwifi stripped driver as well as calling the script which loads the Click kernel module and mounts the Click proc like file system to communicate with the Click router running as a kernel module (*/home/roofnet/scripts/start_click.sh*). After loading the Click kernel module the router is not yet initialized with a configuration. The router can now be accessed through the handlers of the router and the individual elements through the /click file system.

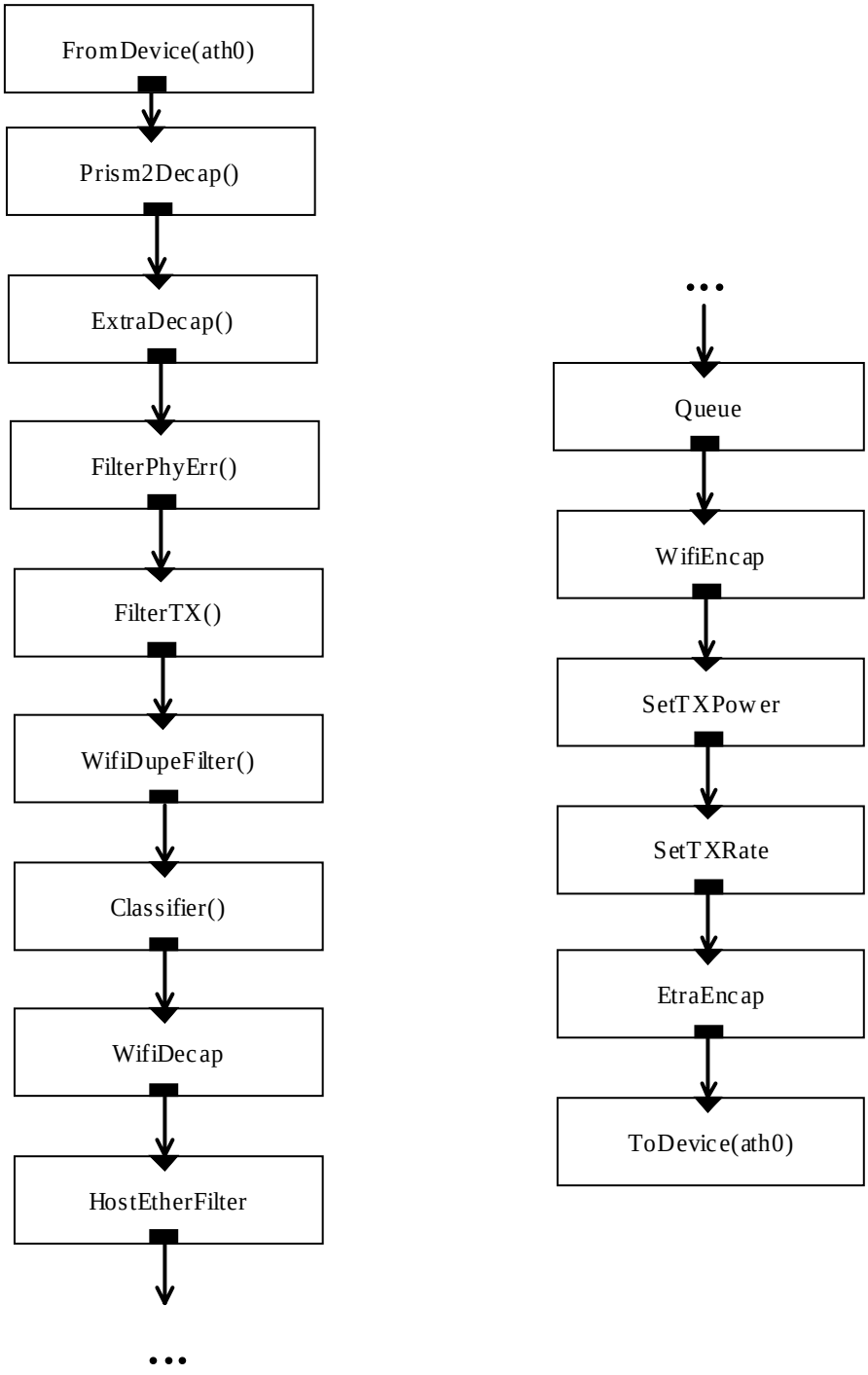
The script now determines the number of installed wireless interfaces (one or two) and calls the corresponding script (*/home/roofnet/scripts/gen_config_teunet_1_interface.pl* or */home/roofnet/scripts/gen_config_teunet_2_interfaces.pl*). The output of this script is redirected to the /click/config handler, thus installing the desired Click configuration.
These scripts configure either one or two wireless interfaces. To do this they can use passed parameters or the default values for configuring the operation mode (a/b/g) and channel. The script assigns the IP-Address to the wireless interface(s). Thereafter they generate the corresponding Click configuration for one or two interfaces dynamically including the assigned IP-Addresses and the MAC-Addresses. By the above mentioned redirection the router is initialized and started.

After completing this initialization routine the node boot up ends at asking for the username and password of the user, while the node already acts as an OLSR node in the background.

Click Configuration

On the nodes Click is run in kernel level. While the elements compiled into the kernel module "/home/roofnet/kernel/click.o" are the same as described in chapter 5.1, the Click configuration was adapted to work similar like the MIT Roofnet project with the special Madwifi Stripped driver. This driver allows to access functions of the driver from within Click elements. In the Click wifi package are some special elements that allow interacting with the driver of the wireless interface. This can for example include receiving signal to noise information or link feedback within Click. The input and output chain are therefore completed by some special elements for the use with wireless network interfaces, as seen in Picture 37. The left part shows the additional elements for the input chain while the right part shows the additional elements for the output chain. Detailed description of each element can be found in the header file of the corresponding element in the Click source code. So far the additionally provided information is not used but future enhancements could easily use this

already available information. For example MPR election or route table computation could be based on signal to noise information or transmission rates on the corresponding links



Picture 37: Additional elements for special wireless interface features

Operating systems of the outdoor nodes

The small embedded devices use a modified mini-Linux distribution which is based upon the pebble mini-Linux distribution. Pebble itself is based upon Redhead Linux, focusing on the programs needed to run a Linux operating system on small embedded devices especially for

wireless purposes. Further packets were removed by the Roofnet people to adapt pebble to the specific needs of a wireless routing test bed on Soekris boards. The installed packets include among others the Perl interpreter, an httpd server, a ssh server and a dhcpd server.

Boot up options

The Soekris Board allows interrupting standard boot up by attaching a serial cable to the serial console of the board and working from another PC over the console on the Soekris board. This way upon boot up and during runtime all information is put on the terminal screen and it can be interacted with the keyboard of the connected PC. During the boot up screen the boot up can be intercepted by typing ctrl^p. This leaves the device in the pre-boot environment, from where the boot process can be altered. Besides the normal boot option it is possible to boot the device over the network (command: *boot f0*).

Network boot up

The device can use the PXE boot up environment to boot up over network rather than from a locally installed operating system. To boot over the network the following server have to be installed within the network and configured with their corresponding configuration files (named in parentheses).

DHCPD [38] (/etc/dhcpd.conf)
PXE [39] (/etc/pxe.conf)
TFTPD [40] (/etc/xinet.d/tftp; /tftpboot)
NFS [41] (/etc/exports; /etc/hosts.allow; /etc/hosts.deny; /export/)

Upon network boot up, the board requests an IP-Address over DHCP. The DHCP configuration of the dhcpd server also includes the name of the file to be given to a PXE client. This file contains the boot image which is transferred by the trivial file transfer protocol (TFTP) to the Soekris board that requested it. The board now boots the file from this image, and mounts as root file system the roofnet root directory as exported from the NFS server over NFS.

During the initialization phase for which the configuration files from the mounted roofnet root directory are used, the device determines whether the boot up occurred over the network or locally. In case of a network boot up, the device starts the reinstallation routine of its local system automatically.

Reinstallation of the software

To install the software on the device the NFS exported directory is mounted (if not already done) and the flash memory of the Soekris board is mounted for read and write. Then the flash memory is repartitioned and reformatted. Thereafter the files from the over NFS mounted roofnet root directory are copied to the local flash memory and form the new root directory of the Soekris board. When the device is automatically reinitialized thereafter, boot up takes place locally from the freshly installed files on the flash memory. This boot up now includes the start and initializations of the OLSR node as described above.

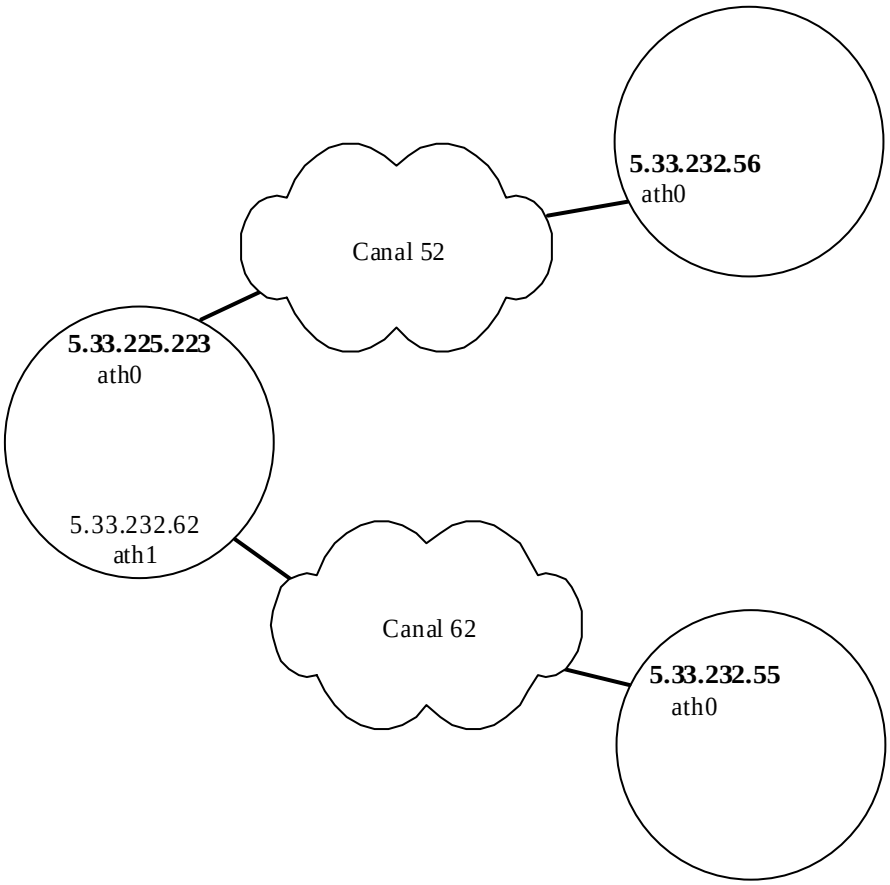
Small changes that do not effect the running system, like only changing the Click configuration generation scripts, can easily be done over a ssh connection over the wireless OLSR network. In this case, the flash device has to be remounted read write and files can be deleted and created or copied on the flash memory.

6.3 Test

As the installation on the roof of the university took a lot more time due to administrative issues, no test could be run with the actually planned network. This work will be continued by follow up master thesis projects.

6.3.1 Test setup

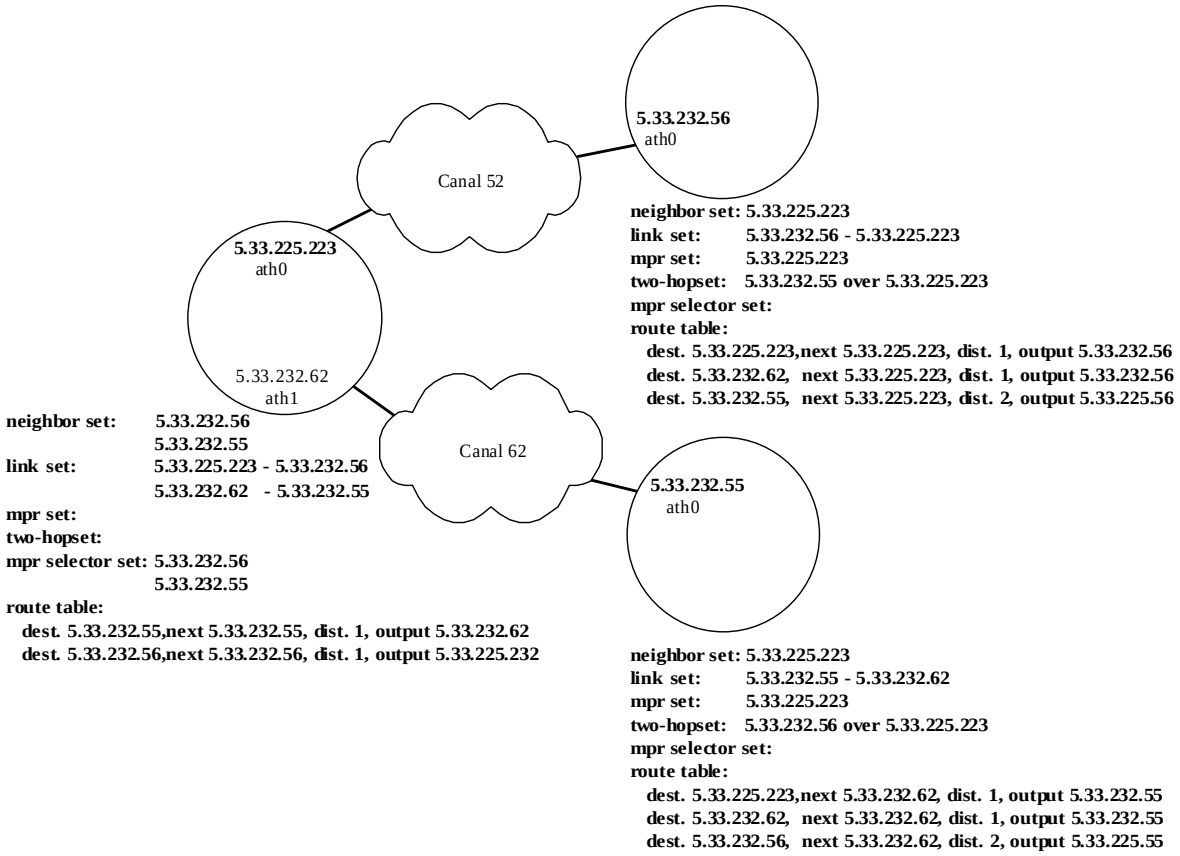
To test the code and the installed devices in a set up similar to step one described in chapter 6.1.2, tests were done still in the laboratory, imitating the links of the directional antennas using different channels. One of the boxes was configured with two interfaces, each operating on another channel while two boxes were configured each with just one interface, operating on the corresponding channels. Thus a three node, two hop network was built, in which the middle node has to forward the traffic, as can be seen in Picture 38.



Picture 38: Laboratory test scenario of test bed

6.3.2 Connection test

Upon connecting and configuring the nodes, the nodes started to exchange HELLO packets and to correctly set up their link sets and neighbor information repository. The middle node correctly was elected as MPR and started to emit TC message, as well as HELLO and MID messages. The two nodes with just one interface correctly did neither emit MID nor TC messages. The route tables were built up correctly, as can be seen in Picture 39.



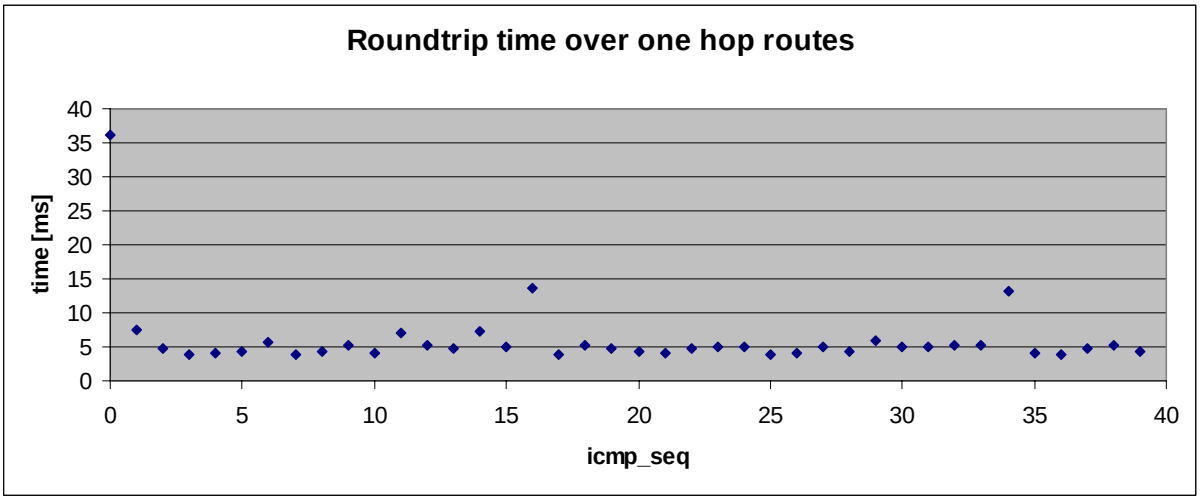
Picture 39: Connectivity test results

6.3.3 Measuring roundtrip times

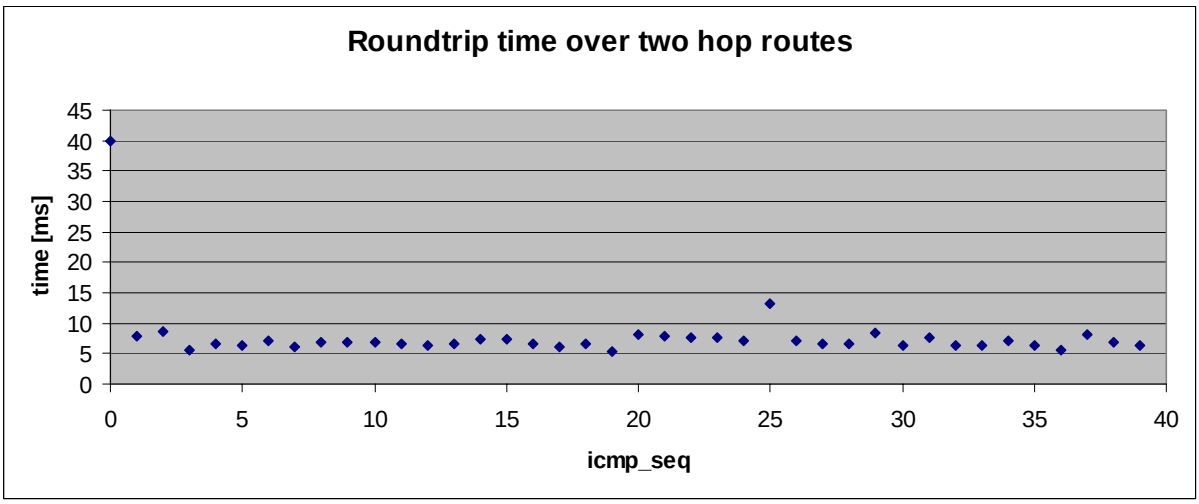
The echo request and reply mechanism (ping) was used to test correct establishment of routes and information exchange between nodes as well as to measure round trip time of packets.

Between the first echo request and the reception of the corresponding echo reply a longer time span was observed. This is due to the initial ARP resolution phase in which ARP packets are exchanged before the echo request and reply can be transmitted. Further echo replies were received within a comparatively small time. Typical medium values for a one hop route (see Picture 40) are around 6 ms while medium values for the two hop route (see Picture 41) are around 8 ms. During a larger serious of echo request and replies, peaks of up to the double of

the medium value in the roundtrip time occurred from time to time. This could be an indication that one or more of the involved devices were busy generating or processing OLSR control packets and thus blocking packet forwarding during this time in the single threaded Click environment.



Picture 40: Roundtrip times over one hop routes



Picture 41: Roundtrip times over two hop routes

6.3.4 Measuring bandwidth

To get a first idea of the possible transmit rate a quick bandwidth check was conducted. A file of 9.5 Mbytes was placed in the temporary directory residing on the ram disk of node A. Due to the limitations of the used device, no larger file was used. The httpd server on this node was configured to allow access to this directory.

First from node B an HTTP request for this file was initiated by using the *wget* command, storing the incoming file after being transmitted over one wireless link on the temporary storage ram disk of node B.

Thereafter the test was repeated for a two hop route initiating the *wget* command on node C, transmitting the file over two wireless links, with node B working as a forwarding node in the middle.

Results

In the first case a medium bandwidth of 441.20 KByte/s or 3,5MBit/s was achieved. The obtained bandwidth of the two hop link did not differ much. A medium value of 410.35 KB/s or 3.3 MBit/s resulted.

Discussion of Results

The obtained bandwidth is the bandwidth of actual transmitted file data, not containing the HTTP and TCP overhead as well. The influence of the varying response times using TCP connections including flow control should be compared to results obtained using UDP transmission of packets.

With the embedded routing devices acting as router as well as data source and sink simultaneously some processor capacity was used for these purposes as well, which otherwise could have been used for pure packet forwarding leading probably to higher bandwidth rates.

Laboratory setup did not include any antennas. The pigtail connector to which usually the antenna cable is connected was the only “antenna” used. With this setup, links sometimes tended to fail or loose packets.

This test therefore demonstrates more the correct working of the implementation with regards to software and hardware, including a middle node acting as pure forwarding node. The results can give an idea of a bottom value, but the above mentioned limitations should be considered when judging the results. Once the installation is completed with actual antennas and connections to the campus network, the test should be redone more profoundly including different nodes acting as packet sources and sinks and using UDP packet generators.

7 Future work

This master thesis initiated a new project in the research group. After this master thesis this project will continue and further work will be necessary and can be assigned as similar master

thesis to other students. So far two follow up master thesis were initiated. One of them deals with the development of a client server structure for monitoring the state of the developed test bed. The other will implement auxiliary OLSR functions into the Click code and try other routing metrics as well as continue the installation work of the network. Furthermore two PhD students of the research group started the XORP based version of OLSR using an advanced forwarding plane based on Click elements.

7.1 OLSR implementation

Further work on the OLSR implementations can involve optimizations of the developed OLSR Click code as well as the development of a multithreaded version for the OLSR control part of the routing implementation.

7.1.1 OLSR Click code

So far the OLSR Click code implements OLSR core functionality and especially the MPR election and route computation scale rather bad with network size. This leads to possible optimizations and extensions of the code.

Optimizations

Performance critic algorithm should be optimized for performance, especially for networks that include mobility. As explained in 5.6.3, scheduling route updates rather than executing immediately the corresponding routine could be tested and compared to the existing implementation.

Extensions

Further extensions to the OLSR Click code should be implemented. These, among others, could include OLSR auxiliary features as specified in the RFC as well as new QOS algorithms or different metrics for routing, security policies or multicast support.

7.1.2 OLSR in XORP

As explained in 5.6.3 the existing implementation is not suitable for large and mobile networks on wireless devices that do not have computing performance in abundance. For these purposes an OLSR implementation in multithreaded environment would be useful, like

a Click based forwarding plane running within OLSR route maintenance implemented in the XORP framework.

7.2 Test bed

As this thesis was the initial work developing the code, designing the test bed and installing the nodes for the test bed a lot of future work on the test bed can be thought of to keep the project alive.

7.2.1 Software

The designed OLSR Click code as well as the installed system on the test bed nodes probably have to be maintained although in the tests they proved to work fine. As mentioned above a thought should be given changing the test bed from a Click based version to a XORP and Click based version.

Maintenance

The current implementation needs to be maintained. This includes upgrading of the roofnet or Click based parts upon the release of bug fixes for the madwifi stripped driver or the used Click elements.

XORP and Click

It should be tried to run the XORP/Click OLSR combination on the small devices installed as a test bed as well and compare the obtained results, especially for more dynamic network topologies that imply more recalculations of MPR sets and route tables. This applies even more when building larger strongly meshed networks based on omni directional antennas and including potentially mobile nodes.

7.2.2 Network

The in chapter 6.1.2 outlined steps and eventually the there mentioned enhancements should be implemented. Further node installations should be planned and executed.
For a more meshed network and tests including mobile nodes, the use of omni directional antennas should be taken into account.

Another possibility to deploy a network consisting of more nodes in a cheap way would be to install an interior test bed as well, which could use the same software and the same embedded board, but would not need outdoor enclosure or external antennas. A large indoor network could thus be installed rapidly and cheaply.

7.2.3 Hardware

A part from the right now used hardware, it can be thought of using different hardware for the installations. The company from where the Mark II boxes were ordered now sells them with the more powerful Soekris 4826 board [42] including a Geode 266 MHz processor. The performance using this more powerful device should be evaluated.

Interesting for future installations could be the WRAP boards from pc-engines [43] as well. Those boards are also equipped with the 266 MHZ AMD Geode processor. Furthermore this board can be configured with 128 MB of RAM rather than the 64 MB of the Mark II boxes. As flash storage they use a standard Compact Flash cards which can be of larger size then the 64 MB compact flash of the Metrix boxes.

8 List of references

[1] W. Brown, V. Marano: *"Future Combat System - Scalable Mobile Network Demonstration Performance and Validation Results"* MILCOM 2003, Boston, Massachusetts

[2] Widens Project: *"Wireless Deployable Network System For Future Public Safety"*, Thales Communications S.A. 2004 – www.widens.org

[3] FleetNet Project: "*Inter-Vehicle Communication*" - http://www.et2.tu-harburg.de/fleetnet/index.html (2003)

[4] Cartalk2000 Project: *"CarTALK 2000 - Safe and Comfortable Driving Based Upon Inter-Vehicle-Communication"* - http://www.cartalk2000.net/ (2004)

[5] Freifunk: "*Freie Funknetze im deutschsprachigen Raum*" http://www.freifunk.net/ (2005)

[6] D. Maltz, D. Johnson, Y Hu: "*The Dynamic Source Routing Protoco for Mobile Ad Hoc Networks*", INTERNET-DRAFT, July 2004.

[7] C. Perkins, E. Belding-Royer, S. Das: *"AODV RFC3561"*, experimental edition, July 2003.

[8] F. Templin, R. Ogier, M. Lewis: "*TBRPF RFC3684"*, experimental edition, February 2004.

[9] Clausen, Jacquet: *"OLSR RFC3626"*, experimental edition, October 2003.

[10] E. Kohler: "The Click Modular Router". PhD Thesis, MIT, February 2001

[11] M.Neufeld, A.Jain & D.Grunwald. *"Nsclick: Bridging Network Simulation and Deployment."*, University of Colorado, Boulder, Colorado 2002

[12] B. Chen, R. Morris: "*Flexible Control of Parallelism in a Multiprocessor PC Router*" USENIX Annual Technical Conference, Boston, Massachusetts June 2001

[13] M. Handley, O. Hodson, E. Kohler: "*XORP: An Open Platform for Network Research*", ICSI Center for Internet Research, Berkeley, California 2002

[14] Niraj Shah, William Plishker, Kurt Keutzer. "*NP-Click: A Programming Model for the Intel IXP1200.*" (HPCA-9), Anaheim, CA, February, 2003.

[15] G.Schelle, D. Grunwald: *"CUSP: A Modular Framework for High Speed Network Applications on FPGAs"*, (FPGA 2005)

[16] "*The Grid Ad Hoc Networking Project*", MIT, Massachusets US - http://www.pdos.lcs.mit.edu/grid/

[17] C.E.Perkins, T.J. Watson: *"Highly dynamic destination sequenced distance vector routing (DSDV) for mobile computers"*, in: ACM SIGCOMM 94 Conference on Communications Architectures, London, UK 1994

[18] A. Tornquist: *"Modular and Adaptive Ad Hoc Routing in Click."* MSc Thesis,

University of Colorado at Boulder, 2001

[19] N.K. Palanisamy: *"Modular Implementation of Temporally Ordered Routing Algorithm."* ME Thesis, University of Colorado at Boulder, 2001

[20] INRIA *"Optimized Links State Routing"* - http://menetou.inria.fr/olsr/ (2004)

[21] A. Tønnesen: *"Impementing and extending the Optimized Link State Routing Protocol"*, UniK University Graduate Center, University of Oslo 2004

[22] D. De Couto, D. Aguayo, J. Bicket, R. Morris, *"A High-Throughput Path Metric for Multi-Hop Wireless Routing"*, (MobiCom '03), San Diego, California, September 2003.

[23] QOLSR: *"Qos with the OLSR protocol"* - http://qolsr.lri.fr (2004)

[24] ProteanForge: *"OLSR Summary"* - http://pf.itd.nrl.navy.mil/projects/olsr/ (2004)

[25] GRC: *"OLSR implementation"* - http://www.grc.upv.es/software/intro_olsr.html (2004)

[26] T. Dørum: *„Implementing OLSR for Click"*, Norwegian University of Science and Technology June 2004

[27] D Aguayo, J. Bicket, S. Biswas, D. De Couto, R. Morris: *"MIT Roofnet Implementation"* - http://www.pdos.lcs.mit.edu/roofnet/doku.php?id=design

[28] BOE: *"ORDEN CTE/2082/2003"* - http://www.setsi.mcyt.es/legisla/teleco/octe208203.htm, July 2003

[29] CNAF: *"Cuadro Nacional de Atribución de Frecuencias"* - http://www.setsi.mcyt.es/espectro/cnaf.htm (2003)

[30] UN-128: *"CNAF 2002: NOTAS UN (121 - 131)"* - http://www.setsi.mcyt.es/espectro/notas_un02/un121_131_02.htm (2003)

[31] Metrix: *"Metrix Kit Mark II"* - http://metrix.net/metrix/products/packages/MET-KIT-MARK-II.html (2004)

[32] Soekris: *"Soekris Engineering 4526"* - http://www.soekris.com/net4526.htm (2004)

[33] J.Bicket: *"Madwifi Stripped / Click Wifi Driver"* - http://www.pdos.lcs.mit.edu/~jbicket/madwifi.stripped/ (2004)

[34] Equinox: *„Equinox SA5224"* - http://www.rfeq.com/pdf/rad_SA5X24.pdf (2004)

[35] Stella Doradus: *„Stella Doradus 564040"* - http://www.stelladoradus.com/pdfs/5.6/56%204040%20(05-01-04).pdf (2004)

[36] European Telecomunications Standard Institute - http://www.etsi.org/ (2004)

[37] C. Feather: *"Country codes in ISO 3166"* - http://www.davros.org/misc/iso3166.html

[38] R. Droms: *"Dynamic Host Configuration Protocol"* - http://www.dhcp.org/ (2003)

[39] M. Johnston: *„DHCP Preboot eXecution Environment (PXE) Options"* Internet draft - http://www.ietf.org/internet-drafts/draft-ietf-dhc-pxe-options-01.txt (2005)

[40] K. Sollins *"The TFTP Protocol"* - http://www.rfc-archive.org/getrfc.php?rfc=1350

[41] Network Working Group: *"NFS: Networks Filesystem Specification RFC 1094"* - http://www.apps.ietf.org/rfc/rfc1094.html

[42] Soekris: "Soekris Engineering net4826" - http://www.soekris.com/net4826.htm (2004)

[43] pcengines: "Wireless Router Application Platform" - http://www.pcengines.ch/wrap.htm (2004)

9 Appendix

9.1 Example Configuration for two interfaces at user level

```
elementclass OLSRnode{$my_ip0,$my_ether0, $my_ip1,$my_ether1,
$hello_period, $tc_period, $mid_period, $jitter, $n_hold, $t_hold, $m_hold,
$d_hold |

tun::KernelTun($my_ip0/24)
joinInput::Join(2);

forward::OLSRForward($d_hold,duplicate_set, neighbor_info,
interface_info,interfaces, $my_ip0)
interface_info::OLSRInterfaceInfoBase(routing_table,interfaces);
interfaces::OLSRLocalIfInfoBase($my_ip0 ,$my_ip1)

arpt :: Tee(2);
in0 ::FromDevice(eth0)
todevice0 :: ToDevice(eth0);
out0 :: SimpleQueue(200)->todevice0;

// Input and output paths for eth0
c0 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
in0 -> SetTimestamp->
HostEtherFilter($my_ether0 , DROP_OWN false, DROP_OTHER true)->c0 ;

joindevice0 :: Join(3)->out0 ;
c0 [0] -> ar0 :: ARPResponder($my_ip0 $my_ether0) -> [1]joindevice0 ;
arpq0 :: ARPQuerier($my_ip0, $my_ether0) -> [2]joindevice0 ;
c0 [1] -> arpt;
arpt[0] -> [1]arpq0 ;
c0 [2] -> Paint(0) -> [0]joinInput;
c0 [3] -> Discard;

output0 :: Join(2)->OLSRAddPacketSeq($my_ip0)->
UDPIPEncap($my_ip0 , 698, 255.255.255.255, 698)->
EtherEncap(0x0800, $my_ether0 , ff:ff:ff:ff:ff:ff)->[0]joindevice0 ;

hello_generator0 :: OLSRHelloGenerator( $hello_period,$n_hold,link_info,
neighbor_info, interface_info, forward, $my_ip0 , $my_ip0,SPREADINIT
false,JITTER 150)
->[1]output0

in1 ::FromDevice(eth1)
todevice1 :: ToDevice(eth1);
out1 :: SimpleQueue(200)->todevice1;

// Input and output paths for eth1
c1 :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
in1 -> SetTimestamp->
HostEtherFilter($my_ether1 , DROP_OWN false, DROP_OTHER true)->c1 ;

joindevice1 :: Join(3)->out1 ;
c1 [0] -> ar1 :: ARPResponder($my_ip1 $my_ether1) -> [1]joindevice1 ;
arpq1 :: ARPQuerier($my_ip1, $my_ether1) -> [2]joindevice1 ;
c1 [1] -> arpt;
arpt[1] -> [1]arpq1 ;
c1 [2] -> Paint(1) -> [1]joinInput;
```

```
c1 [3] -> Discard;

output1 :: Join(2)->OLSRAddPacketSeq($my_ip1)->
UDPIPEncap($my_ip1 , 698, 255.255.255.255, 698)->
EtherEncap(0x0800, $my_ether1 , ff:ff:ff:ff:ff:ff)->
[0]joindevice1 ;

hello_generator1 :: OLSRHelloGenerator( $hello_period,$n_hold,link_info,
neighbor_info, interface_info, forward, $my_ip1 , $my_ip0,SPREADINIT
false,JITTER 150)
->[1]output1

duplicate_set::OLSRDuplicateSet
olsrclassifier::OLSRClassifier(duplicate_set, interfaces, $my_ip0)
check_header::OLSRCheckPacketHeader(duplicate_set);

//Input
ip_classifier::IPClassifier(udp port 698,-)
get_src_addr::GetIPAddress(12)
joinInput->Strip(14)->CheckIPHeader->ip_classifier
ip_classifier[0]->get_src_addr->Strip(28)->check_header->olsrclassifier
check_header[1]->Discard
join_fw::Join (4)

//olsr control message handling
neighbor_info::OLSRNeighborInfoBase(routing_table,
tc_generator,hello_generator0,link_info,interface_info, $my_ip0,
ADDITIONAL_HELLO false);

topology_info::OLSRTopologyInfoBase(routing_table);
link_info::OLSRLinkInfoBase(neighbor_info, interface_info, duplicate_set,
routing_table,tc_generator)

routing_table::OLSRRoutingTable(neighbor_info, link_info, topology_info,
interface_info,$my_ip0);
process_hello::OLSRProcessHello($n_hold,link_info, neighbor_info,
interface_info, routing_table, tc_generator, interfaces, $my_ip0);
process_tc::OLSRProcessTC(topology_info, neighbor_info, interface_info,
routing_table, $my_ip0);
process_mid::OLSRProcessMID(interface_info, routing_table);

olsrclassifier[0]->Discard
olsrclassifier[1]->process_hello->Discard
olsrclassifier[2]->process_tc
olsrclassifier[3]->process_mid
olsrclassifier[4]->[2]join_fw
olsrclassifier[5]->[3]join_fw

process_tc[0]->[0]join_fw
process_tc[1]->Discard

process_mid[0]->[1]join_fw
process_mid[1]->Discard

forward[0]->[0]joinjitter::Join(2)->broadcast::Tee;
forward[2]->OLSRQueue (JUfw,200)->JUfw::JitterUnqueue(0.375)->[1]joinjitter
join_fw->[0]forward;
broadcast[0]->[0]output0
broadcast[1]->[0]output1
forward[1]->Discard

//handling of other ip packets
```

```
dst_classifier::IPClassifier(dst $my_ip0, -);
get_next_hop::OLSRGetNextHop(routing_table,interfaces)
join_cl::Join(2);
nexthopswitch::PaintSwitch;

ttl::DecIPTTL
dst_classifier[1]->ttl
tun->CheckIPHeader->[0]join_cl;
join_cl->dst_classifier
ip_classifier[1]->[1]join_cl
dst_classifier[0]->SetPacketType(HOST)->tun

ttl[0]->get_next_hop
ttl[1]->Discard
get_next_hop[0]->nexthopswitch
get_next_hop[1]->Discard
nexthopswitch[0]->[0]arpq0
nexthopswitch[1]->[0]arpq1

mid_generator::OLSRMIDGenerator ($mid_period,$m_hold,interfaces,SPREADINIT
false,JITTER 500)
tc_generator::OLSRTCGenerator($tc_period,$t_hold, neighbor_info, $my_ip0,
ADDITIONAL_TC false,JITTER 500)

joinforward::Join(2)

tc_generator->[0]joinforward->[1]forward
mid_generator->[1]joinforward

} //end of compound element
AddressInfo(my_ether0 eth0:eth);
AddressInfo(my_ether1 eth1:eth);
AddressInfo(my_ip0 192.168.2.1);
AddressInfo(my_ip1 192.168.2.2);

OLSRnode(my_ip0,my_ether0,my_ip1,my_ether1,1500, 5000, 5000, 0.375, 4500,
15000, 15000, 30);
```

9.2 Configuration generation tool usage

```
usage: make-olsr-config.pl -i IF -a ADDR [OPTIONS]
  Generate a Click OLSR configuration to run on interface IF with IP address ADDR.

Options:
  Run in kernel or at userlevel?   Defaults to kernel.
   -k, --kernel                    Run in kernel.  Only works on Linux with Click kernel module.
   -u, --userlevel                 Run in userlevel
   -s, --simulator                 Run in ns-Click simulator

   -n  --number N                  use n interfaces defaults to 1
   -i, --interface IF1..IFn        Use interface IF.  Required option.
   -a, --address A1..An            Use IP address A.  Required option.


   --ether ETH1..ETHn              Use Ethernet address ETH for interface.  If not specified,
                                   the address will be dynamically discovered.

  Optional OLSR parameters.
   --HELLO-Interval T(msec)        Hello Interval     [default: 1500]
   --HELLO-Jitter T(msec)          Jitter Interval for Hello messages [default 1/10 Hello Interval]
   --TC-Interval T(msec)           TC Interval        [default: 5000]
   --TC-Jitter T(msec)             Jitter Interval for TC messages [default 1/10 TC Interval]
   --MID-Interval T(msec)          Mid Interval       [default: 5000]
   --MID-Jitter T(msec)            Jitter Interval for MID messages [default 1/10 MID Interval]
   --neighb-hold-time T (msec)     Neighbor Holding time [default: 3*Hello Interval]
   --top-hold-time T (msec)        Topology Holding time [default: 3*TC Interval]
   --mid-hold-time T (msec)        MID Holding time [default: 3*MID Interval]
   --dup-hold-time T (sec)         duplicate Holding time [default: 30s]
   --Jitter T (sec)                max Jitter time for forwarded OLSR control messages
                                   [default: hello_Interval/4000] note: in seconds!
   --spread-init (boolean)         Initial emission of hello/Mid messages starts after random time
                                   between 0 and respective emission Interval [default: false]
   --additional-hello-msgs         Send more than just necessary messages to react faster to link failures
   --additional-tc-msgs            Send more than just necessary messages to react faster
                                   to topology changes
   --Jitter-all                    T (sec)[old] all packets are jittered before output on interface x default:0
   --hna                           build configuration with hna message processing
   --hna-gen                       build configuration with hna message generation (needs alternative
routetable code)
   -h, --help                      Print this message and exit.
```