UNIVERSITY OF OSLO
Department of Informatics

Transparent gateways between OLSR networks

Master thesis

Einar Bjerve

8th May 2006

Abstract

Future ad hoc networks will probably see an increasing demand to be able to communicate with external networks. There has already been done research on connecting ad hoc networks with other networks, like the Internet. While one can see that using available connections to external networks to reach other ad hoc networks would be beneficial under certain circumstances, research on this topic has been minimal. In this thesis we look into ways of connecting ad hoc networks using gateways, and analyse possible problems that might occur in such scenarios.

Table of Contents

1. Introduction

Recently we have seen an increasing popularity of wireless networks. This creates a demand for even greater freedom and functionality of such networks. One can imagine networks being formed on demand, whenever devices capable of communication are within range of each other, without using existing infrastructure. This type of on demand networking is called ad hoc networking and currently, extensive research into this issue is carried out.

One can easily see the benefit of such networks, like the possibility of setting up business meetings anywhere and form networks between participants' computers. Another example are military operations, where infrastructure necessary for wired networks may be absent.

One project focusing on the application of ad hoc networks in real world scenarios is the Ad Hoc InfoWare[1] project. The project aims to create a platform in which emergency personnel can efficiently communicate and coordinate their efforts in situation where the infrastructure of common networks like the Internet is unavailable.

In some cases where ad hoc networks are being used, there might be several partitions of the network. At the same time it would be beneficial for these pertitions to be able to communicate with each other. A possible scenario, is a tunnel accident where emergency personnel at either end of the tunnel would need to communicate. Another example are firefighters working on a large wildfire spread out over a large area, where the devices they normally use to communicate are out of range of each other.

To further improve communication, one could use existing infrastructure and let some nodes in the networks act as gateways between different partitions to link partitions subnets together. In this thesis I will examine the possibility for communication between separate ad hoc networks.

1.1 Readers guide

This thesis is divided into 6 parts. The first part gives an overview over the topics covered in this thesis. The second part gives some background to the technologies used. In the third covers design alternatives and choices are being outlined, while the fourth

describes the implementation. The fifth part briefly covers testing and the last presents a conclusion, open problems and future work.

1.2 Ad hoc networks

Mobile ad hoc networks, or MANETs[3], are a relatively new term in wireless networking. The term ad hoc comes from latin and means “for this purpose”. The term “ad hoc” often refers to improvised events or solutions, which fits this network type very well.

An ad hoc network is formed when needed, when nodes capable of forming such networks are in range of each other. Nodes in these types of networks are allowed to roam freely within the network. Some can leave while others may join at any moment. Because of the movement of the nodes within the network, links between them are formed and broken consistently. Due to this, one could say MANETs completely lack infrastructure.

The lack of infrastructure and the mobility of nodes makes conventional routing very difficult in such networks. Routing is therefore done on a hop by hop basis, meaning that nodes forward data towards the destination if it is not meant for that node itself. This again means that every node has to act as a router.

While there are many benefits of ad hoc networks, there are challenges too, especially when it comes to issues like routing, security, service discovery, addressing and Quality of Service.

1.3 Challenges in ad hoc networks

Being a relatively new technology, there are still several problems that need to be solved on the topic of ad hoc networks.

1.3.1 Duplicate address detection

Due to the lack of infrastructure in ad hoc networks, and their dynamic nature (that nodes can randomly enter and leave), it can be hard to ensure that addresses of nodes in the

MANET are kept unique. In wired networks, the DHCP service usually helps ensure uniqueness of an address, together with the network hierarchy, but in a MANET one cannot always be certain that DHCP is available, and they are not hierarchal by nature. Several solutions to this have been proposed, e.g [16] proposes to use route discovery mechanisms to detect duplicate addresses, while [15] proposes to temporary allow duplicate addresses as long as no packets are routed to wrong destination.

1.3.2 Scarce resources

The small devices that a MANET consists of will almost certainly have limited internal resources like computational power and memory, and most importantly, battery power. This should also be considered while developing applications for ad hoc networks.

1.3.3 Service discovery

Service discovery is also a challenging topic within ad hoc networking. Because no node can be guaranteed to be present in the network, and therefore existing solutions from e.g. the Internet are not well suited for ad hoc networks..

1.3.4 Quality of Service

Quality of Service (QoS) is still a difficult question for traditional wired networks, and when it comes to ad hoc networks achieving QoS is even more defficult. Ensuring that certain resources like bandwith, delay and others are available in a rapidly changing MANET is challenging. This is discussed in further detail in chapter 6.3.2.

1.4 Security issues in ad hoc networks

The basic requirements for security in ad hoc networks are mostly the same as for any other network (confidentiality, integrity, availability, authenticity, accountability, non-repudiation), but an ad hoc network requires more well-designed security mechanisms due to the total lack of infrastructure. In a wired network an attacker needs physical access to the network, but in an ad hoc network one only needs to be within range.

Authentication, to verify that a user is who they say they are, is very important. It has do

be done both internally on the ad hoc network and when connecting to nodes on a wired network that is connected to the ad hoc network.

For pure ad hoc networks the lack of infrastructure implies that there is no central server that can be used for authentication, and this is hard to implement distributed in the network.

Authentication on outside networks can be done in two ways. Either the gateways in the ad hoc network take responsibility for all ad hoc nodes, or the traffc of ad hoc nodes is verified at a central authentication server.

Any service on the network should be accessible for an authorized user at any time. This introduces another range of possible attacks such as jamming and denial of service attacks. A malicious node could enter the network and flood it with requests, or false packages.

The network should therefore ensure the integrity of transmitted data. This includes both accidental changes like interference and changes done by malicious intermediate nodes.

To preserve confidentiality means to prevent eavesdropping, that is making sure that no unauthorised persons can access confidential data. Eavesdropping can be prevented by using modern encryption techniques. On the other hand, since mobile nodes have limited battery power, it may be undesirable to send much power on complex encryption algorithms. There is also a problem of key distribution in such infrastructureless networks.

1.5 Routing in ad hoc network

Routing in ad hoc networks follows one of these basic philosophies: one can either route by means of proactive protocols or with reactive protocols.

Proactive routing protocols keep an updated table on every node on the network. The good thing about this approach is that routes are instantly available when needed. If a route is broken nodes can also respond to changes relatively quickly. These protocols do however create quite a bit of overhead since they need to constantly keep an up to date topology. Due to the overhead created these protocols are best suited for mainly

stationary networks which do not have constantly changing topology. Examples include Destination-Sequenced Distance-Vector (DSDV), Wireless Routing Protocol (WRS), Optimized Link-State Routing (OLSR)[2].

Reactive protocols create routes on demand, that means a node only knows about its immediate neighbourhood. When it has to send something it will first attempt to find a route to the destination. After it has finished transmitting it the route will be discarded. These routing protocols are better suited for networks with rapidly changing topology. Examples include Ad hoc On-demand Distance Vector (AODV)[4] and Dynamic Source Routing Protocol (DSRP).

A more thorough description of different ad hoc routing protocols can be found in [8]

1.6 Gateways

A gateway is a node on the border of a network, which can forward traffic to other networks. The term gateway is loosely defined, and a gateway can work in hardware, software or in both. It can also work in different layers in the OSI model.

Since a gateway is on the border of a network, it usually contains other features, like firewalls, or it may be acting like a router.

In an ad hoc scenario, the gateway will most likely be a node that participates in the MANET, but at the same time has a connection with the other networks. This node can be either mobile (e.g. a GSM device) or stationary (e.g. a laptop participating in the manet with a DSL connection to another network).

2. Background

This chapter covers the technologies used in the thesis. It also gives a brief description of the Ad Hoc InfoWare project, which this thesis is built on.

2.1 Ad Hoc InfoWare

The Ad Hoc InfoWare[1][17] project was started to develop middleware services for ad hoc networks used by emergency personnel during rescue situations. One of the essential requirements needed for a successful rescue mission is the abilitity to communicate and share information, and ad hoc networks might be suited to do this.

With respect to this project, one can see the use of gateways between separate ad hoc networks in some of these scenarios, like in the wildfire or tunnel scenarios mentioned earlier.

2.2 OLSR

OLSR[2] is a proactive ad hoc routing protocol. It maintains a picture of the known topology in every node. It also has the benefit that routes are instantly available when they are needed. It suffers from constant control traffic that needs to be broadcast to keep the routing tables up to date, but OLSR has optimized these broadcasts in a rather elegant way. OLSR has also support for multiple interfaces on a single node as well as connections to external networks. It also have a general messaging system, which allows us to use customised packages that are forwarded according to OLSRs standard scheme.

The most interesting part of OLSR is the way it floods messages very efficiently. This is done with the use of MultiPoint Relays, or MPRs. MPRs are the only nodes that are allowed to forward broadcast messages. Each node chooses a set of MPRs from their 1-hop neighbourhood. These MPRs are chosen in such a way that they cover the entire 2-hop neighbourhood of the node, or said in another way, the the node should be able to reach every 2-hop neighbour through only the MRPs. An example of MPRs is shown in the figure 2.1. The numbers indicate the number of hops from the node in question
The algorithm for how MPRs are selected could be changed for different reasons, for example if you want to incorporate QoS into OLSR, e.g QOLSR[10] use this technique to incorporate QoS into OLSR. I will have a brief look at QoS later in the thesis.

Because some nodes may have several devices running OLSR, each node is identified by only one of the addresses used by any of these devices. This address is either called the originator address or the main address. The address of the device that sent the packet is contained in the IP header of the packet, while the originator address is used in the OLSR message header (see header description later).

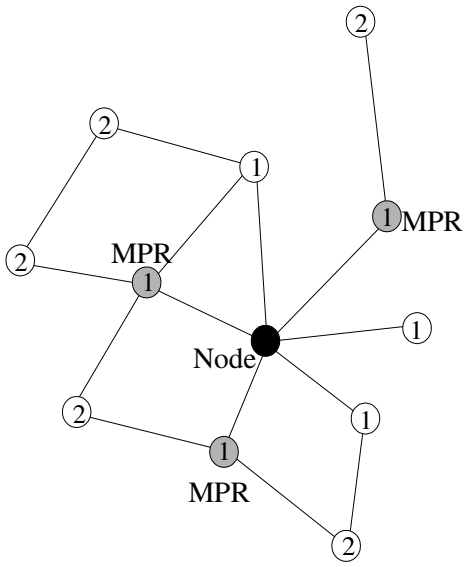


Figure 2.1: Example of MPR selection by a node in OLSR

2.2.1 OLSR Packets

Control packets in OLSR follow a standard scheme, that also allows us to create custom packet types.

A standard OLSR message is laid out as shown in the figure 2.2. A brief description of the various fields follows. The first two fields are considered part of the packet header, the rest is part of the message header, giving the possibility to include several messages in a single OLSR packet.

- Packet length is the length of the packet in bytes, including the header.
- The sequence number is a number to used to indicate whether a packet is out of date or not. It is incremented by one every time a new packet is sent.

- Message Type is the field of most immediate interest for custom packets, as it allows such packets to be broadcast through the network according to the standard OLSR broadcasting scheme. It will also just be forwarded correctly if the reciving node does not know how to handle this type of package.
- Vtime is the validity time of the information in the packet. It allows us to tell OLSR or our custom plugin how long it should take before we should consider the information out of date.
- Message size is the size of the message, including the message header.
- Originator address is the main address of the node that originally generated this message.
- The time to live tells OLSR through how many hops this message will be retransmitted before being discarded
- Hop count tells how many time the message has been retransmitted.
- Message sequence number is a unique identification number assigned to each message.

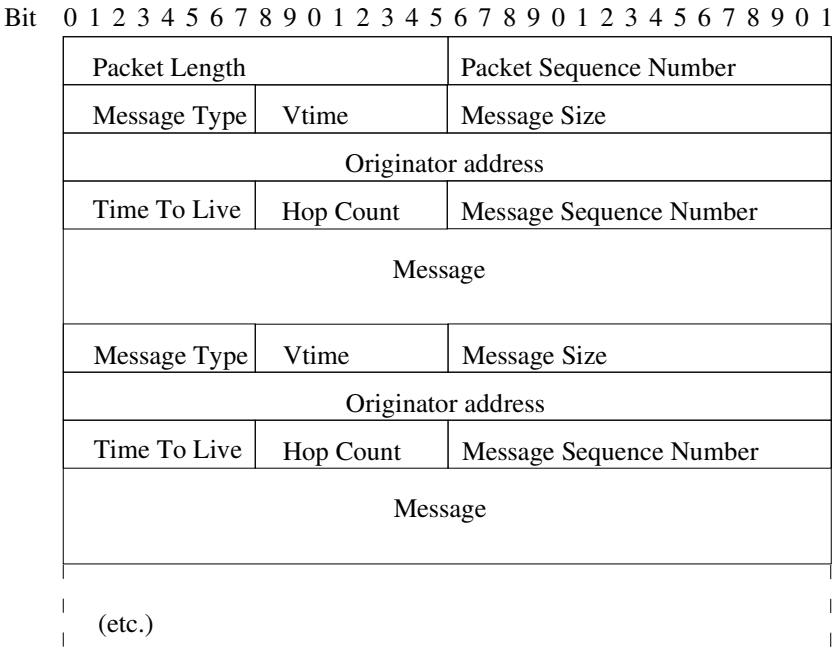


Figure 2.2: OLSR packet

2.2.2 OLSR messages

OLSR uses a standard packet scheme described earlier. The message and values in the header are the only things that differentiate the packet types.

HELLO messages

HELLO messages are sent one hop only to inform neighbouring nodes about what nodes that exists in the one hop neighbourhood of the originator node, together with informing neighbouring nodes of it's own presence. The message contains a list of nodes in its 1-hope neighbourhood, what type of links these are (symmetric, asymmetric, lost or unspecified), which nodes that are selected as MPRs and some information about the node itself (is it willing to forward packets for other nodes).

Unless HELLO messages are received regularly, the information stored about the sender will eventually time out and be removed.

This allows a node to learn its whole 2-hop neighbourhood from HELLO messages only, and from that it can select appropriate MPRs that cover the entiere two-hop neighbourhood. In addition, this will tell neighbouring nodes whether a link is symmetric or asymmetric, since if a node can hear a HELLO, and the node hearing it does not appear on the list of neighbours, one can assume that it is an asymmetric link.

I will later explain how it can be possible to use HELLO messages to relay information about neighbouring networks.

TC messages

Topology Control, or TC messages are used to broadcast information about what nodes the originator node has links to. It contains a list of neighbouring nodes, and must at least include all neighbouring nodes that have selected the transmitting node as MPR. As long as any node has selected another node as MPR, transmission of TC messages is Mandatory for this (another) node.

These messages can possibly be manipulated to send routing information to neighbouring networks, which I will look further into later

HNA messages
I will only touch Host and Network Association (HNA) messages briefly, since they are of little importance to the thesis. As I will discuss later, they could have been used as an alternative way of creating the gateways.

A node participating in the MANET may have several interfaces, possibly one which is part of another network. HNA messages can be used to relay information about the existence of this network to other nodes. The message simply contains the network address of the network in question and the netmask.

MID messages
Multiple Interface Declaration (MID) messages also have very little importance to this thesis, so I will also only touch these briefly.

MID messages are used to inform other nodes about the presence of multiple interfaces, and contains a list of all addresses of all OLSR interfaces on that node. The main address is easily identified through the originator address in the header.

Nodes in the ad hoc network may have multiple interfaces participating in the network. In that way, a node also may have multiple addresses on the same network. If a node have more than one OLSR interface, it will select a main address, which is also called the originator address in the [2]. The main address is used in the originator address field in uniquely identifying the node, no matter what interface that originally sent the message.

2.2.3 OLSRD

An implementation of OLSR is OLSRD[6]. I will use this in my implementation, mainly because it is easy to use and extend, and it works with NEMAN[5], the simulation tool that I will be using.

In addition to the core OLSR function, this implementation also features a plugin interface, which should make it easier to implement the required changes to the routing protocol.

OLSRD is explained in more detail throughout the thesis.

2.3 Real tests, simulation and emulation

When testing an implementation of a networking scenario, there are three different approaches to do it.

First one can test it in a real scenario, using people that walk around with real devices with the software to be tested installed. Another way is to use specialised simulation software to simulate the network and the software. And finally one can also emulate a wireless network, which means simulating the network with real processes.

To test an ad hoc network in real life is an expensive solution, both in terms of equipment and manpower needed. The benefit is that this is the closest you can get to the scenarios where the released product will be used in. On the other hand it requires hard to do repeated tests where exactly the same parameters apply and it is hard to reproduce special errors that might occur due to random events. It require smassive coordination if it is a large network to be tested. So all in all, this is not a very good way of testing, at least not with a product still under development.

The second approach uses simulation software to simulate the network. When simulating a network in this manner it will of course be cheaper in terms of manpower. This type of software allows for accurate testing of large networks, but you have to develop everything twice, once for the simulator and again when programming the release version. Learning the simulator can also be timeconsuming. A minor downside might also be that some existing simulators like ns-2[12] require you to completely recompile the entire simulator, even if you only make minor changes.

The third approach is to emulate a wireless network where real processes are used, that is only the lower network layers are simulated. It will make it easier to repeat exactly the same simulation over and over, which is especially useful when searching for specific bugs that might only appear when certain conditions are met. The true benefits from this are that your software runs as real processes, and thus there is little to no need for changes when it is ported to real devices. A downside is that it can be very hardware demanding to test large networks. A program that can be used to simulate networks in this way is NEMAN, which has been given as part of this assignment.

2.4 NEMAN

NEMAN[5] is an emulation tool used to simulate ad hoc networks on a single computer. It can simulate a large amount of nodes, new links that become established and links that break between nodes. This can also be shown in a easy-to-understand GUI, that displays movement of nodes, the range of their wireless devices, and which nodes are connected to each other.

NEMAN consists of two parts, the topology manager and the GUI front-end.

First, let's have a look at the topology manager. The topology manager, or topoman, is the part of NEMAN that emulates different devices on a single computer. It maintains an image of the current topology of the network. For each node on the network it creates a virtual network device. These devices are available from the Linux kernel and provide virtual network interfaces (TAP interfaces). User processes attach to different TAP interfaces like they were normal network devices.

Topoman controls which TAP interfaces the packets are forwarded to, depending on the current topology it has stored

Topoman also features a special device that can be called the monitoring device, all traffic on the network is also passed to this device, in this case it is TAP0. This device is not part of the topology, and is “connected” to every other TAP interface, making it possible to monitor and analyse traffic, but also induce traffic into the network.

The version I use of NEMAN has scripts to start and stop the routing deamons, in my case olsrd, running on different TAP devices. Thus routing deamons are started and stopped directly from the GUI.

The GUI allows the user to see how the nodes respond to each other. It also notifies topoman of changes in the topology of the network. It uses standard ns-2 scenario files (further description on how to create them can be found at [12] that describe where nodes are positioned, where and when they move and when the gain/lose a connection with another node. The simulation can be paused at any time, allowing the user to see how their software reacts to different type of changes in the topology.

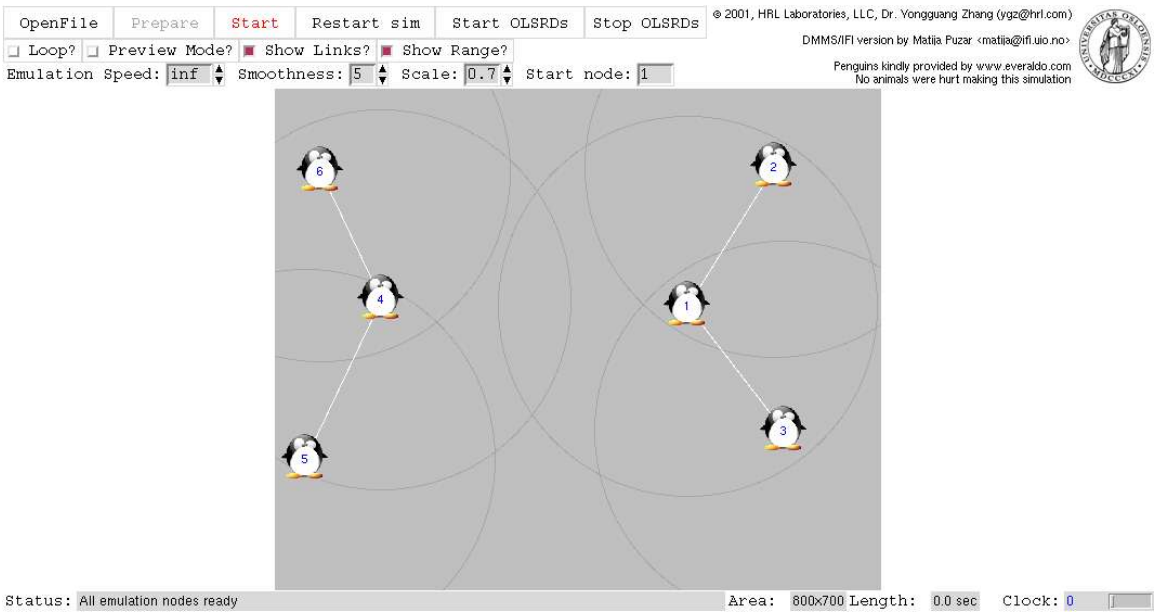


Figure 2.3: Screenshot of NEMAN

As you can see on the screenshot, the GUI represents different nodes as circles, numbered after their IP address following a default scheme. Around each node there is a circle which represents the range of its transmission. A line is drawn between nodes that have a wireless connection.

In respect of this thesis, one could see node 1 and 4 on the screenshot as possible gateways that could connect the two separated ad hoc networks, since there is no wireless connection between them. Data transmitted between them should then use another medium than the TAP interfaces provided by NEMAN

2.5 Related Work

To my knowledge there is noone have previously tried to connect two different ad hoc networks. But there has been done research to connect ad hoc networks to outside networks, e.g. the Internet. Some protocols, like OLSR already have this functionality included, while there other studies examining entire networks being mobile[13], and thus requiring MobileIP to run on the router.

However, when connecting two ad hoc partitions, there are shortcomings to these methods both with similar address ranges on several network partitions and in duplicate address detection.

3. Design

The purpose of this chapter is to analyse different possible implementations, and later on decide on one of them and describe how the chosen solution might be implemented.

3.1 Requirements for gateways

A gateway is a node that forwards traffic between different network partitions. To connect two ad hoc networks we can derive the following rather informal requirements:

The gateway must provide the means to forward traffic between networks
The point of having a gateway is to be able to send data from one network to another, so this one is clearly necessary.

Gateways should reduce overhead/control traffic between themselves as much as possible
As already mentioned, resources are often scarce in ad hoc networks, while it may or may not be on intermediate networks (e.g while an DSL link to the Internet will probably provide sufficient bandwidth for most uses, a GSM link might not. This will not be known beforehand, and therefore gateways should take steps to reduce traffic between them.)

The gateways should either implement or have the possibility to easily extend their functionality to securely transmit data between them over unsecure networks
While the data sent on the ad hoc network should be secure, it does not necessarily mean that it is secure on intermediate networks. This should be ensured when security in the ad hoc network is important.

They should also have the possibility to implement Quality of Service parameters when it becomes possible to use such features in ad hoc networks.
Future ad hoc networks will probably have QoS requirements, and the gateway implementation should be open enough to incorporate QoS parameters in future versions.

Implementation of the gateway should mean as little tampering with the existing code as possible.
When maintaining existing code unchanged it is easier to upgrade the gateways to new

releases of the routing protocol.

Specific requirements for this thesis.
Some requrements are set specifically in this thesis. These have to be followed.

- The routing protocol to be used is OLSR, specifically the olsrd implementation.
- The gateways should be able to run within in NEMAN

Functional requirements

1. Must be able to transmit data between gateways
2. Gateways must be able to exchange information about which nodes that exists on their network.
3. Routing information that is transmitted between gateways should only contain new/left nodes from their respective networks, to minimize traffic load on the intermediate link.
4. Data transmitted between gateways, that has a destination other than the gateway itself, should be forwarded to the correct node according to OLSR policy.
5. Packets transmitted between partitions should not be affected by the type of network between the gateways (e.g. Ethernet or GSM)
6. Routing information should have priority over other information, requiring a scheduler to prioritise packets
7. Nodes accepted on one network (Authenticated + negotiated address) should also be accepted on the neighbouring network.

3.2 Design alternatives

3.2.1 Transparent gateways

Gatways act as normal nodes, i.e. it is not visible to common nodes that they are gateways. Common nodes know of nodes, and possibly a topology (though this might be incorrect, to reduce overhead between gateways) of the neighbouring network.

- The transparent option solves a few problems easily
    - Duplicate address detection can be done with existing technologies. Easier for nodes to travel between networks too, without negotiating a new address. In

this thesis I will assume a duplicate address detection mechanism is present, but making such a mechanism is out of scope of the thesis.
    - Authentication can be done as if there was a single network. Another possibility is to have an authentication gateway, that signs packets again before forwarding it to the neighbouring network (see signing of packets later).
- Same address range can be used on the whole network.
- Every service should be available as if the network was working as a single entity, therefore existing service discovery protocols should be usable without alteration (though this might be possible with traditional gateways to, this will depend on how these service discovery protocols work).
- No need to alter the routing protocol for common nodes
- False routing information should be used to reduce traffic on the link between networks (e.g. hop count will not be correct). In general this is not considered a very good thing to do, but it might make things significantly easier

3.2.2 Traditional gateways

Every node knows there is a gateway node they have to contact to reach the neighbouring network. This gateway only broadcasts the address range of the neighbouring network.

- Networks can be segmented, which is the huge advantage of designing networks in this way.
- Trouble might be with addressing if nodes do not get preset IP addresses. While existing proposals to duplicate address detection exists (TODO: Ref), they do not consider outside networks, which would be needed for gateways between ad hoc networks, since we do not know whether it they will operate on the same address range or not.
- One might need reauthentication when moving between networks.
- Existing nodes need to know about the gateways, possibly a few changes in the routing protocols in common nodes, unless the HNA messages in OLSR are used
- Should be easy to reduce control traffic between gateways, simply because the neighbouring network don't need to know the topology of it's neighbour. This will happen automatically since every node only know of the address range of the neighbouring network.

OLSR HNA
This works pretty much in the same way as traditional gateways. Nodes know there are HNA gateways on the network, and what address ranges they can reach. Works pretty much in the same way as traditional, and gateways are announced through OLSR messages.

No topology information about the other network is provided, only that it exists, what address range it consists of and where the gateway is, and therefore control traffic between the gateways is reduced.

In the Ad Hoc InfoWare project devices will most likely have preset IP addresses. The problem is then that you don't know which nodes that will be on which network, and therefore HNA is not a very good solution. If there are 3 separate networks connected through gateways, all with random nodes from the preset IP range, one will not know which gateway to send packets to address, since both neighbouring networks might broadcast that they can reach the same address range.

Also if preset IP addresses are not used, one will have some possible problems with duplicate addresses.

3.2.3 Summary

While the OLSR HNA and normal traditional gateways are very similar, there are some problems with them. Namely there could be some problems with addressing if they do not use preset IP addresses. And there could also be some problems if preset addresses are used, but it is not known beforehand which subnet each node will participate in.

Developing transparent gateways will not have the same addressing problems, but they will have significant control traffic going between them to keep updated routing tables. Due to the simplification of the addressing problems, I will use transparent gateways in this thesis.

3.3 Multiple gateway problems

Multiple gateways occur when you have more than one gateway on a single network, and these gateways are connected to the same neighbouring network. There are several possible challenges with this, but some might be non-existent, depending on how we solve the problem. Therefore it is important to consider this when designing the gateways.

Increased hop count due to errors in the routing information

If gateways reduce overhead between them by only telling the neighbouring networks about the existence of nodes, and not hop count, routes can possibly become significantly longer and putting further unnecessary strain on links between gateways.

The following example (see figure 3.1) explains this issue. Let's say Node A4 wants to send something to Node A3 in this scenario. With a transparent solution where the gateways pretend that every node on the neighbouring network is 1 hop away, GW A1 will see every node gateway GW B can reach as 1 hop away, and GW B can reach every node that GW A2 can reach. So GW B will tell GW A1 that it has a route to node A3 in 2 hops (through GW A2). Thus, Node A4 will prefer the route through GW A1 → GW B → GW A2 over the internal route, since that is a 4 hop route, while the internal are 6 hops. Depending on how it is implemented, it might be even worse if GW B does not tell about GW A2, just that it has a 1 hop route to Node A3

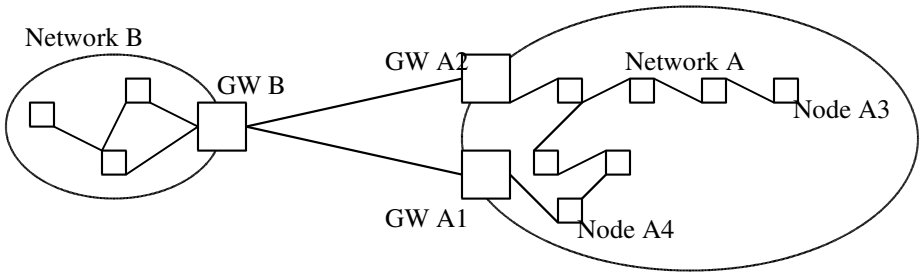


Figure 3.1: Multiple gateways

One possible solution to this is to prevent gateways from telling neighbouring gateways about nodes it knows about on other networks. This however is not a very good solution, since that can exclude certain networks from getting to know of the existence of others,.

The issue can also be avoided by making sure that the gateways do not keep nodes it already known of from its local network as possible destinations through the gateway.

A problem of roughly the same type arises if a node on Network B wants to transmit something to Node A3. The shortest path here is obviously through GW A2, but GW B will not know which gateway is closest.

Retransmitting to both gateways is a solution. But it will lead to double packets on Network A, which again generates unnecessary traffic.

Another solution is to make the gateways transmit either hop count or an estimate of the hop count (to prevent to much control traffic over the gateway-gateway links) to neighbouring gateways. The gateway can then make the right decisions.

Multiple networks with multiple gateways

Some problems arise if you have several network partitions, with several gateways connecting them. You can first have packets that never reach their destination, and it is also possible for routes to become longer than necessary.

Example 1:

Node A (see figure 3.2) wants to send something to Node B. While The number of hops to reach the destination is equally long no matter which route is taken, it does not mean one is better than the other in every case. If a lot of traffic is going on in network A, and all the gateways have DSL connections, the route through Network C will probably be the best. But on the other hand, if every gateway-gateway link is already close to the limits of what it can handle, the route through Network A, and directly to Network B will be the best.

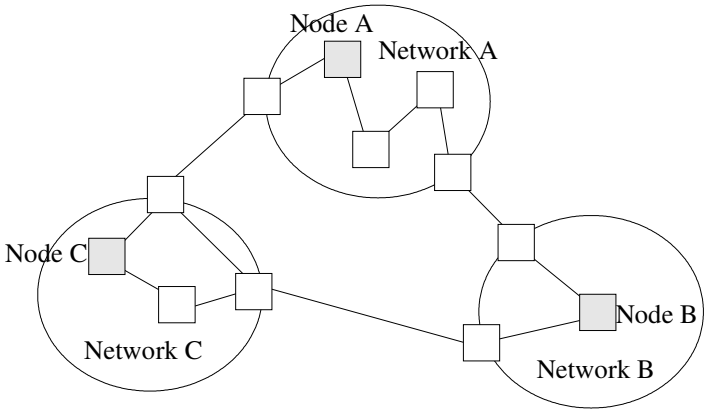


Figure 3.2: Multiple networks

Example 2:
The transparent approach where gateways only supply nodes they know of, not distance to them, is used. Node C in figure 3.2 wants to send something to Node B. The top gateway on Network C have told network C it has two hops to every node on Network A and Network B (since the gateways in A knows of the nodes in B, the a list of all nodes in both A and B will be transmitted to the gateway in C). But when the packet reaches Node A, it believes the closest gateway have 2 hops to Node B, and will therefore send the packet back towards Network C (since the leftmost gateway in Network C will have received a list of every node on Network C and Network B through the gateway in Network C – and therefore Node A's routing tables should forward traffic to network B towards the gateway to network C, since it believes that is the shortest route).

The solution to this problem will also be to add hop count to the notifications between the gateways. It is also necessary to inform local nodes about the range to nodes on other networks.

3.4 Signing of packets

In an emergency scenario, there can be sensitive information on the network which should be protected, and secondly one would avoid bogus nodes injecting false information into the network. By digitally signing packets one can ensure that the node reciving or sending the data is authenticated.

Between two connected ad hoc networks there are two possible ways of signing packets:

1. Network A uses Key A to sign its packet, while network B uses Key B. The gateways use a secure tunnel to transfer the messages.

   This makes the different keys less widely known, and the gateways will need to function as authentication gateways too.

2. Network A and Network B use the same key, or both networks know both keys

   The keys are now widely known, and if a key is compromised the whole network will

be compromised.

Due to time constraints, I have not delved deeply into creating secure gateways, and therefore this is not researched into further detail. If gateways were to be used between such networks, security issues between the gateways would have to be considered more closely.

3.5 Creating transparent gateways

While the communication between the gateway and its own network will remain mostly the same, there are two ways of doing the communication between the gateways.

1. Using HELLO and TC messages to to tell the neighbouring networks about the existence of nodes on its own network.
2. Develop a custom protocol that relays information about which nodes that exists on different networks.

3.5.1 Requirements to transparent gateways

On its own network the gateway node should work exactly as any other node when considering incoming messages. Hello, TC, MID and HNA messages should be processed as usual. Later in this chapter I will show that between gateways some of these control packets should be limited, or even halted completely.

On outgoing messages, it have to be considered whether this is to the same network or a gateway. Outgoing Hello-messages should go as usual to its own network, but some changes will need to be done when forwarding them to the neighbouring network (depending on how GW-GW communication will be implemented as discussed below)

There should be no changes to HNA and probably not MID either

Normal data traffic should be handled by the routing tables built by OLSR, and should thus be tunneled to the neighbouring gateway.

3.5.2 Implementation using HELLO and TC messages

As already discussed, when using transparent gateways it is not necessary to send all routing information. Everything has to pass through the gateway anyway, so strictly speaking you only need to relay information about what nodes that exist on the neighbouring network, or one could pretend every node in a network is part of the 1-hop neighbourhood of the gateway. In that case, one can see it might be possible to create fake HELLO messages that are only sent over the gateway-gateway link to notify its neighbouring gateway about nodes on its network.

If such a solution is chosen, it is very important that such fake Hello messages are sent to the neighbouring network only, and it is not sent to its own network, which would lead to errors in the topology information of the network. If a gateway claims to hear a Node A through a symmetric link, every node in the gateway's 1-hop neighbourhood will think it have Node A in its two hop neighbourhood, even though it can be distant. It can then lead to bogus routes, since OLSR routing table calculation first finds 1-hop neighbours, then 2-hop neighbours and finally n-hop neighbours. If a route to the destination is already found (through 1 hop neighbourhood or 2-hop neighbourhood) when calculation n hop routes, it will be discarded, and thus the real route will never be stored.

Therefore one will need to transmit two sets of hello messages, one to its own network, and one to the neighbouring network.

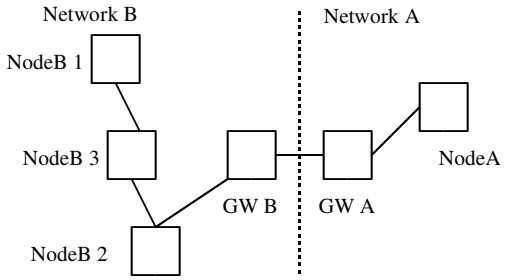
As we see from activity diagram in figure 3.4, HELLO messages to the gateway's own network should be handled as specified by OSLR. This list should then be set as the gateways symmetric 1-hop neighbours in a HELLO message. The resulting HELLO message should then be transmitted to the neighbouring network.

When generating the list of known nodes, it is also important that any list of nodes recived from a neighbouring network is excluded. If not, you can see similar behaviour as mentioned if HELLOs were sent to the gateway's own network, only with the 3 hop neighbourhood instead, as OLSR progressively creates routing table tuples by increasing hops away.

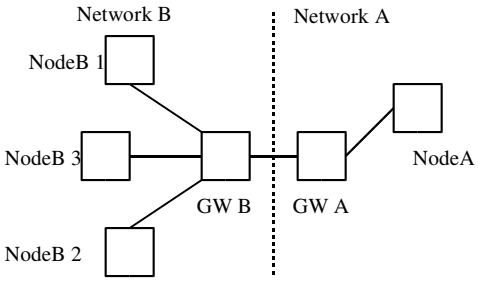
By using this solution, it can be hard to avoid the circular network problems mentioned earlier since there is no way to attach information about the number of hops to another node from the gateway, and will therefore only be usable with 2 partitions, or as long as

every gateway connecting the different partitions is connected to each other. On the other hand, this will probably be rather easy to implement, since most of the functionality needed is already in OLSR.

One also have to consider that since a topology of an neighbouring network should look like the figure 3.3b on the local network (the local network being Network A in the illustration), while the actual topology looks like figure 3.3a. It also means that all other nodes on network B would have selected GW B as MPR if this was a real topology, since then GW B would have been the only means to reach GW A in two hops. That means GW B (or GW A on behalf of GW B) must send fake TC messages too.



3.3a: Real topology



3.3b: Topology as seen on network A

These TC messages should only contain the 1-hop neighbours of the originator of the message, which means all nodes on the remote network.

To reduce overhead between gateways, TC messages from other nodes than the gateway should be stopped at the gateway, and not be passed on to the neighbouring gateway. When it is time for a gateway to generate a TC message, it should generate two sets of TC messages, one to its own network and one to the neighbouring, just like HELLO messages. The one to its own network should be generated in the normal way, but should include nodes on the neighbouring network, and then be sent to its own network. The other one should take all nodes it knows of on its own network and set them as 1-hop neighbours, then transmit this TC message to the other gateway.

The reason fake TC messages can be used is that only nodes that only nodes that are selected as MPRs are required to send TC messages. [2] states that *"A TC message is sent by a node in the network to declare a set of links, called advertised link set which*

MUST include at least the links to all nodes of its MPR Selector set, i.e., the neighbors which have selected the sender node as a MPR". This can be exploited to reduce routing traffic going over the link between gateways, while maintaining transparency. This means only nodes that is selected as MPRs has to send TC messages, which means only the gateway have to do it to preserve the topology.

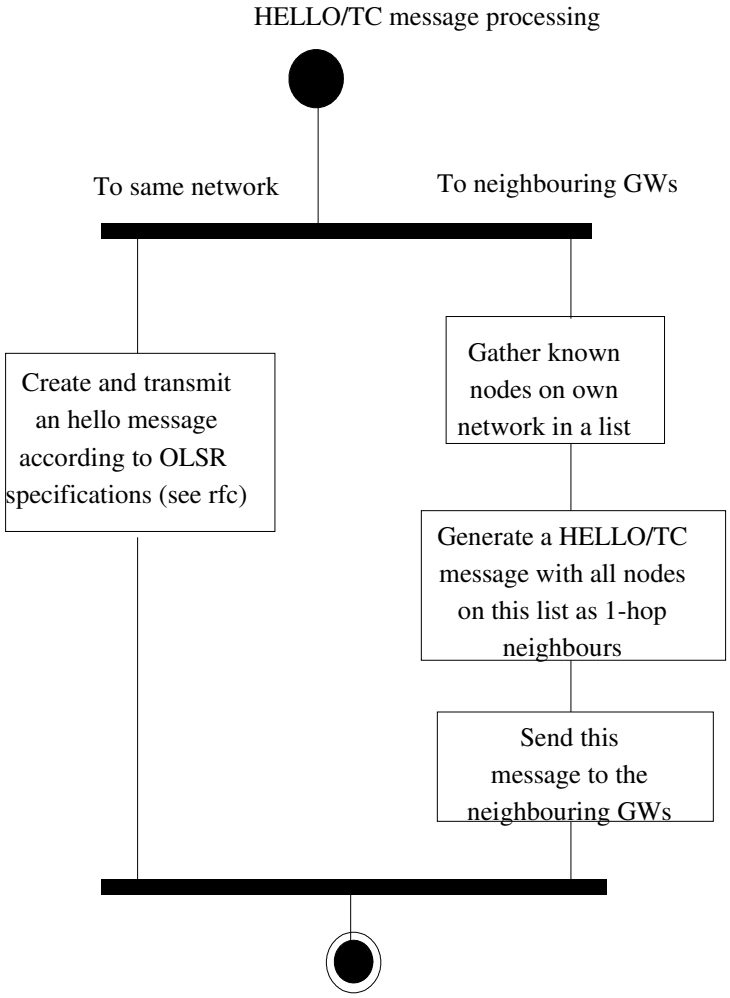


Figure 3.4: Activity diagram showing how HELLO/TC messages should be processed in traditional gateways

Due to how MPRs are selected in OLSR, gateways will always be MPRs to each other, because through MPRs you are supposed to cover the entire two-hop neighbourhood of a node. Therefore, gateways will always be responsible of forwarding the TC-messages of the neighbouring gateway, though there might be a special case where this does not apply, if one gateway are connected to two gateways on the same network. So when a gateway does recive a TC message from another gateway, it will have to forward it

normally to its own network, but if several gateways are connected, only one of them should act as MPR.

3.5.3 Implementation using a custom protocol

In this solution the gateways have a specific protocol for exchanging routing information. There are several ways of solving this. The simplest way is to do it as with TC messages, except we don't wrap the information in TC messages. Other ways can be to only exchange information when the gateways learns that nodes have entered/left the network, and reducing the load on the link significantly in that way.

This solution is also more scalable, since it is easy to add additional information that can be found useful into the messages between gateways (e.g. QoS parameters, or additional routing information like hop count which can be useful if there are several gateway connections between networks,).

The gateway has to maintain a list of what nodes that exists on its current network, and a list of what nodes that exists on the neighbouring network. When changes are made to the list of its own network, they have to be updated on the neighbouring network too. But it is only necessary to send changes, not the whole list of known nodes as you need to do with the HELLO/TC approach.

When a node recives updates from the neighbouring network it have to generate a TC message to notify the nodes on its network about the change. This can also be done in like in the HELLO/TC approach, and create a fake TC which it pretends originated from the neighbouring gateway, thus making every other node on the other network appear to be in it's two-hop neighbourhood, which will work in the same way as the HELLO/TC approach.

Another possible improvement here is that if the number of hops needed to reach the node in question is included in the custom protocol, you can also include that when creating the fake TC message, and in that way keeping routing information relatively intact. You also reduce the control traffic between gateways slightly, since changes about nodes that move without changing distance to the gateway don't have to be told to the neighbouring gateway. You can also choose to only update information when significant changes have happened, while still keeping a relatively correct hop count table (e.g. by

only notifying the neighbouring gateway when the range of a node have changed by 2 hops), and reduce control traffic further. Or one could reduce the reactiveness of hop count lists (e.g. tell the neighbouring gateway about nodes that have different hop counts every 30 seconds, but instantly tell about nodes that enter or leave the network)

Each gateway will also need to keep its own OLSR implementation updated on the existence of the neighbouring gateway, and possibly the neighbouring node. It will have to refresh the OLSR timeout timer for the neighbouring gateway for as long as it has a connection with it, since they will not receive any real HELLO messages from the neighbouring gateway to refresh it.

With the possibilities to extend the custom protocol solution with further functionality, this is the solution I have chosen in my implementation.

4. Implementation

4.1 Plugin Functionality in olsrd

Olsrd comes with an easy interface to create plugins, and this is what has been used in the implementation. It allows a plugin to modify and access data and packages recived by olsrd. To put it shortly a plugin can manipulate every aspect of olsrd, from creating custom packages to provide alternative ways of calculating routes.

Plugins are created as dynamically linked libraries to olsrd. This gives you the possibility of using functions implemented in olsrd. Further documentation can be found at [7].

4.1.1 Required functions for a plugin

We are required to implement some functions to get the plugin to work properly with olsrd. When dynamically linked libraries are loaded a constructor is called, while a destructor is called when it is unloaded. These two has to be implemented.

In addition, olsrd requires you to implement 3 functions.

`void olsr_plugin_init()`
Necessary initialisation for the plugin should be done in this function.

`void olsr_plugin_exit()`
This is called in the the destructior, and should free up allocated memory.

`void plugin_ipc_init()`
All IPC initiallisation should be done in this function.

4.1.2 Data types and data structures

Olsrd provides redefinition of normal data types, which I use to keep my code consistent with the rest of olsrd. This also help ensuring that my plugin will stay compatible with different implementations of olsrd.

A commonly used datastructure in olsrd is the hashed 2 dimensional linked list, or one could say an array of linked lists. The index you store data in is chosen by the a hash value, calculated from an IP address. This structure is created by allocating memory for a set of nodes equal to the amount of possible hash values, and having its previous and next points pointers to itself. Since these nodes are always present, they should always be empty. When an item is stored in the structure, it is inserted into the correct list according to its hash value.

The structure is shown in the figure 4.1, where the grey nodes represent the anchors

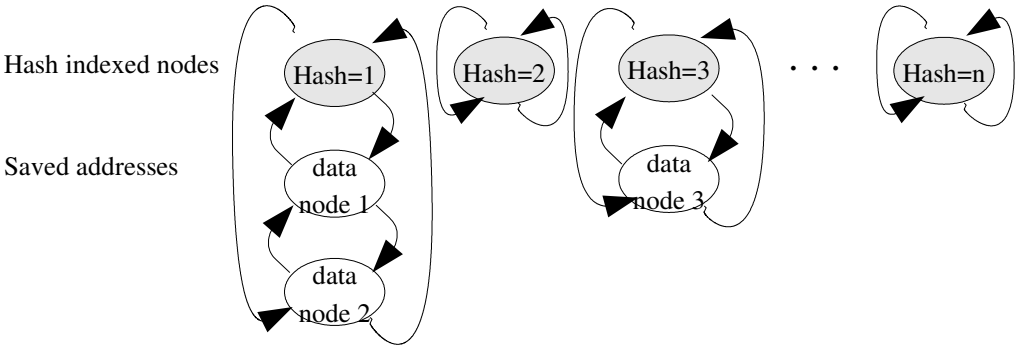


Figure 4.1: An illustration of olsrd's 2 dimensional linked list

For hashing of IP-addresses, olsrd provides us with the following function.

```
olsr_u32_t olsr_hashing(union olsr_ip_addr *)
```

In my implementation, I use this type of datastructure to store addresses of my own network and addresses received from the neighbouring network. Olsrd does not provide an general implementation of this list, and therefore queue, dequeue and search function have to be implemented upon need.

4.1.3 Functions

Scheduled events

Certain events have to be scheduled to happen at certain intervals. An example from my thesis is the transmission of data between gateways. To schedule events to run regularly, olsrd provide the following function to register events with olsrd scheduler.

```
int olsr_register_scheduler_event(void (*event_function)(void *),
                                  void *par,
                                  float interval,
                                  float initial,
                                  olsr_u8_t *trigger)
```

Timeout events

Olsrd also lets us register timeout functions with the scheduler. This functionality can be used to flush out out-of-date data

```
int olsr_register_timeout_function(void (*)(void));
```

Allocating memory

Olsrd provides a function for allocating memory. It is simply a wrapper function for the normal malloc that does check for errors, and quits OLSR if you run out of memory. I use this to stay consistent with the rest of olsrd.

```
void *olsr_malloc(size_t size, const char *id)
```

The string argument is the error message olsrd will display.

4.1.4 Loading plugins

A plugin can only be loaded at startup. To make it load you have to edit the olsrd.conf file, and add the appropriate lines

```
LoadPlugin "path/Plugin_name"
{
PlParam "Parameter_name" "Parameter settings"
}
```

4.2 My implementation

My implementation was mainly going to focus on proving that transparent gateways was a possible solution. But due to time constraints, I have not managed to finish it. In this chapter I will however document what has been done, and how I have planned on doing what isn't finished yet.

The subchapters are organised by first explaining what is done, then how this would have been improved if I had more time. The last subchapter covers what I have not started implementing yet.

4.2.1 Initialisation

The plugin uses the olsrd plugin interface to load. This enables it to work independently of other olsrd nodes, and thus these do not need to know about the gateway plugin. So changes are only needed on gateway nodes.

4.2.2 Data structures

The plugin keeps track of nodes in both its own and the neighbouring networks in two lists, both indexed by the hashed 2-dimensional linked list structure in olsrd, described in 4.1.2. The datastructure that stores these entries is defined as

```
struct gwentry{
  union olsr_ip_addr addr;
  struct gwentry *next;
  struct gwentry *prev;
};
```

where addr is the address of the node in question, while the prev and next pointers are there to let us use the structure in a linked list. It should also be easy to extend this structure if needed, e.g by adding hop count to the node.

The basic idea here is that when a gateway generates a control package to be sent to a neighbouring gateway, it will first look up the nodes in its own network list, generate a package containing the entries from this list and then send it. And when reciving a control package, it will have to store the entries in it in its neighbouring network list.

4.2.3 Output from the plugin

Output is consequently sent to STDERR to work around the fact that the NEMAN script that start olsrd consumes all output olsrd sends to STDOUT.

4.2.4 Gateway-gateway communication, sockets

I follow the common client/server model for the gateways. This is currently hardcoded for testing purposes, with a certain IP address working as client and another as server. While ideally every gateway would work as both a client and a server, to be able to both establish connections with neighbouring gateways on request, it should also be able to receive incoming connections.

Sockets can be seen as end points in communication, where data either go in or come out. A simple comparison could be a telephone. You do not need to know what that lies between, only that what you say into the phone will be heard in the other end. So you could say that sockets are the entry and exit points of communication between over networks or between processes.

In my implementation I create sockets for communication between gateways, and I would still have done that in a real world scenario. However, to prove that the concept of transparent gateways works, I do not need to know about the lower layers, but that could have changed in a real scenario (e.g. if I wanted to add QoS support to the gateways I might need to know parameters like bandwitdh, latency and so on).

The sockets are created with standard C calls, provided by sys/sockets.h. This follows a standard and straightforward procedure. The server first creates a socket (the socket() function), assigns an address to the socket (localhost, through the bind() function), and then tells it to listen for incoming connection. I then registers the socket with olsrds socket parser to fire an event when it gets a connection. The event function will then accept the connection when it arrives (through the accept() function).

The client follows a similar procedure before it tries to connect to the server through the connect() function call.

Now that a connection is made, I use the send() and recvfrom() functions to send and recive data.

Information about which nodes that are on both networks are passed between the gateways in a very simple packet. It is a number indicating the amount of nodes mentioned in the message, followed by a list of nodes. The structure it is stored in is defined as

```
struct gw_msg{
  int size;
  union olsr_ip_addr *nodes;
};
```

There is certainly room for improvement here later. The most obvious one would be to specify specifically the size of `size` in bits using typedefs provided by olsrd.

To add further functionality like hop count and information only on nodes that have joined or left the network, one would need to improve the code further by having a structure defined for each node, and sending a list of these structures. This structure would need to contain the hop count to the node, and a flag indicating whether it left or arrived, while the definition of the message to be sent would look relatively similar to the current one. An example of the code needed follows

```
#define LEFT     0 // flag for left nodes
#define ARRIVED  1 // flag for new nodes

struct gw_msg_node{
  olsr_u8_t flags;
  olsr_u8_t hopcnt;
  union olsr_ip_addr addr;
};

struct gw_msg{
  olsr_u16_t size;
  struct gw_msg_node *nodes;
}
```

4.2.5 Unfinished parts of my implementation, and what is planned.

Updating olsrd's routing tables

To create the routing tables olsrd uses several internal tables, like the neighbourhood table and the two hop neighbourhood table. Both of these are originally generated when a node receives HELLO messages. There are two ways of injecting this information into olsrd.

One way is to update these lists manually, with the neighbouring gateway being a 1-hop neighbour, and every other node on the neighbouring network being a two hop neighbour. The other, and far easier way of doing it is to let the plugin create a HELLO message similar to one that would have originated from the neighbouring gateway, and use olsrd's existing routines for processing HELLO messages to update the internal lists.

Transmitting HELLO and TC messages to the local network
As long as the 1-hop and 2-hop neighbourhood are updated correctly, olsrd should also generate HELLO and TC messages correctly. Information about the neighbouring network should then propagate through the local network.

Tunnel for forwarding of data traffic
Tunneling is to encapsulate data sent on one network in packets suitable to be sent on the intermediate network. This allows packets to between two networks using one protocol to be sent over networks using another protocol.

Tunneling can be used to keep connected networks secure while packets are traveling over an unsecure network through the use of technologies like VPN or IPSec[10].

4.2.6 Installation and running the application

The plugin need to be compiled before it is run. It also needs olsrd installed. However the output is limited to debugging information at the moment.

Installation and compiling is done by extracting the gateway plugin into the /lib subdirectory under olsrds root directory. You can then use “make” to compile, and “make install” to install the program. To set up installation paths, please refer to the documentation following olsrd.

5. Evaluation

5.1 Introduction to testing

Testing is done to validate that that the implementation works according to the specification. This means we have to design tests for every requirement, and ensure that these requirements are met. We also have to test that special events do not make the implementation malfunction.

5.2 What should I test specifically, tests to test the different requirements

The informal requirements cannot be tested, because of their nature. They just have to be followed in the best manner possible. However, the more specific requirements mentioned in the *Requirements for transparent gateways* chapter can fully be tested, and how this should be done is listed below.

1. *Must be able to transmit data between gateways*

Is it possible to send data from a gateway on network A to a gateway on network B? If e.g. a packet from gateway A is possible to read on gateway B, the transmission should work. Since this is done while simulating in NEMAN you have to make sure that there is no direct wireless connection between the gateways, only the gateway connection. By listening to TAP0 (the monitoring channel in NEMAN) one can make sure that the packets do not go through OLSR.

2. *Gateways must be able to exchange information about which nodes that exists on their network.*

A print of the routing table on the node in neighbouring network should provide the necessary information to see if nodes on the neighbouring network do in fact exist in the routing table. If they do, the gateways must be able to exchange information.

One could also compare the topology shown in NEMAN with the list of nodes recived from the neighbouring gateway and verify that the received list is correct.

3. *Routing information that is transmitted between gateways should only contain*

new/left nodes from their respective networks, to minimize traffic load on the intermediate link.

A simple print of packages before they are sent, and after they are recived should reveal what information is transmitted between them. A couple of snapshots should ensure right before and after a node leaves/enters a network should make sure only necessary information is transmitted when it is necessary to send that information.

While verifying this, one should also ensure that the routing tables are kept up to date, since complete routing information is not passed between the gateways

4. *Data transmitted between gateways, that have a destination other than the gateway itself, should be forwarded to the correct node according to OLSR policy.*

TCPdump should provide the functionality to to see if a node receives a packet, and which node it came from. If the source is on another network the implementation should work.

5. *Packets transmitted between networks should not be affected by the type of network between the gateways (e.g. Ethernet or GSM)*

This is not implemented or tested in this thesis. But if the gateways are implemented so that it doesn't matter what type of network lies below, this shouldn't be a problem

6. *Routing information should have priority over other information, requiring a scheduler to prioritise packets*

This is not implemented or tested in this thesis. Not sure if this can be done in a plugin either, maybe you even have to make changes to the OS itself

7. *Nodes accepted on one network (Authenticated + negotiated address) should also be accepted on the neighbouring network.*

Not tested in this thesis, but should work automatically due to the nature of transparent gateways.

5.3 Realistic testing of gateway-gateway communication without the use of OLSR

To implement a realistic test of communication between gateways, while still using the emulation interface of NEMAN, I use a connection through localhost (127.0.0.1). The reason I chose localhost is that a first that this works independently of the TAP interfaces NEMAN uses, and therefore avoids using OLSR interfaces for gateway-gateway communication. Localhost also works as a regular interface, and is thus relatively realistic.

5.4 Results

Since the implementation is not complete, there are very few results to show.

So far the sockets between the gateways are set up, as well as the datastructures containing local addresses are also set up. The datastructures containing remote addresses use the same code as local addresses, and I assume that these work too.

With further implementation I would have hoped to prove that transparent gateways are a possible solution for connecting mobile ad hoc networks.

6. Conclusion

6.1 Summary

In this thesis I have looked into whether it might be possible to connect different ad hoc networks through gateways. Further, I have analysed possible solutions, like using OLSRs built in gateways to foreign networks and using a different approach through transparent gateways. I decided to try to implement and simulate two ad hoc networks communicating with each other through transparent gateways, though time didn't permit that.

Taking a look back at the beginning, there are obvious scenarios where connecting ad hoc networks would be beneficial, like the tunnel rescue scenario. So hopefully, future will prove that connecting MANETs through other networks is possible.

6.2 Open problems

This section describes the work I would have done if I had a little more time.

With the work I have done so far, it seems like transparent gateways are at least possible. But there are still a lot of open problems, and I deeply regret not being able to finish the implementation, since if I had time to do that I could have analysed several problems in further detail, and maybe seen others that needed to be worked out. While I planned having a deeper look into the security issues that might arise with gateways too, I did not have time.

Also, if I had had more a lot more time, I would have had a better look at the proposed QoS solutions to OLSR, and possibly adjusted my gateways to be able to work with these protocols. It would probably have required some alterations to the routing calculation algorithms, as the communication between gateways would not work in the same way as communication between normal nodes.

6.2.1 Finish implementation

Since my implementation is not complete, except the part of simulating communication between gateways, this would have been my first priority. While the implementation is

far from finished, I feel I have planned it well enough so it would not take that much time to get it done.

Though, what is missing is storing of neighbouring addresses, updating of routing tables, transmission of routing information.

6.2.2 Deeper look into security issues.

While I originally planned on making secure gateways, I did not have time have a deeper look into that either. This would be my next point to research.

6.3 Future work

Some problems like QoS in ad hoc networks were not covered in any detail in this thesis but were partly researched during the work process. And the HNA based gateways where discarded for the transparent one. To broaden the horizon, I would very much have liked to had a better look into that solution too.

6.3.1 Further investigation into HNA based and traditional gateways

These solutions would not only give a more conventional gateway, but would also keep the gateways more similar to normal networking gateways. While choosing the transparent approach solved some problems easily, like addressing issues, others did arise like keeping traffic between the gateways down.

By having a look into HNA based gateways, it might have been possible to find suitable solutions without going through the unconventional transparent ones. By researching HNA gateways between OLSR networks more closely, I would probably also have to look deeper into duplicate address detection algorithms too, and maybe the key to get HNA gateways to work lies within altering one of these algorithms, or developing a new one.

Ideally, I would made a comparison between transparent and HNA gateways before

making the final choice, as besides addressing issues and gateway-gateway traffic, there might be some other issues like service discovery that will make them different.

If one could guarantee that addressing and routing would not be an issue, both with fixed addresses (the 3+ networks problem described in the design) and negotiated addresses, HNA gateways might have been a better choice.

6.3.2 Bringing Quality of Service to the gateways

I would like to see if a QoS scheme can improve the choice of gateway, and improve the traffic flow over it. However, this would require the gateways to be implemented on an network that supports QoS, e.g. QOLSR[10]To find this we first need to identify all the problems related to this, before we can search for solutions.

When there exist several gateways in an ad hoc network the question of which should be used and how to decide which araise. First the available resources, e.g. bandwidth, will be an important question. Another important question in ad hoc networks is the total hop count. One would not want gateways to become bottlenecks, so another thing that should be avoided if possible is air congestion (too many nodes try to transmit at the same spot). And finally, one would most likely try to avoid local traffic from being sent to a gateway.

Many of these problems should be split into smaller categories while we examine them. First we may have different bandwidths in different parts of the network. For example, the source and destination network may not be operating at same bandwidth (for now, I assume all nodes in the same network operate at the same speed). We also have another bandwidth between the gateways (see figure 6.1). And since it seems wise to choose the gateway with the highest bandwidth, one should combine this with the hopcount on each side of the gateway, gateway therefore we need to calculate hop count on both sides of the GW and the total hop count (figure 6.2).

Several of the problems described are dependent of each other. Hop count from source node to gateway, and from gateway to destination (and gateway-gatewat if traversing several networks), are the same, but in different networks. Hop count from source to gateway and source to destination are related, since node to gateway is a part of the source – destination count. In the same way is also bandwidth in source – gateway (and

gateway – destination) and bandwidth between gateways related.

The available resource(bandwidth) problem and the hop count problem are closely related. As low hop count as possible may not always be preferable, same as as high bandwidth as possible. It might be good to trade one hop extra for a DSL gateway instead of a GSM gateway. So an algorithm uniting hop count and bandwidth would need to be used. The hop count seems like a path-optimization(PO) problem (as defined in by Chen and Nahrsted[9]) on a part of the total path, while the bandwidth is a link-constrained (LC) problem.

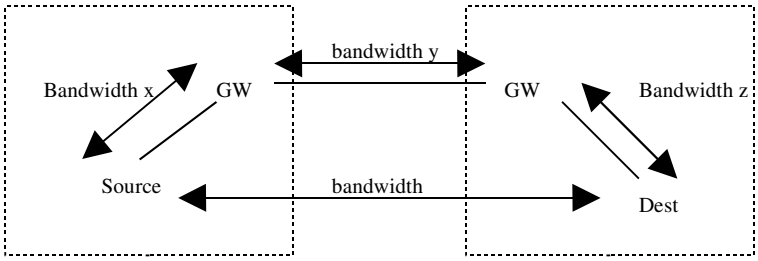


Figure 6.1: Simple illustration of where the bandwidth problems are located, shown as bandwidth x, y and z

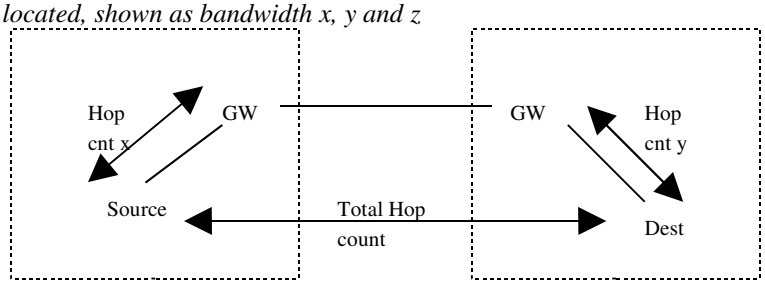


Figure 6.2: Simple illustration of where the hop count problems are located, shown as hop cnt x, y

To avoid air congestion traffic should be routed over several different gateways if possible. If one gateway seems like the ideal one to many nodes, too many might use it, causing a lot of collisions around the gateway. If another less desireable gateway is available, it should be used to relive the first of some of the traffic if it becomes congested. This is actually another bandwidth-problem, and bandwidth from the source node to the gateway should be reserved. When a gateway has to many reservations, it should either refuse new connections so they can be routed through other gateways if available, or tell the transmitting nodes to slow down (but strictly speaking, this is against

the resource reservation philosophy of QoS...). This is a LC problem.

It is also worth noting that two different gateways that are connected, but opreates in different networks may have different loads. This is because one partition may have high internal traffic while the neighbouring partition may have low. Therefore one gateway should be able to tell it's *sister* to slow down. And then we also have another connection between source-gateway and gateway-destination.

One should also try to avoid loops over several partitions. If there are three partitions (see figure 6.33), each with two gateways connecting both neighbouring networks, a connection may span an unnecessary network. In most cases this should be avoided because it causes an increase in traffic in the intermediate partitions. But it can also decrease the total load on the entire network, if the total hop count is lower when travelling over the additional partition. This again seems like another hop-count problem, and is a PO problem on the whole path.

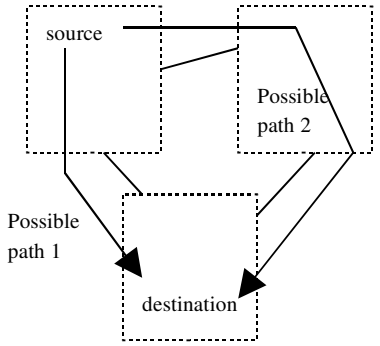


Figure 6.3: loops over several cells

Challenges of finding a common measurement of how adequate a route is that a paths are measured in different ways. The link-constrained/optimization problems is never better than the weakest link, while the path-optimization/constrained is the sum of all weaknesses/strengths over the whole path. To combine these directly can be difficult. Therefore it can be practical to find several solutions that satisfy one of the QoS requirements of one of the problems first (either LC or PO), and then see if there are any routes among the first that satisfy the other requirements.

Another challenge when using QoS in ad hoc networks with gateways is that this type of network spans several types of technology. First you have the wireless ad hoc domain, which has other problems than the wired domain between the gateways. So should we use two different algorithms (one for wired and one for wireless), or should they be treated as a single entity.

A problem with transparent gateways is how one should find a QoS path, when some of the metrics applies to the tunnel between the gateways. Here the common node will know

that one interface in the virtual gateways has a path to the destination, but they will not know of the bandwidth or the prolonged delay between the interfaces (because the data has to travel through the tunnel). In addition, if the gateways deliver simplified routing information to their networks (e.g. Delivered routing info is “I have a route to A”, while the actual info is “I have a route to A in 5 hops”), a node has the wrong hop count to the destination. A possible solution would be to make the QoS algorithm distributed, but this would remove many of the benefits of a proactive protocol.

By grouping same types of problems together we get a multi-PO multi-LC problem. This type of problem however is not mentioned to be neither NP-complete or solvable in polynomal time by [9], so I have to find out if it's easily solvable. Since the bandwidth problems in a way are dependent of each other, it should be possible to find a common metric for them, and the same applies to the hop-count problems. This would reduce the problem to a PO LC problem.

It seems like a QoS scheme could secure a more smooth transfer of data between the networks. It is important that this QoS scheme would have to contain a set of parameters based on the problems mentioned above. A challenge is that since some of these are very closely related, some sort of algorithm to unite them should be developed.

Some problems can most likely be modelled mathematically (like when will air congestion occur, and when will it be useful to prevent loops, and the bandwidth problems). If the QoS parameters needed are only link-constrained and path-optimization (maybe path-constrained is better, e.g. no more than one hop more than minimum, but one would still have to calculate minimum then), maybe some existing algorithm can be modified slightly, and then used (none of the algorithms mentioned by Chen and Nahrsted handles a LC PO, or a multi LC multi PO problem though).

We also have the question whether it is possible to create a QoS scheme that selects an appropriate GW in a reactive protocol. The problem here is that none of the nodes has any stored routing information (except active routes of course), and thus they cannot know of the gateways (it probably is a bit more advanced than this, but I’ve haven’t studied any reactive protocol in very much detail yet). With a transparent gateway route reqests will be sent over the gateway unrestricted. Therefore it seems difficult to to reduce routing traffic with transparent gateways (but it might be possible to at least select the best gateway with a distributed QoS scheme, which analyses several paths during the

route discovery procedure). With non-transparent gateways, the nodes need to know that the gateway exists, but it might be easier to control the traffic (the gateways could announce their *attractiveness* when they announce their services).

Hopefully, by using a set of QoS parameters one can achive a certain increase in performance of the gateways. It seems important that one chooses the right gateway for every connection. The challenges seems to be how the nodes should be able to choose an appropriate path without complete topology information, and how nodes should discover/gain access to all the metrics needed to calculate the best path (should the QoS algorithm be distributed?). To use an QoS algorithm that floods the network seems inapproriate (much wasted resources in a wireless invironment), and one that traverses several paths simultaniously removes many of the benefits of using a proactive protocol. I have to study the QoS algorithms (an maybe especially in ad hoc networks) more in detail, to get an idea of which one to choose, or if one would need to be developed.

7. References

[1] T. Plagemann, et al., "Middleware Services for Information Sharing in Mobile Ad-Hoc Networks - Challenges and Approach", Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004.

[2] T. Clausen, and P. Jacquet. “Optimized Link State Routing Protocol (OLSR)”, October 2003, RFC3626

[3] S. Corson and J. Macker. “Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations”, January 1999. RFC2501

[4] C. Perkins. Nokia Research Center, E. Belding-Royer, “Ad hoc On-Demand Distance Vector (AODV) Routing” July 2003, RFC3561

[5] M. Pužar. and T. Plagemann. “NEMAN: A Network Emulator for Mobile Ad-Hoc Networks”, Department of Informatics, University of Oslo.

[6] A. Tønnesen. “Implementing and extending the Optimized Link State Routing Protocol”

[7] http://www.olsr.org

[8] C.E. Perkins. “Ad hoc networking”. December 2000.

[9] S. Chen and K. Nahrstedt. “An Overview of Quality-of-Service Routing for the Next Generation High Speed Networks: Problems and Solutions”. Department of Computer Sience, University of Illinois at Urbana-Champaign

[10] H. Badis, and K. Al Agha. "QOLSR, QoS routing for Ad Hoc Wireless Networks Using OLSR", *European Transactions on Telecommunications*, Vol. 15, No. 4, 2005.

[11] S. Kent, R. Atkinson. “Security Architecture for the Internet Protocol”, November 1998. RFC4301

[12] http://www.isi.edu/nsnam/ns/

[13] S. Aust, D. Proetel, C. Görg, C. Pampu. “Mobile Internet Router for Multihop Ad hoc Networks”. 2003

[14] S. Glass, T. Hiller, S. Jacobs, C. Perkins. “Mobile IP Authentication, Authorization, and Accounting Requirements”. October 2000

[15] N. H. Vaidya, “Weak Duplicate Address Detection in Mobile Ad Hoc Networks”. 2002.

[16] C. E. Perkins, Elizabeth M. Royer, Samir R. Das. “IP Address Autoconfiguration for Ad Hoc Networks”

[17] O.V. Drugan, T. Plagemann, E. Munthe-Kaas. “Building resource aware middleware services over MANET for rescue and emergency applications”