University of Oslo
Department of Informatics

Wireless Extensions to OSPF: Implementation of the Overlapping Relays Proposal

Kenneth Holter

Master thesis

2nd May 2006

Abstract

Current wireless networks only allow users a limited freedom of mobility. This follows from the observation that these networks typically rely on existing infrastructure such as access points. To overcome this restriction, a new type of wireless networks is emerging - the Mobile Ad Hoc Network (MANET). MANETs are infrastructure-less, self-configuring networks that consist of mobile nodes moving in an arbitrary fashion.

During the past few years, MANETs have been subject to heavy investigation by both the academic and the industrial communities. Various research efforts have resulted in numerous routing protocols for such networks which are originally assumed to run in a standalone fashion. A recent growing trend in ad hoc networking, however, indicates that the inter-networking between MANETs and existing wired networks, e.g. the Internet, is an interesting topic of practical importance. Being a standard interior routing protocol widely deployed in the global Internet, Open Shortest Path First (OSPF) appears to be the most promising routing solution, not only for inter-connected MANETs, but also for standalone ad hoc networks.

This master thesis deals with the topic of how to extend OSPF for IPv6 (known as OSPFv3) for (enhanced) operation in MANETs. Based on an OSPF-MANET extension proposal published by Cisco Systems as an Internet-draft, referred to as *Overlapping Relays (OR)*, the thesis has designed and implemented the scheme which consists of several mechanisms for wireless extensions of OSPF. Aiming at reducing routing overhead imposed on MANETs, the OR proposal adopts the MultiPoint Relay (MPR) functionality of the Optimized Link State Routing (OLSR) protocol for optimized flooding in such networks, but ensures the reliable transmission assumed by OSPF. *Incremental Hellos*, a mechanism for reducing the size of Hello packets by signaling only *changes* in state, is also defined. The above mechanisms call for exchanging additional information on a link, and is implemented by means of Link Local Signaling (LLS); Existing OSPFv3 packets are appended by a special data block carrying the extra information, thus preserving the format of these packets.

These wireless extensions are implemented by modifying and extending the OSPFv3 source code shipped with the Quagga routing software suite. One design objective is to reuse parts of the UniK OLSR source code, which is done when implementing the overlapping relays mechanism. Another design objective is to minimize changes to the existing OSPFv3 source code, which for the most part is accomplished by having the extensions code reside in separate files.

The work on this thesis has produced a working implementation of the wireless extensions described in the above mentioned draft, and a testbed has been used for proving the implementation's correctness and operability. Issues regarding the design, implementations, as well as the draft itself, are presented in details in the thesis.

Acknowledgments

This thesis concludes my Master's Degree in Computer Science, and is submitted to the Department of Informatics at the University of Oslo.

I would like to thank Andreas Hafslund for his excellent guidance and helpful advice. Whenever I lost sight of the objectives of this thesis, he skillfully led me back on track.

I would also like to thank Frank Y. Li for his support and enthusiasm. He has enhanced my knowledge and understanding of the art of writing. My thanks also goes to Knut Øvsthus.

Thanks to my family and friends for their patience and encouragement. Finally, a special thanks goes to Marianne for her understanding and moral support.

Contents

Chapter 1

Introduction

As wireless communication technology is becoming more and more popular, people expect to be able to use their network terminals anywhere and anytime. Examples of such terminals are PDAs and laptops. Users wish to move around while maintaining connectivity to the network (i.e., Internet), and wireless networks provide them with this opportunity.

Wireless connectivity to the network gives users the freedom of movement they desire. Most wireless networks today require an underlying architecture of fixed-position routers, and are therefore dependent on existing infrastructure. Typically, the mobile nodes in such networks communicate directly with access points (APs), which in turn route the traffic to the corresponding nodes. Today, another type of wireless network is emerging, namely ad hoc wireless networks. These networks consist of mobile nodes and networks which themselves create the underlying architecture for communication. Because of this, no fixed-position routers are needed.

Figure 1.1 on the following page depicts a typical situation in which an ad hoc wireless network can be applied. In this battlefield scenario the soldiers move toward the enemy while maintaining network connectivity. The network itself is mobile in this scenario, thus forming a mobile ad hoc network (MANET) [9]. Soldier A is within range of the backbone, and can therefor act as a gateway between these two networks.

The latest trend in networking is the use of wireless links; Wi-Fi, GSM and Bluetooth have gained enormous popularity. The future lies in wireless communication. Although wireless networks have been deployed for quite some time, the wireless link does not allow for full mobility; current wireless networks typically depend on centralized control in the form of access points. This limitation restricts full mobility since wireless networks are dependent on infrastructure.

During the past few years, mobile ad hoc networking have been subject to a lot of research. Ad hoc networks allow for full mobility; Networks are created dynamically and without the need for existing infrastructure. One important issue with such networks is *routing*, which is the subject of study in this thesis.

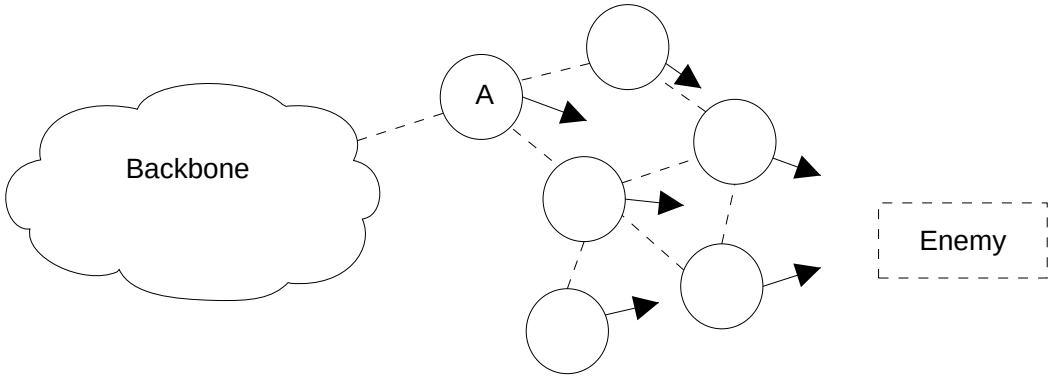


Figure 1.1: Wireless ad hoc network scenario. The soldiers (circles) maintain a mobile ad hoc wireless network while moving towards the enemy.

1.1 Problem statement

This thesis designs and implements a wireless extensions to OSPF for IPv6 (usually referred to as OSPFv3). These extensions are based on a wireless OSPF proposal which describes extensions defined for utilization on OSPF-MANET interfaces. The extensions include modifications to the flooding procedure to adapt to the multi-hop environment of MANETs. In addition, optimizations to the flooding procedure and Hello protocol are defined to diminish the routing overhead imposed on the MANET.

As of this writing, the OSPF-MANET research community provides two implementation of the above mentioned extensions. The implementation described in this thesis is one them, and the other implementation is provided by Boeing, and can be downloaded from [24].

Further research on wireless extensions to OSPF will benefit from access to different implementations. Details regarding the design, implementation or analysis described in this thesis may provide useful observations to researches who are interested in this topic.

1.1.1 Motivation for extending OSPF

During the past few years several routing protocols have been standardized by the IETF Mobile Ad-hoc Network (MANET) working group. A typical category is developed based on traditional link state routing protocols. Optimized Link State Routing (OLSR) [7], for example, is the most representative routing protocol of this kind, where protocol overhead minimization is achieved through *unreliable* flooding of network topology messages via Multipoint Relays (MPRs). On the other hand, another link state routing protocol, Open Shortest Path First (OSPF), which is the current Internet standard for interior routing, has been extensively studied and widely deployed in the wired Internet. A natural question that has recently received much attention within the IETF is: Why not adopt OSPF in MANETs?

The motivation for adopting OSPF in a MANET environment is twofold: Interoperability and familiarity. With both wired and wireless support, a MANET-capable OSPF router could operate properly when plugged into a wired OSPF network. By extending and building an OSPF framework, the transition and interoperability between wired and wireless networks would become seamless. Furthermore, OSPF is already a mature routing protocol. Experience and lessons learned from OSPF can be of great help when extending OSPF to wireless networks.

However, adopting OSPF to MANETs is not an easy task since OSPF was originally designed for more or less static, wired networks. There are several reasons that the OSPF protocol cannot be deployed directly in a MANET environment. First of all, OSPF does not have a suitable interface type for a wireless broadcast environment which is characterized by a multicast-capable transmission medium and where routers do not necessarily form a full mesh. Moreover, to adapt to the unpredictable behavior of mobile nodes, OSPF will have to increase the amount of topology dissemination messages, leading to a prohibitively high routing overhead when adopted in MANETs. Consequently, OSPF does not scale well even for a quite small MANET [31]. Furthermore, since links in a wireless environment cannot be assumed to be bi-directional, the Designated Router (DR) mechanism, a common OSPF flooding optimization mechanism, will not perform correctly in MANETs as this mechanism assumes a true multi-access network.

So far, two proposals, [5] and [20], have received the most attention. In this thesis I have designed and implemented wireless extensions to OSPFv3, based on mechanisms described in [5].

1.2 Chapter overview

Chapter 2 Wireless Network An overview of wireless technologies and issues related to computer network.

Chapter 3 Mobile Ad-hoc Networks (MANETs) Outlines and comparisons of two differing MANET routing protocols.

Chapter 4 Open Shortest Path First (OSPF) An introduction to OSPF.

Chapter 5 Wireless Extensions to OSPF The Internet draft[5] published by Cisco Systems. This draft describes the wireless extensions implemented in this thesis.

Chapter 6 The Quagga Routing Software Suite Outlines the routing software framework used in this thesis.

Chapter 7 Implementation Design Described the design and implementation of the wireless extensions described in Chapter 5. Details regarding data structures, modifications to OSPFv3, and so forth, are described. Issues regarding the implementation are also outlined.

Chapter 8 Proof of Concept - Test of the Implementation Several test scenarios are set up to illustrate how the wireless extensions responds to changes in topology.

Chapter 9 Other Issues Relevant issues such as auto-configuration and scalability of OSPF-MANETs are shortly outlined in this chapter.

Chapter 10 Conclusions and Future Work Conclusions on the implementation and suggestions for future work.

Chapter 2

Wireless Networks

This chapter provides a general overview of wireless networks and some related problems.

2.1 Wireless networks

Numerous different wireless networks exist. They vary in the way the nodes interconnect. One can roughly classify them into two types:

- Infrastructure dependent networks
- Ad hoc wireless networks

Current cellular networks are *infrastructure dependent*. These networks are characterized by their use of access points, or base stations. In addition to acting as a router within the network, an access point can also act as a bridge connecting, for example, the wireless network to a wired network. GSM, and its 3G counterpart UMTS, are examples of well known cellular networks.

In ad hoc wireless networks the nodes themselves are responsible for routing and forwarding packets. Hence, the nodes need to be more intelligent in order to function both as routers as well as regular hosts.

Centralized routing and resource management by an AP implies less complicity than distributed routing. An AP, as opposed to individual nodes, usually possess more information about the network, and is therefore able to make intelligent routing decisions.

2.2 The OSI reference model

The Open Systems Interconnection (OSI) reference model was developed by the International Organization for Standardization (ISO) aiming to standardize the protocols used in various network layers.

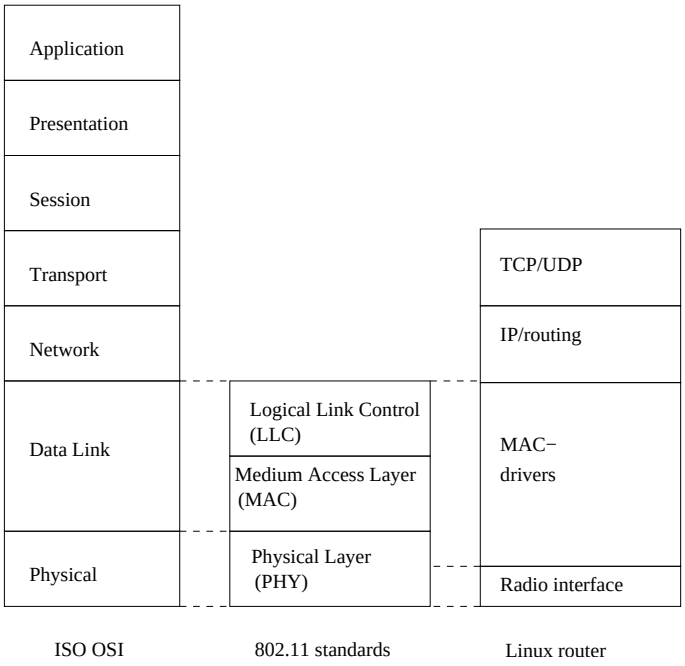


Figure 2.1: IEEE 802.11 standards mapped to the OSI reference model and a standard linux router implementation.

IEEE 802.11[8] is a family of specifications for Wireless Local Area Networks (WLANs). Like all IEEE 802.11 standards, the 802.11 works on the two lower levels of the OSI model. Although wireless networks are not restricted to any special hardware, nodes in such networks are likely to operate according to the IEEE 802.11.

Figure 2.1 shows the IEEE 802.11 standards mapped to the OSI reference model. The figure also shows how the implementation of a typical linux router corresponds to these models.

In wireless networks, nodes typically use radio frequency channels as their physical medium. This corresponds to the lowest layer in the OSI model. Since the nodes need not be physically connected, the network offers data connectivity along with user mobility.

The IEEE 802.11 MAC layer corresponds to the data link layer in the OSI model. The main objective of the OSI data link layer is to provide error free transmission of data across a physical link. IEEE 802.11 protocols' version of this scheme consists of two sub-layers: Logical Link Control (LLC) and Medium Access Control (MAC). The (possibly) most important services that the LLC offers is error- and flow control. The MAC directly interfaces with the physical layer, and provides services such as addressing, framing, and medium access control.

2.3 Radio technology

Nodes in wireless networks typically utilize radio transmission. Today, such wireless links are likely to be made up of interfaces operating according to the IEEE 802.11 family of specifications.

Wireless LANs use electromagnetic waves (radio or infrared) to communicate. The waves propagate through wireless medium (even in a vacuum).

Different frequencies have different qualities. Lower frequencies are easy to generate, can travel long distances, and can penetrate buildings easily. Radio waves operate on lower frequencies than infrared waves, making it more suitable for most wireless networks. High frequency waves, on the other hand, *may* allow an increase in the amount of information transmitted per second, but are far more sensitive to physical obstacles such as walls.

Frequency Hopping Spread Spectrum (FHSS) and Direct Sequence Spread Spectrum (DSSS) are the two radio transmission techniques supported by IEEE 802.11. The idea behind FHSS is that the transmitter hops from frequency to frequency hundreds of times per second. The hop pattern is known to both the sender and receiver, and receivers that are not aware of the pattern are not likely to be able to detect the transmission. DSSS, on the contrary, does not hop from one frequency to another, but distributes the signal over the entire frequency band simultaneously.

2.4 Issues in wireless networks

There are a number of issues to consider when designing operations of wireless networks. The next subsections describe a selected few of them.

2.4.1 Hidden terminals

In figure 2.2 on the next page, node A and node C are in range for communicating with node B, but not with each other. In the event that both try to communicate with node B simultaneously, A and C might not detect any interference on the wireless medium. Thus, the signals collide at node B, which in turn will be unable to receive the transmissions from either node. [22]

The typical solution for this so-called “Hidden terminal” problem is that the nodes coordinate transmissions themselves by asking and granting permission to send and receive packets. This scheme is often called RTS/CTS (Request To Send/Clear To Send). The basic idea is to capture the channel by notifying other nodes about an upcoming transmission. This is done by stimulating the receiving node to output a short frame so that nearby nodes can detect that a transmission is going to take place. The nearby nodes are then expected to avoid transmitting for the duration of the upcoming (large) data frame. The scheme is illustrated in Figure 2.3 on the following page.

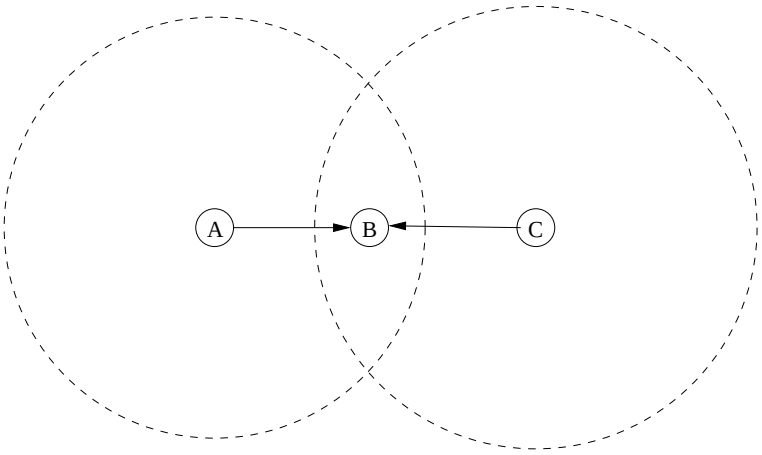


Figure 2.2: The hidden terminal problem. Node A and C try and communicate with B simultaneously, but cannot detect the interference.

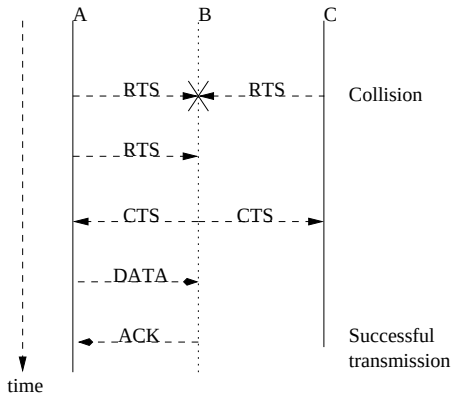


Figure 2.3: A Request To Send (RTS) and Clear To Send (CTS) scheme. First, A and C each transmit a packet simultaneously, causing a packet collision at B. Then A retransmits the packet before C does, thus capturing the channel.

2.4.2 Exposed terminals

Consider a topology similar to that of Figure 2.2 on the page before, but with an added node D only reachable from node C.

Furthermore, suppose node B communicates with node A, and node C wants to transmit a packet to node D. During the transmission between node B and node A, node C senses the channel as busy. Node C falsely concludes that it may not send to node D, even though both the transmissions (i.e., between node B and node A, and between node C and node D) would succeed. Bad reception would only occur in the zone between node B and node C, where neither of the receivers are located. This problem is often referred to as “the exposed terminal problem”.

Both the hidden and the exposed terminal problem cause significant reduction of network throughput when the traffic load is high.

2.4.3 Neighbor discovery

Discovering neighbors is a central link layer operation in wireless networks. In some cases, the node might be interested in just one particular kind of neighbor, or all neighbors. In either case, the node needs to discover its neighbors and determine their types. Since the topology of the network tends to be highly dynamic, the neighborhood information should be updated periodically. If the topology undergoes rapid changes in connectivity that is too rapid, i.e. the nodes are unsuccessful in exchanging topological information, flooding is the only way to get data to a particular destination. [18]

Chapter 3

Mobile Ad-hoc Networks (MANETs)

MANETs are a subset of wireless networks, as they can be viewed as wireless networks not dependent on existing infrastructure. An overview of MANETs is given in this chapter, along with an introduction to the two most prominent MANET routing protocols.

3.1 Introduction

In ad-hoc networks, as mentioned above, the nodes are responsible for the routing and forwarding of packets. If the wireless nodes are within range of each other, no routing is necessary. But if the nodes have moved out of range of each other, and are not able to communicate directly, intermediate nodes are needed to make up the network in which the packets are to be transmitted.

Figure 3.1 gives an illustration of a multi-hop (ad-hoc) network.

There are a number of situations in which ad-hoc networks are suited. Examples include emergency operations where no fixed infrastructure exists and military operations where the existing infrastructure might not be trustworthy.

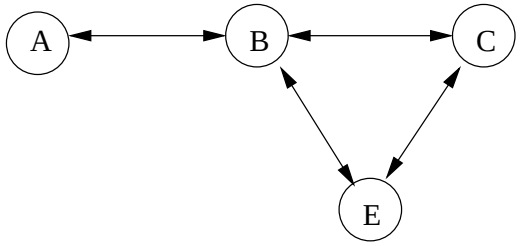


Figure 3.1: Ad-hoc network. The nodes make up the network themselves. C and E cannot reach A directly, so B routes and forward the traffic.

In cellular networks, nodes in an ad-hoc network are responsible for dynamically discovering which other nodes they can directly communicate with.

There are several issues that need to be considered when it comes to ad- hoc networking. A brief overview of some of these follows:

Medium access scheme The medium access protocol (MAC) needs to be designed to allow for certain characteristics of wireless networks. Typical for wireless networks the nodes move about, and this leads to the hidden terminal problem previously described. Fair access to the medium and minimization of collisions, must also be taken into account. The MAC protocol should also be able to adjust the power used for transmissions, because reducing transmission power at a node causes a decrease in interference at neighboring nodes, and increases frequency reuse. [18]

Routing Traditional routing protocols are not designed for rapidly changing environments such as ad-hoc networks and customized routing protocols are needed. Examples of such protocols are AODV[23] and OLSR. Routing is further discussed below.

Security Due to the fact that the nodes in a wireless ad-hoc network communicate on a shared medium, security becomes an important issue. In combination with the lack of any central coordination this makes the network more vulnerable to attack than wired networks. There are different ways of compromising wireless networks, including:

- Denial of service. An attacker makes services unavailable to others by keeping the service provider busy.
- Resource consumption. Battery power of critical nodes is depleted because of unnecessary processing caused by an attacker, or the attacker causes buffer overflow which may lead to important data packets being dropped.
- Host impersonation. As the name suggests, a compromised node may impersonate a host, and thereby cause wrong route entries in routing tables elsewhere in the network.

Quality of service Providing quality of service (QoS) in a wireless ad-hoc network is a difficult task. Nodes in such a network usually act both as clients and service providers making, contrary to most networks, the boundary between network and host less clear. Hence, to achieve QoS, a better coordination between the nodes is required. Furthermore, wireless communication usually implies limited resources, and this, in addition to the lack of central coordination, exacerbates the problem.

- Parameters. Different applications have different QoS parameter requirements. Multimedia applications require high bandwidth and low delay whereas availability is the primary requirement for search-and-rescue applications.
- Routing. To make sure that applications are provided with the services they require, QoS parameters should be considered for route decisions. Throughput, packet loss rate, and reliability are examples of such parameters.

3.2 Routing in ad-hoc wireless networks

As the nodes in a wireless ad-hoc network can be connected in a dynamic and arbitrary manner, the nodes themselves must behave as routers and take part in discovery and maintenance of routes to other nodes in the network.

The goal of a routing algorithm is to devise a scheme for transferring a packet from one node to another. One challenge is to define/choose which criteria upon which to base the routing decisions. Examples of such criteria include hop length, latency, bandwidth and transmission power.

[18] lists some of the challenges in designing a routing protocol for ad-hoc wireless networks, and a brief overview of these is given below.

Mobility The network needs to adopt to rapid changes in the topology due to the movement of the nodes, or the network as a whole.

Resource constraints Nodes in a wireless network typically have limited battery and processing power, and these resources must be managed optimally by the routing protocol.

Error-prone channel state The characteristics of the links in a wireless network typically vary, and this calls for an interaction between the routing protocol and the MAC protocol to, if necessary, find alternate routes.

Hidden and exposed terminal problem This is described in 2.4.1 and 2.4.2.

MANET routing protocols are typically subdivided into two main categories: proactive routing protocols and reactive on-demand routing protocols.

3.2.1 Proactive protocols

In networks utilizing a proactive routing protocol, every node maintains one or more tables representing the entire topology of the network. These tables are updated regularly in order to maintain up-to-date routing information from each node to every other node.

To maintain up-to-date routing information, topology information needs to be exchanged between the nodes on a regular basis which in turn leads to relatively high overhead on the network. The advantage is that routes will always be available on request.

Many proactive protocols stem from conventional link state routing, including the Optimized Link State Routing protocol (OLSR) which is discussed in section 3.4 on page 16.

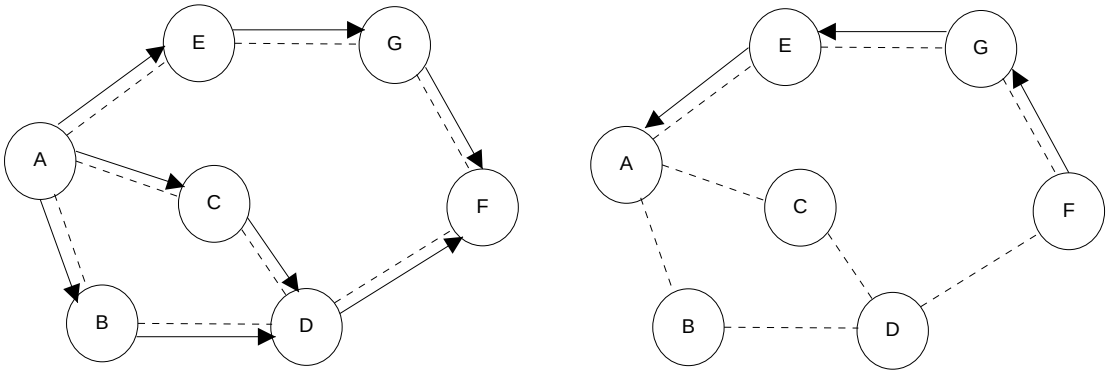


Figure 3.2: AODV route discovery.

3.2.2 Reactive protocols

Unlike proactive routing protocols, reactive routing protocols do not make the nodes initiate a route discovery process until a route is required. This leads to higher latency than with proactive protocols, but lower overhead. Ad-hoc On-Demand Distance-Vector routing protocol (AODV) is further discussed in section 3.3.

3.2.3 Hybrid protocols

These types of protocols combine proactive and reactive protocols to exploit their strengths. One approach is to divide the network into zones, and use one protocol *within* the zones, and another *between* them.

3.3 Ad-hoc On-demand Distance Vector (AODV)

This section describes the AODV routing protocol. Some details on the route request mechanism and link sensing are provided, along with an example.

3.3.1 Introduction to AODV

AODV is an on-demand routing algorithm that determines a route only when a node wants to send a packet to a destination. It is a relative of the Bellman-Ford distant vector algorithm, but is adapted to work in a mobile environment. Routes are maintained as long as they are needed by the source. AODV is capable of both unicast and multicast routing.

In AODV every node maintains a table containing information about which direction to send the packets in order to reach the destination.

Source address	Request ID	Destination address	Source sequence #	Destiantion sequence #	Hop count

Figure 3.3: The format of a ROUTE REQUEST packet.

Sequence numbers, which are one of the key features of AODV, ensures the freshness of routes.

3.3.2 Control Messages

Three message types are defined by AODV:

RREQ When a route is not available for the desired destination, a *route request* packet is flooded throughout the network. Figure 3.3 shows the format of such a packet.

RREP It a node either is, or has, a valid route to the destination, it unicasts a *route reply* message back to the source.

RERR When a path breaks, the nodes on both sides of the link issues a *route error* to inform their end nodes of the link break.

3.3.3 Sequence numbers

AODV differs from other on-demand routing protocols in that it uses *sequence numbers* to determine an up-to-date path to a destination. Every entry in the routing table is associated with a sequence number. The sequence numbers act as a route timestamp, ensuring the route remains up-to-date. Upon receiving a RREQ packet, an intermediate node compares its sequence number with the sequence number in the RREQ packet. If the sequence number already registered is greater than that in the packet, the existing route is the most up-to-date.

Counting to infinity

The use of sequence numbers for every route also helps AODV avoid the “count to infinity” problem. This problem arises in situations where nodes update each other in a loop. “The core of the problem ”, as Tanenbaum [32] put it, “ is that when X tells Y that it has a path somewhere, Y has no way of knowing whether it itself is on the path”. So if Y detects that the link to, say, Z is down, but X says it has a valid path, Y assumes X in fact *does* have a path, thus registering X as the next neighbor toward Z. If the path X assumed is valid runs through Y, X and Y will start updating each other in a loop.

3.3.4 Route discovery

Route discovery is initiated by issuing a RREQ message. The route is established when a RREP message is received. However, multiple RREP messages may be received, each suggesting different routes to the destination. The source only updates its path information if the RREP holds information about a more up-to-date route than already registered. Thus, every incoming RREP packet is examined to determine the most current route.

When a intermediate node receives either a RREQ or a RREP packet, information about the previous node from which the packet was received is stored. This way, next time a packet following that route is received, the node knows which node is the next hop toward the source or destination, depending on which end node originated the packet.

The next subsection illustrates route discovery by providing an example.

Example of a Route Discovery

Consider the ad-hoc network of Figure 3.2 on page 13. In this example, node A wants to send a packet to node F. Suppose A has no table entry for F. A then needs to discover a route to F. In our example, we assume that neither of the nodes knows where F is.

The discovery algorithm works like this:

Node A broadcasts a special ROUTE REQUEST packet on the network. The format of the ROUTE REQUEST (RREQ) packet is shown in figure 3.3 on the preceding page. Upon receiving the RREQ packet, B, C and E check to see if this RREQ packet is a duplicate, and discards it if it is. If not, they proceed to check their tables for a valid route to F. If a valid route is found, a ROUTE REPLY (RREP) packet is sent back to the source. In the case of the destination sequence number in the table being less than the destination sequence number in the RREQ, the route is not considered up-to-date, and thus no RREP packet is sent. Since they don't know where F is, they increment the RREQ packet's hop count, and rebroadcast it. In order to construct a route back to the source in case of a reply, they also make an entry in their reverse route tables containing A's address.

Now, D and G receive the RREQ. These go through the same process as B, C and E. Finally, the RREQ reaches F, which builds an RREP packet and unicasts it back to A.

The Expanding Ring search

Since RREQ packets are flooded throughout the network, this algorithm does not scale well to large networks. If the destination node is located relatively near the source, issuing a RREQ packet that potentially passes through every node in the network is wasteful. The optimization AODV uses is the *expanding ring* search algorithm. The source node searches successively larger areas until the destination node is found. This is done by incrementing the *time to live*

(TTL) value carried in every RREQ packet for every RREQ retransmission until a route is found thus expanding the “search ring” in which the source is centered.

3.3.5 Link Breakage

When a link breaks, a RERR message is propagated to both the end nodes. This implies that AODV does not repair broken links locally, but rather makes the end nodes discover alternate routes to the source. Moreover, link breakage caused by the movement of end nodes also results in initialization of a route discovery process.

When an RERR packet is received by intermediate nodes, their cached route entries are removed.

3.4 Optimized Link State Routing (OLSR)

In this section the proactive routing protocol OLSR is described, with emphasis on the *MultiPoint relay (MPR)* mechanism. As the *overlapping relays* mechanism defined for WOSPF-OR is essentially an MPR scheme for OSPFv3 networks, OLSR (especially the MPR mechanism) is more thoroughly examined than AODV.

3.4.1 Introduction to OLSR

The Optimized Link State Routing (OLSR) is a table-driven, proactive routing protocol developed for MANETs. It is an optimization of pure link state protocols that reduces the size of control packets as well as the number of control packet transmissions required.

OLSR reduces the control traffic overhead by using Multipoint Relays (MPR), which is the key idea behind OLSR. An MPR is a node’s one-hop neighbor which has been chosen to forward packets. Instead of pure flooding of the network, packets are forwarded by a node’s MPRs. This delimits the network overhead, thus being more efficient than pure link state routing protocols.

OLSR is well suited to large and dense mobile networks. Because of the use of MPRs, the larger and more dense a network, the more optimized link state routing is achieved.

MPRs helps providing the shortest path to a destination. The only requirement is that all MPRs declare the link information for their MPR selectors (i.e., the nodes which have chosen them as MPRs).

The network topology information is maintained by periodically
exchange link state information. If more reactivity to topological changes is required, the time interval for exchanging of link state information can be reduced.

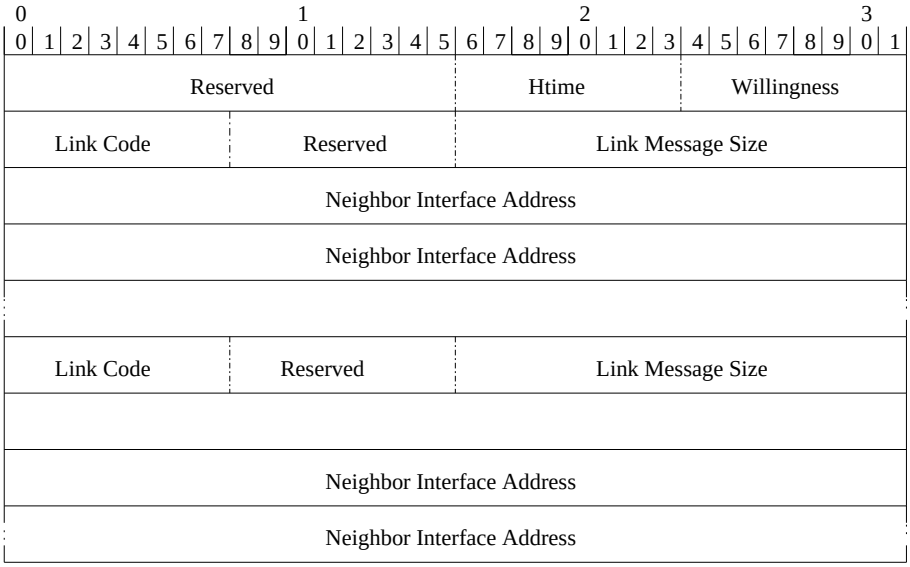


Figure 3.4: The format of a OLSR HELLO packet.

3.4.2 Control messages

OLSR uses three kinds of control messages: HELLO, Topology Information (TC), and Multiple Interface Declaration (MID).

A Hello message is sent periodically to all of a node's neighbors. Hello messages contain information about a node's neighbors, the nodes it has chosen as MPRs (i.e., the MPR Selector set), and a list of neighbors with whom bidirectional links have not yet been confirmed. Figure 3.4 shows the format of the Hello message.

Every node periodically floods the network with a TC message using the multipoint relaying mechanism. This message contains the node's MPR Selector set.

A MID message is used for announcing that a node is running OLSR on more than one interface. The MID message is flooded throughout the network by the MPRs.

3.4.3 Multipoint Relays

A node N selects an arbitrary subset of its 1-hop symmetric neighbors to forward data traffic. This subset, referred to as an MPR set, covers all the nodes that are two hops away. The MPR set is calculated from information about the node's symmetric one hop and two hop neighbors. This information is extracted from HELLO messages. Similar to the MPR set, an MPR Selectors set is maintained at each node. An MPR Selector set is the set of neighbors that have chosen the node as their MPR.

Upon receiving a packet, a node checks it's MPR Selector set to see if the sender has chosen the node as MPR. If so, the packet is forwarded, else the packet is processed and discarded.

3.4.4 Selection of Multipoint Relay Nodes

The MPR set is chosen so that a minimum of one-hop symmetric neighbors are able to reach all the symmetric two-hop neighbors. In order to calculate the MPR set, the node must have link state information about all one-hop and two-hop neighbors. Again, this information is gathered from HELLO messages. Only nodes with willingness different than WILL_NEVER may be considered as MPR.

3.4.5 Neighbor discovery

As links in an ad-hoc network can be either unidirectional or bidirectional, a protocol for determining the link status is needed. In OLSR, HELLO messages serve this purpose. HELLO messages are broadcast periodically for neighbor sensing. When a node receives a HELLO message in which it's address is found, it registers the link to the source node as symmetric.

As an example of how this protocol works, consider two nodes A and B which have not yet established links with each other. Firstly, A broadcasts an empty HELLO message. When B receives this message and does not find its own address, it registers in the routing table that the link to A is asymmetric. Then B broadcasts a HELLO message declaring A as an asymmetric neighbor. Upon receiving this message and finding its own address, A registers the link to B as symmetric. A then broadcasts a HELLO message declaring B as a symmetric neighbor, and B registers A as a symmetric neighbor upon reception of this message.

Topology Information

Information about the network topology is extracted from *topology control* (TC) packets. These packets contain the MPR Selector set of a node, and are broadcast by every node in the network, both periodically and when changes in the MPR Selector set are detected. The packets are flooded in the network using the multipoint relaying mechanism. Every node in the network receives such TC packets, from which they extract information to build a topology table.

3.4.6 Route Calculation

The shortest path algorithm is used for route calculations, which are initiated when a change is detected in either of the following: *the link set*, *the neighbor set*, *the two-hop neighbor set*, *the topology set*, or *the Multiple Interface Association Information Base*.

To calculate the routing table, information is taken from *the neighbor set* and *the topology set*. The calculation is an iterative process, in which route entries are added starting from one-hop neighbors, increasing the hop count each time through. A more detailed outline is found in [7].

3.5 Comparing the protocols

In this section we will compare the described protocols. In section 3.5.1 a comparison overview is provided, and in sections 3.5.2 through 3.5.4 the protocols are compared with respect to resource usage, mobility, and route discovery delay.

3.5.1 Overview

Being a proactive protocol, OLSR imposes large control traffic overhead on the network. Maintaining an up-to-date routing table for the entire network calls for excessive communication between the nodes, as periodic and triggered updates are flooded throughout the network. The use of MPRs decreases this control traffic overhead, but for small networks the gain is minimal. The traffic overhead also consumes bandwidth.

The reactivity of AODV is more sensitive to resource usage. As control traffic is almost only emitted during route discovery, most of the resource and bandwidth consumption is related to actual data traffic.

3.5.2 Resource usage

Because information about the entire network needs to be maintained at all times, OLSR requires a large amount of storage complexity and usage. Hence, there is a greater demand for storage capacity for nodes in such networks. The control overhead adds to the necessary processing in each node also increasing the battery depletion time. Another downside to OLSR is that it must maintain information about routes that may never be used, wasting possibly scarce resources.

AODV greatly simplifies the storage complexity and reduces energy consumption by keeping only information about active routes stored at a node. The processing overhead is less than with OLSR, as little or no useless routing information is maintained.

3.5.3 Mobility

OLSR and AODV have different strengths and weaknesses when it comes to node mobility in MANETs. Unlike wired networks, the topology in wireless ad-hoc networks may be highly dynamic, causing frequent path breaks to on-going sessions. When a path break occurs, new routes need to be found. As OLSR always has up-to-date topology information at hand, new routes can be calculated immediately when a path break is reported.

Because AODV is a *reactive* protocol, this immediate new route calculation is not possible, so a route discovery must be initiated.

In situations where the network traffic is sporadic, OLSR offers less routing overhead due to having found the routes pro-actively. AODV, on the other hand, will have to first discover a route before the actual information can be transmitted. This calls for extensive control overhead

per packet. In cases where the network traffic is more or less static (i.e., the traffic has a long duration), however, AODV may perform better, as the amount of control overhead per packet decreases.

3.5.4 Route discovery delay

When a node in a network running the OLSR protocol wishes to find the route to a host, all it has to do is a routing table lookup, whereas in a AODV network, a route discovery process needs to be initialized unless no valid route is cached. It goes without saying that a simple table-lookup takes less time than flooding the network, making the OLSR protocol performance superior in delay-sensitive networks.

Chapter 4

Open Shortest Path First (OSPF)

OSPF is a very comprehensive and complex routing protocol, and has implemented several extensions to adapt to different network types. This chapter describes only a selected few of OSPF's features, focusing on features relevant to the wireless extensions described in Chapter 5.

Whereas OSPFv3 (described in RFC 2740[1]) is essentially equivalent to OSPFv2 (described in RFC 2328) with support for IPv6, there are some essential differences between the two specifications (see subsection 4.1.1). Note that most of the functionality outlined in this thesis is equal in both specifications.

The term "OSPF", as used in this thesis, implies functionality defined for OSPFv2[2] but also applies to OSPFv3. When describing OSPFv3 specific details the explicit term "OSPFv3" is used.

4.1 Overview

OSPF is one of the most commonly known Link State (LS) protocols around. It uses the Shortest Path First (SPF) algorithm for calculating the shortest path to any destination known by the router that runs OSPF. OSPF classified as an Interior Gateway Protocol (IGP), distributing routing information within an Autonomous System (AS) (often referred to as routing domain).

In contrast to distance vector protocols such as the Routing Information Protocol (RIP) [14], OSPF maintains a complete map of the network. This map is known as the LS database, which in fact is a distributed database; Every router within the same area (see section 4.8.2) has an identical database. To keep this distributed database up-to-date, a great deal of effort goes into synchronizing it between the routers. Of all the different packets defined for OSPF, nearly all of them are defined for database synchronization.

[1] This specification is not a complete, stand-alone document. Essentially, it merely describes differences with regards to RFC 2328

[2] OSPFv1 was not much deployed, and was quickly replaced by its successor

4.1.1 Some differences between OSPFv2 and OSPFv3

OSPFv3 differs from OSPFv2 in some essential ways. Most relevant for this thesis are these observations:

- OSPFv3 packets, and most of the LSAs (described later), does not carry addressing semantics. All OSPFv3 routers are identified by their router ID, leaving a network-protocol-independent core.
- Two routers can become neighbors regardless of their associated network prefix. This follows from that link-local addresses are used for all inter-router communication.
- Two new LSAs are defined for OSPFv3, namely the *Link LSA* and the *Intra-Area-Prefix-LSA*. The former one has, as the same suggests, a link-local flooding scope, and provide the neighbors on the link with information about (amongst others) its link-local address, and its IPv6 prefixes associated with the link. The latter LSA type carry all IPv6 prefix information that in OSPFv2 were carried in Router LSAs and Network LSAs.

With many respects, OSPFv2 and OSPFv3 are quite similar. The above list includes only a few differences between the two specifications.

4.2 The Link State Database

As mentioned above, the link state database represents each router's map of the network topology. Within an area, this database is synchronized between all the routers. When a router first joins such an area, it mirrors one of its neighbors' LS database (resulting in the routers becoming *adjacent*, as described in later subsections). After the initial synchronization, every router floods routing updates for every other router in the area for analyze and comparison with its own view of the topology.

4.2.1 The routing table

The core of OSPF is the SPF algorithm. As the name suggests, the Open *Shortest Path* First protocol aims at calculating the shortest path possible to any given destination. The input to these calculations is the router's view of the network - the LS database. Based on the entries in this database the router builds an SPF tree from itself to the given destination. Based on this shortest path tree the router decides to *which* neighbor the packet in question should be sent. In other words, the LS database and the SPF tree calculation serves one purpose only: To decide to whom a given packet should be forwarded. The SPF tree yields a routing table for the router.

It is possible to assign a cost to each link in the network and take such metrics into account when calculating the SPF tree. This method has the advantage that factors such as bandwidth and delay may be considered. Calculating the SPF is a matter of running the well known

Dijkstra's algorithm on relevant information maintained in the link state database, where links in the network represent *edges* in the graph.

4.2.2 The Link State Advertisement (LSA)

Routing updates are messages in the form of LSAs. These messages describe the connectivity of the routers running OSPF. Several different types of LSA messages are defined, and a router constructs these messages to indicate its adjacencies, prefixes associated with connected networks, and so forth. The LSAs are carried in LS Update packets.

LSAs are flooded over links on which the receiving router is adjacent (described in the next subsection) to the transmitting router. As soon as two routers start the initial database synchronization, future LSAs are flooded on this link as long the (beginning) adjacency is maintained.

To illustrate the concept of LSAs, let us examine one such message type - the router LSA. This LSA type describes the state of the router's interfaces to the area (for information on areas, see 4.8.2) by including descriptions of fully adjacent neighbors on the interface. Each interface originates such LSAs, and new instances of the LSA are originated, for example, when new adjacencies are formed. An adjacent neighbor is described by *link descriptors*, indicating (amongst others) the cost of reaching the neighbor, as well as the neighbor's router ID.

4.2.3 Initial synchronization of the LS database

Upon joining a new network, a router must learn the LS database of this network. This is done by performing a database synchronization with one of its new neighbors. Note that, depending on the interface type used, the neighbor with whom to synchronize is not random. This topic will be discussed in later subsections.

The synchronizing routers exchange *database description (DD)* packets describing their current LSAs for the area. If a router discovers any discrepancies it will request the discrepant (or missing) LSAs from its neighbor.

The *database exchange* process leaves the two routers with an identical LS database, and the two routers are said to be *fully adjacent*. This process is referred to as initial synchronization.

4.2.4 Maintaining the LS database

The initial database synchronization is designed to ensure that newly joined routers are up-to-date on the view of the network. This process is performed only once per new router. To adapt to changes occurring *after* this initial synchronization the routers exchange LSAs on a regular basis.

4.2.5 Adjacencies - being adjacent

Two routers are said to be *adjacent* when they form a relationship for the purpose of exchanging routing information. They are described as adjacent as soon as the database synchronization process is initiated, since LSAs from this point on will be flooded along this link. The routers are said to be *fully* adjacent when their databases are synchronized. Only fully adjacent neighbors are included in LSAs. In networks where the DR scheme is utilized, the routers form an adjacency with the DR only. The number of adjacencies in such networks are thus kept at a minimum.

The number of adjacencies formed is usually a key parameter when measuring the impact of OSPF's overhead on different networks. This follows from the definition provided above that states that adjacencies are formed for routing information exchange. The number of adjacent neighbors is proportional to the amount of routing traffic needed to be sent on the network. The number of adjacent routers is a drawback in the point-to-multipoint interface type described below.

4.3 The Hello protocol

At a regular time interval a router announces its existence to other routers by emitting Hello packets. When a new router joins a network, it emits a Hello packet indicating its presence. Most likely it will not know of any neighbors at this point, and the list of neighbors carried in the Hello packet is empty. Neighboring routers receive this packet, and thus discover the new router. When the new router receives a packet from a neighbor who heard the router's Hello packet, the new router's ID is included in the packet. In this way the new router knows the neighbor has heard its Hello message, and creates an entry for the neighbor in its local neighbor data structure. As the router now knows that they can hear one another, the router marks the communication as being bi-directional. In the consecutive Hello packets from the new router, the neighbor's ID is included. Thus, this protocol establishes neighbor relationships.

As long as the router is running OSPF, it regularly emits Hello packets, to make sure its neighbors know it is still alive. The Hello protocol also elects a DR, if needed. RFC 2328 defines this emission interval to 10 seconds.

4.4 Neighbor states

When two routers become neighbors, they go through several states describing their relationship with one another. One such state is "TWOWAY" in which bi-directional connectivity is established. If the routers decide to become adjacent, they transition to (among others) state "EXCHANGE" in which initial database synchronization is started, and state "FULL" which indicates that the adjacency is brought up and the routers are fully synchronized. State "DOWN" indicates that the routers are not (longer) known.

4.5 OSPF packets

4.5.1 Packet types

Table 4.1 lists the packet types defined for OSPF. Put simply, these packet types can be divided into two categories; The Hello packet is used for acquiring and maintaining neighbor relationships, while the others serve the purpose of ensuring that the routers maintain a consistent view of the network. Hence, of the five packet types defined, four are used for maintaining the link state database.

Packet name	Protocol function
`Hello`	Discover/maintain neighbors
`Database Description`	Summarize database contents
`Link State Request`	Database download
`Link State Update`	Database update
`Link State Ack`	Flooding acknowledgment

Table 4.1: OSPF packet types

4.5.2 Bundling of routing messages

Each routing message that is sent on a link is carried on packets made up of the message itself and a necessary header. There is a coherence between message size and packet overhead; the smaller the message contained within a packet, the greater the amount of overhead associated with the transmission. Thus, each packet should contain as much necessary information as possible, since the cost of transmitting the packet over a link is inversely proportional with message size. This observation is taken into account when sending LS Update and LS Ack packets.

When transmitting an LSA to a neighbor the transmitting router awaits an acknowledgment for the LSA. If no such acknowledgment is received within a pre-defined time interval (RFC 2328 defined this interval to 5 seconds) the LSA is retransmitted. If an acknowledgment for the LSA is received within this interval then retransmission of the LSA is canceled. This feature may be exploited by routers having received an LSA: Instead of immediately responding with an acknowledgment message (carried in an LS Ack packet), it waits for a short period (less than the sending node's retransmission interval) before sending the LS Ack packet. This way several acknowledgment messages may be bundled into one LS Ack packet when additional LSAs are received within a certain time interval.

The same scheme is utilized for LSAs - a router may decide to wait a small amount of time before sending an LS Update packet. During this time new LSAs may be scheduled for transmission, and these are also included in the LS Update packet.

4.6 Network types

OSPF is designed to work over different network types. The most common network type on which OSPF is run is broadcast networks such as Ethernet. This network type assumes bidirectional connectivity between *all* participating routers on the link. OSPF routers in such a network implement OSPF's *broadcast* interface type, allowing the use of a DR.

Other network types supported by OSPF are *point-to-point*, *NBMA*, *virtual links*, and *point-to-multipoint*. The wireless extensions described in this thesis assumes the use of an interface describing the latter network type. The point-to-multipoint interface type is the most relevant with regards to the interface type defined for these extensions, and is examined in the next subsection.

4.6.1 Point-to-MultiPoint

In networks where a router only can reach a subset of the network's participating routers the *point-to-multipoint* interface may be the best choice. A router running a point-to-multipoint interface describes its neighbors (i.e. the link between the local router and the neighbor) as point-to-point links, e.g. defining link metric on a per-neighbor basis. A point-to-multipoint interface can be viewed as an interface maintaining a collection of point-to-point links.

No DR is elected in a point-to-multipoint network, since the DR mechanism assumes broadcast capabilities on the link. Instead all routers form adjacency with all of their neighbors.

Hellos and adjacencies

In point-to-multipoint networks the Hello protocol operates slightly different from networks such as broadcast and NBMA. The Hello packet is received only by the neighbors with which the sending neighbor can communicate directly. In special cases, such as in full meshed networks, this might include all routers on the network. The Hello protocol, then, operates as on a broadcast network.

The Hello packet is sent to each attached neighbor on a regular basis.

All routers connected on point-to-multipoint networks always become adjacent. This decision is made when bi-directional connectivity to a neighbor is established.

4.7 Timers

Consider the Hello protocol. An OSPF router periodically emits Hello packets to signal its ongoing presence. Moreover, when the router transmits an LSA, but fails to receive an acknowledgment within a predefined time interval, the LSA is retransmitted. These features depend on *timers*.

Timers are very essential in OSPF as they are set to trigger actions such as the ones just described. When a router emits a Hello packet, the routing daemon sets off a timer set to expire within a predefined period - the `HelloInterval`. In OSPF this interval is set to 10 seconds, which means that the router will emit a Hello packet every 10 seconds. Equivalent timers exist for retransmission of LSAs, to determine whether a neighbor has not been heard from in a long time (i.e. if it should be dropped), and so forth.

4.8 Scaling OSPF for large networks

Topology information is flooded throughout the network in LS Update packets. However, pure flooding will in most cases result in unnecessary retransmissions, and waste of network resources such as bandwidth. To limit the extents of flooding, several schemes have been designed. One such scheme is the DR mechanism described in the following subsection. Another scheme is the use of *areas*, outlined in subsection 4.8.2.

4.8.1 The Designated Router

OSPF defines several optimizations in order to minimize OSPF packets' overhead imposed on the network. One such optimization is the use of *designated routers (DRs)*.

A DR is merely a regular OSPF router elected to relay LSAs on behalf of the other routers in the network. In addition, the DR originates information on behalf of the network. Each router in the network synchronizes with the DR only. This scheme is designed as an optimization in broadcast networks, as having *one* router generating router information on behalf of the network, and at the same time limiting the number of database synchronization to $n - 1$, restricts the amount of routing overhead OSPF imposes on the network. Note that the DR scheme is designed for NBMA (nonbroadcast multiple access), and broadcast networks such as Local Area Networks (LANs) only.

As OSPF is typically deployed on LANs or other broadcast networks, the DR mechanism is one of the most known features of OSPF.

4.8.2 Areas

The idea behind areas is to divide OSPF networks into sub-domains in order to diminish flooding in the network, and limit the size of the routers' LS database. Only the routers within the same area knows the area's exact topology, while routers outside the area is only advertised a summary of the topology. Thus, when changes occur, flooding the local area may suffice, reducing the flooding overhead on the other parts of the routing domain. These areas are what constitute the AS. A special area referred to as the *backbone* must be present within an AS. Figure 4.1 depicts the relationship between the terms *backbone* and *area*; All areas must connect directly to the backbone. These areas (including the backbone) will typically reside one AS.

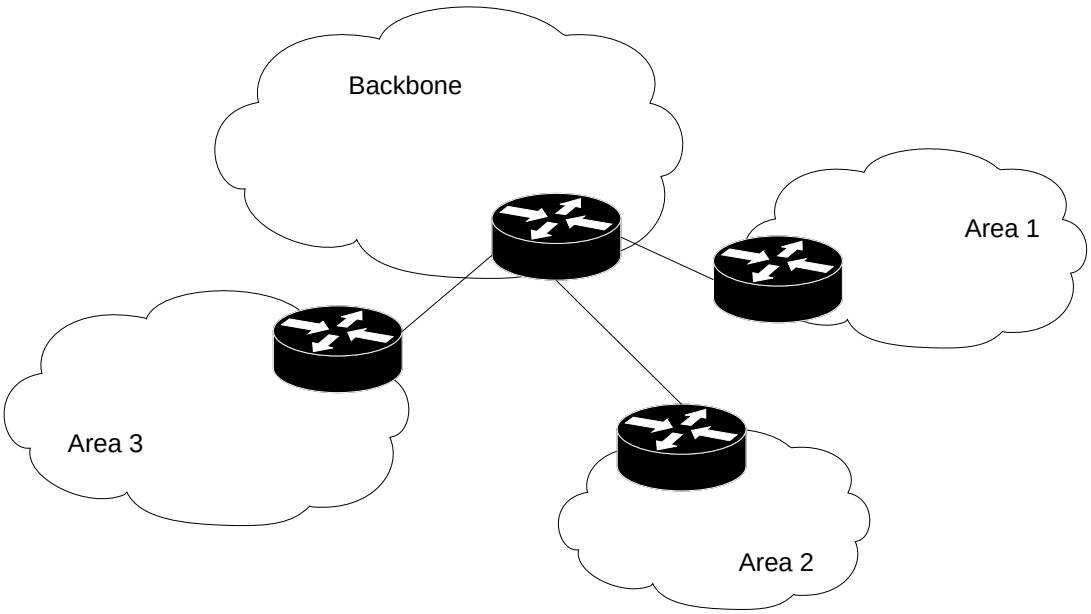


Figure 4.1: An example network partition

Within one area several different networks may exist. Consider Figure 4.2. Both networks reside in the same (sample) area. As the figure illustrates, an OSPF router may have interfaces to different networks. In the figure, router A is attached to two networks, which both reside in the same area.

4.9 Example OSPF operation

Figure 4.3 illustrates normal operation in a link state routing based network. Initially the network consist of a number of routers (and normal hosts) including router B. At this point router A is either not operable (for example it is not running OSPF on the interface, or the interface is down), or it does not have a link to router B. When the link between router A and router B is established, and both router A and router B is running an OSPF daemon, they commence an exchange OSPF packets.

First of all, the routers need to discover one another and establish connectivity. This is the responsibility of the Hello protocol.

Next, a decision is made whether the neighboring routers should form an adjacency or not. Assume router B is the DR of the network, or that they run a point-to-multipoint interface, and they hence decide to synchronize. This step of the operation is the *database exchange* process as described in section 4.2.3.

When the databases are synchronized the routers are noted as fully adjacent. At this point router A will start receiving LSAs from router B. These LSAs are both the LSA originated by router

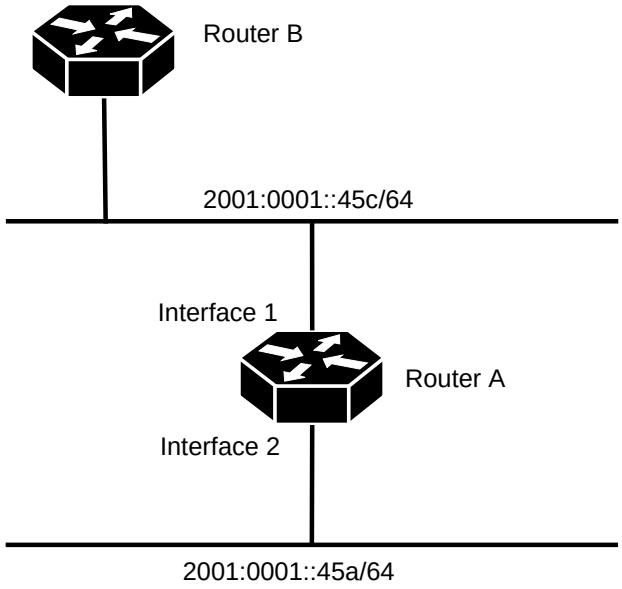


Figure 4.2: An example OSPFv3 network

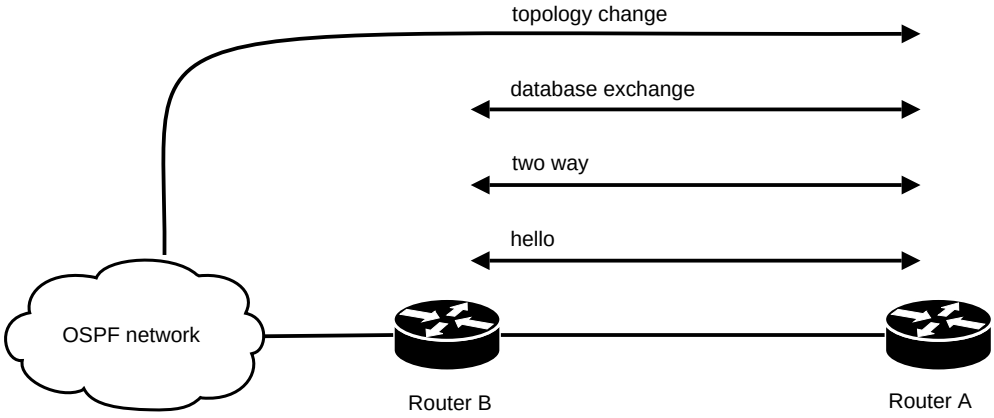


Figure 4.3: Normal operation of a link state protocol

B, as well as LSAs originated by the other routers in the network.

Chapter 5

Wireless Extensions to OSPF

This chapter describes the extensions to OSPFv3 needed to support mobile ad hoc networking. The extensions are based on the ideas proposed in Cisco's Internet-draft [5] proposing a MANET extension to OSPFv3. Throughout this thesis, the term "Cisco's draft" will refer to [5].

Throughout this thesis, the terms *WOSPF-OR* (derived from Wireless OSPF with Overlapping Relays) and *WOSPF-OR extensions* will refer to the extensions to OSPF for utilization on MANET interfaces. References to WOSPF-OR interfaces are equivalent to interfaces implementing the WOSPF-OR extensions. The term "OSPF-MANET" is a general reference to OSPF based MANETs, while a "WOSPF-OR network" is a OSPF-MANET based on WOSPF-OR interfaces. Recall the definition of the terms "OSPF" and "OSPFv3" (given in Chapter 4) as used in this thesis.

5.1 Overview

Deploying a legacy routing protocol defined for wired networks in an OSPF-MANET calls for modifications and optimizations. First of all, non-MANET routing protocols are not designed for operation in a multi-hop environment. Second, dissemination of routing packets in a network whose topology is (possibly) rapidly changing requires intelligent and optimized techniques, unless resource consuming, pure flooding is to be used.

OSPF-MANET interfaces should take into account the different aspects of (probably) resource constrained OSPF-MANET environments; the bandwidth may be scarce, topology is unpredictable, and link quality poor. The wireless extensions described in this chapter aim to define an interface that can cope with these properties.

5.1.1 Deployment issues

To maintain the distributed LS database in OSPF networks, the routers exchange neighborhood information on a regular time interval. In addition, Hello packets are emitted quite frequently in order to announce their (ongoing) presence. On wired networks the routing overhead is very small as the links usually have relatively high capacity, and the network topology remains more or less constant.

OSPF-MANETs, on the other hand, are typically subject to low capacity links, and resource constrained routers (for example PDAs) in terms of processing power and memory capacity. Not only does the periodic transmission of routing packets impose more resource overhead compared to the wired networks mentioned above, but also the unpredictable behavior of OSPF-MANETs implies higher routing packet emission frequency to converge.

The extensions discussed in this thesis aim to minimize the intra-area routing overhead for OSPFv3 in MANETs. A number of issues await further study, including deployment of OSPF's *areas* for scaling large OSPF-MANETs, and the use of OSPF-MANETs as transit networks.

5.1.2 Proposed extensions for OSPF-MANET interfaces

Flooding is perhaps the most dominating routing overhead factor in networks. Although OSPF *does* have some flooding restrictions such as the DR mechanism and a restriction on which interface an LSA is to be retransmitted on, operating in a MANET call for specialized optimizations.

The mechanisms implemented in this thesis can be roughly divided into three:

Signaling additional information The extra non-OSPF information needed to be exchanged on a link is signaled by the use of Link Local Signaling (LLS), which was first introduced in [42]. The basic idea is to piggyback information onto existing OSPF Hello packets, thus preserving the existing packet format. Section 5.2 outlines this mechanism.

Optimize flooding Section 5.3.1 describes an optimization to the flooding scheme similar to the OLSR's MPR mechanism adapted to a MANET environment. The idea here is to limit the number of retransmission needed to disseminate LSAs throughout the network. As OSPF relies on acknowledgments for reliable transmission, the amount of LS Ack packets in such networks consume scarce bandwidth. An optimization to the acknowledgment scheme is also proposed.

Reduce size of Hello packets A modification to the Hello protocol is proposed, where only changes in state are signaled. The aim here is to minimize the size of one of the most frequently transmitted packets in OSPF, the Hello packet. This mechanism is outlined in section 5.7.

Note that there is a coherence between some of the mechanisms described above.

5.2 Link Local Signaling (LLS)

To preserve the format of the existing OSPF packet format, the extra information needed to be exchanged on a link in a WOSPF-OR network is exchanged using *Link Local Signaling (LLS)*. The existing format is preserved but the OSPF packets are appended a special data block that only routers that support LLS will consider the content of. Since all additional information needed to be exchanged in MANETs is appended to already existing OSPF packets, the routers that do not support LLS can silently ignore the LLS. Thus, OSPF routers that do not support LLS can operate seamlessly with LLS capable OSPF routers. However, as only WOSPF-OR interfaces are to utilize the LLS mechanism, such interoperability is not considered further in this thesis.

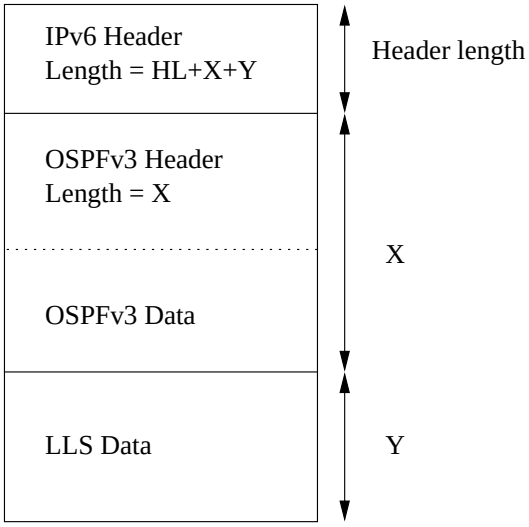


Figure 5.1: A Link Local Signaling data block attached to an OSPF packet. As this thesis implements extensions to OSPF *for IPv6*, the packet is encapsulated in an IPv6 header.

Figure 5.1 illustrates how the LLS data block is appended to an OSPF packet. Note that the OSPF packet length field denotes the length of the OSPF packet only. The length of the LLS data block is included in the IP packet length field.

5.2.1 The LLS data block

The LLS data block consists of a checksum, the length of the entire LLS packet, and the set of TLVs (described in the following subsection). Figure 5.2 depicts the LLS data block format.

All TLVs are carried in an LLS data block, which again is appended to Hello packets. The LLS data block may be appended to any OSPF packet type, but Cisco's draft defines TLVs as being transmitted with Hello packets only. Any reference to TLVs being emitted or included in a Hello packet must implicitly be interpreted as the TLV being included in a LLS data block which is appended to a Hello packet.

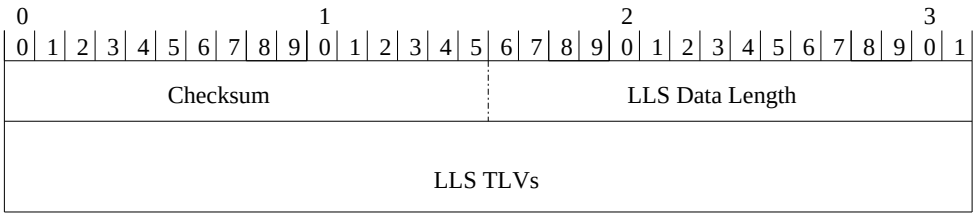


Figure 5.2: The format of an LLS data block.

5.2.2 Type/Length/Value triplets

The data in LLS data blocks are carried in Type/Length/Value (TLV) triplets. As the name suggests, the block consists of:

Type Several types of TLVs are proposed, each carrying differing types of information.

Length The Value field is in many TLV types variable in length, so the length field contains the length of the Value field.

Value The information being signaled in TLVs is carried in this field. As described in later sections of this chapter, such information may include a list of dropped neighbors, the willingness to utilize the extensions described in this thesis, and so forth.

An Extended Options (EO) TLV is also described, and may be used to signal link-specific OSPF capabilities. As no additional information needed to be signaled on a link, as described in Cisco's draft, is carried in this TLV type, it will not be further discussed in this thesis.

5.3 The WOSPF-OR interface

Currently, there are no interfaces defined that accurately describe the different characteristics of wireless networks. For example, wireless links may behave as both multi-access and point-to-multipoint links. The links are multi-access in that any transmission will reach the neighbors who are within transmission range, so that two simultaneous transmissions may collide, and as the transmission of a packet may reach only a subset of the routers in the network, the network may be modeled as point-to-multipoint. However, because of differences in for example radio signal strength, one cannot assume that two routers on the same link have bi-directional connectivity.

These issues indicate that an OSPF-MANET interface needs to be defined. Of the interfaces already defined for OSPF, the *point-to-multipoint* interface has been chosen to represent WOSPF-OR interfaces, and will have the following properties:

- All connections are considered point-to-point links.

- Each link metric is defined pr neighbor.
- As the wireless interface supports link layer broadcast, multicast is used for network layer routing packets (OSPFv3 packets).

Note that this thesis assumes that all wireless links are made up of WLAN interfaces, i.e. interfaces using the IEEE 802.11 family of standards. All packets transmitted on such an interface are received by *all* neighbors (the hidden node problem is disregarded here) within the interface's transmission range. Hence, as a single physical message is received by all attached neighbors the link layer is said to support link layer broadcast.

A node transmitting a packet can reach a (sub)set of the nodes in the network (hence the point-to-multipoint interface). To take advantage of this property OSPFv3 packets are, with few exceptions, transmitted to a well known multicast address so that all the receivers will process the packets.

5.3.1 WOSPF-OR interface operation

The WOSPF-OR interface is defined for operation in a MANET. Communication with non-MANET routers is done through normal interfaces such as Ethernet, ATM and so forth. Note that being merely a new interface type for OSPF, standard OSPF operations such as flooding LSAs are still carried out.

LSAs, while kept in a router's LS database, are removed from the database when their age exceeds a given threshold. For this reason an LSA is refreshed by having the originating router retransmit the LSA on a regular basis (usually every 30 minutes in wired OSPF networks). To lessen the routing overhead in WOSPF-OR networks LSAs are note aged, and (unnecessary) retransmission of LSAs is avoided. To indicate that a given LSA is not to be aged, the DC bit is set in the OSPFv3 option field and the DoNotAge bit is set in the LS Age field. See RFC 2740 for details on these bits.

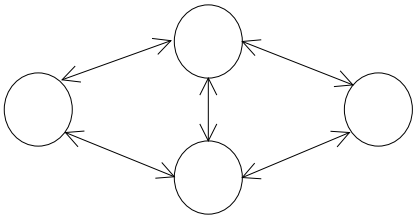
5.3.2 Using IPv6

As the extensions implemented in this thesis are defined for OSPFv3, a WOSPF-OR interface is configured with IPv6 addresses.

The use of IPv6 addresses works around some issues associated with connecting WOSPF-OR networks to the Internet. This is further examined in section 9.4.1 on page 113.

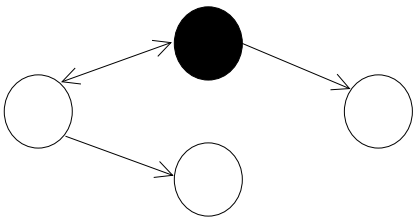
5.4 Optimized flooding

One of the most deployed flooding optimizations used in OSPF networks today, the DR mechanism, will not perform correctly in OSPF-MANETs. This is because OSPF-MANETs



(a) a

Figure 5.3: Pure flooding in a wireless network.



(a) b

Figure 5.4: Wireless network with an optimized flooding scheme.

are not true multi-access networks, as is a DR assumption; in OSPF-MANETs, two nodes on the same network segment cannot be assumed to have two-way connectivity. Therefore, we will adopt the ideas behind the OLSR optimized flooding scheme *MPR Relays*, and implement the *overlapping relays* mechanism.

In wired networks, a common optimization is to not relay packets on the interface through which the packet was received. This, of course, is not possible on wireless networks, as the wireless interface may be the only interface through which other wireless nodes may be reached. This is typically the case for OSPF-MANETs. In such networks, if using pure flooding, a node receives *one copy* of a packet *pr symmetric neighbor*. Figure 5.3 depicts a very simple scenario, consisting of only four nodes. It is assumed that the leftmost node is the originator of the packet, although the flooding would be exactly the same for either of the nodes.

As the figure shows, every node (except the packet originator) retransmits the packet. Hence, we have $n - 1$ retransmissions. This scheme is not optimal, as many duplicate packet received are the result of an unnecessary transmission. This scheme could benefit from some flooding modifications designed especially for OSPF-MANETs. A WOSPF-OR interface uses the *overlapping relays* mechanism for this purpose.

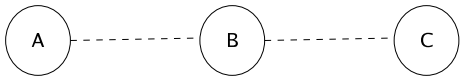


Figure 5.5: Non-overlapping VS two-hop.

5.4.1 Overlapping Relays

When a router receives a router LSA from an adjacent neighbor, the router compares the LSA's list of routers (i. e., the list of adjacent neighbors of the originator of the packet) to its local neighbor list. Neighbors known to the local router, but not known by the originator, are locally registered as non-overlapping neighbors of the originator. For the originator, however, this set is a set of two hop neighbors reachable through that neighbor but not reachable directly by the originator itself. Throughout this thesis, unless otherwise specified, the term "two hop neighbor" will be used on a two hop neighbor which is not also a one hop neighbor (a neighbor's neighbor may also be a one hop neighbor of the local router).

Figure 5.5 illustrates the difference between the terms non-overlapping and two-hop. Node C is a two-hop neighbor of node A, but for node B, C is a non-overlapping neighbor for node A.

When the local router receives an LSA from a neighbor, the LSA is relayed to the set of non-overlapping neighbors of the sender. However, as a two hop neighbor may be a neighbor of more than one one-hop neighbor, only one of the one-hop neighbors need to relay the LSA. The sender should select a set of one-hop neighbors to relay routing information *first* (this is further examined below). This set should be calculated such that every two-hop neighbor can be reached via at least one one-hop neighbor. Optimally, every two-hop neighbor should be covered by only *one* one-hop neighbor, but this has proved to be an NP-complete problem. In Cisco's draft, this set is known as the *Active Overlapping Relays (AOR)* set. A heuristic for calculating this set is based on the MPR selection algorithm described in [7], and will be examined in detail shortly.

Locally calculated sets

A node maintains different sets with regards to the overlapping relays mechanism. These sets include:

AOR set This is the set of neighbors the local router has elected to serve as an AOR.

AOR Selector set When the local node is signaled that it has been elected to serve as an AOR, the sender is included in this set. This set is then the set of neighbors for which the local router should act as an AOR for.

5.4.2 The Active Overlapping Relays set

All neighbors of a router are responsible for relaying LSAs. The AOR set is merely the subset of neighbors elected to retransmit the LSAs *first*. An AOR relays LSAs immediately upon reception, whereas a non-AOR (i.e., a neighbor not elected as an AOR) must retransmit the LSAs if the AORs fails to do so (due to, for example, poor link quality). In situations where the AOR set is not updated (the AOR set was calculated before link state changes occurred) a non-AOR will notice a two hop neighbor of the transmitter not receiving the LSA. It should then retransmit the LSA according to the algorithm outlined in the next subsection.

A router that signals that it most likely will not relay routing packets must not be chosen as an AOR unless that router is the only neighbor to reach two-hop neighbors. A router can also convey its willingness to serve as an AOR.

5.4.3 The AOR set selection algorithm

The overlapping relay selection algorithm is not concerned with *how* a neighbor is reached, but rather, *which* neighbor it can reach. In other words, the election of overlapping relays is not link-based, but node-based, and is hence done across WOSPF-OR interfaces.

The suggested algorithm for calculating the AOR set can be outlined as:

1. Add to the AOR set the routers that will always flood LSAs.
2. For all two hop neighbors reachable by only *one* one-hop neighbor, add the one-hop neighbor to the AOR set.
3. While there exists two-hop routers not yet covered by a node in the AOR set:
   (a) For each one-hop neighbor, calculate the number of two-hop neighbors it can reach, but that are not already covered by a router in the AOR set. This number indicates the neighbor's *reachability*.
   (b) Add to the AOR set, in decreasing order of willingness, one-hop neighbors that can reach at least one two-hop neighbor. If multiple neighbors have equal willingness, select the one with the highest reachability. If multiple neighbors have equal reachability, select the one with the highest number of two-hop neighbors. A final tie breaker is to select the neighbor with the highest router ID.
4. Optimize the AOR set by removing redundant entries, if any are found.

This algorithm is based on the MPR selection algorithm described in OLSR, with only minor variations. Note that a two-hop neighbor, as defined for the algorithm above, is a two-hop neighbor who is not also a one-hop neighbor.

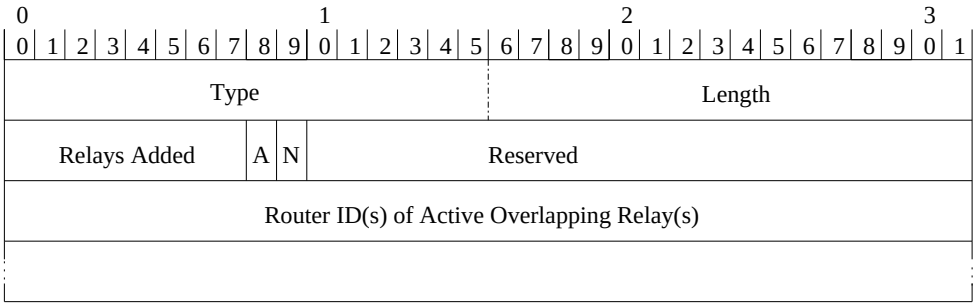


Figure 5.6: The format of the AOR-TLV data block.

Field	Description
`Type`	Set to 4
`Length`	Length of the value (i.e., length of the TLV *not* including the type and length fields)
`Relays Added`	The number of AOR listed in the TLV
`A bit`	Set if the node *always* will flood LSAs regardless of it being an AOR or not
`N bit`	Set if the node will most likely not relay LSAs
`Reserved`	Reserved for future use
`Router IDs`	Router IDs of the AORs elected by the local node. The first `length` router IDs are AORs, whereas the remaining list of router IDs are neighbors not longer in the AOR set.

Table 5.1: The fields in the AOR TLV

5.4.4 Signaling AOR election and willingness

To know that it has been elected to act as an AOR, a router must be signaled accordingly by the selector. The signaling is done by the use of LLSs. This also applies for signaling of willingness to relay LSAs (i.e. act as an AOR).

Figure 5.6 and table 5.1 illustrate and describe the TLV which is used for signaling the AOR set elected by a router, respectively. In addition, this TLV type allows for signaling the list of neighbors who are not longer elected to serve as an AOR for the sending router. This TLV type will be referred to as the AOR TLV.

A router may also signal its willingness to serve as an AOR. Based on, for example, link capacity or power status, the router may signal the willingness such that its neighbors can take this into account when performing AOR calculations. Nodes with low willingness are less likely to be elected as an AOR for a neighbor, so this way an extra metric is included in the flooding decision. This may be very advantageous, as this new metric describes factors known only locally to the node, and its neighbors' flooding decisions will consider this metric in their calculations. OSPF *does* have metrics associated with links, but these are not affected by the neighbor's processing

power, energy level and so forth. The Willingness TLV is detailed in figure 5.7 and table 5.2

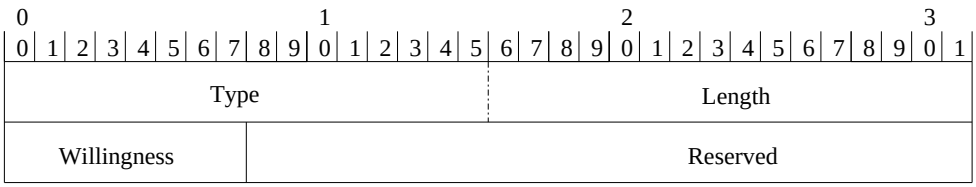


Figure 5.7: The format of the Willingness TLV data block.

Field	Description
`Type`	Set to 5
`Length`	4 bytes
`Willingness`	Willingness to serve as an AOR. The range of values is 0-255, where 128 is the default willingness
`Reserved`	Reserved for future use

Table 5.2: The fields in the AOR TLV

5.4.5 Flooding and Relay Decisions

As described in the previous subsections, a node makes a relay decision based on its role in the neighborhood. The decision is made according to the following algorithm:

1. If the router is elected as an AOR for the sending router the LSA is relayed immediately.
2. Else, the router is not an AOR for the sender, and should wait a specified time before deciding whether to re-flood. During this time:
   (a) If all of its neighbors receive the LSA, the retransmission is suppressed. Otherwise the LSA is transmitted on timer expiration.
   (b) If a neighbor which covers its non-overlapping neighbors re-floods the LSA, the router resets its timer. If all of its neighbors receive the LSA the retransmission is suppressed. Otherwise the LSA is transmitted on timer expiration.
   (c) Old LSAs should be ignored, and it should not respond to the sender with a unicast packet with the most recent LSA.
3. An LSA received unicast should not be re-flooded if the LSA has already been received from another neighbor. If the LSA has not already been received, the router should perform as if the LSA was received multicast.

To decide whether all neighbors have received the LSA or not, the router registers its neighbors' acknowledgment or retransmission of the LSA, which indicates that the LSA has been received and processed. If not all neighbors have received the LSA, it is retransmitted even though the router was not elected as an AOR in the first place. In effect, the non-AORs function as backup for the AORs in that they relay information in cases where the AORs fail. Also, the flooding of LSAs are ensured to be reliable, since each router is responsible for its neighbors' reception of the LSAs.

5.5 Intelligent Acknowledgments

One of OSPF's prerequisites is that every router in a flooding domain will receive all the routing information exchanged within the domain. To accomplish this, each router responds with an acknowledgment indicating the reception of the particular routing update. If a neighbor does not respond with an acknowledgment, the routing update is re-flooded until such an acknowledgment is received. These updates are signaled in LSAs, and when an LSA is received an LS Ack packet containing the LSA is sent back to the transmitting router. Note that many acknowledgments may be carried within one LS Ack packet.

5.5.1 Implicit acknowledgments

To utilize reliable flooding of LSAs the transmitting router's neighbors must signal reception of the LSA. Normally, as mentioned above, this is done by explicitly notifying the transmitting router by sending back an LS Ack packet. However, OSPF allows for a technique referred to as *implicit* signaling of LSA reception. Let us examine this technique by providing an example. Consider Figure 5.8.

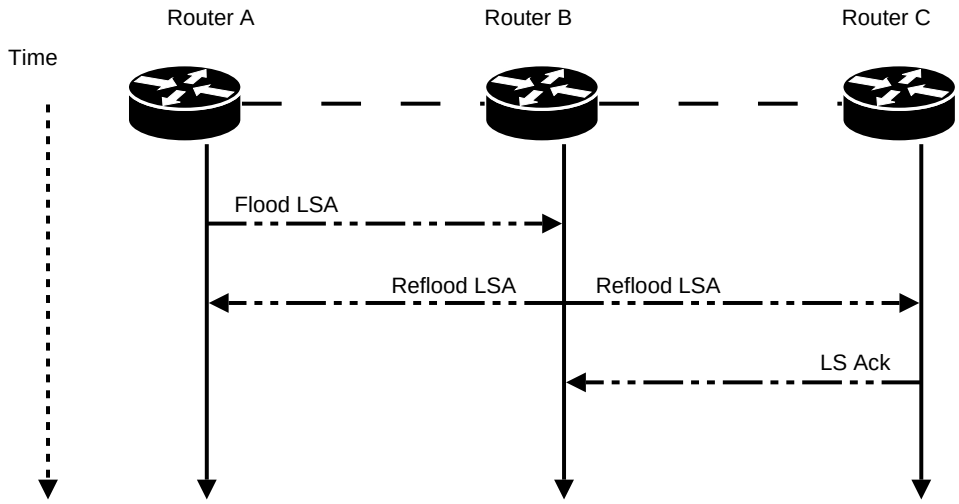


Figure 5.8: Utilization of implicit acknowledgments.

The figure shows three routers. As router B is the only router to reach router C, router A elects router B to act as an AOR. The same goes for router C. In this scenario router A floods an LSA. Since router B is an AOR for router A it immediately re-floods the LSA. This LSA re-flood is received by both router A and router C (bi-directional connectivity is assumed). Note that router A receives the same LSA itself flooded in the first place. Because it hears router B re-flooding the LSA it knows that router B has received it. This is the essence of implicit acknowledgments: A re-flood serves just the same purpose as sending LS Acks - both imply receiving the LSA in the first place.

The use of implicit acknowledgments clearly has benefits with regards to routing overhead, since LS Ack packets may be suppressed due to implicit acknowledgments. Considering, once again, figure 5.8 on the preceding page, we see that router B will never[1] have to send an LS Ack packet. Intuitively, for each LSA flooded by either router A or router C, *one* OSPFv3 packet is suppressed from transmission. However, as OSPF allows for *bundling* of messages, i.e. sending multiple LSAs or acknowledgments within one OSPF packet, the exact number of suppressed packets is not known.

5.5.2 Receiving duplicate LSAs

In WOSPF-OR networks nearly all acknowledgments are sent multicast. This feature takes advantage of the broadcast nature of MANETs as, due to the broadcast nature of wireless links, all neighbors within transmission range will hear the local router's acknowledgment of an LSA. These neighbors then know that the local router has received the LSA, and the local router will not be considered when deciding whether to retransmit the LSA.

As noted earlier, LSAs (carried in LS update packets) are sent multicast and are hence received by all one hop neighbors. These neighbors may decide to re-flood the LSAs, and a given neighbor will therefore typically receive multiple copies of the LSAs. Figure 5.9 illustrates one such scenario. Router B is elected to act as an AOR by router A. First, router A floods an LSA which is received by router B and router C. Router C responds with an acknowledgment, since it is not an AOR for router A. Next, being an AOR for router A, router B re-floods the LSA thus making router C receive two copies of the same LSA. Having already acknowledged the LSA, router C defers from taking any such action.

Note that router B must implement some kind of mechanism for registering router C's acknowledgment for an LSA itself has not (re)flooded yet.

5.5.3 Acknowledgments as "keep-alive" messages

An OSPF router emits Hello packets at regular time intervals to indicate its ongoing presence. If a neighbor is not heard from within `RouterDeadInterval` (defined as 40 seconds in RFC 2328) the neighbor is considered down. Every time a Hello packet is received from the neighbor, a timer, the *inactivity timer*, associated with the neighbor is reset to `RouterDeadInterval`.

[1] An exception is when LSAs are sent unicast, as is done for example during initial database synchronization

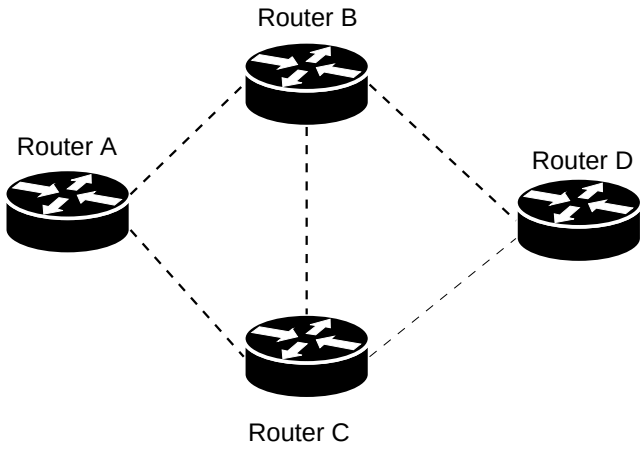


Figure 5.9: Acknowledging only the first LSA copy.

In other words, if the neighbor is not heard from within `RouterDeadInterval` the inactivity timer expires and necessary operations are performed.

Since all activity registered by the neighbor is an indication of its ongoing presence, WOSPF-OR interfaces allow for resetting of the inactivity timer whenever an LS Ack from the neighbor is heard. In addition, if no state change is to be signaled the local router may reset its *hello send timer*, i.e. wait additional `HelloInterval` seconds before sending the next Hello packet.

In effect, the "keep-alive" feature will make the acknowledgment serve as a Hello packet with no state change. These features have the advantage of reducing the routing overhead by reducing the number of packets transmitted on the network.

5.6 The incremental Hellos TLVs

To reduce routing overhead, only changes in state are listed in Hello packets. Instead of transmitting the complete list of neighbors, Hello packets only contain lists of new or dropped neighbors. Recall that a dropped neighbor is a neighbor no longer known, i.e. it has not been heard from within `RouterDeadInterval`. As a result, the size of Hello packets are reduced, thus decreasing the routing overhead imposed on the network. This mechanism is referred to as *incremental Hellos*, and is described in detail in the sections 5.6.1 through 5.6.1.

Although the reduction in packet size is relatively small, over time it will be significant, especially with regards to WOSPF-OR interfaces; For WOSPF-OR networks the `HelloInterval` should be smaller than on wired, relatively static networks, as changes in topology may be more rapid. Consequently, an WOSPF-OR interface will typically emit Hello packets at a much higher rate than in wired networks, but as the packet size is reduced the total amount of overhead is not increased linearly with the emission interval.

Four TLVs are defined to facilitate the operation of incremental Hello packets:

State Check Sequence TLV (SCS-TLV) The current state of the transmitting router is indicated in this TLV. Requesting full state from neighbors is also done using the SCS-TLV.

Neighbor Drop TLV The list of neighbors that are no longer known is contained in this TLV.

"Request From" TLV (RF-TLV) The transmitting router can request current state from a subset of its neighbors. These neighbors are listed in this TLV. Requesting full state from *all* neighbors is a special case and is further examined shortly.

"Full State For" TLV (FS-TLV) Full state for a subset of its neighbors can be requested by including them in this TLV. Similar to the RF-TLV, including full state for *all* neighbors is considered a special case, and is examined shortly.

The above described TLVs are transmitted using the LLS mechanism. Full state is defined as all the state changes occurring since the last SCS change. The SCS number is much like a sequence number indicating the current state of the transmitting router. Changes in state are indicated by increasing the local SCS number.

The term "Hello TLVs" will refer to the TLVs described in this section. An SCS-TLV with the R bit set is referred to as a Hello request.

5.6.1 Signaling state

Following is a more detailed overview of the TLVs associated with the incremental Hellos scheme.

The State Check Sequence (SCS) TLV

The SCS number is signaled to indicate freshness of state. Whenever a router's state changes, the SCS number is incremented, and full state is transmitted to its neighbors.

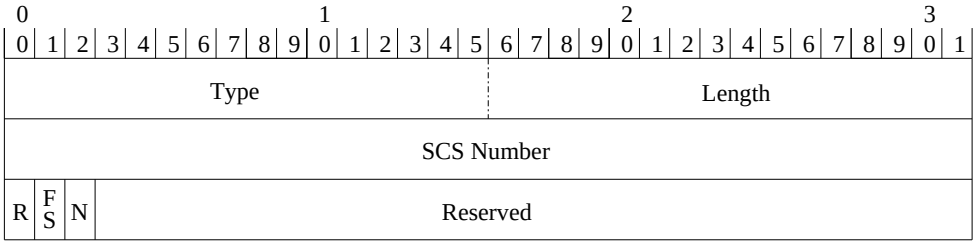


Figure 5.10: The format of the SCS-TLV data block.

For certain signaling operations, the SCS-TLV depends on other TLVs. This applies for situations where the local router either requests the current state from, or contains full state for, a subset of its neighbors. Let us examine this in detail.

Field	Description
`Type`	Set to 2
`Length`	Set to 4
`SCS Number`	A 16 bit unsigned integer indicates current state
`R bit`	Set if the router is requesting state from its neighbors. If state is requested from only a subset of the neighbors, the neighbors' IDs are listed in a RF-TLV
`FS bit`	Set if the Hello packet contains full state for its neighbors. If the full state is intended for only a subset of the neighbors, the neighbors are listed in the FS-TLV.
`N bit`	Set if there is more state associated with the SCS number than included in the Hello packet.
`Reserved`	Reserved for future use.

Table 5.3: The fields in the SCS-TLV.

If the local router emits a *Hello request*, an RF-TLV may be appended to signal which neighbors should respond to the request. If no RF-TLV is appended, each neighbor responds with its full state included.

If full state is to be sent to a subset of its neighbors, the local router will list these neighbors in an FS-TLV. If no such TLV is appended, *all* neighbors process the packet.

The Neighbor Drop TLV

As the incremental Hello packets are designed to include state changes *only*, a known neighbor may not be included in the Hello packet's neighbor list. In OSPF, this implies that a neighbor is no (longer) known. Contrary to OSPF, an empty neighbor list does not imply that no neighbors are known, so the list of dropped neighbors has to be explicitly signaled. However, as the incremental Hellos scheme is only utilized with adjacent neighbors, only such neighbors in state "EXCHANGE" or above are removed from the neighbor list reported in Hello packets. Therefore, when a neighbor transitions to a state less than "EXCHANGE" its router ID is added to the dropped neighbor list reported in the Neighbor Drop TLV. Note that neighbors who do not support the incremental Hellos mechanism are always reported in the neighbor list.

The packet format is illustrated in Figure 5.11, and table 5.4 describes the fields.

The "Request From" TLV

Whenever a router notices an error in a neighbor's state signaling, it emits an SCS-TLV with the R-bit set. (Recall that all TLVs are transmitted using the LLS method described in section 5.2). The neighbor's router ID is added to the RF-TLV, and transmitted along with the SCS-TLV.

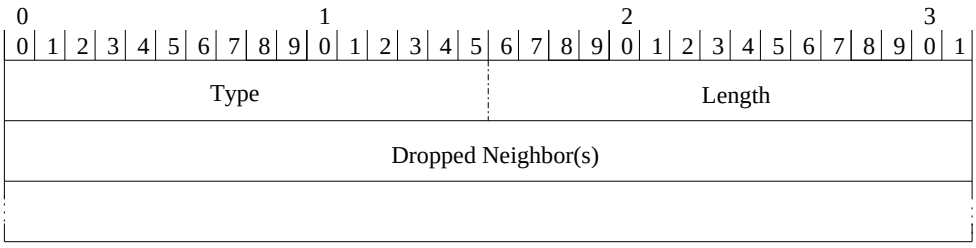


Figure 5.11: The format of the Neighbor Drop TLV.

Field	Description
`Type`	Set to 3
`Length`	The number of bytes used by the dropped neighbor list.
`Dropped Neighbor(s)`	The list of neighbors being dropped by the transmitter.

Table 5.4: The fields in the Neighbor Drop TLV

The router may wish to request full state from *all* neighbors, for example, when it first joins the network. Requesting full state from all neighbors is accomplished by setting the R-bit in the SCS-TLV, but not attaching the RF-TLV. Neighbors finding that the R-bit is set with either its router ID listed in the RF-TLV, or that the RF-TLV is not present, will respond with full state.

Figure 5.12 and table 5.5 describes the RF-TLV.

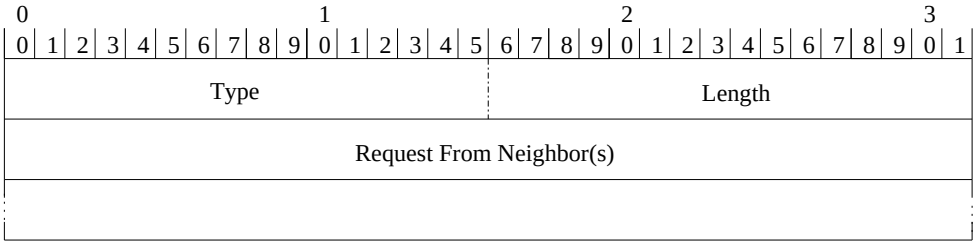


Figure 5.12: The format of the RF-TLV.

The “Full State For” TLV

A router receiving a Hello request responds with full state, and adds the requesting neighbor to the “full state for” list in the FS-TLV.

Figure 5.13 and table 5.6 describes the FS-TLV.

Field	Description
`Type`	Set to 4
`Length`	The number of bytes used by the “request from” list.
`Dropped Neighbor(s)`	The list of neighbors from which full state is requested.

Table 5.5: The fields in the RF-TLV

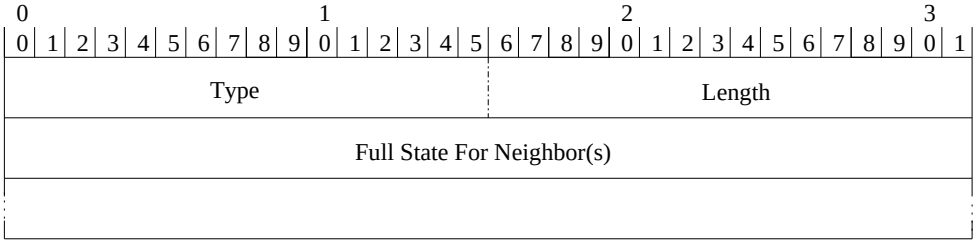


Figure 5.13: The format of the FS-TLV.

5.7 The incremental Hello protocol

The incremental Hellos mechanism aims at reducing the size of the packet type which may be the packet type being transmitted the most number of times during operation - the Hello packet.

Utilizing the incremental Hello protocol requires some modifications to OSPF’s Hello protocol. The next subsections examine these modifications in detail.

5.7.1 Neighbor Adjacencies

A neighbor who supports the incremental Hellos scheme signals this by setting setting the I bit in the packet’s option field. This enables the local router to exclude the neighbor’s router ID in its Hello packets, thus preserving network bandwidth by transmitting less bytes per Hello packet transmitted on the link. Upon first contact, however, the neighbor’s router ID must be included in order to establish bi-directional connectivity. When such connectivity is established and the routers start forming an adjacency, the neighbor’s router ID is excluded from the neighbor list reported in the Hello packet.

When an adjacency fails, the neighbor must be explicitly signaled as “dropped”. This is done by adding the neighbor’s router ID to a Neighbor Drop TLV which is transmitted with the next Hello.

Field	Description
`Type`	Set to 5
`Length`	The number of bytes used by the "full state for" list.
`Full State For Neighbor(s)`	The list of neighbors for which full state is included

Table 5.6: The fields in the FS-TLV

5.7.2 The State Check Sequence number

The current state of a router is represented by the State Check Sequence (SCS) number. When a change of state occurs, the router increments the SCS number. In essence, the SCS number functions as a measure of state freshness.

Whenever an adjacency is formed or broken, the SCS number used by the local router is incremented to signal state change. The next Hello will then include full state.

5.7.3 Sending Hellos

The modifications to OSPF's Hello protocol were described in the previous subsections. However, the incremental Hello scheme supports several additional interesting features which are briefly mentioned above, but fully examined in later chapters.

- A router joining a WOSPF-OR network must synchronize with its neighbors before it can utilize the incremental Hellos scheme. Recall that a router can request full state by transmitting a Hello request (i.e., an SCS-TLV with the R bit set), and that omitting the RF-TLV causes every neighbor to respond with full state. This feature can be exploited upon initialization, where the new router may choose to send such a request immediately. Alternatively, the router must initially utilize OSPF's Hello protocol.

- The incremental Hellos mechanism introduces flexibility regarding the amount of information sent in LLS data blocks. By default, when the SCS number is increased, full state associated with the SCS number is transmitted in the next Hello. As long as there are no changes in state, the TLVs are removed from the LLS data block. In other words, TLVs are only included once per state change. However, the incremental Hello mechanism allows for sending TLVs *persistently*; One can define TLVs always to be included in any number of consecutive packets. In wireless networks, where links typically have a high drop rate, this feature will increase reception of these TLVs. Recall that (incremental) Hello packets are not acknowledged by the receivers, and the transmitting router has no assurance of reception by its neighbors. Therefore, signaling information persistently will only *increase* probability of reception, not ensure it.

Note that the TLVs indicated in this section are Hello TLVs, i.e. TLVs used by the incremental Hello mechanism. However, the persistent signaling technique also applies to AOR TLVs.

The features just described will be subject to further study in the consequent chapters.

5.7.4 Receiving Hellos

As the SCS number carried in the SCS-TLV indicates the current state of the transmitter, several integrity checks must be performed to ensure that it correctly updates the transmitting router's state change information.

When a router receives an incremental Hello with a different SCS number than the last SCS number registered with that particular neighbor, the router must check its repositories for the validity of the Hello. The following lists the possible SCS number events:

- The SCS number is *lower* than the current number. Unless the SCS number is a wrap around, the router should require current state from the neighbor as it has received an old state change update.
- The SCS number is *equal* to the current SCS number registered with the neighbor. This will only occur if the transmitting router is signaling full state. As the local router already has received full state (the SCS number has already been received and processed), the packet should be ignored.
- The SCS number is *current SCS number + 1*. The increase in SCS number indicates a change of the neighbor's state. Changes other than the local router being signaled as dropped (in a Neighbor Drop TLV) will cause the local router to update the SCS number associated with that neighbor. A Hello without state change information will cause the local router to send a Hello request.
- The SCS number is at least the *current SCS number + 2*. The local router should send a Hello request, as it may be missing some neighbor state.

5.8 Communicating with OSPF routers

The WOSPF-OR interface type is defined for communication in WOSPF-OR networks, using the overlapping relays mechanism and optionally the incremental Hellos mechanism to lessen the WOSPF-OR network's routing overhead.

When speaking in an WOSPF-OR network, a router must do this by using an interface that utilize the WOSPF-OR extensions. When speaking in other non-WOSPF-OR (wired) OSPFv3 networks, however, the WOSPF-OR interface should *not* be used. Figure 5.14 on the next page depicts a scenario where a router implements two such interfaces. As seen in the figure, router A implements one interface (a WOSPF-OR interface) when communicating in a WOSPF-OR network, while communication with router B is utilized via another interface (for example

a point-to-point interface). Router A is thus a gateway between these two networks, and implements one[2] interface to each network.

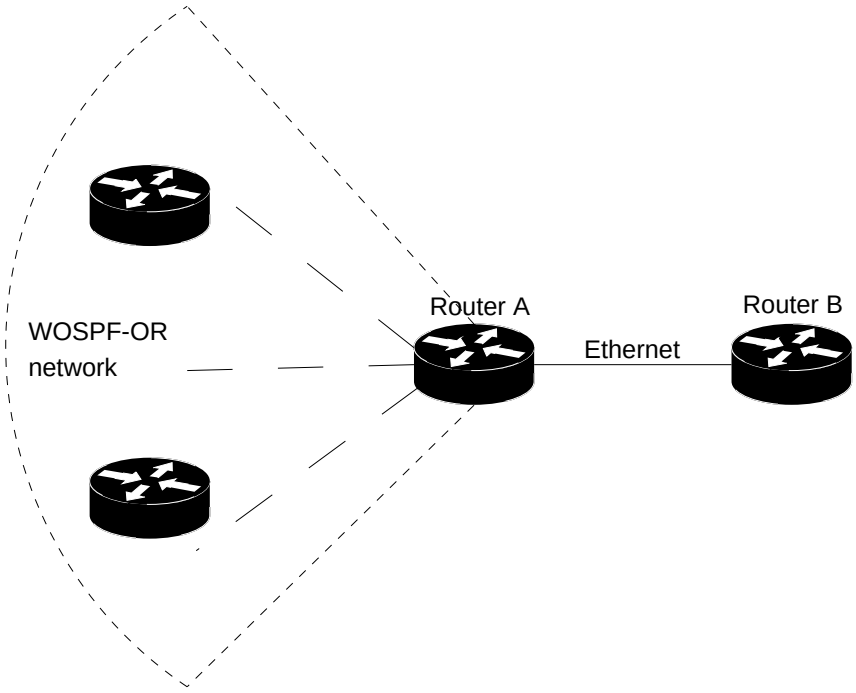


Figure 5.14: A gateway between a WOSPF-OR network and an OSPF network implements one interface to each of the networks.

A WOSPF-OR interface is based on OSPFv3's point-to-multipoint interface in which no designated routers are elected. A designated router should not be elected in a network where it cannot reach each participating router.

5.9 Related work and drafts

The last few years there has been a growing interest in deploying an already well known and widely deployed routing protocol like OSPF in MANETs. One of the first works on this matter was [3] which discussed several problem areas regarding these networks.

Since the winter of 2005 a working group, referred to as the *OSPF MANET design team*, within the IETF have been focusing on two active Internet drafts. One of these is Cisco's draft described in this thesis. The other (and competitive) draft is a draft published by Boeing [20]. The approach proposed in [20] is often referred to as OSPF MANET Designated Router (OSPF-MDR), while the approach proposed in Cisco's draft is referred to as Overlapping Relays.

[2]Router A could in principle implement several interfaces to each network, but this is not considered in this scenario.

The OSPF-MDR draft

The MDR approach aims at generalizing the designated router (DR) mechanism for use in OSPF-MANETs. Instead of electing *one* router to act on behalf on the other routers in the network, the MDR approach elects several DRs such that the MDRs form a source independent *Connected Dominating Set (CDS)* (i.e. all non-(B)MDR routers have at least *one* MDR router as a neighbor). An interesting feature of this approach is that the relay router set (the set of routers included in the CDS) is elected independent of the participating routers. A (Backup) MDR forms adjacency with only a subset of its neighbors, thus reducing the number of adjacencies formed.

Other drafts

[1] was published in May 2004 and proposes a wireless extension to OSPFv2. Similar to OLSR the routing information is disseminated in an unreliable fashion, and a subset of a router's neighbors are elected to relay such information. Similar to the Overlapping Relays scheme no DR is elected on wireless interfaces.

At the end of March 2006 [2] was published proposing a scheme inspired by the MPR functionality of OLSR whereby routing packets are re-flooded by MPRs. Adjacencies are formed only with MPRs or MPR Selectors, hence reducing the number of adjacencies formed.

5.9.1 Comparison

As Boeing's and Cisco's drafts are the two proposals that have incited the most interest and debate, these are discussed in this section.

OSPF-MDR and Overlapping Relays differ in the core approach for flooding: OSPF-MDR elects a source independent set[3] of routers that are to relay routing packets, while in the Overlapping Relays scheme this election is router specific (i.e. source dependent). Furthermore, they differ in the way information about a router's two hop neighborhood is retrieved - OSPF-MDR extends the OSPFv3 Hello packet to carry such information, while Overlapping Relays relies on router LSAs for this task. However, the drafts do have similarities in the use of reduced Hello packets size (differential/incremental Hellos) and backup relay sets.

Although no consensus on the OSPF-MDR and the Overlapping Relays scheme has been reached, as of this writing Boeing's proposal is much more discussed on the OSPF-MANET mailing list. As a result, or maybe because of this, Cisco's draft has expired but not been replaced by an updated version. The OSPF MANET design team *does*, however, show some promising results with regards to the OSPF-MDR approach as of April 2006.

[3]When used for communicating computer network properties, one can define a source independent set as a set calculated without regards to a specific router's view on the network. The set is thus equal for all routers in the network.

5.10 Discussions on the chosen draft

Cisco's draft was chosen because of its close connection with OLSR. As OLSR has had much success with its MPR scheme it was interesting to look into how this scheme would perform in an OSPFv3 based wireless network. It is also worth mentioning that the decision to implement the ideas proposed in Cisco's draft was made *before* any preliminary results from the OSPF MANET design team were published. Had the goal of this thesis been to implement the most promising and heavily debated proposal our choice *may* have been different.

5.10.1 The proposed extensions

The overlapping relays mechanism and the incremental Hellos mechanism proposed in Cisco's draft have very interesting properties with regards to minimizing routing overhead imposed on the network.

The basic idea behind optimizing flooding in a network is to perform flooding in such a way that every participating router received the routing update only *once*. The overlapping relays scheme aims to achieve this by reducing the relay router set to a bare minimum. As OSPFv3 relies on every router to receive each routing update the flooding is utilized in a reliable manner through the use of acknowledgments. The overlapping relays scheme is therefore compliant with RFC 2740 on this matter. A drawback is the number of adjacencies formed in such WOSPF-OR networks.

The incremental Hello protocol is similar to the Hello protocol defined in TBRPF (RFC 3684) [27] in that neighbor change is reported in a small number of consecutive packets. The incremental Hellos protocol, however, increment the sequence number associated with the state at most *once* per Hello interval. This may be an improvement over the TBRPF Hello protocol, where the sequence number is incremented with each Hello.

As information contained in a Hello packet (and related TLVs) is reported in a number of consecutive Hellos one can adjust this parameter based on link characteristics - if delivery rate is fairly high one can choose to lower the number of consecutive packets in which information is sent.

The use of implicit acknowledgments, as well as backup relays (i.e. non-AORs), are also interesting features proposed in Cisco's draft. These mechanisms ensure reliable dissemination of routing updates while reducing the number of retransmissions.

5.10.2 Issues with the proposed extensions

Although the extensions proposed in Cisco's draft do have very interesting and promising features, there are definitely some drawbacks not addressed in the draft. One issue is the number of adjacencies formed in WOSPF-OR networks; a router forms adjacencies with all its neighbors. Recall that forming and maintaining an adjacency imposes overhead during LS

database initial synchronization as well as maintenance by the use of flooding (LSAs are only flooded along adjacencies).

Another issues is the use of backup relays, the non-AORs. Backup relays are to re-flood routing updates whenever AORs of the transmitting router fail to do so. To reduce the possibility of two non-AORs re-flooding at the same time, *jitter* is used to create random pushback intervals. In very dense networks it is likely that a fraction of these will decide to re-flood, resulting in a flooding overhead not longer linear to the neighbor of routers in the network.

One issue that is briefly mentioned in the draft, however, is the amount of parallel link created when using multiple interfaces in a WOSPF-OR network. The draft does not indicate possible solutions to this problem other than stating that a mechanism for preventing such parallel links should be added.

5.10.3 Implementation details

Unfortunately, Cisco's draft does not go into much detail regarding an implementation of the extensions. For example, the flooding and relay procedure (see section 5.4.5 on page 40) specified in the draft only indicates that a mechanism for determining whether the re-flood of an LSA will result in a redundant retransmission, is needed. Specifically, the draft says that

> [...] a router can determine whether further flooding a LSA will only result in a redundant transmission by already having heard link state acknowledgments (ACKs) or floods for the LSA from all of its neighbors.

This high-level outline of required mechanisms is very vague with regard to an prospective implementation of the extensions.

5.10.4 Willingness granularity

Cisco's draft defines the willingness as a value between 0 and 255, occupying one byte of packet space. This defines a very fine granularity in willingness, as each router has a set of 255 values for which willingness to serve as an AOR can be signaled. In comparison, OLSR defines the willingness as a value between 0 and 7.

First of all, by indicating willingness with a very fine granularity such as 255, more bits than if the willingness were an integer between 0 and 7 are occupied in the willingness TLV.

Second, calculating the AOR set consumes more resources than for a lower granularity. For example, when optimizing the AOR set, every router in the AOR set is processed in increasing order of willingness. It goes without saying that processing 255 values of willingness burdens the CPU somewhat more than 7 such values. So in CPU sensitive (routers and) networks, decreasing the set of willingness values could have a positive effect.

5.10.5 Incremental Hellos emission frequency

The use of incremental Hellos reduces the size of Hello packets, thus imposes less routing overhead on the network. However, reducing the per packet overhead allows for sending the packets more frequently. This can be exploited to cope with one of the major issues in MANETs, namely router mobility; in a network with rapid change in topology it is crucial to react to topology changes faster than in network with less mobility. As Hello packets are emitted to discover link failure, amongst other things, one could use incremental Hellos as "keep-alive" messages sent at a higher rate than full state Hellos. The incremental Hellos could be viewed as link probes in that the packets are sent at a high rate in order to more quickly discover link failure. Full state Hello packets impose a somewhat greater amount of overhead on the network, and a router should therefore be restrained from emitting these packets more often than necessary.

Cisco's draft does not indicate any changes to the Hello emission interval.

5.10.6 Two hop neighbor sensing: Hellos vs LSAs

The Overlapping Relays mechanism relies on router LSAs to learn information on two hop neighbors. Other proposals such as that of Boeing's specify extensions to the Hello packet to include two hop information.

One of the major advantages of the WOSPF-OR extensions is that the original OSPF packets are not modified. All extra information needed to be exchanged on a link is included in LLS data blocks, making this protocol interoperate with OSPF routers seamlessly. However, Hello packets have the advantage of being designed for the possibility of being emitted on demand, as opposed to being pure timer based. Thus, when information changes, a router may choose to emit a Hello packet to signal this to its neighbors. The neighbors process this Hello packet and reset their timers associated with the sending router. Changes in the two hop neighborhood may thus be registered within a very short time interval.

Router LSAs, on the other hand, are emitted on a regular time interval as specified in RFC 2328. When a change in neighborhood occurs, the router must wait a specified time interval to signal this, increasing the convergence time of WOSPF-OR networks. Although router LSAs are, in fact, emitted whenever changes in a router's local neighborhood occur, RFC 2328 states that no LSAs can be updated within a predefined time interval since last update. This restriction has the consequence that the two hop neighborhood of the local router is updated with a minimum predefined interval. With regards to this, extending Hello packets for two hop neighbor sensing might be beneficial. It should be noted that time intervals defined in OSPF, as well as the WOSPF-OR extensions, are fully adjustable. Adjusting these parameters may overcome some of these shortcomings.

5.10.7 LLS and DD packet issues

Cisco's draft specifies that LLS data blocks may be attached to both Hello and DD packets. This makes sense, as both packet types are to be transmitted on a *per link* basis and are not to be re-transmitted on other links. Hence the name *Link Local* Signaling.

Attaching an LLS data block to a DD packet would have positive effects on the routing overhead *if* the information contained within the LLS data block were specific for that particular adjacency only. Recall that DD packets are used in the process of initially synchronizing databases during adjacency forming, and such packet are sent unicast and hence processed only by *one* router - the intended receiver. For example, Boeing's draft defines a TLV which is adjacency specific, and thus to be included in an LLS data block which again is to be appended to a DD packet. This TLV carries information about a router's election of which (Backup) MDR it is to synchronize with. This information is naturally adjacency specific.

In WOSPF-OR networks, on the other hand, no TLVs carry information relevant only to one specific adjacency. All TLVs related to the incremental Hellos mechanism, as well as all TLVs related to the overlapping relays mechanism, may contain information relevant to more then one neighbor. One example is the Willingness TLV, which include information of interest to all the neighbors (willingness to serve as an active overlapping relay is of interest to its neighbors, as the neighbors use this information in the AOR calculations). Another example is the Dropped Neighbors TLV, in which every neighbor should process the TLV in case it is itself listed.

5.10.8 Draft specification vagueness

The draft specification on incrementation of the local SCS number is somewhat vague. It states that whenever a Hello from a new neighbor is received, or an existing adjacency fails, "(...), increment the router's SCS number that it will transmit in its next Hello".

This, however, may be interpreted as *whenever a state change occurs, increment the SCS number*. Incrementing the SCS more than once per Hello emission interval is obviously not necessary, as the SCS number is defined to indicate the current state of the local router, *not* the number of state changes it has gone through. Therefore the specification should explicitly state that the SCS number is incremented at most *once* per Hello emission interval, regardless of the number of state changes it has detected.

In general, several aspects of the proposed extensions are outlined in a quite vague manner, but further investigation (studying the draft closely, reading up on the subject on IETF's OSPF MANET mailing list, and so forth) enlighten the subject.

5.10.9 Unnecessary AOR elections

For simplicity, consider a network consisting of two routers, router A and router B. A new router, router C, joins the network and starts synchronizing with the existing routers. However, as such synchronization is performed with one neighbor at the time, the following may happen: Router

C first synchronizes with router B. This causes B to originate a new router LSA in which both router C and router A are listed. As router A has only received a router LSA from router B, and AOR calculations are performed based on such LSAs, it will elect router B as an AOR (router B has a neighbor not yet reachable from router C). Until router C synchronizes with router A, it will continue to use router B as an AOR. Even if synchronization with A is performed within a short interval after election of router B as an AOR, router B will either be included in the AOR set or the dropped AOR set of router C.

5.10.10 Routing architectures

There are a number of ways of designing a MANET routing architecture. As routing, according to the well-known ISO stack, is performed at the network layer (layer 3), it is interesting to try and develop a routing protocol (or an interface type) designed for MANETs. This architecture assumes modifications to the (network layer) routing protocol only, and is the architecture behind the proposals in Cisco's draft.

There are other architectures that could be deployed, mainly focusing on interaction with the link layer. Consider the following architectures proposed in [39]:

- Modify the network layer to optimize performance in wireless networks. This is the architecture implemented in Cisco's draft.
- Modify the link layer to appear as a "full mesh" to the network layer. The routing protocol can then operate with a broadcast-based interface, similar to LANs.
- Integrate the network layer with the link layer to achieve the best of two worlds. Sharing information in this way may have positive effects on for example responsiveness (link changes are detected faster at the link layer than at the network layer, so the link layer could be used to trigger changes at the network layer) and neighbor discovery (this could be managed by the link layer only).

The architecture chosen in Cisco's draft is, as already mentioned, the former of the bullets above. However, all the architectures above have both pros and cons, and should be investigated further.

Chapter 6

The Quagga Routing Software Suite

The wireless extensions to OSPFv3 described in this thesis was implemented by modifying the OSPFv3 implementation provided by the Quagga routing software [25]. Quagga is a GPL licensed routing software suite providing implementations of routing protocols including Border Gateway Protocol (BGP), RIP, and OSPFv3. Quagga is actually a fork of the GNU Zebra routing software [41], focusing on building a more active development community than the centralized model of GNU Zebra.

This chapters gives an overview of the Quagga routing software suite.

6.1 Why Quagga?

While there exists several open source routing suites such as XORP [34], BIRD [33] and Quagga/GNU Zebra, the latter one, as far as the author knows, is the only framework that provides an implementation of OSPFv3. When deciding whether to use Quagga or GNU Zebra as a basis for our implementation Quagga was chosen because of its active development community. There seems to have been very little activity around GNU Zebra for the last couple of years - the latest release was in the end of 2003. Quagga, on the other hand, provides bug-fixes and general implementation updates on a regular basis.

Quagga's design provides:

Modularity Each protocol running on Quagga is a separate process. As a result, modifying a protocol can be done without affecting the other components of Quagga.

Reliability The module based approach works so that an event of failure in *one* deamon does not affect the other daemons running on the system. The system is thus more resistant to failure.

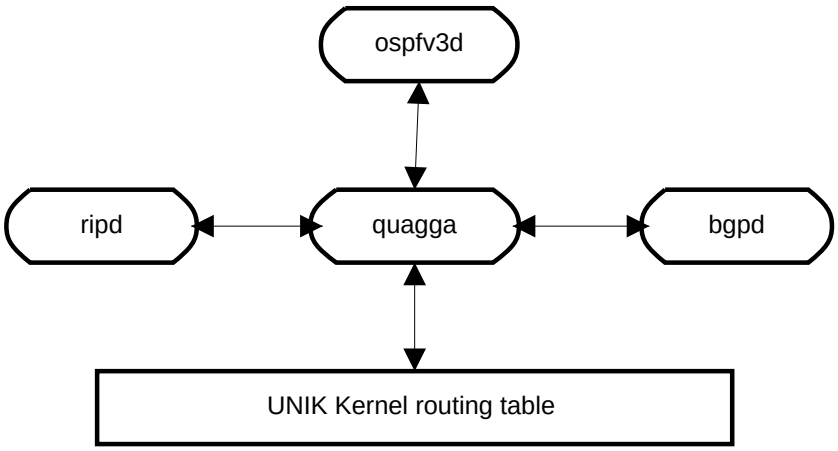


Figure 6.1: The Quagga System Architecture.

However, there are some drawbacks to the Quagga architecture. Running multiple processes calls for inter-process communication. Also, updating the kernel routing table is slow, as is redistribution of routes.

6.2 Software Architecture of Quagga

While traditional routing software is designed so that *one* process manages the entire routing functionality, Quagga separates the routing daemons into different processes. This way, if a routing deamon fails, others will not be affected. As a result reliability and modularity are achieved.

The different routing protocols are handled by routing daemons. For example, the OSPFv3 deamon (referred to as ospfv3d) handles the OSPFv3 routing protocol. A dedicated deamon is responsible for changing the kernel routing table, and distribute routing information between the routing daemons. This is the quagga deamon [1]. Basically, it is just a routing manager. Figure 6.1 depicts the Quagga system architecture. The figure illustrates the module based approach of Quagga.

As shown in the figure, the routing daemons are really just modules in the Quagga system. Not only does this make the system more robust (since upgrading and modification can be done separately), but adding a new routing protocol to the system does not require changing any of the existing software.

Quagga utilizes a strict layered architecture, as figure 6.1 shows. This complements the module based approach of Quagga; as the routing protocols are only concerned with interaction with the Quagga Manager, for example adding a routing protocol will only require knowledge about the communication interface between the Routing Protocol layer and the Quagga Manager layer.

[1] Actually, this daemon is still referenced to as the zebra daemon, but this is disregarded in this thesis

6.2.1 Threads in Quagga

Quagga implements a user-space thread mechanism for performing most of its operation. Such threads are used for scheduling function execution and timer expiration, and must not be mistaken for being threads in such as supported in operating systems. Quagga's threads are managed by the daemons themselves, and may be manipulated (i.e. re-scheduled, canceled) and originated through simple function calls.

The following code excerpt is taken from initialization function in the OSPFv3 daemon:

```
/* Start finite state machine, here we go! */
while (thread_fetch (master, &thread))
  thread_call (&thread);
```

After the necessary data structures have been initialized, the above code is executed. This piece of code is run until the daemon is to abort, and new threads will be added to a this list of user-space threads.

6.2.2 Components of Quagga

Figure 6.2 illustrates how the different components of zebra interact. An interesting observation is that all the components make use of the library functions defined for Quagga. These libraries include list structures, checksum calculations, customized memory allocation and so forth. All of these are Quagga specific.

Moreover, note that none of the actual routing daemons interact directly with either the vtysh or the operating system. This enables the routing daemons to do what they are designed to do, namely *route*. The human-to-computer interaction is handled by the quagga daemon. With regards to the operating system on which the routing daemons are run, porting the software over to another operating system is a matter of adapting only *one* component of the Quagga software suite to interact with a new operating system.

6.3 Virtual Terminal Interface

A user may wish to communicate with a deamon in order to change the daemon's current configuration or view its internal data structures. For this task, all daemons listen for incoming command line interface *VTY* (Virtual Terminal Interface) on a particular port. Communication with the daemon is done via the VTY shell which is an integrated shell for Quagga routing software. This shell is referred to as `vtysh`.

The `vtysh` is an excellent tool for debugging of a routing protocol. Below an excerpt of the ospfv3d testbed used in this thesis is provided. The command line prompt is the `vtysh`, and the command entered is one of many possible commands accepted by the ospfv3d.

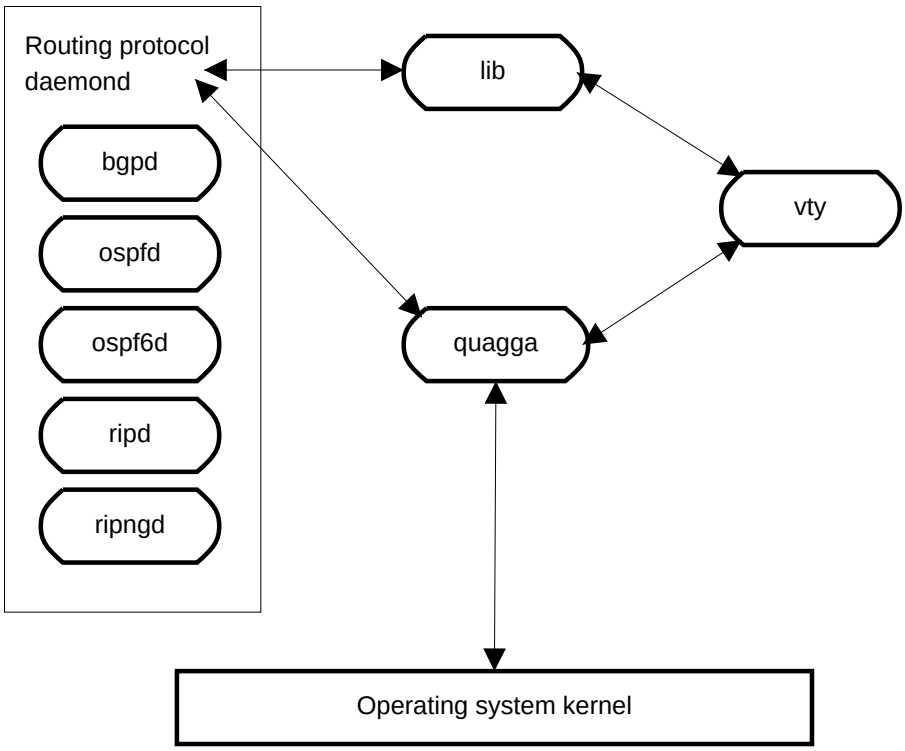


Figure 6.2: Interaction between the components of Quagga.

```
ospf6d@plant# show ipv6 ospf6 spf tree
+-3.3.3.3 [0]
   +-1.1.1.1 [1]
   +-2.2.2.2 [1]
```

The output shows the shortest path first calculation done by the router “3.3.3.3” (or more accurately, the router having “3.3.3.3” as its router ID). This command is clearly related to the debugging of (in this case) ospfv3d.

As mentioned above, the `vtysh` may also be used for changing the router’s current configuration. Consider the following command:

```
ospf6d@plant(config-if)# ipv6 ospf6 cost 9
```

This command sets the cost associated with the interface to nine. When the router later on updates its neighbors on the link cost, they will register node “3.3.3.3”’s cost on this interface as being changed to nine and perform necessary operations. This way of performing dynamic updates of the router’s configuration has the advantage that the router does not have to be restarted whenever changes are registered.

To enable the `ospfv3d vtysh` the users log onto the VTY process by issuing the command `telnet localhost 2606`. Note that “2606” denotes the port on which the ospfv3d vtysh runs, and that every routing daemon is assigned an unique port.

6.4 Configuration files

One interesting feature of Quagga is that commands given when configuring a daemon may be passed via both the vtysh and the configuration file. Consider the link cost command given in the previous section. In this case the command was given by entering it in the vtysh. However, this could just as well have been entered in the configuration file for ospfv3d like this:

```
interface eth0
  ipv6 ospf6 cost 1
  <other options for this interface>
```

Note that the link cost command is given within an interface “structure”, indicating that this commands applies to the noted interface (interface “eth0”). This is quite alike the `vtysh` way of passing commands, but in a vtysh one must navigate (much like navigating in a file system) to the appropriate interface before issuing the command.

This feature is quite neat in that configuring a daemon uses the same programming interface (i.e. makes a call to the same function) regardless of whether the command was given in a configuration file read at startup or whether the command is given during operation.

6.4.1 Defining commands

Defining commands for a daemon is done by defining the command name, description, code to execute, and so forth. This is implemented in the source files of the daemon. As an example, consider the following code excerpt:

```
DEFUN (ipv6_ospf6_cost,
       ipv6_ospf6_cost_cmd,
       "ipv6 ospf6 cost <1-65535>",
       IP6_STR
       OSPF6_STR
       "Interface cost\n"
       "Outgoing metric of this interface\n"
       )
{

<code to execute>

}
```

This code excerpt allows for changing/setting the metric associated with a given interface. The metrics command in the configuration file excerpt above actually executes this code excerpt. Functions defined in this manner may be executed either through definitions in the daemons configuration file, or thorough command entered in the vtysh.

Chapter 7

Implementation Design

Implementing extensions to a relatively complex routing protocol such as OSPFv3 should be done with care - modifying one part of the protocol may have an unexpected impact on seemingly separate parts of the routing system. However, one should minimize modifications to the protocol's source code, yet maximize performance. The implementation design of the proposed extensions to OSPFv3 has tried to balance these two objectives.

This chapter describes the implementation design of the extensions outlined in chapter 5.

Throughout this chapter, the term *module* will refer to a function or collection of functions that handles a special task (not unlike classes in object oriented languages).

Any reference to neighboring routers are implicitly neighbors on WOSPF-OR interfaces, unless otherwise specified.

7.1 Overview

The wireless extensions to OSPFv3 are defined for utilization on OSPF-MANET interfaces. Communication in such networks should rely on the flooding optimization technique provided by the overlapping relays mechanism, and preferably also the incremental Hellos mechanism.

As the overlapping relays mechanism relies on a somewhat different flooding scheme than defined for OSPF, modifications to the existing flooding procedure are implemented. These modifications are utilized only when the outgoing interface (i.e. the network onto which the LSAs are to be flooded) is a WOSPF-OR interface. For non-WOSPF-OR interfaces there is no change with regards to the existing flooding schemes.

Implementation of the incremental Hellos mechanism has lead to only minor modifications to the existing Hello protocol, as most of the extension code is implemented separately than from the existing code.

Use of the extensions

Cisco's draft states that it is possible to use the two optimization mechanisms proposed in the draft, the overlapping relays mechanism and the incremental Hellos mechanism, independently of one another. This makes sense, as the former optimizes the flooding of routing updates, while the latter minimizes the size of neighbor maintenance packets.

This thesis has implemented incremental Hellos as an optional feature, and *may* be used on WOSPF-OR interfaces. Overlapping relays, on the other hand, is a required feature on WOSPF-OR interfaces. This follows from the fact that OSPF does not support the multi-hop nature of WOSPF-OR networks, so in order to make sure LSAs are disseminated throughout the network the re-flood operation of overlapping relays is needed. Naturally, one could extend OSPF-routers in WOSPF-OR networks to always relay all LSAs (i.e. disseminate through pure flooding), but this is far from desirable. Thus, all WOSPF-OR interfaces *must* utilize the overlapping relays mechanism, while the incremental Hellos scheme is optional. Note that interfaces to WOSPF-OR networks must be explicitly specified in the configuration file, and this will make the interface utilize the overlapping relays mechanism.

7.1.1 Information repositories

The wireless extensions rely on a number of repositories maintaining information related to AOR elections, neighbors who have requested full state, and so forth. These sets are maintained with regards to WOSPF-OR interfaces.

The following repositories are maintained with regards to WOSPF-OR environments and are constructed as sets, described below.

Neighbor set The set containing all known one-hop neighbors. Note that this set is *not* equal to the set of neighbors associated with an interface, which is already implemented in OSPFv3. This set contains entries for all neighbors attached to WOSPF-OR interfaces.

Two-hop Neighbor set The set containing all registered two-hop neighbors.

AOR set Each node must calculate a set of one-hop neighbors such that every two-hop neighbor is a neighbor of at least one one-hop neighbor.

AOR Selector set Nodes must know which neighbors they are to relay traffic from, that is, neighbors that have chosen the node as an AOR.

Dropped AOR set AOR calculations may result in changes in the AOR set. The former AORs are added to this set

Dropped Neighbor set Neighbors who are transitioned to state "down" are added to this set.

When LLS data blocks are constructed, TLVs are constructed based on the information maintained in these sets. In addition, there are several sets used for mechanisms such as the Hello request scheme and the *full state for* scheme:

Full state requestor set Neighbors who have requested full state are added to this set.

Full state for set The set of neighbors for which full state is intended.

The sets described above are implemented as double-linked lists[1].

7.1.2 Implementation considerations

Although the wireless extensions to OSPF described in this thesis are based on Cisco's draft, some of the implementation design details are inspired by those described in another Internet-Draft proposing wireless extensions to OSPF, namely Boeing's draft. The latter draft was described in 5.9. The WOSPF-OR extensions proposed in those Internet-Drafts are different in many aspects, especially with regard to whether the set of LSA relay nodes are calculated in a source dependent or source independent manner.

In spite of the differences, similarities are also present. A few of these are:

- Both proposals rely on flooding through use of active and non-active relays. A non-active relay node is a node required to relay information whenever an active relay node fails to do so.
- Flooding is done in a reliable manner, by requiring each node to acknowledge reception of the routing information. The acknowledgment may be either explicit of implicit.
- Most packets are sent to a well known multicast address, thus exploiting the broadcast nature of MANETs. To avoid the *broadcast storm problem* [30], an LSA is acknowledged only once.

Because of these similarities, and because Cisco's draft provides a relatively small amount of detail regarding implementation of the extensions, in as many cases as possible such details are retrieved from Boeing's draft. As the two proposals are different in nature, the implementation details provided in this draft have been modified for use in this thesis.

7.2 Configuring WOSPF-OR parameters

Various WOSPF-OR parameters may be configured to adapt to the different characteristics of wireless networks. The configuration file must explicitly define which interfaces are to utilize the optimization mechanisms described in this thesis, and may optionally include definitions used for tuning of the protocol performance.

The configuration file is named `ospf6d.conf` and is usually located in the folder `/usr/local/etc/` on the host system. This file also allows the router administrator to tune

[1] For esthetic reasons, the illustration of such lists in this chapter make use of single-arrow representation.

most of the parameters defined for OSPFv3. Such parameters include `HelloInterval`, the cost of transmitting a packet on a particular interface, and to which area an interface is to connect to.

7.2.1 Time Intervals

Two timer intervals should be defined in the configuration file, and these are PushbackInterval and AckInterval. If *not* defined, these parameters are calculated based on the LSA retransmission interval defined in RFC 2328. The default values are calculated as follows:

PushbackInterval

A non-AOR node should wait `PushbackInterval` time before further flooding of an LSA. The `PushbackInterval` is denoted in seconds, and should be so that

```
(P_Int + prop_delays) < (RxmtInt / 2)
```

where P_Int is the PushbackInterval, prop_delay is the propagation delays of the link, and RxmtInt is the RxmtInterval as described in RFC 2328.

AckInterval

To reduce the number of transmissions, several LSA acknowledgments may bundled into *one* LS Ack packet. This is done by delaying the transmission of an LS Ack packet. The AckInterval is denoted in seconds, and should be so that

```
(A_Int + prop_delays) < P_Int
```

where A_Int is the AckInterval, , prop_delay is the propagation delays of the link, and P_Int is the PushbackInterval.

7.2.2 Persistent signaling

The use of incremental Hellos allows for sending select information persistently, i.e. sending the same information in a number of consecutive incremental Hello packets (recall that such information implicitly is carried in LLS blocks). For example, one may wish to signal the list of dropped neighbors in `k` consecutive Hello packets (or more precisely: in a Neighbor Drop TLV). This is done by including the newly constructed Neighbor Drop TLV in `k` consecutive packets.

The number of consecutive Hello packets in which such information should be included may be manually configured. Any given TLV type may be signaled in consecutive Hello packets, and the number of consecutive packets it is to be included in is defined in the configuration file. The advantage of this feature is that the router administrator (i.e. the user) or the router itself may choose to always send certain (or all) TLVs in `k` consecutive packets based on, for example, link quality calculations. If the links are considered stable the overhead associated with sending *all* TLVs in several consecutive Hello packets may not be worth the increased probability of reception by its neighbors. Consider the following persistent signaling setup:

```
#define WOSPF_DROPPED_NEIGHBOR_PERS 3
#define WOSPF_WILL_PERS 1
#define WOSPF_AOR_PERS 2
#define WOSPF_DROPPED_AOR_PERS 1
```

This indicates that the list of dropped (i.e. not heard from in `RouterDeadInterval`) neighbors is to be included in the next three Hello packets. As TLVs signaling neighborhood information is of high interest to the neighbors, these are sent in three consecutive packets. However, willingness and AOR election may be of lesser importance to spread to the neighbors, as WOSPF-OR implements functionality ensuring dissemination even in cases of outdated AOR Selector sets by the neighbors (recall that non-AOR are to re-flood information if needed).

When signaling information persistently it is important to avoid sending outdated information. In this implementation, information which are to be sent persistently is added to a list, and deleted from this list when it has been sent `k` times. The information may, however, be outdated even before it is sent in the first place. When a dropped neighbor re-establish contact within a short time interval the neighbor relationship should obviously not be signaled as dropped in the consecutive Neighbor Drop TLVs. Care must be taken to remove outdated information from such lists. This implementation removes all *local neighborhood* information from the corresponding persistent lists whenever a change in the local neighborhood occurs. When a change in the two hop neighborhood of the local node is detected the persistent lists such as *new_aors* and *dropped_aors* (these lists are examined in later subsections) are cleared.

TLVs for which the persistent count is *not* defined in the configuration file are sent only once, i.e. the persistent count defaults to 1.

7.2.3 Setting thresholds for omitting TLVs

When a router wish to request full state from all of its neighbors, it may choose to set the R bit in the SCS-TLV, but omit the RF-TLV. This will cause all of its neighbors to respond with full state. The same applies to FS-TLVs.

Consider the following definition:

```
#define DROP_REQ_TLV_THRESHOLD 0.5
```

This statement indicates that in cases where the router are requesting full state from at least 50 percent of its neighbors, the RF-TLV is omitted. This feature allows for tuning protocol performance according to characteristics such as network density or the cost of transmitting additional bytes on the link. A similar definition exists for FS-TLVs. These definitions reside in the file `wospf_protocol.h`.

7.2.4 Defining WOSPF-OR Interfaces

The configuration file allows the user to define which interfaces are to utilize the WOSPF-OR extensions. For example, the gateway between a WOSPF-OR network and a wired OSPF network should be configured to run *one* routing daemon, utilizing one interface type to each of the networks. The following example shows how the configuration file could be defined to enable a OSPF/WOSPF-OR gateway as described above.

```
router ospf6
  <misc config>

  interface eth0 area 1.1.1.1
  interface eth0 is_wireless
  interface eth0 enable incr_hellos

  interface eth1 area 0.0.0.0
```

A network interface card attached to a wireless network (WOSPF-OR network) must define the `is_wireless` option. This will cause the node to utilize the overlapping relays mechanism. The incremental Hellos mechanism, on the other hand, is optional, as this feature is defined only to further reduce overhead associated with frequently transmitted packets. Utilizing the incremental Hellos mechanism on a wireless interface is not required, as the Hello protocol defined in RFC 2328 is fully supported in WOSPF-OR networks.

Recall that these mechanisms may be used independent of one another, and if neither are defined the interface operates according to OSPFv3.

7.3 Link Local Signaling

As LLS data blocks may be appended to Hello packets, such packets are intercepted by a WOSPF-OR function responsible for processing the attached LLS data block. The presence of the L bit in the OSPFv3 packet's option field indicates that an LLS block is appended, and the LLS block is sent to the LLS processing function for analysis. The function is implemented in `wospf_intercept.c`:

```
int
wospf_process_message(struct ospf6_header *, int, char *, struct
                      ospf6_interface *);
```

The function loops through the existing TLVs, making function calls to process the different TLV types. Processing of the TLVs is more or less straight forward, except for the SCS-TLV; Processing and interpreting the information contained in the SCS-TLV is rather tricky, as this information is relative to information contained in the other TLVs. For example, if the R bit is set in the SCS-TLV's bit field, the local router's action is dependent of whether a "Req FS from"-TLV is present of not. This matter is further examined in later sections of this chapter.

All code associated with LLS resides in *one* file, `wospf_intercept.c`. Hello packets are intercepted when they are received as well as when they are to be transmitted.

7.3.1 Attaching LLS data blocks

Outgoing Hello packets are intercepted by the function below and (possibly) appended an LLS data block.

```
int
wospf_append_lls(struct ospf6_header *, struct ospf6_interface *);
```

The above function iterates the lists associated with state change and AOR selection, and for each non-empty list the associated TLV is constructed and appended. Note that several lists may be associate with *one* TLV type (for example, the AOR-TLV retrieves information from both an *added AOR list* and a *dropped AOR list*).

7.3.2 Piggybacking LLS blocks

Network programming using a relatively low-level programming language as such C calls for some considerations related to the possible difference in architecture on the receiving machine. For example, as different machine architecture may use different byte ordering care must be taken to ensure that packets sent and received on the link are read correctly. Commands such as `htons` and `ntohs` are useful in these circumstances.

Fortunately, the Quagga framework has already implemented many of the mechanisms necessary needed for this task. Moreover, as the LLS blocks are to be piggybacked on existing OSPF packets, the implementation of the extensions described in this thesis does not interfere with the sending and receiving of packets at all. More precisely: When a Hello packet is to be sent on the interface it is put in a buffer, and then the buffer is flushed out on the interface. The WOSPF-OR implementation exploits this by simply extending the buffer and adding the LLS block after the OSPF packets. Naturally, at the receiving end of the link a buffer holds the received information in a similar fashion to the sending buffer.

7.3.3 Signaling information persistently

As information carried in the TLVs may be sent persistently, the information is associated with a counter. This is best explained by providing an example:

Suppose that the local router has dropped neighbor A. The local router wish to signal this is 3 consecutive packets. Obviously, one needs some kind of counter to keep track of in how many more packets this information (i.e. dropping this particular neighbor) is to be included. Associating such a counter with the information is accomplished by including these information bits in a data structure:

```
struct wospf_neighbor_node
{
  char *name;
  u_int32_t router_id;
  int persistent_count;

  struct wospf_neighbor *w_neighbor;
};
```

It is instances of this data structure that are added to a global list named `dropped_neighbors_list`. When initialized, the `persistent_count` is, in this case, set to 3. Each time the `router_id` is included in a Neighbor Drop TLV the `persistent_count` is decremented, and upon reaching 0 the data structure is removed from the `dropped_neighbors_list`. The technique ensures that each dropped neighbor's router ID is added to 3 consecutive packets.

Similar data structures and signaling schemes exist for AOR and willingness signaling. This implementation has chosen *not* to send request for full state, as well as full state intended for, persistently. See section 7.9.3 for further details.

7.4 The WOSPF-OR interface

The point-to-multipoint interface type described in RFC 2740 was not implemented in the OSPFv3 code shipped with Quagga version 0.98.5[2], so the necessary features of this interface type was implemented in parallel with the WOSPF-OR interface. The actual point-to-multipoint interface was *not* implemented, but its characteristics were incorporated into the WOSPF-OR interface.

The OSPF-MANET routers communicate through WOSPF-OR interfaces, to support the characteristics of wireless ad hoc networks. As it is assumed that the OSPF-MANET routers' wireless links are made up of WLAN (802.11x) interfaces, the broadcast capability of the link

[2]During the implementation of the extensions described in this thesis, Quagga's maintainers released new beta-releases of the Quagga software suite. Unfortunately, neither these releases came with an implementation of OSFPv3's point-to-multipoint. For the time being, being such an unusual and non-commercial interface type it seems are the point-to-multipoint interface type will not be implemented.

is per definition broadcast. Hence, no layer-2 API is implemented to provide layer-3 with the information about the broadcast capabilities of the link.

The Quagga daemon provides information about existing interfaces and their configurations. To communicate with the daemon, the OSPFv3 code implements this interface, which allows for communication with the manager daemon. The interface API is declared in the *zclient.h* library.

7.4.1 The Point-to-MultiPoint characteristics

The WOSPF-OR interface is based on the point-to-multipoint interface described in OSPFv3. This section describes the modifications to the OSPFv3 source code made to support the point-to-multipoint characteristics of WOSPF-OR interfaces.

Forming adjacencies

As all bidirectional WOSPF-OR neighbors should be adjacent, the decision whether to become adjacent or not was manipulated to always become so. The decision is made in the function

```
int need_adjacency (struct ospf6_neighbor *on)
```

If the router is running a WOSPF-OR interface, the function returns 1 to indicate that the statement is true.

Interface state machine

A WOSPF-OR interface in state “DOWN” will transition to “OSPF6_INTERFACE_POINTTOPOINT” upon registering an “InterfaceUp” event. This event is registered when the interface is activated, usually during the routing daemon startup.

Building router-LSAs

On a WOSPF-OR interface packets originated by a node will reach a (sub)set of the nodes in the network. The local router will form adjacency with all of these directly reachable neighbors, and their IDs must therefore be included in router-LSAs. So when originating a router-LSA the router will add one link description per neighboring router. The number of link descriptions in the router-LSA is the same as the number of adjacencies of the router, which again is the number of neighbors it has. The router-LSAs are constructed in the following function, implemented in *ospf6_intra.c*:

```
int
ospf6_router_lsa_originate (struct thread *thread)
```

For WOSPF-OR interfaces *one* point-to-point link description is added per fully adjacent neighbor associated with the interface. A point-to-point link description contains information such as the cost to reach the neighbor, the interfaces of both the local router and the neighbor, and the neighbor's ID. In addition to the point-to-point link descriptors, the node adds a stub network link descriptor.

7.5 Overlapping relays

To calculate the AOR set, connectivity information about one- and two hop neighbors is required. This information is fetched from the router LSAs sent by the local router's neighbors. Router LSAs describe a router's local topology, including the list of neighbors. Thus, by gathering the received router LSAs, the local router will have local topology information needed. Based on the one- and two hop neighbor information, the local router now can calculate the AOR set.

When LS Update packets are received by a router, the LSAs contained in the packet are examined and installed in the link state database. Every router LSA is, in addition to being installed in the link state database, also examined for information such as the list of the transmitting neighbor's neighbors. So, when the router LSA is installed in the link state database, the following function is called which updates the two hop neighbor information. The function is implemented in *wospf_intra.c*:

```
void
wospf_process_router_lsa(struct ospf6_lsa *,
                         struct ospf6_neighbor *,
                         struct ospf6_lsa_header *);
```

Further, the list of one hop neighbors is used for querying the link state database for the router LSAs received from the neighbors. The retrieved router LSA's lists the neighbors known to the originator of the LSA, i.e. one and two hop neighbors of the local node. The neighbor's neighbors that are not neighbors of the local node are included in the two hop neighbor set.

7.5.1 Detecting changes

Changes in the local topology is detected when a one- or two hop neighbor data structure is created or deleted. The global boolean variable `changes_neighborhood` is set, indicating the need for a recalculation of the AOR set.

triggered. When a Hello packet is to be transmitted on the interface the following code is executed:

```
  if (changes_neighborhood) {
```

```
    wospf_calculate_aor();
  }
```

If `changes_neighborhood` is set this test causes an update to the *AOR set* and *dropped AOR set*, which are used during the building of the upcoming LLS data block.

7.5.2 Calculating the AOR set

The algorithm for calculating the AOR set is based on the algorithm which OLSR uses for selecting the MPR set. Therefore, the MPR calculation code is directly applied in the AOR set calculation, with only minor implementation specific modifications. Most important of these is the tie-breaker choosing the neighbor with the highest router ID in cases where two neighbors have the same amount of reachability and equal number of non-overlapping relays.

7.5.3 Signaling willingness

A node may announce its willingness to act as an AOR based on, for example, power status or available memory. The willingness is signaled in the Willingness TLV, which is included in the LLS. Unless defined in the configuration file, the willingness defaults to 128 (WILL_DEFAULT). Moreover, unless willingness is different from the default value, the willingness is not signaled. Neighbors assume default willingness unless explicitly signaled otherwise. As a consequence, this implementation adds the following constraints:

- Since willingness is assumed to be WILL_DEFAULT unless otherwise signaled, to save bandwidth the local node does not transmit Willingness TLVs unless willingness differs from the default value.
- A Willingness TLV is sent persistently in WILL_PERS_COUNT packets only when a new neighbor is registered with the local node.

The integer variable `include_will_tlv` indicates the number of consecutive packets a Willingness TLV is to include in the LLS data block. The two bullets above were implemented by including the following code in the `wospf_insert_neighbor_table` function, hence the code is executed every time a new neighbor is registered with the local node:

```
if (wospf_cfg->willingness == WILL_DEFAULT) {
  include_will_tlv = 0;
}
else include_will_tlv = WOSPF_WILL_PERS;
```

When building an LLS data block a decision is made whether or not to include a WILL TLV. The above code excerpt implements the first bullet in the above list. However, the position of the code, i.e. in the `wospf_insert_neighbor_table` function, ensures that whenever a new neighbor is registered with the local node, the Willingness TLV is included in the next `WOSPF_WILL_PERS` packets. The `include_will_tlv` variable is decremented by 1 for each time the TLV is included in an LLS data block.

This implementation could be extended to adjust willingness dynamically based on, for example, current power status of the node. To allow for dynamic signaling of willingness even when no changes in state have occurred the latter bullet above should be replaced by the following:

- A Willingness TLV is sent persistently in WILL_PERS_COUNT packets whenever a change in willingness is registered with the local node. This may be due to change in power level or other issues that are of concern to the local node.

7.5.4 Processing router-LSAs

The two hop neighbor data set is populated by iterating the “neighbor list” contained in router-LSAs. However, as router-LSAs are flooded throughout the network, only router-LSAs received from adjacent neighbors are processed.

When receiving router-LSAs from its neighbors the local node must compare the router-LSA’s neighbor list with the neighbor list already associated with that neighbor. This is done as follows:

1. Mark all the neighbor’s neighbors as not listed in the router-LSA.
   (a) For every router-ID listed in the router-LSA, mark the corresponding two hop neighbor as listed.
2. For every two hop neighbor not listed, delete from neighbor’s neighbor list and two hop neighbor table.

Note that all two hop neighbor data structures have been added a variable indicating whether or not the two hop neighbor was listed in the router-LSA.

7.6 Incremental Hellos

In OSPF, the known neighbors are listed in the Hello packet. This scheme is different from the incremental Hellos mechanism, in which only changes in state are signaled. Hello packets exchanged between incremental Hellos capable routers do not include the router ID of adjacent neighbors.

OSPF's Hello protocol serves the purpose of establishing and maintaining neighbor relationships. As OSPF assumes bi-directional connectivity between neighbors, nodes list neighbors recently heard form in the Hello packets neighbor list. A node, finding its router ID listed in the Hello packet, registers the neighbor as bi-directional (they have discovered each other). Bi-directional connectivity is thereafter assumed until the node fails to find its router ID listed in the received Hello packet.

The incremental Hellos mechanism, however, diverts from the OSPF Hello protocol in that neither known or unknown/dropped neighbors are listed. As soon as two nodes start the adjacency forming, bi-directional connectivity is assumed until explicitly signaled using LLS.

As routers utilizing the incremental Hellos mechanism do not list known neighbors, a small modification to OSPF's Hello protocol has been implemented. The modification states that if an incremental Hello packet (indicated with the I bit set) is received, and the neighbor is adjacent, the *two way received* event is triggered indicating no change in connectivity.

7.6.1 Maintaining state

To indicate current state on a WOSPF-OR interface, the router appends an SCS-TLV to the Hello packets. The SCS number increases as state changes. The SCS number in the neighbor structure is used for ensuring the validity of received incremental Hello packets. This is done by comparing the SCS number (found in the SCS-TLV) to the last registered SCS number associated with that neighbor. If the update is valid, the information is processed by the receiving system.

The local SCS number is incremented at most *once* per `HelloInterval`. All changes in state during this interval are signaled in the next Hello and associated TLVs.

7.6.2 Receiving Hello TLVs

TLVs related to the incremental Hello scheme, along with the AOR related TLVs, are processed by looping through the TLV contained in the LLS data block. Most of this implementation is straight forward, but a few issues are described here:

The SCS-TLV must be processed *after* the other Hello TLVs, as the SCS-TLV may refer to other TLVs. For example, depending on whether the FS-TLV is appended or not, a FS bit set in the SCS-TLV has a different meaning. Thus, the other Hello TLVs must be processed prior to processing of the SCS-TLV.

The received SCS number (contained in the SCS-TLV) indicates current state of the transmitting neighbor. An unexpected SCS number makes the local node request full state from the neighbor by adding the neighbor's router ID to the global *request state from* list. The next LLS data block will then construct an RF-TLV listing this neighbor, and possibly others if the list contains such entries.

If a Neighbor Drop TLV is received, and the node's router ID is listed, the data structures associated with the neighbor are destroyed. Note that destroying an OSPFv3 neighbor data structure also destroys the corresponding WOSPF-OR neighbor data structure.

A neighbor requesting full state sets the R bit in the Hello TLV. If the local node's router ID is listed in the RF-TLV, or if the RF-TLV is not present, the neighbor's router ID is added to the *full state for* set. The next LLS data block will thus construct a FS-TLV including the neighbor's router ID.

7.6.3 WOSPF-OR neighbor table

In OSPF *areas* have been defined to limit the scope of routing update flooding. As WOSPF-OR is intended as an intra-area extension only, all WOSPF-OR interfaces are assumed to belong to the same area. Because of this, and because the AOR calculation is to be done on a *per node* basis rather than a *per interface* basis, a list of *all* adjacent (or in the process of becoming adjacent) neighbors is maintained more or less separately from the neighbor list of the different interfaces. This list of WOSPF-OR neighbors is referred to as *the neighbor table*.

When a new neighbor is discovered the local node creates an OSPF neighbor data structure. Next, this data structure is encapsulated in a WOSPF neighbor data structure. This structure is as follows:

```
struct wospf_neighbor
{
  <pointer to the corresponding OSPF neighbor
   data structure>
  <Last known SCS number>
  <Other WOSPF-OR neighbor specific information>

};
```

Every WOSPF-OR neighbor data structure is added to the neighbor table. We have a double linked list containing *all* WOSPF-OR neighbors of the local node, and for each of these neighbors a double linked list containing the neighbor's neighbors is maintained. In effect, we have a three dimensional double linked list.

Note that this encapsulation design prevents the OSPFv3 neighbor data structure from being altered, and allows for easy extension of information related to WOSPF-OR neighbors (i.e., if more information about WOSPF-OR neighbors is required this can be added to the `wospf_neighbor` data structure rather than extending the original OSPFv3 neighbor data structure).

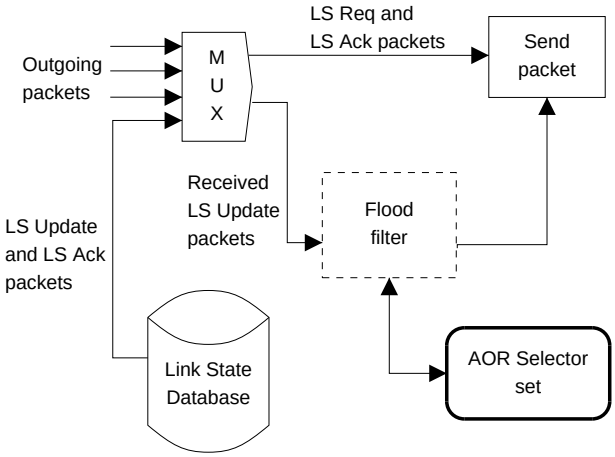


Figure 7.1: The local node forwards the LSA if it is an AOR for the sender.

7.6.4 Dropping neighbors

If a neighbor is not heard from (i. e. a Hello packet is received from this neighbor) within the RouterDeadInterval, the neighbor's state is transitioned to "DOWN". Either the neighbor or the link has ceased to operate so the neighbor is dropped. To signal this state change, the neighbor's router ID is added to the Dropped Neighbor set. Upon next Hello emission, the neighbor's router ID is then added to the Neighbor Drop TLV.

7.7 Relaying control traffic

Figure 7.1 illustrates the process of relaying LS Update packets. The "Relay Decision" box can be viewed as a forwarding filter: When an LSA is received, the router should forward it to its neighbors in order to disseminate the routing information throughout the network. However, the router should not immediately relay information if it is not elected to act as an AOR for the sender - recall that the non-AORs function as backup for the AORs.

OSPF has already defined a forwarding filter described in RFC 2740 as *the flooding procedure*. This procedure suppresses re-flooding of the LSA if for example it is old or if it has already been received by all neighbors in the interface. To exploit the nature of WOSPF-OR networks some modifications to this flooding procedure are implemented. The "Flood filter" box represents the flooding decisions defined in RFC 2328, but the decisions are further complicated by consulting (amongst others) the AOR Selector set.

The design of the modified forwarding procedure described in this section is based on the proposed forwarding procedure described in [20].

To utilize the overlapping relays mechanism and the intelligent transmission of LS Acks, several data structures are maintained. These are described in the following subsections.

7.7.1 Receiving LSAs from AOR Selectors

If the local node has been elected to serve as an AOR for the transmitter, the LSA is scheduled for immediate transmission. The retransmission will serve as an implicit acknowledgment for the sender, and therefore it is not necessary to explicitly acknowledge this LSA. The LSA is thus not added to the list of acknowledgments scheduled for transmission on the interface.

7.7.2 The pushbacked LSAs list

Upon reception of a new LSA, neighbors who are assumed to *not* have received a specific LSA are added to the pushbacked neighbor list associated with the LSA. More precisely: Whenever an LSA is received, all neighbors who have not received this LSA are added to a list. This list is known as the *BackupWait Neighbor (BWN)* list, and is unique for each registered LSA. The LSA for which at least *one* neighbor has not received is added to the *pending LSAs list*. The pending LSAs list is made up of instances of the following data structure:

```
struct wospf_pushbacked_lsa {

  struct ospf6_lsa *lsa;
  struct list *backup_wait_list;
  struct thread *backup_timer;
  struct ospf6_interface *oi;

};
```

The BWN list (named `backup_wait_neighbor` in the above data structure) is a list of the neighbors which have not yet received this LSA. The `wospf_pushbacked_lsa` structs, referred to as *LSA nodes*, are again organized as a list. Thus, each interface has a two-dimensional list made up of pushbacked LSAs and corresponding neighbors. Figure 7.2 depicts this data structure. The horizontal list represents nodes in the pushbacked LSAs list, while the vertical list represents the associated BWN list nodes. Note that the same neighbor may be added to several BWN lists.

Each pushbacked LSA node is associated with a timer. Upon timer expiration the LSA is scheduled for transmission on the interface *unless* the node's BWN is empty. Every neighbor who announces reception of the LSA is removed from the node's BWN list. In other words, if the list is empty, then the local node knows that each of its neighbors have received the LSA, so re-flooding the LSA on the interface will result in a redundant transmission. The LSA node is thus removed from the pushbacked LSA list.

Avoiding redundant transmissions

Cisco's draft has the following statement regarding redundant retransmissions:

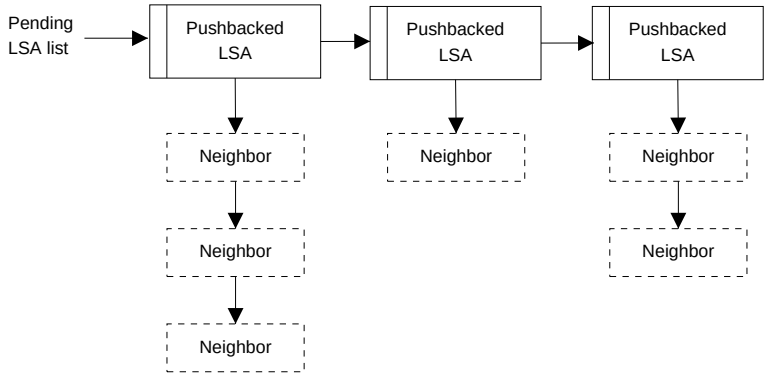


Figure 7.2: The two dimensional list representing the LSAs not having been received by the corresponding neighbors.

> [...] a node can determine whether further flooding a LSA will only result in a redundant transmission by already having heard link state acknowledgments (ACKs) or floods for the LSA from all of its neighbors.

Put in other words, if a node hears an explicit or implicit acknowledgment for the LSA from all its neighbors, a transmission is considered redundant. To accomplish this, acknowledgments received from its neighbors must be registered with the local node. When the local node decides to defer from transmitting the LSA since it is not an AOR for the sender, it must add to the LSA node's BWN list, the list of neighbors not having received the LSA. For each neighbor on the interface, the following tests are made. Each node failing all of these tests is added to the LSA node's BWN list:

- The neighbor originated the LSA.
- The LSA was received from this neighbor.
- The neighbor has already acknowledged the LSA.

The two first bullets are simple to implement. The last one, on the other hand, requires a type of *acknowledgment cache*. This scheme is described in subsection 7.7.3.

Adding jitter to PushbackInterval

The `PushbackInterval` is extended with a small amount of time - *jitter* - to try and avoid several non-AOR retransmitting a pushbacked LSA at the same time. The jitter is calculated by executing `rand() \% 1000`, located in the function `wospf_jitter`. Setting an LSA node to expire in `PushbackInterval` plus jitter is done by executing the following code excerpt:

```
unsigned long interval =
  (unsigned long) (oi->pushback_interval * 1000) +
     wospf_jitter();

lsa_node->backup_timer =
   thread_add_timer_msec (master, wospf_pushback_timeout,
                    lsa_node, interval);
```

The `wospf_jitter` function returns a value between 0 and 1000, hence adding between 0 and 1000 milliseconds to the `PushbackInterval`.

7.7.3 Registering (implicit) acknowledgments - the Ack cache

As a non-AOR is to re-flood LSAs whenever its neighbors belonging to the AOR set for the transmitting neighbor fail to do so, the node must keep track of the LSAs and acknowledgments received by each individual neighbor. If, after the `PushbackInterval` plus jitter for a given pushbacked LSA node, at least *one* neighbor still has not received the LSA, the LSA is scheduled for transmission.

Basically, the non-AOR is only concerned with deciding if it has received (implicit) acknowledgment from all of its neighbors. If this is not the case, the LSA is re-transmitted on the interface.

Each neighbor data structure holds a list, called the *Acked LSA list*, of the LSAs this neighbor has acked but for which the LSA itself has not been received by the local node. This is because a neighbor only acks an LSA the first time it is received on the interface. The LSAs may have been acked either explicitly (acknowledging the LSA in an LS Ack packet) or implicitly (the neighbor have retransmitted the LSA).

When the node receives an (implicit) acknowledgment for an LSA the local node already has received (i.e., the instance of the LSA is being acknowledged), the transmitting neighbor is removed from all the node's BWN lists as the neighbor has now received the LSA.

Figure 7.3 depicts the data structure maintaining the list of acked LSAs.

This scheme of registering acknowledgments is essentially an *acknowledgment cache*. Received (implicit) acknowledgments are cached so that the local router can make re-flood decisions based on the bullets outlined in subsection 7.7.2.

Let us examine this caching scheme by studying an example. Consider figures 7.4 and 7.5. Figure 7.4 provides an example scenario in which an LSA is flooded by router A. Let us call this LSA "x". Router A's flood of LSA "x" reach router B and router C. Router C responds with an acknowledgment, which is received by both router A and router D. At this time, as router D registers an acknowledgment for an LSA itself has not yet received, it registers the acknowledgment in its acknowledgment cache.

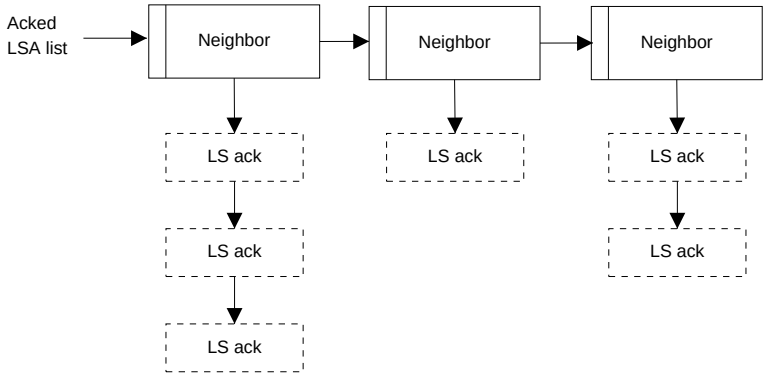


Figure 7.3: The two dimensional list representing the list of LSAs the corresponding neighbors have acked but for which the LSA itself has not been received by the local node.

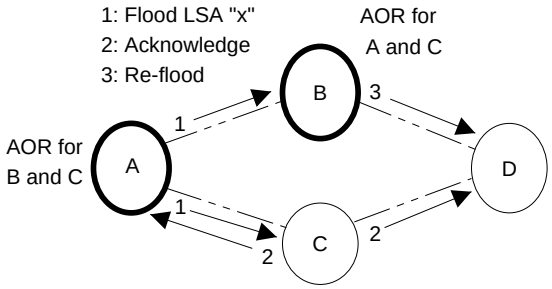


Figure 7.4: The acknowledgment cache

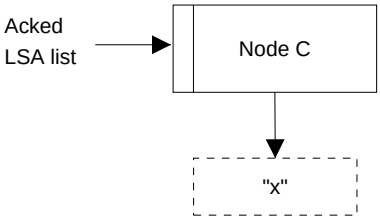


Figure 7.5: The Ack list for node D.

Router B, being an AOR for router A, re-floods the LSA. When router D received this LSA it must take appropriate action according to the algorithm outlined in subsection 7.7.2. As the LSA was received by router B, and router C has already acknowledged the LSA, there is no need to re-flood the LSA.

7.7.4 Pushbacked LSA Timer Expiration

When scheduling a pushbacked LSA node for possible transmission after a `PushbackInterval` plus jitter, the following function is scheduled for execution:

```
int wospf_pushback_timeout(struct thread *)
```

The thread argument (see section 6.2.1 on page 59) is the pushbacked LSA node for which the backup timer has expired.

Define `x` as the time interval from the `wospf_pushback_timeout` for when the LSA node was scheduled for execution, and the time the function was actually executed. `x` will

typically be between 2 and 3 seconds[3]. During `x` all neighbors contained in the LSA node's `backup_wait_list` may have (implicitly) acknowledged the LSA, and have hence been removed from the list. When the function is executed, re-flooding of the LSA is aborted since all neighbors have already received the LSA. Re-flooding will thus result in a redundant transmission. However, if at least *one* of the neighbors in the `backup_wait_list` is still a neighbor (this check is performed in case the neighbor has been dropped during `x`), it has not received the LSA, and the LSA is scheduled for transmission.

Furthermore, re-flooding of the LSA causes the Link State retransmission list for any neighbor to be rescheduled if the LSA is in such a list. Whether the retransmission is the result of a timeout of the neighbor's retransmission list or the LSA is retransmitted for some other reason (timeout of the pushbacked LSA node in our case) is irrelevant.

7.7.5 Receiving Link State Acknowledgments

The observant reader may have noticed the similarity between the *pending LSAs list* and the *Acked LSA list* described in the previous subsections. They have similar structure, and are closely connected to one another:

When a received LSA is to be re-flooded on the interface, the *flooding procedure* defined in Section 13.3 of RFC2328 described an algorithm for determining whether all neighbors on the interface has received the LSA. As the use of the `acked_lsa_list` (which is required since a node acknowledges the LSA only the first time it is received) allows for the local node to register acknowledgments even before the LSA itself is received, this list is consulted when deciding whether to re-flood. Each neighbor which by the mechanisms described in RFC2328, and the use of `acked_lsa_list`, is not known to have received the LSA, is added to the `backup_wait_list` of the LSA (unless the local node is an AOR for the transmitting neighbor). The `acked_lsa_list` of the neighbor is consulted before possibly being added to the LSA node's `backup_wait_list`.

7.7.6 The flooding procedure

The flooding procedure outlined in Section 13.3 in RFC 2328 remains unchanged for non-WOSPF-OR interfaces. The overlapping relays mechanism calls for some modifications due to the intelligent acks scheme and pushback done by non-AORs.

As the node registers acknowledgments of LSAs it has not yet received, Step 1(c) in Section 13.3 is added a second constraint saying that if an acknowledgment for the LSA has been received, examine next neighbor.

Furthermore, steps 2 through 5 are replaced by the following steps:

[3]This assumes the use of the time intervals defined in RFC 2328

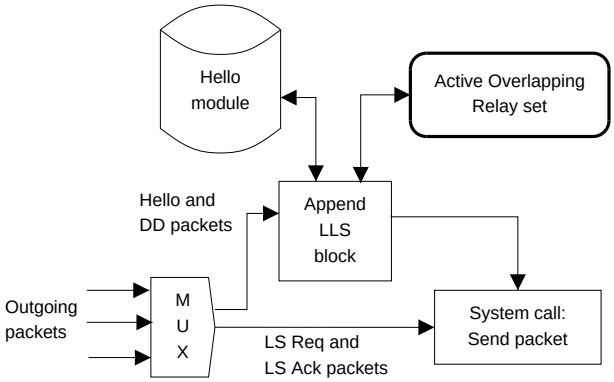


Figure 7.6: Outgoing Hello packets on WOSPF-OR interfaces are intercepted and appended an LLS data block.

Step 2 If every neighbor associated with the interface either has received the LSA or is covered (the LSA must have been received on an WOSPF-OR interface) there is no need to flood the LSA out this interface. Examine next interface.

Step 3 If the LSA was received on this interface, and the router is a non-AOR for this interface, the router waits `PushbackInterval` before deciding to retransmit the LSA. This is accomplished by adding to the BackupWait Neighbor List the neighbors which failed the three conditions listed in section 7.7.2.

Step 4 If the router is an AOR for the neighbor, i.e. the neighbor can be found in the AOR Selector set, the LSA is flooded out this interface.

Note that a neighbor is defined as covered if it is listed as a neighbor of the transmitting router.

All LS Ack packets sent on a WOSPF-OR interface are multicast using the IPv6 address `AllSPFRouters` defined in OSPFv3 . Retransmitted LSAs, and LSAs sent during database synchronization are, however, sent unicast.

7.8 Appending LLS blocks

LLS data blocks are appended to Hello packets when non-OSPF (i.e. information not currently defined for being signaled in the predefined packet types) link local information is to be signaled on the link. TLVs make up such data blocks.

7.8.1 Intercepting outgoing Hello packets

As illustrated in figure 7.6 Hello packets are intercepted by a module responsible for appending the TLVs which are to be transmitted on the link. Quagga transmits packets by adding the packet

to a buffer, and the buffer is then transmitted on the link. The size of this buffer denotes the IPv6 packet size, and is found in the struct field `iovector[0].iov_len`. Appending LLS data block is a matter of extending this buffer, calculating the new buffer (and thus IPv6 packet) size, and adding the result to the above mentioned struct field. The following code shows the simplicity of this operation:

```
int new_end = wospf_append_lls(oh, oi);
extra_length = new_end - (int)end;
iovector[0].iov_len = extra_length + iovector[0].iov_len;
```

Before the LLS data block is appended, the `iovector` contains a pointer to the OSPFv3 packet's header, as well as the size of the packet (retrieved from the header's length field). For this reason, appending the LLS data block is very straight forward. The LLS data block is added to the memory area following the OSPF packet's header, and the length of the `iovector` is extended to include the LLS data block. This is the code shown above.

The TLVs are constructed based on global lists, as briefly mentioned in section 5.2. The lists are updated due to, for example, changes in state, which are signaled in the corresponding TLVs.

The following subsections describes the construction of each of the TLV types.

7.8.2 Including Hello TLVs

The TLVs associated with incremental Hello signaling manages signaling of dropped neighbors, for whom the full state included is intended, and which neighbors full state is being requested. The appending of each of these TLV types is now further examined.

State Check Sequence TLV To indicate current state, an SCS-TLV is appended to outgoing Hello packets. Please refer to section 7.9.1 for further discussion on this TLV type.

The local SCS number is a global variable, and is updated upon state change. In case several state changes occur in a `HelloInterval`, the local SCS number is updated *only* when the SCS-TLV is built. The SCS number should be updated only once per HelloInterval. To insure correct update of this number, the local router keeps track of the previous SCS number. Every state change is followed by the statement `wospf->scs_number = wospf->last_scs_number + 1`, where `wospf` is the struct maintaining information regarding the WOSPF-OR extensions. As the `wospf->last_scs_number` is not updated until the next SCS-TLV is built, it is assured that the `wospf->scs_number` is updated only once per Hello interval.

Neighbor Drop TLV If the global `dropped_neighbors_list` is non-empty, a Neighbor Drop TLV is added to the LLS data block. The router IDs contained in the list are listed in the TLV, and the TLV length is set to the number of bytes occupied by this list.

'Request From' TLV If the *request full state from* set is non-empty, a RF-TLV is added to the LLS data block. The router IDs contained in the set are listed in the TLV, and the TLV

length is set to the number of bytes occupied by this list. If the set is identical to the neighbor set, the RF-TLV is omitted and the R bit in the SCS-TLV is set. This indicates that the full state is requested by all neighbors.

'Full State For' TLV If the *full state for* set is non-empty, a FS-TLV is added to the LLS data block. The router IDs contained in the set listed in the TLV, and the TLV length is set to the number of bytes occupied by this list. If the set is identical to the neighbor set, the FS-TLV is omitted and the FS bit in the SCS-TLV is set. This indicates that the full state Hello packet is intended for all neighbors.

7.8.3 Extended sequence number

As described in Cisco's draft, an SCS-TLV carries a 16 bit sequence number indicating current state. For security threats such as *replay attacks* [13] I have extended the length of the sequence number to 32 bits.

The current format of the SCS-TLV is shown in Figure 7.7. Due to the extended SCS field each SCS-TLV will have an additional length of 4 bytes.

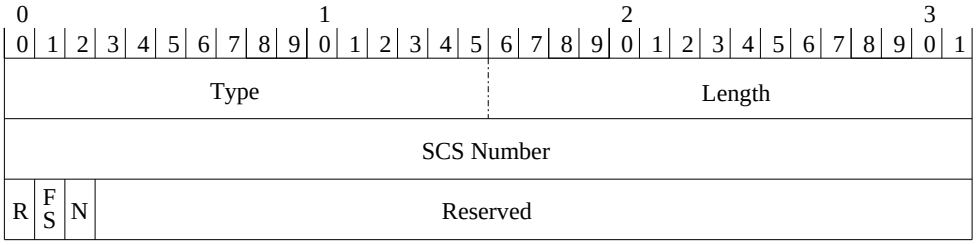


Figure 7.7: The format of the extended SCS-TLV.

7.8.4 Including Overlapping Relays TLVs

Two TLVs are defined for use in AOR signaling. These are the AOR TLV and the Willingness TLV.

As the TLVs described below only signal selected and dropped AORs, as well as willingness to serve as an AOR, only changes in the *AOR set*, the *dropped AOR set* and also the power level of the router, cause the router to include AOR information in the LLS block.

AOR TLV The newly calculated AORs are appended to a list consisting of data structures similar to that used when signaling dropped neighbors. These data structures maintain information such as a counter indicating the number of packets the AOR's router ID has been included in the packet. This scheme allows for signaling AOR elections persistently, as outlined in section 7.3.3. The same approach is used for signaling dropped AORs, i.e. neighbors who are not longer elected to serve as AORs.

Willingness TLV The willingness is signaled upon willingness change, which may be the result of difference in power status, available memory, etc. However, this thesis does not implement automated willingness calculation, so willingness is only altered upon input from the router administrator.

Care has been taken to avoid unnecessary election of AORs, as described in section 5.10.9. This has been implemented by not signaling dropped AORs unless the AORs have been signaled as an AOR in the first place.

7.9 Design analysis

This section discusses issues related to the implementation design.

7.9.1 Including the SCS-TLV

Cisco's draft does not explicitly discuss whether the SCS-TLV should be included in *every* Hello packet, or only when changes in state are signaled. When describing reception of Hello packets, however, it states the following: "If a device receives a Hello from an adjacent neighbor with an SCS number less than [...]". This statement *may* indicate that the SCS-TLV is present in all Hello packets. Let us discuss this in more detail.

If the SCS-TLV is *not* present in all Hello packets (it is, for example, only included in k Hello packets after a state change). The situation may arise that a neighbor, due to poor link quality, does not receive k Hello packets in a row, and is not aware of the transmitting node's state change.

If, on the other hand, the SCS-TLV is included in *all* Hello packets, this adds to the overhead associated with the periodically sent Hello packet. As the SCS-TLV consists of eight bytes, each Hello packet transmission is added eight bytes. If a node has only one neighbor, removes this neighbor's router ID from the Hello packet's neighbor list but adds an SCS-TLV, the incremental Hellos scheme will in fact add to the total Hello protocol overhead. In scenarios where the transmitting node has more than two neighbors, however, one can see that even with an added SCS-TLV the overhead will lessen. Intuitively, the denser the network, the greater the benefit of utilizing the incremental Hellos scheme.

In my implementation I have chosen to append the SCS-TLV in *all* Hello packets, in order to exclude the possibility of a neighbor missing change of state signaled by the transmitting node.

7.9.2 List structures in persistent signaling

To allow for persistent signaling, the information which is to be sent persistently must be associated with some sort of counter. This implementation has designed persistent signaling in such a way that any particular piece of information may be sent persistently, not only the TLV

type itself. More precisely put: Setting the `WOSPF_DROPPED_NEIGHBORS_PERS` definition to 3 ensures that any dropped neighbor is included in 3 consecutive Neighbor Drop TLVs (i.e. in 3 consecutive Hellos). This is regardless of other router ID's listed in the TLV - the dropped neighbor *will* be signaled 3 times.

Optionally, persistent signaling could be implemented such that a given TLV type is sent in a predefined number of consecutive packets. However, cases where for example a neighbor has been dropped just when the `k`th Neighbor Drop TLV is built, this neighbor would either be listed in *one* such TLV, or the entire TLV would be sent in additional `k` consecutive packets.

Cisco's draft does not discuss this matter in detail, other than stating that TLVs may be sent persistently.

7.9.3 Not sending FS-TLVs and RF-TLVs persistently

This implementation does not support ordinary persistent signaling of either request for full state or replies with full state. More precisely, defining such information to be signaled in a number of consecutive packets (as is done for, for example, Neighbor Drop TLVs) is not allowed. The argument for this decisions is as follows: Request for full state is signaled once a neighbor's state signaled is found erroneous or missing. As long as the neighbor does not respond to the request for full state, the local node requests this by default. Request for full state is thus signaled indirectly until full state is received by the neighbor, or the neighbor is dropped. Similarly, as a neighbor will request full state until the local node responds to the request, full state is only sent *once* upon such request. For example, if Hello packets containing such TLVs are lost, the request for (and possibly also responses to these requests) will implicitly be signaled persistently.

7.9.4 OSPF-MANET multicast using the AOR calculations

A function has been implemented to return a truth statement regarding whether the node is an AOR for the transmitting neighbor. This functions allows for optimizing flooding of *all* multicast packets, as other software components on the router can make forwarding decisions based on the WOSPF-OR interface's topology information. The function has the following definition:

```
int
is_AOR(u_int32_t neighbor_id)
```

This function could be used for disseminating application layer real-time traffic throughout the OSPF-MANET. The traffic would be relayed by each node's AOR set. However, this feature is dependent on the presence of a mechanism for avoiding re-flooding duplicate packets. The WOSPF-OR extensions implement such an mechanism, but can only be applied for OSPF routing updates.

7.10 Implementation issues

7.10.1 Programming in Perl

As the author had close to no experience with Perl programming, quite a lot of time and effort was put into learning what was needed to successfully develop the scripts. Fortunately, numerous web forums exist where tricky issues can be posted and reviewed by experienced Perl programmers, and some parts of the scripts were developed based on discussions on such forums.

Perl programming skills where acquired by the author since the author concluded that this tool would increase development efficiency.

7.10.2 Linux issues

Fortunately, the Quagga routing software does not depend on any rare programming libraries, so the installation was quite smooth. However, the developing tools installed by fetching Debian/Ubuntu packages from the respective repositories did not interoperate well. There were compatibility issues due to different release versions of the development tools, but after the correct versions were installed, the Quagga code would compile.

7.10.3 Using debugging tools

Having access to good debugging tools is essential when developing software systems. In particular, programming in C is (as most programmers would agree) synonymous with pointer errors, memory corruption, and so forth. Tools such as the GNU debugger [15] and valgrind [36] proved to be crucial. Especially valgrind proved to be of help with debugging memory corruption error, which is one of the hardest types of error to track down.

7.10.4 Bugs in Quagga and the OSPFv3 daemon

Being an open source community based project, the protocols implemented in Quagga may contain errors. However, these are quickly patched by the volunteer developers as soon as the errors are identified.

During the process of implementing software the better part of the time used is often associated with finding and correcting errors. This is very common, as programming often involves rather complex logic. Extending a relatively comprehensive and complex protocol like OSPFv3, which in Quagga's implementation of version 0.98.5 is made up of some 20.000 lines of C code, is quite difficult as the OSPFv3 code itself may contain errors. Quite a lot of time was spent trying to determine whether errors were occurring due to errors in the OSPFv3 code, or due to the extensions described in this thesis.

One protocol behavior in particular took a lot of time and energy to analyze, and resulted in an (for now) unresolved discussion on the Zebra mailing list. In short the problem was as follows: When a neighbor transitions to state "FULL", the local node schedules a stub area advertisement. The Intra-Prefix LSA is then prematurely aged and flushed from the routing domain, and removed from the LS database. The problem arises when the neighbors acknowledge the LSA, and the local node has removed it from its database. We then have a situation where an ack is received for a newly sent LSA, but for which the LSA is no longer in the database. In other words, an ack for a self-originated LSA is received, but the local node does not have a copy of it in its database. For a more comprehensive outline of the problem the interested reader should see the relevant posts on the Zebra mailing list.

To this day the behavior described above remain unclassified as to whether this is a bug or if it is compliant with the OSPF specification. As soon as this thesis is completed, by request the author will continue the discussion on the mailing list.

Another bug that took time to track down was the bug causing routers to emit newly generated LSAs without waiting the required `MinLSInterval`. RFC 2328 states that a new instance of a particular LSA type is not to be flooded within `MinLSInterval` time since the previous instance. This bug caused a lot of unexpected behavior in the routing daemon, as routers rejected the new instances and LS database discrepancies became a fact. Due to time constraints, this bug was not fixed by adding the timer, but a hack was added saying that no new instances are to be rejected even though they arrive in less that `MinLSArrival` apart.

7.10.5 The point-to-multipoint interface

The OSPFv3 source code shipped with the Quagga software, version 0.98.5, was not fully compliant to RFC 2740. For example, the point-to-multipoint interface was not implemented, so this interface was developed along with the protocol described in draft. The details on this implementation were given in section 7.4 on page 70. It is important to note that the point-to-multipoint interface is not implemented for use on non-WOSPF-OR interfaces; this thesis implements only the *characteristics* of point-to-multipoint interfaces, and these characteristics are incorporated into the WOSPF-OR interface. The distinction is rather fine, but implementing a point-to-multipoint interface separate from the WOSPF-OR interface should require only minor modifications.

Chapter 8

Proof of Concept - Test of the Implementation

The WOSPF-OR extensions implement *two* mechanisms for reducing routing overhead in an OSPF-MANET, namely *overlapping relays* and *incremental Hellos*. These mechanisms aim at optimizing two different aspects of routing in such networks, and may be used independently of one another. However, as described in section 7.1, only the use of incremental Hellos is optional on WOSPF-OR interfaces; The use of overlapping relays is mandatory.

This chapter demonstrates the implementation and its operability. Several examples are provided to illustrate how the implementation works in different scenarios set up to emulate real-world usage. As WOSPF-OR is a *routing protocol* extension, operating on the network layer of the ISO model, whether the links in the network are wireless or wired should not be of any significance. Wireless challenges such as the hidden node problem are not examined further in this thesis, as this is irrelevant for the implementation of WOSPF-OR itself (this is left to future research on the routing protocol and link layer protocol). Functionality of WOSPF-OR is demonstrated by setting up different scenarios on a wired testbed consisting of four nodes[1]. Link breakage and mobility is emulated by manipulating the `ip6tables` of each node. To show that the implementation works as specified, proof of concept is applied to test different aspects of the extensions in relevant scenarios. To aid in this, output from the nodes' log files are provided and examined.

To prove that the WOSPF-OR implementation works, several test scenarios are provided in this chapter. Sections 8.2 and 8.3 prove the functionality of the overlapping relays mechanism and the incremental Hellos mechanisms, respectively. These tests are relatively small, and aim at proving some of the main features of the extensions. In section 8.4 a more comprehensive example is given. This example utilizes both the overlapping relays mechanism as well as the incremental Hellos mechanism. All the tests come with log output added informational

[1] A node is equivalent with a computer connected to the network. The nodes in our tests function both as hosts and routers

comments, included in Appendix A. For the latter scenario, parts of the log output are provided along with the scenario description.

The first section describes the testbed used for testing the implementation. The consecutive sections describe the tests used for proving the implementation's correctness.

8.1 Testbed setup

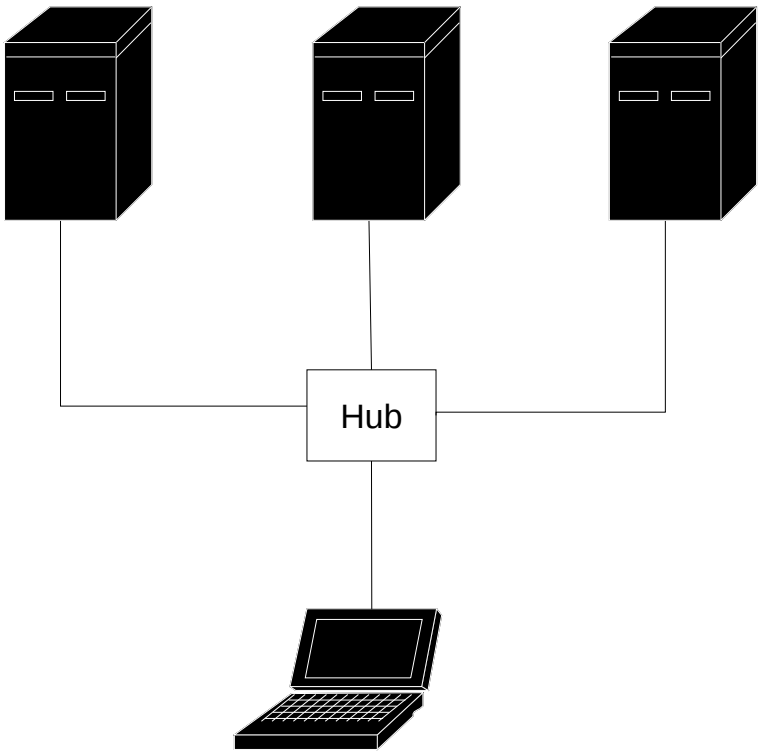


Figure 8.1: The testbed used for implementation.

The WOSPF-OR extensions were developed and tested on a testbed consisting of four routers (actually, these routers were merely ordinary computers running the discussed protocol) connected via a hub. Figure 8.1 depicts the testbed. Four routers is sufficient for setting up different scenarios for utilizing the wireless extensions. Emulating the multi-hop environment of MANETs was done by bringing links up and down, forming more or less random topologies typical for MANETs.

Only one machine (the laptop) was used for the actual implementation, whereas the three network machines (i.e. machines A, B and C in the figure) were only to run the implementation.

Recognizing that the network machines may have different architecture, it was decided that, for each test run, each of these machines would be updated with the current source code, then compile and then run. This way, the implementation would run correctly independent of the underlying architecture of the network machines.

To automate the task of distributing the source files, a script written in the programming language Perl was developed to update the network machines with the latest version of the source files. This script was run on the implementation machine when the implementation was to be tested.

When using the described testbed to emulate a mobile environment, manipulation of the *ip6tables* comes in handy. To allow for efficient manipulation of these tables, another script was developed. This script allowed for interpreting instructions on the form `<machine name> symblock <machine name>` and `<machine name> block <machine name>`. The former emulates a symmetric link breakage, while the latter emulates the loss of link resulting in that a machine `A` can reach machine `B`, but `B` can not reach `A`.

Several more Perl scripts was developed to automate tasks, such as synchronizing time between the machines, fetching and parsing log files, and so forth. In addition, a few primitive shell scripts were created for the same purpose as the Perl scripts - automating frequently performed tasks.

8.2 Scenarios proving Overlapping Relays functionality

To prove the basic functionality of having a node elect an AOR, a three node network will suffice. The nodes will form a chain, and the two end nodes will elect the middle node for relaying their LSAs.

However, in a four node network a node may elect, as well as be elected by, nodes to relay LSAs. This makes testing of the functionality more realistic since real-world OSPF-MANETs most likely will see nodes taking on both tasks.

The first scenario shows a full meshed four node network. As depicted in figure 8.2, all nodes are in direct communication range of one another, and relaying information on behalf of others should not be necessary.

The second scenario, depicted in figure 8.3, shows the effect of having a node moving in and out of transmission range of one end-point of a network. The scenario is set up so that this end-point node should both elect, and be elected as, an AOR.

These scenarios are now studied in more detail. Throughout this chapter the term “node” will be used for routers implementing the WOSPF-OR interface. Also, the router names “1.1.1.1”, “2.2.2.2” and so forth will be represented by the capital letters “A”, “B” and so forth in the provided illustrations and commentaries. Initially, the log output provided by Quagga included the complete time and date of each of the events registered. This information has been parsed

such that only minutes and seconds are present. Each line in the log file now starts with `<x>:<y>: <information>`, where `x` denotes minutes, while `y` denotes the seconds.

Full mesh

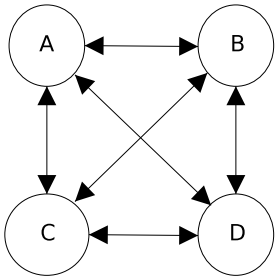


Figure 8.2: Full meshed four node network.

For log output see Appendix A, section A.1.

In a full meshed network there should be no need to relay information. Every node has connectivity to every other node in the network, and a re-flood of routing information should result in a redundant transmission.

It is worth noting, however, that a node during initialization time *may possibly* elect AORs. This is because the node may not be up to date with regards to its local neighborhood; LSAs are only transmitted along neighbors who either are in the process of synchronizing their LS databases, or have completed this process. Consider Figure 8.2: If, say, node D is the last node to join the network (by either have being brought withing transmission range of the other nodes, or having been powered on) it must synchronize with its new neighbors. If the first node it synchronizes with is node B then node B is the only adjacent of node D and thus all LSAs are relayed to node B only. Through node B it learns of node A and node C, and node D elects node B to relay LSAs to these nodes. Node B keeps relaying LSAs until node D synchronizes with the other nodes and LSAs are transmitted directly to these nodes.

Chain topology

For log output see Appendix A, section A.2. The log output is an excerpt of node C's log messages.

At startup node A though C form a line. Node B is an AOR for node A and node C. After a while node D moves within transmission range of node C, thus forming a line of four nodes. After C has sent its router LSA to node D, the latter node elects node C as an AOR. Node C is now both an AOR and an AOR Selector.

An interesting observation in this scenario is that half of the nodes in this network are elected to serve as an AOR. Also, the AORs are both elected by each other for this task.

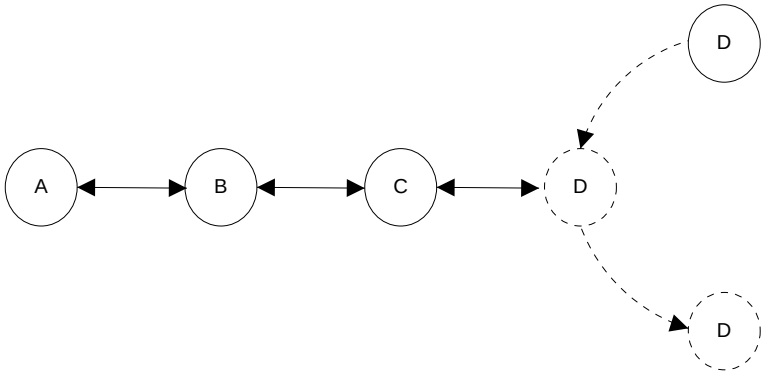


Figure 8.3: Three nodes forming a chain topology. Node D moves in and out of node C's transmission range

8.3 Scenarios proving incremental Hellos functionality

Hello packets are emitted to signal current state of the transmitting node. As each state is associated with a sequence number, receiving a Hello packet with an erroneous sequence number will cause a node to request current state from the transmitting node.

The first scenario, depicted in figure 8.4 shows two static nodes within transmission range of each other. After a small initialization time they omit each other's router ID in the Hello packets, thus reducing the packets size. In this simple scenario the reduction in packet size is very small, but as the number of neighbors increase the reduction will be greater. Over time this should have significant effect on routing overhead imposed on the network.

To show what happens when a node becomes neighbors with a new node having an SCS number greater than 1 (indicating that the new neighbor have undergone state changes in the past), the second scenario includes a wandering third node which travels in the vicinity of a two hop neighbor. This scenario is depicted in figure 8.5.

Two static nodes

The basic idea behind incremental Hellos is to signal only *changes* in state, rather than transmitting full state in every Hello packet. In the following scenario two nodes become neighbors, and the incremental Hellos mechanism causes the two nodes to omit the neighbor list usually carried within the Hello packet.

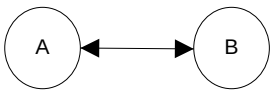


Figure 8.4: Two static nodes.

For log output see Appendix A, section A.3. The log output is an excerpt of node C's log messages.

OSPFv3 specifies that each Hello packet should include a list of *all* nodes heard from recently, whereas in WOSPF-OR networks this list is omitted unless changes in state occur. As seen in the log output, as soon as the neighbor state reaches "EXCHANGE" (see description of neighbor states in RFC 2328 section 10.1) the incremental Hellos mechanism is enabled. Every WOSPF-OR neighbor utilizing the incremental Hellos mechanism is removed from the Hello packet's neighbor list as soon as the adjacency starts forming. As long as the adjacency is maintained node A will not include node B in its neighbor list carried in its Hello packets, unless node B for some reason requests full state.

This very simple scenario illustrates the essence of the incremental Hellos mechanism. Note that utilizing incremental Hellos with only *one* neighbor is not an optimal scenario. Discussion of this issue can be found in section 7.9.1.

Wandering third node

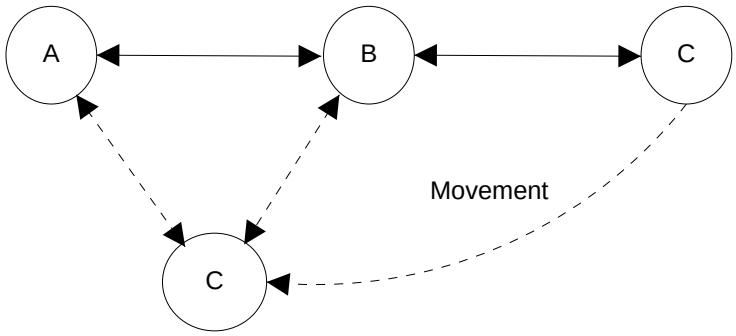


Figure 8.5: Wandering third node.

For log output see Appendix A, section A.4. The log output is an excerpt of node C's log messages.

Figure 8.5 shows how the incremental Hellos mechanism operates when a node (node C) is being introduced to a new node (node A) having an SCS number greater than 1. As node A have a SCS number greater than one it must have undergone state changes, and when node C sees this it requests full state from node A. The same goes for node A - learning that node C has undergone state changes of its own node A requests full state from node C.

One neat feature implemented in this thesis (it was briefly mentioned in Cisco's draft as an optional mechanism) is that a node may omit for example a "Request FS from"-TLV if it wishes to request full state from all of its neighbors. The implementation of this was described in section 7.2.3, and in the scenarios described in this chapter the threshold for omitting the TLV is 0.5, meaning that if the node wishes to request full state from a minimum of half its neighbors

the TLV is omitted. In the scenario depicted in Figure 8.5 both node A and node C utilize this feature.

8.4 Scenarios using both Overlapping Relays and incremental Hellos

The previous scenarios have shown the incremental Hellos and overlapping relays mechanisms used independently of one another. The following scenario utilizes *both* these techniques simultaneously.

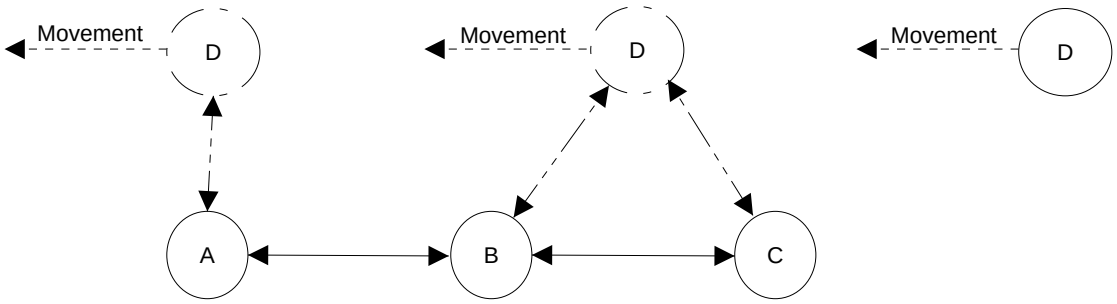


Figure 8.6: Both incremental Hellos and overlapping relays.

Figure 8.6 depicts a scenario that typically would occur on a freeway. The cars are here represented by nodes (circles). Nodes A, B and C are moving in a chain formation. Node D is moving at a somewhat higher speed, and is slowly catching up and passing the other cars in the overtaking lane.

The complete log output is provided in Appendix A section A.5, but some key issues are now further examined. Each subsection starts with an excerpt from the log file, followed by information comments. The "`[...]`" indicates that lines have been left out. The issues are listed in chronological order to be consistent with the actual course of events.

Most of the log output is retrieved from node D's log file, but additional log output from node B is also provided. This is noted.

The description in the following subsections follows the process as node D overtakes other nodes.

8.4.1 Requesting full state

```
32:56: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
32:56: Got a new incremental Hello with no state change ->
   requesting full state from 3.3.3.3
32:59: Including 3.3.3.3 in neighbor list
```

```
32:59: NOT setting the SCS TLV's N bit
32:59: Requesting FS from 1 of 1 neighbors, dropping the TLV
```

Node C is utilizing the incremental Hellos mechanism when communicating with node B. When the link becomes operational (the local node moves within transmission range of node C), the local node overhears such a packet being transmitted by node C. This causes the local node to request full state from node C.

The threshold for omitting a RF-TLV is set to 0.5 (i.e. 50 percent of its neighbors), and when the node requests full state from 100 percent of its neighbor the TLV is naturally dropped.

8.4.2 Initial database synchronization

```
32:59: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]

Synchronize databases.

32:59: Sending LS Req to 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:1.1.1.1]

[...]

32:59: Got [Link Id:0.0.0.2 Adv:1.1.1.1] from 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:1.1.1.1]
32:59: Age: 69 SeqNum: 0x80000001 Cksum: f325 Len: 56
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Delayed acknowledgment scheduled in 2 seconds

[...]

32:59: Got LS Req from 3.3.3.3 - send [Link Id:0.0.0.2 Adv
   :4.4.4.4] unicast
32:59: Sending LSA(s) UNICASTS to neighbor 3.3.3.3:
32:59: [Link Id:0.0.0.2 Adv:4.4.4.4]
32:59: Age: 70 SeqNum: 0x80000001 Cksum: 5e66 Len: 56
```

When the local node starts the initial synchronization process with node C, a request for node 3's database entries are made. As node C has not had any direct contact with node 1, these LSAs have been received by node B. This indicates that the LSAs are flooded throughout the multi-hop environment. Note that node C may have received these LSAs either through database synchronization with node B (assuming node had node A's LSA in its database at the time), or through flooding. A more explicit example of *flooding* LSAs in such a multi-hop environment is provided in a later subsection.

As node C doesn't have a copy of the local node's LSAs, it makes a request for these in LS Req packets.

8.4.3 Re-flooding due to pushback timeout

```
33:02: Received a duplicate LSA [Link Id:0.0.0.2 Adv:4.4.4.4]
   from 3.3.3.3
```

The LSAs node C received from the local node must be flooded to the other nodes in the network. Since node C is not (yet) an AOR for the local node, it waits `PushbackInterval` before deciding to re-flood. Upon timer expiration, at least one of its neighbors (node B) hasn't received the LSA, and node C re-floods it on the interface.

8.4.4 Signaling the election of AOR

```
33:09: Signaling my election of AORs (pers: 3):
33:09: 3.3.3.3
33:19: Signaling my election of AORs (pers: 2):
33:19: 3.3.3.3
33:29: Signaling my election of AORs (pers: 1):
33:29: 3.3.3.3
```

The information carried in the "AOR"-TLV is in this scenario to be sent in three consecutive TLVs.

8.4.5 A node becoming an AOR

The following log output is retrieved from node C's log file. Note that these listings were recorded approximately simultaneously with the log output in the previous subsection.

```
33:09: I'm an AOR for 4.4.4.4
33:09: AOR Selector table:
33:09: 1: 2.2.2.2
33:09: 2: 4.4.4.4

[...]

33:59: Got [Router Id:0.0.0.0 Adv:4.4.4.4] from 4.4.4.4
33:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
33:59: Age: 1 SeqNum: 0x80000002 Cksum: 5689 Len: 56
33:59: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 2.2.2.2
33:59: 4.4.4.4 originated the LSA..
33:59: 4.4.4.4 is the sender if the LSA..
33:59: At least one neighbor hasn't received the LSA..
33:59: I'm an AOR for 4.4.4.4 -> reflood [Router Id:0.0.0.0 Adv
   :4.4.4.4] on interface
```

```
[...]

33:59: I'm an AOR -> NOT sending ack to 4.4.4.4
```

An AOR is to immediately re-flood LSAs received from nodes in its AOR Selector set. Being an AOR for node 4, node C performs as expected. In addition, since the re-flood serves as an implicit acknowledgment no explicit acknowledgment is sent back to node 4.

8.4.6 Second synchronization

```
33:59: Neighbor state change 2.2.2.2%eth0: [ExStart]->[ExChange
   ]
33:59: Neighbor state change 2.2.2.2%eth0: [ExChange]->[Full]
```

When the local node establishes connectivity with node B, and the start the database synchronization process, they find that they are already synchronized. This is due to node 3's relaying of LSAs (see the next subsection).

8.4.7 Inserting and updating the pending LSA list

```
33:59: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 2.2.2.2
33:59: [Router Id:0.0.0.0 Adv:2.2.2.2]
33:59: Age: 1 SeqNum: 0x80000003 Cksum: b80d Len: 72
33:59: Flood: Add [Router Id:0.0.0.0 Adv:2.2.2.2] to retrans-
   list of 3.3.3.3
33:59: 2.2.2.2 originated the LSA..
33:59: 2.2.2.2 is the sender if the LSA..
33:59: At least one neighbor hasn't received the LSA..
33:59: Adding LSA([Router Id:0.0.0.0 Adv:2.2.2.2]) node to the
   pending LSA list. Waiting:
33:59: 3.3.3.3

[...]

33:59: 3.3.3.3 acked the pending LSA [Router Id:0.0.0.0 Adv
   :2.2.2.2], removed from backup wait list
```

At this point, the local node is adjacent to both node B and node C. Since LSAs are flooded out adjacencies only, and at least one of these presumably hasn't received the LSA yet, the LSA is added to the pending LSAs list with node C as a pending neighbor.

Immediately after, however, node C explicitly acknowledges the LSA, and is removed from the backup wait list.

8.4.8 Implicit acknowledgments

```
33:59: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 3.3.3.3
33:59: Implicit acknowledgment(reflood of [Router Id:0.0.0.0
   Adv:4.4.4.4]) from 3.3.3.3!
33:59: Removing [Router Id:0.0.0.0 Adv:4.4.4.4] from 3.3.3.3's
   retrans list
```

By now node C is an AOR for the local node, and re-floods LSAs immediately. Such a re-flood serves as an implicit acknowledgment, which is shown in the log output above.

8.4.9 Aborting re-flood after pushback

```
34:02: Flooding [Router Id:0.0.0.0 Adv:2.2.2.2] will result in
   a redundant transmission - abort
```

Initially, node C was added to the above LSA's backup wait list. As node C (implicitly) acknowledged the LSA it was removed from this list. After the `PushbackInterval` timeout the backup wait list is empty, and there is no need to re-flood the LSA.

8.4.10 Dropping neighbors

```
34:46: Neighbor state change 3.3.3.3%eth0: [Full]->[Down]
34:46: 3.3.3.3's state transitioned to less than EXCHANGE:
   Delete from neighbor table
34:46: Change in state: deleting neighbor 3.3.3.3!
34:46: ** Neighbor table:
34:46: 1: 2.2.2.2 (will: 200)
34:46: Neighbor 1: 1.1.1.1
34:46: Neighbor 2: 3.3.3.3
34:46: Increasing my SCS number from 3 to 4

[...]

34:49: Building Dropped TLV:
34:49: - added 3.3.3.3 (pers: 1)
```

An entry in the neighbor table is removed when the corresponding neighbor transitions to a state less than "EXCHANGE". The change in state is indicated by increasing the local SCS number.

This scenario defines information carried in "Neighbor Drop"-TLVs only to be signaled in one TLV.

8.4.11 Receiving full state from up-to-date neighbors

```
35:01: Got a new incremental Hello with no state change ->
   requesting full state from 1.1.1.1
35:01: Including 1.1.1.1 in neighbor list
35:01: NOT setting the SCS TLV's N bit
35:01: Requesting FS from 1 of 2 neighbors, dropping the TLV

[...]

35:03: Ignoring 2.2.2.2's Hello packet: FS bit set, but SCS is
   same
```

Setting the R bit and omitting the "Req FS from"-TLV causes every neighbor to respond with full state. In this case, both node A (the intended requestee) and node B respond with full state. However, full state from node B is not needed, and the TLVs are thus ignored.

8.4.12 Receiving a request for full state

```
35:11: Got SCS from 1.1.1.1 with R bit set but no Request From
   TLV added -> sending full state
35:11: Including 1.1.1.1 in neighbor list
35:11: 1 of 2 neighbors requesting full state, dropping the TLV
```

Node A has requested full state from all of its neighbors. The local nodes respond with full state. Note that the SCS TLV is implicitly added, although not shown in the log output.

8.4.13 Inserting a neighbor's Acked LSAs list

```
35:43: Neighbor state change 2.2.2.2%eth0: [Full]->[Down]
35:43: 2.2.2.2's state transitioned to less than EXCHANGE:
   Delete from neighbor table

[...]

35:45: ** Reg ack [Router Id:0.0.0.0 Adv:2.2.2.2] received from
    1.1.1.1

[...]

35:46: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 1.1.1.1
35:46: [Router Id:0.0.0.0 Adv:2.2.2.2]
35:46: Age: 5 SeqNum: 0x80000004 Cksum: 39b1 Len: 56
35:46: 1.1.1.1 is the sender if the LSA..
35:46: 1.1.1.1 has already acked the LSA..
35:46: All my neighbors have received the LSA - abort flooding
35:46: Delayed acknowledgment scheduled in 2 seconds
```

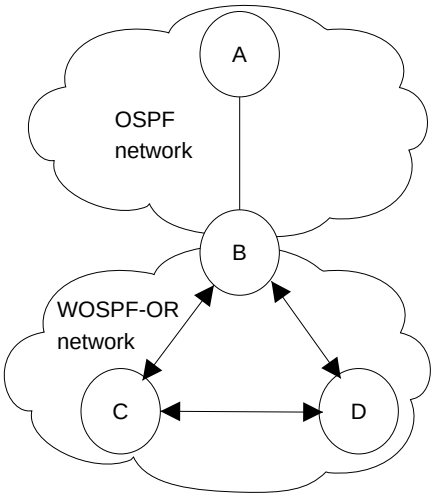


Figure 8.7: Interconnecting a WOSPF-OR network with an OSPF network.

Here we see that node A explicitly acknowledges the router LSA received from node B. As node B has not signaled node A to act as an AOR, node A pushes back the LSA, and then re-floods it upon timer expiration. However, the explicit acknowledgment sent by node A was overheard and registered with the local node. The acknowledgment was added to the `Acked LSA list` maintained in every `wospf_neighbor` data structure. When the LSA is received, the local node knows that node A has received it because it has already heard the acknowledgment. In this particular scenario, however, node A is also the transmitter of the LSA so even if the local hadn't heard the ack it would have known that node A had received the LSA.

8.5 Communicating with OSPF routers

The wireless extensions implemented in this thesis are defined for use on WOSPF-OR interfaces. When the outgoing interface is to an WOSPF-OR network, the extensions are utilized for optimization reasons. Interfaces to other OSPF networks, on the other hand, do not make use of these extensions. To prove that the wireless extensions do not affect operation on non-WOSPF-OR interfaces, a scenario where a router has one interface to a WOSPF-OR network, and one interface to a OSPFv3 based LAN, is set up.

The scenario is depicted in Figure 8.7, and shows node A and node B being connected on OSPF interfaces. Node B has *two* interfaces, one running OSPF for communicating with node A, and another interface for communicating with nodes in the WOSPF-OR network. Nodes C and D are WOSPF-OR nodes.

When communicating in a WOSPF-OR network, a node utilizes the overlapping relays scheme, and possibly (but this is not required) also the incremental Hellos scheme. Hence, when communicating in a WOSPF-OR network the local node must enable at least the overlapping relays scheme on that interface.

As the figure shows, node A is connected to node B on one interface, while the other interface is a WOSPF-OR interface on which it communicates with node C and node D.

8.5.1 Election of Designated Router on LAN

The first listing shows output from node A's vty shell when executing the `show ipv6 ospf6 neighbor drchoice` command.

```
RouterID State/Duration DR BDR I/F[State]
3.3.3.3 Full/00:00:21 0.0.0.0 0.0.0.0 eth0[PointToPoint]
4.4.4.4 Full/00:00:40 0.0.0.0 0.0.0.0 eth0[PointToPoint]
2.2.2.2 Full/00:00:37 2.2.2.2 1.1.1.1 eth1[BDR]
```

On interface `eth1`, node A is running OSPFv3. On this interface the designated router scheme as defined in RFC 2328 is utilized. However, on the WOSPF-OR interface `eth0` no (B)DRs are elected, as was expected.

8.5.2 Flooding routing information

Moreover, being a DR on its broadcast link, node B originates Network LSAs to indicate which routers are connected to the link. The following output is provided when the command `show ipv6 ospf6 database network detail` is executed in node D's vty shell:

```
  Area Scoped Link State Database (Area 0.0.0.0)

Age: 60 Type: Network
Link State ID: 0.0.0.2
Advertising Router: 2.2.2.2
LS Sequence Number: 0x80000001
CheckSum: 0x56bc Length: 32
    Options: --|R|-|--|E|V6
    Attached Router: 2.2.2.2
    Attached Router: 1.1.1.1
```

8.5.3 Advertising prefixes

Because all the nodes in the scenario reside in the same *area*, LSAs with an area flooding scope such as this are flooded through all the networks in the area. In addition, node B floods LSAs indicating its network's prefix:

```
Age: 251 Type: Intra-Prefix
Link State ID: 0.0.0.2
Advertising Router: 2.2.2.2
```

```
LS Sequence Number: 0x80000001
CheckSum: 0x1327 Length: 44
    Number of Prefix: 1
    Reference: Network Id: 0.0.0.2 Adv: 2.2.2.2
    Prefix Options: --|--|--|--
    Prefix: 2002:700:500:49::/64
```

8.5.4 Flooding procedure

As the optimized flooding scheme is applied to WOSPF-OR interfaces *only*, if the outgoing interface is not WOSPF-OR then the flooding procedure described in RFC 2328 is utilized. Consider the following output from node A's log file:

```
First run:

2006/04/18 18:11:57 OSPF6: Got [Network Id:0.0.0.2 Adv:2.2.2.2]
    from 2.2.2.2
2006/04/18 18:11:57 OSPF6: [Network Id:0.0.0.2 Adv:2.2.2.2]
2006/04/18 18:11:57 OSPF6: Age: 1 SeqNum: 0x80000001 Cksum: 56
   bc Len: 32
2006/04/18 18:11:57 OSPF6: Flood: Add [Network Id:0.0.0.2 Adv
   :2.2.2.2] to retrans-list of 3.3.3.3
2006/04/18 18:11:57 OSPF6: Flood: Add [Network Id:0.0.0.2 Adv
   :2.2.2.2] to retrans-list of 4.4.4.4
2006/04/18 18:11:57 OSPF6: At least one neighbor hasn't
   received the LSA..
2006/04/18 18:11:57 OSPF6: Adding LSA([Network Id:0.0.0.2 Adv
   :2.2.2.2]) node to the pending LSA list. Waiting:
2006/04/18 18:11:57 OSPF6: 3.3.3.3
2006/04/18 18:11:57 OSPF6: 4.4.4.4

Second run:

2006/04/18 18:11:57 OSPF6: Got LSA [Network Id:0.0.0.2 Adv
   :2.2.2.2] from 2.2.2.2
2006/04/18 18:11:57 OSPF6: Outgoing interface is not MANET -
   flood [Network Id:0.0.0.2 Adv:2.2.2.2] as specified in RFC
   2338
```

As seen, the Network LSA is run through the flooding procedure twice - one for each of the node's interfaces. The first run is for the MANET interface, and the LSA is added to the pending LSA list. This is correct behavior for MANET interfaces. The second run is for the non-MANET interface, and so the flooding procedure as defined in RFC 2328 is utilized.

8.5.5 The Hello protocol

On non-MANET interfaces, the Hello protocol described in RFC 2328 is utilized. The following output is retrieved from node A's log file:

```
2006/04/18 18:12:12 OSPF6: Including 2.2.2.2 in neighbor list (
   incr Hellos not enabled)
```

Since it is communicating with node B it is implicitly using its OSPFv3 interface. The incremental Hellos mechanism is not utilized on such interfaces, and all neighbors should thus be included in the neighbor list carried in Hello packets.

To summarize, this scenario shows that WOSPF-OR capable nodes operate as expected; on non-WOSPF-OR interfaces they behave as a legacy OSPF node, while on WOSPF-OR interfaces they utilize the overlapping relays scheme and possibly also the incremental Hellos scheme. This feature is very neat, as the WOSPF-OR implementation does not affect normal OSPF behavior - only interfaces explicitly defined in the configuration file utilize the wireless extensions.

8.6 Advertising external prefixes

As all routing updates are received by all routers in a WOSPF-OR networks, advertising connection to external networks is no different from legacy OSPFv3 networks. Consider the network consisting of four WOSPF-OR nodes depicted in figure. Node A has been added a second interface card and configured with the IPv6 prefix `2001:700:500:49::/64`.

When issuing the command `show ipv6 ospf6 spf tree` in node D's vty shell, the following output is generated:

```
+-4.4.4.4 [0]
  +-3.3.3.3 [1]
    +-2.2.2.2 [2]
      +-1.1.1.1 [3]
```

Clearly, the only path to node A is through the other nodes in a multi-hop fashion. Consider the following output generated by issuing the command `show ipv6 ospf6 database intra-prefix det` also in node D's vty shell:

```
    Area Scoped Link State Database (Area 0.0.0.0)

Age: 41 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 1.1.1.1
LS Sequence Number: 0x80000002
CheckSum: 0x92ed Length: 56
```

```
Number of Prefix: 2
Reference: Router Id: 0.0.0.0 Adv: 1.1.1.1
Prefix Options: --|--|--|--
Prefix: 2001:700:500:49::/64
Prefix Options: --|--|--|--
Prefix: 2002:700:500:49::/64
```

The command output shows that node A advertises to the WOSPF-OR network the prefix associated with its new interface card.

Chapter 9

Other Issues

As this thesis focuses on routing, non-routing issues are left for discussion in other research projects. However, deploying OSPFv3 in MANETs *do* have a number of issues (in addition to the reduction of routing overhead implemented in this thesis) that need to be addressed. This chapter deals with some of these issues. Note that most of the issues discussed in this chapter are not WOSPF-OR, as the use of the term "OSPF-MANET" indicates.

9.1 Overview

Although packet radio networks have been deployed by the US army since the 70's, it was not until a few years age these networks developed into popular research topic. The technology is as of yet far from mature, and as a result there are numerous issues needed to be addressed before MANETs can mature sufficiently for more commercial use. OSPFv3 based MANETs are fine examples of this; Of all the different technologies associated with MANETs (consider for example the ISO stack) even the basic functionality of *routing* is under heavy investigation by researchers in every parts of the world.

The mechanisms implemented in this thesis are designed as *intra-area* optimizations only; They optimize OSPFv3's performance in MANETs aiming at reducing routing overhead to an acceptable level. In addition, the multi-hop nature of MANETs are taken into account by having each node (potentially) relay routing information. These features benefit routing overhead within an OSPFv3 area only.

Scaling OSPF-MANETs for larger networks is very important for successful deployment in real-world scenarios. Recall OSPFv3's use of routing areas used for limiting the flooding domain as well as reducing the routers' routing table. Deploying such a scheme in OSPF-MANETs is under heavy investigation, but it not destined to work. Scaling of OSPF-MANETs is further examined in section 9.2

As MANETs are designed to operate in a decentralized manner, some way of configuring IP addresses is required. Some good approached to this have been proposed, and is outlined in

section 9.3.

Deploying the WOSPF-OR interface in stand-alone MANETs will not suffice in a real-life scenario, since users expect to access the Internet and so forth regardless of the underlying network type (i.e. a MANET). Section 9.4 looks into this issue.

9.2 Scalability of WOSPF-OR networks

The wireless extensions to OSPFv3 described in this thesis focus on intra-area routing optimizations. Both the incremental Hellos mechanism and the optimized flooding is designed to minimize the routing overhead within one OSPF area. As stated in the OSPF-MANET Working Group meeting notes of October 2005, the scaling goal using these extensions is 50-100 nodes. For real life usage this may not be sufficient. Thus, better scaling schemes are required.

Standard OSPF has employed *areas* for scaling purposes. In a wired network, configuration of areas is manageable as topology changes are sparse. OSPF-MANETs, on the other hand, must cope with possibly rapid and unpredictable node mobility, and manual configuration of areas is not a very attractive solution unless the user of the OSPF-MANET device is proficient with computer networks - routing should, in the general case, not require much insight into networking, as one cannot assume that the typical user is an OSPF expert.

Baker [3] states two issues that need to be addressed when adapting OSPF to a MANET environment. First, an interface type must be defined specifically for MANET. The extensions implemented in this thesis aims at solving this. Second, one must address the problem of area mobility - how do wandering nodes act when facing nodes of a different area? *Area* configuration in MANETs is a non-trivial task, and this chapter discusses some of the issues on this topic.

9.2.1 OSPF-MANET Area design

OSPF has very good scaling properties by the use of *areas*. Deploying the area model in MANETs, however, is not destined to work because of the problem of area formation in such mobile networks. This is one shortcoming in the OSPF-MANET area design proposed in [11]. Although [11] proposes a two-step solution to area design in such networks using both a classical graph partitioning algorithm as well as an ad-hoc heuristic, wireless links and mobile nodes are not evaluated in this approach. This approach is, nevertheless, very interesting in that it not only clusters the network into areas, but also elects a backbone for the network. Utilizing this in a distributed manner is currently under investigation by the authors.

Area roaming

RFC2328 states that nodes can not become neighbors across *areas*. In a wired network where areas have been manually configures this seems reasonable, as the node mobility is very low. In current OSPF-MANETs scenarios, this restriction in fact *limits* the node mobility as the node must stay within communication range of other nodes in its area. It is possible to detect and auto-join new areas, but this may cause the node to be part of two areas without being configured as an area border router. In addition, if neither areas is the backbone a virtual link to the backbone must be created. [3] states that this is problematic in MANETs.

These issues are just a small selection of problems associated with deploying OSPF's area scheme in MANETs.

9.2.2 Adjacency forming

One drawback of the proposed extensions to OSPF described in this thesis is the excessive forming of adjacencies between neighbor nodes. Cisco's draft describes modifications to the Hello protocol and the forwarding algorithm, but does not address the overhead associated with deploying a point-to-multipoint interface on a MANET; Nodes will form adjacency with every bi-directional neighbor, and in a dense network, protocol packets used for (initial) database synchronization may dominate the bandwidth. Furthermore, mobility increases the overhead since adjacency is formed even with short-time neighbors, so in cases where a node just temporarily is within transmission range of another node they will form adjacency.

Moreover, as wireless links may be relatively unstable for reasons such as distance, radio interference or difference in antenna capacity/strength, a node may waste bandwidth listing neighbors with which it may not be capable of forming even bi-directional connectivity with. This issue has been discussed in [27], which proposes listing one-way neighbors for only a limited number of Hellos.

Smart Peering

The WOSPF-OR extension could clearly benefit from implementing some sort of adjacency reduction scheme. In July 2005 Cisco Systems proposed a complementary draft[28] referred to as *smart peering*, which aims at further optimizing the Overlapping Relays mechanism by reducing the number of adjacencies formed. The basic idea is to *intelligently* form adjacency with only selected neighbors, based on a heuristic which ensures reachability to every possible destination in the network.

The OSPF-MANET working group has incorporated the smart peering extension into the overlapping relays proposal, referring to it as *Overlapping Relays via Smart Peering*. Simulation results presented at the OSPF and MANET working group meetings of November 2005 [16] show that as efficient flooding, up to a certain point, has overhead gains as the number of nodes increase, adjacency reduction will have additional benefits beyond this point. Thus, optimization

of flooding by using the overlapping relays mechanism *only* will get only the routing protocol scalability so far.

9.2.3 Temporary LS database

Another optimization scheme published by Cisco System involves the use of a temporary link state database to store LSAs received from non-adjacent neighbors. This has the benefit that when a new adjacency is formed with a neighbor, the initial database synchronization may be minimized (or even eliminated) as the two neighbors already have an identical copy of the LSAs. In effect, routers listen in promiscuous mode storing all LSAs heard on the link, and possibly reducing the build time for future adjacencies.

9.2.4 Other scalability proposals

Several papers [12][40][4] have been published proposing scalability schemes in MANETs. These may not necessarily have been designed with OSPF explicitly in mind, but adapting these ideas to OSPF could be interesting. For example, the idea behind the Fisheye State Routing (FSR) is to update close neighbors at a higher rate than distant neighbors. Packets transmitted to distant locations are routed more and more precisely the closer they get to the destination. This idea is very interesting in that areas with high mobility are updated more frequently than low-mobility areas, thus limiting frequently routing updates to areas with rapidly changing topology. Moreover, [40] and [4] aim at supporting hierarchical routing (not unlike administrative areas) to MANETs. These schemes might be applicable to OSPF-MANETs.

9.3 Autoconfiguration of IP addresses in OSPF-MANETs

All routable hosts (including routers) on a network must be associated with an unique IP address. In OSPFv3 based MANETs the hosts are associated with *IPv6* addresses. The use of IPv6 in OSPF-MANET has benefits, especially with regards to address configuration. This section describes how routers in a OSPF-MANET (i.e. a MANET consisting of OSPFv3 routers utilizing the WOSPF-OR extensions) may configure IPv6 addresses.

9.3.1 Address autoconfiguration with IPv6

The transition from IPv4 to IPv6 has many advantages, including incorporated security, support for mobility, and the ability of nodes to autoconfigure addresses. The latter is outlined in this subsection.

There are roughly two types of autoconfiguration: *stateful* and *stateless*:

Stateful Rather than manually assigning one or more IPv6 addresses to each host in a network, mechanisms such as DHCPv6 (Dynamic Host Configuration Protocol for IPv6) [26] are utilized; The DHCPv6 server manages and assigns addresses to requesting unconfigured hosts. This centralized approach, in which one entity is responsible for the addresses' uniqueness, is classified as *stateful* address autoconfiguration.

Stateless A host can generate its own address by using a combination of the network prefix(es) announced on the link by the attached routers, and local information such as the MAC address of the network interface card. This address is only a tentative address, and may only be applied to the interface once a Duplicate Address Detection (DAD) scheme has been run (see below). Routers on the link may announce a prefix associated with the subnet, the host may use this information (in addition to a well-known site-local prefix) to construct a site-local address. If no routers are present, a host may generate only a link-local address.

The autoconfiguration schemes described above assumes that the hosts are connected on the same link such as on an Ethernet based LAN. The stateful and stateless approaches complement one another.

Duplicate Address Detection

If the stateless approach described above is used for configuring an IPv6 address, a host utilize a technique for ensuring that its address is unique on the link. This follows from the these observations:

An IPv6 based interface may configure its own IPv6 address based on for example its MAC address. Ideally *all* MAC-addresses should be unique, but since this cannot be assumed due to reasons such as the ability to manually manipulate a network card's MAC address, the nodes must utilize a Duplicate Address Detection (DAD) scheme to ensure address uniqueness.

The host must insure that the address is unique, and issues a *Neighbor Solicitation (NS)* message containing its tentative address. A node in the network will respond with a *Neighbor Advertisement (NA)* message if it finds that it has already configured an address equal to the received tentative address. If the host does not receive any NAs, the tentative address is used on the interface.

[29] describe the process of autoconfiguring IPv6 addresses, and performing DAD in order to ensure the configured address' uniqueness, and the interested reader should consult this and related works for more details on this subject.

9.3.2 Address autoconfiguration in MANETs

In MANETs, the use of a DHCP server is not feasible as this would require the nodes to maintain a connection with the DHCP server. Following this, several proposals ([21][6][19]

are a selection of these) on IPv6 *stateless* address autoconfiguration for MANETs schemes has been published, in which nodes themselves select their address and query the other nodes in the network to assure the address' uniqueness. The proposals take into account the multi-hop nature of MANETs.

Duplicate Address Detection in MANETs

The standard DAD scheme is not suitable on MANETs for several reasons. First, the scheme is not designed for the multi-hop nature of MANETs. Second, since an IP address based on for example a node's MAC address cannot be assumed to be unique, correctly processing address duplication detection in merging of such networks is needed.

Several papers [38][35][21] have been published proposing DAD schemes (often incorporated in proposed autoconfiguration schemes) for MANETs. One of these [35] describes *weak DAD* [1] which informally means that a packet is routed to the intended receiver, even if two nodes have the same address. This is accomplished by associating an unique identifier to each node in the network. This identifier, or key, is added to routing tables (and hence routing packets) to insure correct lookup when forwarding packets. Weak DAD is especially useful for handling address duplication due to partitioning and merging in MANETs.

9.3.3 Router IDs in OSPFv3

OSPFv3 identify routers by their router ID. Consequently, the router ID *must* be unique, otherwise a loop would originate as a result of an OSPF re-flooding what would seem to be a self-originating LSA with erroneous sequence number.

[17] proposes router ID being set to the lowest IP address associated with the router, and this mechanism is already implemented in Quagga's OSPFv3 code. This, however, assumes that the IP address is unique, and the node should run a DAD scheme.

Router IDs in WOSPF-OR networks

Every router in an area is identified by its router ID. However, if two routers within one area is configured with the same router ID, problems arise. This is due to the protocol specification indicating that whenever a router receives a router-LSA from itself with the LSA sequence number greater than the last originated router-LSA sequence number, it issues a new router-LSA with a sequence number greater than the received one. As both routers will perform these actions, an infinite loop will be formed.

RFC 2328 states that router IDs should be unique, but it is not specified *how* a router ID should be chosen. This is implementation specific. The specification *does* propose router IDs being set to the lowest IP address associated with the router.

[1] *Weak* DAD

Typically the router IDs will have to be configured manually. Quagga implements a mechanism for routers to configure router IDs based on the lowest IP address, as proposed above. However, no protocol for performing duplicate router ID checks are defined.

9.4 Internetworking with other networks

This section describes the interaction between OSPF-MANETs and other networks, for example the global Internet.

9.4.1 Global connectivity

Gateways in IPv4-based MANETs often implements a NAT or Mobile IP mechanism to cope with the scarcity of global IPv4 addresses. This, in addition to the use of default routes normally allowed in IPv4 MANETs, greatly complicates global connectivity [10].

Using IPv6-based nodes simplifies external connectivity, especially in multi-homed MANETs. As an IPv6-based MANET node can configure an unique global address based on the prefix of one of the gateways, there is no need for a NAT or Mobile IP agent at the gateways, and no default route.

External access

An OSPF-MANET consisting of nodes implementing the WOSPF-OR interface will operate well on its own. However, as users of such network may wish to access external networks, the OSPF-MANET must provide one or more network gateways. A typical scenario may be a network where the users wish to gain access to the Internet.

To gain access to another network (e.g. the Internet) at least one node must be able to reach it through one of its interfaces. This node, which will act as a gateway for the OSPF-MANET, may for example have connectivity to the other network through an Ethernet interface. The network(s) reachable through its Ethernet interface will be advertised into the OSPF-MANET in link state advertisements.

Multi-Homed MANETs

An OSPF-MANET may consist of multiple gateways to the same network. Such a network is said to be *multi-homed*.

Nodes in a multi-homed OSPF-MANET should (for as long as possible) avoid changing gateways when accessing other networks. For example, when changing a gateway all TCP sessions will break due to the change in paths.

In IPv6 one can work around this problem due to the ability of an OSPF-MANET node to configure a global address itself (as described above). To ensure that outgoing and incoming traffic is routed through a particular gateway, the OSPF-MANET node configures its address from a global prefix managed by the gateway [37].

9.4.2 Interaction with other networks

The OSPF-MANET interface type implemented in this thesis is defined for communicating in OSPF-MANET environments. Being an *interface type* for OSPFv3 merely indicates that when communicating in OSPF-MANETs, the OSPFv3 router should utilize the WOSPF-OR extensions. Communicating with other OSPFv3 networks is very straight forward in that it is merely a matter of adding and configuring network interface cards to these networks. Recall that from the outside, the OSPF-MANET network is considered a point-to-multipoint OSPFv3 network, which is one of the already defined network types supported in RFC 2740. Note that in such cases, *one* OSPFv3 routing daemon is used for all attached OSPF networks, including OSPF-MANETs.

Similarly, communicating with other non-OSPFv3 networks is no different from having non-MANET OSPFv3 routers communicate with these networks. The OSPFv3 routing daemon running on a gateway between the OSPF-MANET and the other network is configured straightforwardly: The OSPFv3 daemon is attached to the OSPF-MANET, while another routing daemon is attached to the other network.

Chapter 10

Conclusions and Future Work

This chapter concludes the thesis. The first section provides some final conclusions on the implementation of the WOSPF-OR extensions, as well as additional remarks from the author. The second section outlines suggestions for future work with regards to implementation and testing.

10.1 Conclusions

In this thesis, a proposal for wireless extensions to OSPFv3 has been designed and implemented. The extensions are defined for operation on OSPF-MANET interfaces, and are based on an Internet-Draft published by Cisco Systems, where a mechanism for optimizing flooding in an OSPF-MANET, and a mechanism for reducing the size of Hello packets, are proposed. These mechanisms are referred to as *overlapping relays* and *incremental Hellos*, respectively. These wireless extensions are in this thesis referred to as WOSPF-OR.

This thesis has resulted in a working implementation of the mechanisms described in Cisco's draft. A testbed based on my implementation has shown that the extensions operate as specified, and that the existing OSPFv3 functionality has not been altered for non-WOSPF-OR interfaces.

The main focus on the proposed extensions was to optimize flooding, utilized through the overlapping relays mechanism. Fortunately, the MPR set election code found in UniK's OLSR implementation could be adapted to the WOSPF-OR AOR set calculation implemented in this thesis, greatly reducing the time spent on this matter. The implementation of reliable flooding and intelligent acknowledgments turned out to be the most challenging tasks, as these mechanisms called for modification to some very basic (but complex) features of OSPFv3. Although the implementation design of these mechanisms may seem quite straight forward, getting the in-depth insight into OSPFv3, as well as the nature of these mechanisms, proved to be very difficult, especially with regards to the actual implementation.

The implementation of the overlapping relays mechanism turned out to be quite intrusive into the OSPFv3 flooding procedure in particular. For this reason it would be interesting to try and

implement the WOSPF-OR extensions as a plugin to OSPFv3, instead of applying the extension code directly into the existing OSPFv3 code. However, one of the design objectives was to avoid making too many modifications to the existing OSPFv3 code, and this objective has with many regards been accomplished - the code implementing OSPFv3's Hello protocol has only been moderately modified, as most of the functionality reside in separate files. This also goes for implementation of the LLS scheme, where close to all of the code resides in separate files.

Being merely a new interface type, the WOSPF-OR extensions should not have any affect on OSPFv3's operation in other network types. Tests have provided showing that non-WOSPF-OR interfaces[1] have not been affected by the implemented MANET extensions.

Although there exists another implementation of Cisco's draft, my implementation should be of interest to researchers on OSPF-MANETs; Design choices such as extending the SCS number field carried in SCS TLVs (see section 7.9.1 on page 86) for security reasons, avoid signaling full state request persistently, and so forth, may provide a basis for enlightening discussions and considerations for further study on the subject.

The goal of this thesis has been to design and implement the wireless extensions to OSPFv3 based on Cisco's draft. It is important to note that issues related to the protocol itself and its performance in WOSPF-OR networks do not fall within the scope of this thesis, so the interested reader should consult the IETF MANET working group for further discussions of such matters.

As the extensions implemented in this thesis are based on a GPL licensed software suite, the same license naturally applies to the resulting software. The source code can be downloaded from this site[2]: *http://folk.uio.no/kenneho/index.php?page=studies&subpage=wospf*.

10.1.1 Additional remarks from the author

The time limit set by a master thesis hand-in deadline is in many ways just long enough to scratch the surface of the subject of study. Doing research as well as the actual implementation with respect to this thesis has left a number of open questions which, due to time constraints, unfortunately must be left to future research and work. Some of these issues were discussed in Chapter 9.

This thesis, as well as related topics which were studied in respect to this thesis, has provided me with a thorough insight into issues with MANETs as well as networking in general.

A lot of time has been put into issues not directly related to the implementation. Such issues include learning some Perl programming, learning how to use and install ssh-keys, learning how to manipulate `ip6tables` on Linux machines, and other technical issues needed for this thesis. Furthermore, quite a few papers on (OSPF-)MANETs, as well as general networking in general, have been studied in order to gain an in-depth knowledge on state-of-the-art technologies and

[1] Actually, only the OSPFv3 broadcast interface was used in the test, but there is nothing that indicates that other interface types are affected by the WOSPF-OR extensions

[2] As of this writing the site has not been created, and will not be operable until some time after this thesis has been submitted

research results. Some of these papers have influenced this thesis with regards to design choices and suggestions for future work.

The active development around Quagga has proven to be quite beneficial to my implementation, as participants on Quagga's forums (and the maintainers in particular) have been more than willing to discuss and clear up both general routing issues, as well as more technical implementation issues.

Finding and learning how to use debugging tools such as the GNU Debugger and valgrind, as well as learning basic programming in Perl, proved to be of great value when developing the OSPF extensions. These examples illustrate what I have found to be one of the most important aspects of education - learning new technologies as needed, and not attempting to re-invent the wheel.

Being such a comprehensive and complex routing protocol, reading up on and understanding OSPFv3 proved to be very challenging. As RFC 2740 for the most part describes differences to RFC 2328, both specifications had to be studied on order to aquire the in-depth knowledge needed for thesis. Put toghether, these specifications consist of some 300 pages.

10.2 Future work

During the process of implementing the extensions described in this thesis, new ideas and solutions have come to mind as a result of a deeper understanding and overview of the problem statement and the technical aspects of the Quagga framework. However, the time constraints set by a master thesis allows only for a limited amount of research and work. Consequently, improving the source code in term of further reducing the amount of (not yet known) bugs as well as implementing smarter and more effective solutions, have not been a very high priority, as this to some extent might have come in the way of having an implementation that works.

Below is a short discussion on some issues related to the design choices made with regards to the implementation. Also, suggestions for further work regarding testing of the implementation are provided.

10.2.1 Implementation

Some of the choices made during both the design process and the implementation process were made with regards to keeping the implementation simple and consistent with the existing OSPFv3 source code. Below is an overview of some changes I would have made to the implementation given I had more time.

Lists VS lists with hashing The list data structure more or less exclusively used in the existing OSPFv3 source code is the `struct list` data structure, which implement a double linked list. For example, the neighbors list implemented on all OSPFv3 interfaces use the `struct list` data structure. The neighbor table, which was implemented with regards

to the wireless extensions, is consistent with this code style, and is also implemented by the use of this data structure. However, for efficiency reasons the neighbor table could be implemented similar to the UniKs OLSR implementation which combines such a double linked list with a hashing function. This reduces the time to perform an item lookup, which would be beneficial to devices having very sparse processing power. However, as OSPFv3 (and hence WOSPF-OR) is a relatively CPU- and memory consuming routing protocol, it will probably be run on devices having greater processing power.

OSPFv3 source code modification Most of the WOSPF-OR specific parts within the OSPFv3 source code are very small, typically implementing only message intercept functionality. As noted above, however, the modifications to OSPFv3's flooding procedure are quite intrusive. To minimize the impact of the extension code other design choices could be made. One such choice would be to make a WOSPF-OR version of each relevant OSPFv3 function, only making modifications to the lines where these functions are to be called. In other words, instead of modifying for example parts of the `ospf6_flood` function, a new function named `wospf_flood` could be implemented, consisting of the modifications to the OSPFv3 source code necessary to implement the wireless extensions. This would lessen the impact of WOSPF-OR specific code, as only calls to the `ospf6_flood` function would have to be replaced with `wospf_flood`. Furthermore, this idea could be extended to implement the extension code as plug-ins to the source code. The scheme would be very similar, with the main difference being that the WOSPF-OR source code would reside in a plug-in library instead.

Parsing LLS data blocks When receiving LLS data blocks, the contents of the data block are processed and analyzed. The existing code for this task could benefit from being rewritten; The incoming LLS data block should be parsed into internal messages (i.e. structs). A parsing function could build these messages, and the contents could be interpreted by other functions. This would be a streamlined way of processing incoming LLS data blocks.

Memory leaks Debugging tools such as *valgrind* indicates that some memory leakage is present in the implementation. However, this is caused not only by the extensions implemented in this thesis, but also the existing code. Some of the leaks in the code have been removed[3], but further work on this matter is needed.

Rescheduling of explicit acknowledgments sent by non-AORs The non-AORs have been designed to respond with an explicit acknowledgment for each received LSA. Many acknowledgments may, as already described, be bundled into on LS Ack packet in order to reduce the cost of transmitting the acknowledgments. Nevertheless, if pushbacked LSAs are re-flooded, the re-flood will serve as an implicit acknowledgment. An optimization might be to suppress the transmission of explicit acknowledgments until the decision whether to re-flood the pushbacked LSA or not have been made. It would be interesting

[3]It was not until quite late in the implementation process that I came across, and started using, valgrind for debugging and for locating memory leaks. More leaks might have been removed if I had come across this tool at an earlier stage.

to look into this matter, as environments in which non-AORs frequently must re-flood pushbacked LSAs might benefit from utilizing such a scheme.

Note that the mentioned improvements are, as the name suggests, *improvements* - the implementation does work, but to (slightly) increase performance and code readability the improvements could be implemented.

There are two small features that, due to time constraints, have not yet been implemented. The first is removal of old acknowledgments from WOSPF-OR neighbors' acked LSA list (i.e. the acked LSA list maintained in each neighbor's data structure). Unless explicitly removed, entries in this list should time out. Note that while this detail does result in a memory leak, this leakage is very small. However, this feature should be implemented before the protocol is to be deployed for a long-term service. The second feature is the use of the `DoNotAge` bit and the `DC`-bit, described in section 5.3.1. This feature is used in order to avoid refreshing and flooding of already known information. However, as LSA updates are flooded relatively seldom once the network has converged, this feature should not be the most significant routing overhead factor in WOSPF-OR networks.

10.2.2 Testing

A routing protocol such as OSPFv3 should, in theory, be independent of the underlying link technology - whether the link is wireless or not should not be of great importance. For this reason the testbed used in this thesis has made use of Ethernet network cards.

I have had four machines at my disposal, which have been sufficient for implementing and testing the functionality of the WOSPF-OR extensions. However, further implementation on testbeds consisting of even more nodes could reveal bugs or other weaknesses in the code, and should therefore be performed before the code is considered stable.

Performing tests on real-life wireless ad hoc networks would thus be of interest, as one could study the effects on lossy links, hidden node problem, and so forth. Naturally, these tests are not very relevant to the implementation of the protocol itself, but rather for performance analysis. Finally, a study of the interaction between wired and wireless OSPF in real-world networks should be conducted. As of this writing, OSPFv3 based networks are not much deployed, and I have hence not had the opportunity to have the WOSPF-OR network interact with real-world OSPFv3 networks. Looking into this matter would be of interest, as one of the wireless extension's objectives is to make the junction point between wired and wireless OSPFv3 networks as fluid as possible.

Bibliography

[1] J. Ahrenholz, T. Henderson, P. Spagnolo, E. Baccelli, T. Clausen, and P. Jacquet. "OSPFv2 Wireless Interface Type". IETF Internet Draft, draft-spagnolo-manet-ospf-wireless-interface-01.txt (expired), May 2004.

[2] E. Baccelli, T. Clausen, and P. Jacquet. "OSPF MPR extension for ad hoc networks". IETF Internet Draft, draft-baccelli-ospf-mpr-ext-01.txt, March 2006. Work in progress.

[3] F. Baker. "An outsider's view of MANET". IETF Internet Draft, draft-baker-manet-review-01.txt (expired), March 2002.

[4] Josh Broch, David A. Maltz, and David B. Johnson. "Supporting Hierarchy and Heterogeneous Interfaces in Multi-Hop Wireless Ad Hoc Networks". Proceedings of I-SPAN'99, June 1999.

[5] M. Chandra. "Extension to OSPF to Support Mobile Ad Hoc Networking". IETF Internet Draft, draft-chandra-ospf-manet-ext-03.txt (expired), April 2005.

[6] T. Clausen and E. Baccelli. "Simple MANET Address Autoconfiguration". IETF Internet Draft, draft-clausen-manet-address-autoconf-00.txt (expired), January 2005.

[7] T. Clausen and P. Jacquet. "Optimized Link State Routing (OLSR) Protocol ". RFC 3626, October 2003.

[8] IEEE Computer Society LAN MAN Standards Committee. *IEEE 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[9] S. Corson and J. Macker. "Mobile Ad Hoc Networking (MANET)". RFC 2501, January 1999. Informational edition.

[10] P. E. Engelstad, A. Tønnesen, A. Hafslund, and G. Egeland. "Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks". Proceedings of IEEE ICC'04, Paris, France, June 2004.

[11] S. Galli, H. Luss, J. Sucec, A. McAuley, S. Samtani, D. Dubois, K. DeTerry, R. Stewart, and B. Kelley. "A Novel Approach to OSPF-Area Design for Large Wireless Ad-Hoc Networks". Proceedings of IEEE ICC'05, Seoul, Korea, 2005.

[12] Tsu-Wei Chen Guangyu Pei, Mario Gerla. "Fisheye State Routing in Mobile Ad Hoc Networks". Proceedings of ICDCS'00, Taipei, Taiwan, April 2000.

[13] A. Hafslund, A. Tønnesen, R. B. Rotvik, J. Andersson, and Ø. Kure. "Secure Extension to the OLSR protocol". Proceedings of the OLSR Interop and Workshop, San Diego, 2004.

[14] C. Hedrick. "Routing Information Protocol (RIP)". RFC 1058, June 1988.

[15] The GNU Project Debugger. http://www.gnu.org/software/gdb/.

[16] OSPF and MANET WG meetings. Proceedings of IETF 64, November 2005.

[17] J. Moy. "OSPFv2 version 2". RFC 2328, April 1998.

[18] C. Siva Ram Murthy and B. S. Manoj. *Ad Hoc Wireless Networks, Architectures and Protocols*. Prentice Hall, 2004.

[19] S. Nesargi and R. Prakash. "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network". Proceedings of IEEE INFOCOM'02, New York, USA, June 2002.

[20] R. Ogier and P. Spagnolo. "MANET Extension to OSPF using CDS Flooding". IETF Internet Draft, draft-ogier-manet-ospf-extension-07.txt, March 2006. Work in progress.

[21] C. Perkins, J. T. Malinen, R. Wakikawa, E. Belding-Royer, and Y. Suan. "IP Address Autoconfiguration for Ad Hoc Networks". IETF Internet Draft, draft-ietf-manet-autoconf-01.txt (expired), November 2001.

[22] Charles E. Perkins. "Mobile networking in the Internet". *Mob. Netw. Appl.*, 3(4):319–334, 1999.

[23] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. "Ad hoc On-Demand Distance Vector (AODV) Routing Protocol". RFC 3561, July 2003.

[24] IETF MANET OSPF design team code and documentation repository. http://hipserver.mct.phantomworks.org/ietf/ospf/.

[25] The Quagga Routing Suite. www.quagga.org.

[26] Ed. R. Droms. "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)". RFC 3315, July 2003.

[27] F. Templin R. Ogier and M. Lewis. "Topology Dessemination Based on Reverse-Path Forwarding (TBRPF)". IETF Internet Draft, draft-ietf-manet-tbrpf-05.txt (expired), July 2003.

[28] A. Roy and Ed. "Adjacency Reduction in OSPF using SPT Reachability". IETF Internet Draft, draft-roy-ospf-smart-peering-01.txt, November 2005. Work in progress.

[29] T. Narten S. Thomson. "IPv6 Stateless Address Autoconfiguration". IETF Internet Draft, RFC 2462 (expired), December 1998.

[30] Y.-S. Chen S.-Y. Ni, Y.-C. Tseng and J.-P. Sheu. "The Broadcast Storm Problem in a Mobile Ad Hoc Network". Proceedings of MOBICOM'99, Seattle, Washington, USA, August 1999.

[31] P. A. Spagnolo T. R. Henderson and J. H. Kim. "A Wireless Interface Type for OSPF". Proceedings of IEEE MILCOM'03, Boston, USA, October 2003.

[32] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[33] The BIRD Internet Routing Daemon. http://bird.network.cz.

[34] eXtensible Open Router Platform. www.xorp.org.

[35] N. H. Vaidya. "Weak Duplicate Address Detection in Mobile Ad Hoc Networks". Proceedings of MOBIHOC'02, New York, USA, June 2002.

[36] Valgrind. http://valgrind.org.

[37] R. Wakikawa, J. Malinen, C. Perkins, A. Nilsson, and A. Tuominen. "Global connectivity for IPv6 Mobile Ad Hoc Networks". IETF Internet Draft, draft-wakikawa-manet-global6-04.txt (expired), July 2005.

[38] K. Weniger. "Passive Duplicate Address Detection in Mobile Ad Hoc Networks". Proceedings of IEEE WCNC'03, March 2003.

[39] Boeing Phantom Works. "Interlayer routing issues for wireless networks". NRL Cross-Layer Workshop, June 2004. Slides presentation.

[40] L. Villasenor Y. Ge, L. Lamont. "Improving Scalability of Heterogeneous Wireless Networks with Heirarchical OLSR". Proceedings of WIMOB'05, August 2004.

[41] The Zebra Routing Suite. www.zebra.org.

[42] A. Zinin, B. Friedman, A. Roy, L. Nguyen, and D. Yeung. "OSPF Link-local Signaling". IETF Internet Draft, draft-nguyen,ospf-lls-05.txt (expired), September 2004.

Appendix A

Log results

A.1 Full meshed four node scenario

Four nodes in a full mesh.

```
50:52: OSPF6d (Quagga-0.98.5 ospf6d-0.9.7o) starts: vty@2606
50:52: ** MY WILL: 200

The local node’s willingness to serve as an AOR is 200.

50:54: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
50:54: Neighbor state change 1.1.1.1%eth0: [Down]->[Init]
50:58: Neighbor state change 4.4.4.4%eth0: [Down]->[Init]
50:58: Neighbor state change 4.4.4.4%eth0: [Init]->[ExStart]
50:58: Neighbor state change 4.4.4.4%eth0: [ExStart]->[ExChange
   ]
50:58: Neighbor state change 4.4.4.4%eth0: [ExChange]->[Loading
   ]
50:58: Neighbor state change 4.4.4.4%eth0: [Loading]->[Full]
51:00: Changes in the AOR set: New AOR(s)

Node D has become adjacent to the one or both of node A and
   node C,
and this is advertised in D’s router LSA. Node D is elected
as an AOR for the local node, since the other nodes are not
   reachable
by the local node.

51:12: Signaling my election of AORs (pers: 3):
51:12: 4.4.4.4
```

```
51:12: Neighbor state change 1.1.1.1%eth0: [Init]->[ExStart]
51:12: Neighbor state change 3.3.3.3%eth0: [Init]->[ExStart]
51:12: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]
51:12: Neighbor state change 1.1.1.1%eth0: [ExStart]->[ExChange
   ]
51:12: Neighbor state change 3.3.3.3%eth0: [ExChange]->[Full]
51:12: Neighbor state change 1.1.1.1%eth0: [ExChange]->[Full]
51:12: Changes in the AOR set: Dropped AOR(s)

The local node can now flood LSAs directly to the other nodes
   in the
network. No need for node D to relay LSAs.

51:24: Signaling my election of dropped AORs (pers: 3):
51:24: 4.4.4.4

Signaling dropped AORs are done persistently in 3 consecutive
   packets.

51:34: Signaling my election of dropped AORs (pers: 2):
51:34: 4.4.4.4
51:44: Signaling my election of dropped AORs (pers: 1):
51:44: 4.4.4.4
```

A.2 Four nodes in a line

Three nodes in a line formation. Node 4 moves within transmission range of node 3 for a short while.

```
22:52: OSPF6d (Quagga-0.98.5 ospf6d-0.9.7o) starts: vty@2606
22:52: Originating [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
22:52: Originating [Link Id:0.0.0.2 Adv:4.4.4.4]
Unblocking connection between 4 and 3

The link is now up.

23:28: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
23:32: Neighbor state change 3.3.3.3%eth0: [Init]->[ExStart]
23:32: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]
23:32: Got [Link Id:0.0.0.2 Adv:1.1.1.1] from 3.3.3.3
23:32: Got [Link Id:0.0.0.2 Adv:2.2.2.2] from 3.3.3.3
```

```
23:32: Got [Link Id:0.0.0.2 Adv:3.3.3.3] from 3.3.3.3
23:32: Got [Router Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
23:32: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3
23:32: Got [Router Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
23:32: New twohop neighbor: 2.2.2.2 (via 3.3.3.3)
23:32: Got [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
23:32: Got [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3
23:32: Got [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3

The LS databases are now synchronized.

23:32: Neighbor state change 3.3.3.3%eth0: [ExChange]->[Full]
23:32: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
23:32: Changes in the AOR set: New AOR(s)
23:32: Got [Router Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
23:34: 3.3.3.3 acked [Router Id:0.0.0.0 Adv:4.4.4.4] - remove
   from retrans list

Node C acknowledged the local nodes's router LSA.

23:44: Signaling my election of AORs (pers: 3):
23:44: 3.3.3.3
23:52: Originating [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
23:52: Originating [Link Id:0.0.0.2 Adv:4.4.4.4]
23:52: Implicit acknowledgement(reflood of [Intra-Prefix Id
   :0.0.0.0 Adv:4.4.4.4]) from 3.3.3.3!

Be an AOR for the local node, node C reflooded the LSA.

23:52: Implicit acknowledgement(reflood of [Link Id:0.0.0.2 Adv
   :4.4.4.4]) from 3.3.3.3!
23:54: Got [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3

Node is reflooding LSAs received by its neighbor (node B). This
    may be
due to a pushbacked LSA timeout, or that node B has elected it
   to
serve as an AOR.

23:54: Got [Link Id:0.0.0.2 Adv:2.2.2.2] from 3.3.3.3
23:54: Signaling my election of AORs (pers: 2):
23:54: 3.3.3.3
23:56: Got [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
23:56: Got [Link Id:0.0.0.2 Adv:1.1.1.1] from 3.3.3.3
```

```
23:56: Got [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
23:56: Got [Link Id:0.0.0.2 Adv:3.3.3.3] from 3.3.3.3
24:04: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3
24:04: Got [Router Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
24:16: Signaling my election of AORs (pers: 1):
24:16: 3.3.3.3
```

A.3 Two static nodes

Two nodes utilizing the incremental Hellos mechanism.

```
35:09: OSPF6d (Quagga-0.98.5 ospf6d-0.9.7o) starts: vty@2606
35:09: ’ My hostname is hal2004 (WOSPF-OR router) ’
35:09: ** MY WILL: 200
35:21: Change in state: new neighbor w/incr Hellos!

Got a Hello packet with the I bit set from node D.

35:21: Increasing my SCS number from 1 to 2
35:21: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
35:29: Including 3.3.3.3 in neighbor list
35:29: NOT setting the SCS TLV’s N bit

This packet contains full state.

35:29: Neighbor state change 3.3.3.3%eth0: [Init]->[ExStart]
35:29: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]
35:29: Neighbor state change 3.3.3.3%eth0: [ExChange]->[Full]
35:41: Including 3.3.3.3 (pers: 1) in neighbor list

This information is sent persistently (however, only once).

35:41: Increasing 3.3.3.3’s SCS number from 1 to 2

Node C has a new neighbor (the local node), and has increased
   its SCS
number.

35:51: Not including 3.3.3.3 in neighbor list...

Node C supports the incremental Hellos mechanism, so do not
   include
```

```
its router ID in the Hello packets neighbor list.
```

A.4 Third node wandering

Line formation on startup. Node 1 does not support incremental Hellos.

```
34:29: OSPF6d (Quagga-0.98.5 ospf6d-0.9.7o) starts: vty@2606
34:29: ’ My hostname is lapdance (WOSPF-OR router) ’
34:29: ** MY WILL: 200
34:33: Change in state: new neighbor w/incr Hellos!
34:33: Increasing my SCS number from 1 to 2
34:33: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
34:39: Including 3.3.3.3 in neighbor list
34:39: NOT setting the SCS TLV’s N bit
34:39: Neighbor state change 3.3.3.3%eth0: [Init]->[ExStart]
34:39: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]
34:39: Neighbor state change 3.3.3.3%eth0: [ExChange]->[Full]
34:55: Increasing 3.3.3.3’s SCS number from 1 to 2
34:57: Including 3.3.3.3 (pers: 1) in neighbor list
Unblocking connection between 4 and 2

The local node moves withing transmission range of node B.

35:07: Not including 3.3.3.3 in neighbor list...
Unblocking connection between 4 and 1

The local node moves withing transmission range of node A.

35:13: Change in state: new neighbor w/incr Hellos!
35:13: Increasing my SCS number from 2 to 3

New incremental Hello capable neighbor.

35:13: Neighbor state change 2.2.2.2%eth0: [Down]->[Init]
35:13: Full state requested by 2.2.2.2

Node B may have overheard the local node’s incremental Hello
   just
sent. This causes node 2 to request full state.

35:13: Expected SCS 1 from 2.2.2.2, got 3 -> request full state
```

```
Node B has undergone state changes. The local node must request
    full state.

35:13: Neighbor state change 2.2.2.2%eth0: [Init]->[ExStart]
35:13: Neighbor state change 2.2.2.2%eth0: [ExStart]->[ExChange
   ]
35:13: Neighbor state change 2.2.2.2%eth0: [ExChange]->[Full]
35:15: Not resetting HelloInterval - (Req) FS list is not empty
35:17: Including 2.2.2.2 (pers: 1) in neighbor list
35:17: Not including 3.3.3.3 in neighbor list...
35:17: NOT setting the SCS TLV's N bit
35:17: Requesting FS from 1 of 2 neighbors, dropping the TLV
35:17: Setting the R bit

The threshold for omitting a ''Req From''-TLV is set to 0.5,
   meaning
that if the local node is requesting full state from at least
   half the
neighbors, then the TLV is omitted. All neighbors will
   interpret this
as a full state request.

35:17: 1 of 2 neighbors requesting full state, dropping the TLV
35:17: Setting the FS bit

Similar to the ''Req FS''-TLV as described above.

35:23: Got full state: Increasing 2.2.2.2's SCS number from 1
   to 3
35:25: Ignoring 3.3.3.3's Hello packet: FS bit set, but SCS is
   same

Node C responded to the local node's request for full state,
   but the
local node is not missing any state with regards to node C.

35:27: Change in state: new neighbor!
35:27: Neighbor state change 1.1.1.1%eth0: [Down]->[Init]
35:27: Neighbor state change 1.1.1.1%eth0: [Init]->[ExStart]
35:27: Neighbor state change 1.1.1.1%eth0: [ExStart]->[ExChange
   ]
35:27: Neighbor state change 1.1.1.1%eth0: [ExChange]->[Full]
35:27: 1.1.1.1 does not support incr Hellos - including in
   neighbor list
```

```
35:27: Not including 2.2.2.2 in neighbor list...
35:27: Not including 3.3.3.3 in neighbor list...
35:27: NOT setting the SCS TLV's N bit
35:43: 1.1.1.1 does not support incr Hellos - including in
   neighbor list
35:43: Not including 2.2.2.2 in neighbor list...
35:43: Not including 3.3.3.3 in neighbor list...
```

A.5 Testing both Overlapping Relays and incremental Hellos

Full test, node D.

```
31:49: OSPF6d (Quagga-0.98.5 ospf6d-0.9.7o) starts: vty@2606
31:49: ' My hostname is hal2004 (WOSPF-OR router) '
31:49: ** MY WILL: 200
31:49:
31:49: Originating [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
31:49: [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
31:49: Age: 0 SeqNum: 0x80000001 Cksum: 33fb Len: 44
31:49: Sending LSA(s) on interface:
31:49: [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
31:49: Age: 0 SeqNum: 0x80000001 Cksum: 33fb Len: 44
31:49:
31:49: Originating [Link Id:0.0.0.2 Adv:4.4.4.4]
31:49: [Link Id:0.0.0.2 Adv:4.4.4.4]
31:49: Age: 0 SeqNum: 0x80000001 Cksum: 5e66 Len: 56
31:49: Sending LSA(s) on interface:
31:49: [Link Id:0.0.0.2 Adv:4.4.4.4]
31:49: Age: 0 SeqNum: 0x80000001 Cksum: 5e66 Len: 56

Unblocking connection between 4 and 3

The link between node C and the local node is now up.

32:56: Change in state: new neighbor w/incr Hellos!
32:56: Increasing my SCS number from 1 to 2
32:56: ** Neighbor table:
32:56: 1: 3.3.3.3 (will: 110)
32:56: Neighbor state change 3.3.3.3%eth0: [Down]->[Init]
32:56: Got a new incremental Hello with no state change ->
   requesting full state from 3.3.3.3
```

```
The link became operational just before node C sent an
   incremental
Hello packet (to node B).

32:59: Including 3.3.3.3 in neighbor list
32:59: NOT setting the SCS TLV's N bit
32:59: Requesting FS from 1 of 1 neighbors, dropping the TLV

The threshold for omitting the ''Req FS from''-TLV is 0.5.

32:59: Neighbor state change 3.3.3.3%eth0: [Init]->[ExStart]
32:59: Neighbor state change 3.3.3.3%eth0: [ExStart]->[ExChange
   ]

Synchronize databases.

32:59: Sending LS Req to 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:1.1.1.1]
32:59: [Link Id:0.0.0.2 Adv:2.2.2.2]
32:59: [Link Id:0.0.0.2 Adv:3.3.3.3]
32:59: [Router Id:0.0.0.0 Adv:1.1.1.1]
32:59: [Router Id:0.0.0.0 Adv:2.2.2.2]
32:59: [Router Id:0.0.0.0 Adv:3.3.3.3]
32:59: [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1]
32:59: [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2]
32:59: [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3]
32:59:
32:59: Got [Link Id:0.0.0.2 Adv:1.1.1.1] from 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:1.1.1.1]
32:59: Age: 69 SeqNum: 0x80000001 Cksum: f325 Len: 56
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Delayed acknowledgement scheduled in 2 seconds

Note that LSAs are flooded along links in state EXCHANGE or
greater. As soon as a neighbor transitions to state EXCHANGE,
   LSAs
received from this neighbor are run through the flooding
   procedure.

32:59: Got [Link Id:0.0.0.2 Adv:2.2.2.2] from 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:2.2.2.2]
32:59: Age: 70 SeqNum: 0x80000001 Cksum: afb1 Len: 56
32:59: 3.3.3.3 is the sender if the LSA..
```

```
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Link Id:0.0.0.2 Adv:3.3.3.3] from 3.3.3.3
32:59: [Link Id:0.0.0.2 Adv:3.3.3.3]
32:59: Age: 67 SeqNum: 0x80000001 Cksum: aef3 Len: 56
32:59: 3.3.3.3 originated the LSA..
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Router Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
32:59: [Router Id:0.0.0.0 Adv:1.1.1.1]
32:59: Age: 61 SeqNum: 0x80000001 Cksum: 26e9 Len: 40
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3
32:59: [Router Id:0.0.0.0 Adv:2.2.2.2]
32:59: Age: 60 SeqNum: 0x80000002 Cksum: 3daf Len: 56
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Router Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
32:59: [Router Id:0.0.0.0 Adv:3.3.3.3]
32:59: Age: 59 SeqNum: 0x80000001 Cksum: e91e Len: 40
32:59: 3.3.3.3 originated the LSA..
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: New twohop neighbor: 2.2.2.2 (via 3.3.3.3)
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1] from 3.3.3.3
32:59: [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1]
32:59: Age: 69 SeqNum: 0x80000001 Cksum: f650 Len: 44
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
```

```
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3
32:59: [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2]
32:59: Age: 70 SeqNum: 0x80000001 Cksum: 0b34 Len: 44
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59:
32:59: Got [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
32:59: [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3]
32:59: Age: 67 SeqNum: 0x80000001 Cksum: 1f18 Len: 44
32:59: 3.3.3.3 originated the LSA..
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
32:59: Got LS Req from 3.3.3.3 - send [Link Id:0.0.0.2 Adv
   :4.4.4.4] unicast

Node C doesn't have a copy of this LSA, and is hence requesting
    a copy.

32:59: Got LS Req from 3.3.3.3 - send [Intra-Prefix Id:0.0.0.0
   Adv:4.4.4.4] unicast
32:59: Sending LSA(s) UNICASTS to neighbor 3.3.3.3:
32:59: [Link Id:0.0.0.2 Adv:4.4.4.4]
32:59: Age: 70 SeqNum: 0x80000001 Cksum: 5e66 Len: 56
32:59: [Intra-Prefix Id:0.0.0.0 Adv:4.4.4.4]
32:59: Age: 70 SeqNum: 0x80000001 Cksum: 33fb Len: 44
32:59: retransmitting LSA(s):
32:59: Neighbor state change 3.3.3.3%eth0: [ExChange]->[Full]
32:59:
32:59: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
32:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
32:59: Age: 0 SeqNum: 0x80000001 Cksum: 21dd Len: 40
32:59: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 3.3.3.3
32:59: Sending LSA(s) on interface:
32:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
32:59: Age: 0 SeqNum: 0x80000001 Cksum: 21dd Len: 40
32:59: Adding 3.3.3.3 to new_aor list
```

```
32:59: Changes in the AOR set: New AOR(s)
32:59: [Router Id:0.0.0.0 Adv:3.3.3.3] was received within
   MinLSarrival, but don't discard
32:59:
32:59: Got [Router Id:0.0.0.0 Adv:3.3.3.3] from 3.3.3.3
32:59: [Router Id:0.0.0.0 Adv:3.3.3.3]
32:59: Age: 1 SeqNum: 0x80000002 Cksum: 04dc Len: 56
32:59: 3.3.3.3 originated the LSA..
32:59: 3.3.3.3 is the sender if the LSA..
32:59: All my neighbors have received the LSA - abort flooding
32:59: Acknowledgements are due for transmission in less than 2
    sec
33:01: Sending the following acks on the interface:
33:01: [Link Id:0.0.0.2 Adv:1.1.1.1]
33:01: [Link Id:0.0.0.2 Adv:2.2.2.2]
33:01: [Link Id:0.0.0.2 Adv:3.3.3.3]
33:01: [Router Id:0.0.0.0 Adv:1.1.1.1]
33:01: [Router Id:0.0.0.0 Adv:2.2.2.2]
33:01: [Router Id:0.0.0.0 Adv:3.3.3.3]
33:01: [Intra-Prefix Id:0.0.0.0 Adv:1.1.1.1]
33:01: [Intra-Prefix Id:0.0.0.0 Adv:2.2.2.2]
33:01: [Intra-Prefix Id:0.0.0.0 Adv:3.3.3.3]
33:01: Not resetting HelloInterval - (Req) FS list is not empty
33:02: 3.3.3.3 acked [Router Id:0.0.0.0 Adv:4.4.4.4] - remove
   from retrans list
33:02: [Router Id:0.0.0.0 Adv:4.4.4.4]
33:02: Age: 4 SeqNum: 0x80000001 Cksum: 21dd Len: 40
33:02: Received a duplicate LSA [Link Id:0.0.0.2 Adv:4.4.4.4]
   from 3.3.3.3

The local node has not yet signaled node C to act as an AOR.
   This LSA
is relayed by node C after a pushback interval.

33:02: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 3.3.3.3
33:02: Received a duplicate LSA [Intra-Prefix Id:0.0.0.0 Adv
   :4.4.4.4] from 3.3.3.3
33:06: Got full state: Increasing 3.3.3.3's SCS number from 1
   to 3
33:09: Including 3.3.3.3 in neighbor list
33:09: Signaling my election of AORs (pers: 3):
33:09: 3.3.3.3
33:19: Signaling my election of AORs (pers: 2):
```

```
33:19: 3.3.3.3
33:29: Signaling my election of AORs (pers: 1):
33:29: 3.3.3.3

Unblocking connection between 4 and 2

The local node can now reach node B directly.

33:53: Change in state: new neighbor w/incr Hellos!
33:53: Increasing my SCS number from 2 to 3
33:53: ** Neighbor table:
33:53: 1: 3.3.3.3 (will: 199)
33:53: Neighbor 1: 2.2.2.2

Node B is a neighbor of node C.

33:53: 2: 2.2.2.2 (will: 110)

The local node has not received any new router LSAs from node B
   , and
does not know of node B's neighbors.

33:53: Neighbor state change 2.2.2.2%eth0: [Down]->[Init]
33:53: Got a new incremental Hello with no state change ->
   requesting full state from 2.2.2.2
33:59: Including 2.2.2.2 in neighbor list
33:59: NOT setting the SCS TLV's N bit
33:59: Requesting FS from 1 of 2 neighbors, dropping the TLV
33:59: Neighbor state change 2.2.2.2%eth0: [Init]->[ExStart]
33:59: Neighbor state change 2.2.2.2%eth0: [ExStart]->[ExChange
   ]

The databases are already synchronized, since other nodes (node
    C in
this case) have relayed routing information continuously.

33:59: Neighbor state change 2.2.2.2%eth0: [ExChange]->[Full]
33:59:
33:59: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
33:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
33:59: Age: 0 SeqNum: 0x80000002 Cksum: 5689 Len: 56
33:59: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 2.2.2.2
```

```
33:59: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 3.3.3.3
33:59: Sending LSA(s) on interface:
33:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
33:59: Age: 0 SeqNum: 0x80000002 Cksum: 5689 Len: 56
33:59: Adding 3.3.3.3 to dropped_aor list
33:59: Changes in the AOR set: Dropped AOR(s)
33:59:
33:59: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 2.2.2.2
33:59: [Router Id:0.0.0.0 Adv:2.2.2.2]
33:59: Age: 1 SeqNum: 0x80000003 Cksum: b80d Len: 72
33:59: Flood: Add [Router Id:0.0.0.0 Adv:2.2.2.2] to retrans-
   list of 3.3.3.3
33:59: 2.2.2.2 originated the LSA..
33:59: 2.2.2.2 is the sender if the LSA..
33:59: At least one neighbor hasn't received the LSA..
33:59: Adding LSA([Router Id:0.0.0.0 Adv:2.2.2.2]) node to the
   pending LSA list. Waiting:
33:59: 3.3.3.3

If node C haven't acknowledged the LSA within AckInterval
   seconds, the
local node will transmitt it on the interface.

33:59: New twohop neighbor: 1.1.1.1 (via 2.2.2.2)
33:59: New twohop neighbor: 3.3.3.3 (via 2.2.2.2)
33:59: Delayed acknowledgement scheduled in 2 seconds
33:59: Received a duplicate LSA [Router Id:0.0.0.0 Adv:2.2.2.2]
    from 3.3.3.3
33:59: Implicit acknowledgement(reflood of [Router Id:0.0.0.0
   Adv:2.2.2.2]) from 3.3.3.3!
33:59: Removing [Router Id:0.0.0.0 Adv:2.2.2.2] from 3.3.3.3's
   retrans list
33:59: [Router Id:0.0.0.0 Adv:2.2.2.2]
33:59: Age: 2 SeqNum: 0x80000003 Cksum: b80d Len: 72
33:59: 3.3.3.3 acked the pending LSA [Router Id:0.0.0.0 Adv
   :2.2.2.2], removed from backup wait list
33:59: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 3.3.3.3

Node C is still a AOR for the local node, and have therefore
   reflooded
the LSA.
```

```
33:59: Implicit acknowledgement(reflood of [Router Id:0.0.0.0
   Adv:4.4.4.4]) from 3.3.3.3!
33:59: Removing [Router Id:0.0.0.0 Adv:4.4.4.4] from 3.3.3.3's
   retrans list
33:59: [Router Id:0.0.0.0 Adv:4.4.4.4]
33:59: Age: 2 SeqNum: 0x80000002 Cksum: 5689 Len: 56
34:01: Sending the following acks on the interface:
34:01: [Router Id:0.0.0.0 Adv:2.2.2.2]
34:01: Adding 2.2.2.2 to new_aor list
34:01: Changes in the AOR set: New AOR(s)
34:02: Not resetting HelloInterval - (Req) FS list is not empty

As LLS blocks are appended Hello packets, and in this case
   there are
information waiting to be signaled in the LLS block, the next
   Hello
packet is not rescheduled for later transmission.

34:02: Pushback timeout...
34:02: Flooding [Router Id:0.0.0.0 Adv:2.2.2.2] will result in
   a redundant transmission - abort
34:02: 2.2.2.2 acked [Router Id:0.0.0.0 Adv:4.4.4.4] - remove
   from retrans list
34:02: [Router Id:0.0.0.0 Adv:4.4.4.4]
34:02: Age: 4 SeqNum: 0x80000002 Cksum: 5689 Len: 56
34:02: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 2.2.2.2
34:03: Full state requested by 2.2.2.2
34:03: Got full state: Increasing 2.2.2.2's SCS number from 1
   to 3
34:06: Ignoring 3.3.3.3's Hello packet: FS bit set, but SCS is
   same
34:09: Including 2.2.2.2 in neighbor list
34:09: 1 of 2 neighbors requesting full state, dropping the TLV

The threshold for omitting the ''FS for''-TLV is 0.5.

34:09: Signaling my election of AORs (pers: 3):
34:09: 2.2.2.2

Changes in the AOR set is signaled in three consecutive packets
   .

34:09: Signaling my election of dropped AORs (pers: 3):
```

```
34:09: 3.3.3.3

Blocking connection between 4 and 3

34:19: Signaling my election of AORs (pers: 2):
34:19: 2.2.2.2
34:19: Signaling my election of dropped AORs (pers: 2):
34:19: 3.3.3.3
34:29: Signaling my election of AORs (pers: 1):
34:29: 2.2.2.2
34:29: Signaling my election of dropped AORs (pers: 1):
34:29: 3.3.3.3
34:46: Neighbor state change 3.3.3.3%eth0: [Full]->[Down]
34:46: 3.3.3.3's state transitioned to less than EXCHANGE:
   Delete from neighbor table
34:46: Change in state: deleting neighbor 3.3.3.3!
34:46: ** Neighbor table:
34:46: 1: 2.2.2.2 (will: 200)
34:46: Neighbor 1: 1.1.1.1
34:46: Neighbor 2: 3.3.3.3
34:46: Increasing my SCS number from 3 to 4
34:46:
34:46: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
34:46: [Router Id:0.0.0.0 Adv:4.4.4.4]
34:46: Age: 0 SeqNum: 0x80000003 Cksum: ca36 Len: 40
34:46: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 2.2.2.2
34:46: Sending LSA(s) on interface:
34:46: [Router Id:0.0.0.0 Adv:4.4.4.4]
34:46: Age: 0 SeqNum: 0x80000003 Cksum: ca36 Len: 40
34:46: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 2.2.2.2
34:46: Implicit acknowledgement(reflood of [Router Id:0.0.0.0
   Adv:4.4.4.4]) from 2.2.2.2!
34:46: Removing [Router Id:0.0.0.0 Adv:4.4.4.4] from 2.2.2.2's
   retrans list
34:46: [Router Id:0.0.0.0 Adv:4.4.4.4]
34:46: Age: 2 SeqNum: 0x80000003 Cksum: ca36 Len: 40
34:49: NOT setting the SCS TLV's N bit
34:49: Building Dropped TLV:
34:49: - added 3.3.3.3 (pers: 1)

Information in the ''Dropped Neighbors''-TLV is signaled only
   in one
```

```
packet. Note that this value is adjustable.

34:49:
34:49: Got [Router Id:0.0.0.0 Adv:3.3.3.3] from 2.2.2.2
34:49: [Router Id:0.0.0.0 Adv:3.3.3.3]
34:49: Age: 2 SeqNum: 0x80000003 Cksum: e520 Len: 40
34:49: 2.2.2.2 is the sender if the LSA..
34:49: All my neighbors have received the LSA - abort flooding
34:49: Delayed acknowledgement scheduled in 2 seconds
34:51: Sending the following acks on the interface:
34:51: [Router Id:0.0.0.0 Adv:3.3.3.3]

Unblocking connection between 4 and 1

35:01: Change in state: new neighbor w/incr Hellos!
35:01: Increasing my SCS number from 4 to 5
35:01: ** Neighbor table:
35:01: 1: 2.2.2.2 (will: 200)
35:01: Neighbor 1: 1.1.1.1
35:01: Neighbor 2: 3.3.3.3
35:01: 2: 1.1.1.1 (will: 110)
35:01: Neighbor state change 1.1.1.1%eth0: [Down]->[Init]
35:01: Got a new incremental Hello with no state change ->
   requesting full state from 1.1.1.1
35:01: Including 1.1.1.1 in neighbor list
35:01: NOT setting the SCS TLV's N bit
35:01: Requesting FS from 1 of 2 neighbors, dropping the TLV
35:01: Neighbor state change 1.1.1.1%eth0: [Init]->[ExStart]
35:01: Neighbor state change 1.1.1.1%eth0: [ExStart]->[ExChange
   ]
35:01: Neighbor state change 1.1.1.1%eth0: [ExChange]->[Full]
35:02:
35:02: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
35:02: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:02: Age: 0 SeqNum: 0x80000004 Cksum: 6d78 Len: 56
35:02: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 1.1.1.1
35:02: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 2.2.2.2
35:02: Sending LSA(s) on interface:
35:02: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:02: Age: 0 SeqNum: 0x80000004 Cksum: 6d78 Len: 56
35:02:
35:02: Got [Router Id:0.0.0.0 Adv:1.1.1.1] from 1.1.1.1
```

```
35:02: [Router Id:0.0.0.0 Adv:1.1.1.1]
35:02: Age: 1 SeqNum: 0x80000002 Cksum: 40a8 Len: 56
35:02: Flood: Add [Router Id:0.0.0.0 Adv:1.1.1.1] to retrans-
   list of 2.2.2.2
35:02: 1.1.1.1 originated the LSA..
35:02: 1.1.1.1 is the sender if the LSA..
35:02: At least one neighbor hasn't received the LSA..
35:02: Adding LSA([Router Id:0.0.0.0 Adv:1.1.1.1]) node to the
   pending LSA list. Waiting:
35:02: 2.2.2.2
35:02: New twohop neighbor: 2.2.2.2 (via 1.1.1.1)
35:02: Delayed acknowledgement scheduled in 2 seconds
35:02: Received a duplicate LSA [Router Id:0.0.0.0 Adv:1.1.1.1]
    from 2.2.2.2
35:02: Implicit acknowledgement(reflood of [Router Id:0.0.0.0
   Adv:1.1.1.1]) from 2.2.2.2!
35:02: Removing [Router Id:0.0.0.0 Adv:1.1.1.1] from 2.2.2.2's
   retrans list
35:02: [Router Id:0.0.0.0 Adv:1.1.1.1]
35:02: Age: 2 SeqNum: 0x80000002 Cksum: 40a8 Len: 56
35:02: 2.2.2.2 acked the pending LSA [Router Id:0.0.0.0 Adv
   :1.1.1.1], removed from backup wait list
35:02: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 2.2.2.2
35:02: Implicit acknowledgement(reflood of [Router Id:0.0.0.0
   Adv:4.4.4.4]) from 2.2.2.2!
35:02: Removing [Router Id:0.0.0.0 Adv:4.4.4.4] from 2.2.2.2's
   retrans list
35:02: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:02: Age: 2 SeqNum: 0x80000004 Cksum: 6d78 Len: 56
35:03: Ignoring 2.2.2.2's Hello packet: FS bit set, but SCS is
   same

Node B received the local node's Hello request (the local node
   sent a
Hello packet with the R bit set, but no ''Req FS from''-TLV
   included), and responded with
full state.

35:04: Sending the following acks on the interface:
35:04: [Router Id:0.0.0.0 Adv:1.1.1.1]
35:04: Not resetting HelloInterval - (Req) FS list is not empty
35:04: 1.1.1.1 acked [Router Id:0.0.0.0 Adv:4.4.4.4] - remove
   from retrans list
```

```
35:04: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:04: Age: 4 SeqNum: 0x80000004 Cksum: 6d78 Len: 56
35:04: Pushback timeout...
35:04: Flooding [Router Id:0.0.0.0 Adv:1.1.1.1] will result in
   a redundant transmission - abort

Blocking connection between 4 and 2

35:11: Got full state: Increasing 1.1.1.1's SCS number from 1
   to 3
35:11: Got SCS from 1.1.1.1 with R bit set but no Request From
   TLV added -> sending full state
35:11: Including 1.1.1.1 in neighbor list
35:11: 1 of 2 neighbors requesting full state, dropping the TLV
35:43: Neighbor state change 2.2.2.2%eth0: [Full]->[Down]
35:43: 2.2.2.2's state transitioned to less than EXCHANGE:
   Delete from neighbor table
35:43: Change in state: deleting neighbor 2.2.2.2!
35:43: ** Neighbor table:
35:43: 1: 1.1.1.1 (will: 200)
35:43: Neighbor 1: 2.2.2.2
35:43: Increasing my SCS number from 5 to 6
35:43:
35:43: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
35:43: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:43: Age: 0 SeqNum: 0x80000005 Cksum: 748e Len: 40
35:43: Flood: Add [Router Id:0.0.0.0 Adv:4.4.4.4] to retrans-
   list of 1.1.1.1
35:43: Sending LSA(s) on interface:
35:43: [Router Id:0.0.0.0 Adv:4.4.4.4]
35:43: Age: 0 SeqNum: 0x80000005 Cksum: 748e Len: 40
35:43: Adding 1.1.1.1 to new_aor list
35:43: Changes in the AOR set: New AOR(s)
35:45: ** Reg ack [Router Id:0.0.0.0 Adv:2.2.2.2] received from
    1.1.1.1

Node A is not (yet) elected to act as an AOR for node B, and
explicitly acknowledges the LSA. The local node overhears this
   and
registers the acknowledgement in its ack cache.

35:45: 1.1.1.1 acked [Router Id:0.0.0.0 Adv:4.4.4.4] - remove
   from retrans list
35:45: [Router Id:0.0.0.0 Adv:4.4.4.4]
```

```
35:45: Age: 4 SeqNum: 0x80000005 Cksum: 748e Len: 40
35:46: Received a duplicate LSA [Router Id:0.0.0.0 Adv:4.4.4.4]
    from 1.1.1.1
35:46:
35:46: Got [Router Id:0.0.0.0 Adv:2.2.2.2] from 1.1.1.1
35:46: [Router Id:0.0.0.0 Adv:2.2.2.2]
35:46: Age: 5 SeqNum: 0x80000004 Cksum: 39b1 Len: 56
35:46: 1.1.1.1 is the sender if the LSA..
35:46: 1.1.1.1 has already acked the LSA..
35:46: All my neighbors have received the LSA - abort flooding
35:46: Delayed acknowledgement scheduled in 2 seconds
35:48: Sending the following acks on the interface:
35:48: [Router Id:0.0.0.0 Adv:2.2.2.2]
35:58: NOT setting the SCS TLV's N bit
35:58: Building Dropped TLV:
35:58: - added 2.2.2.2 (pers: 1)
35:58: Signaling my election of AORs (pers: 3):
35:58: 1.1.1.1

Blocking connection between 4 and 1

36:08: Signaling my election of AORs (pers: 2):
36:08: 1.1.1.1
36:18: Signaling my election of AORs (pers: 1):
36:18: 1.1.1.1
36:45: Neighbor state change 1.1.1.1%eth0: [Full]->[Down]
36:45: 1.1.1.1's state transitioned to less than EXCHANGE:
   Delete from neighbor table
36:45: Change in state: deleting neighbor 1.1.1.1!
36:45: No neighbors listed in neighbor table
36:45: Increasing my SCS number from 6 to 7
36:45:
36:45: Originating [Router Id:0.0.0.0 Adv:4.4.4.4]
36:45: [Router Id:0.0.0.0 Adv:4.4.4.4]
36:45: Age: 3600 SeqNum: 0x80000005 Cksum: 748e Len: 40
36:45: Sending LSA(s) on interface:
36:45: [Router Id:0.0.0.0 Adv:4.4.4.4]
36:45: Age: 3600 SeqNum: 0x80000005 Cksum: 748e Len: 40
36:48: NOT setting the SCS TLV's N bit
36:48: Building Dropped TLV:
36:48: - added 1.1.1.1 (pers: 1)
```

Appendix B

Design and Implementation of Wireless OSPF for Mobile Ad Hoc Networks

Kenneth Holter, Andreas Hafslund, Frank Y. Li and Knut Øvsthus.

To be presented at the 6th Scandinavian Workshop on Wireless Ad-hoc Networks

Stocholm, Sweden, May 3-4 2006

Design and Implementation of Wireless OSPF for Mobile Ad Hoc Networks

Kenneth Holter*, Andreas Hafslund†, Frank Y. Li*, and Knut Øvsthus*
*UniK - University Graduate Center, P. O. Box 70, N-2007 Kjeller, Norway
†Thales Norway AS, N-0609 Oslo, Norway
Email: {kenneho, fli, knuto}@unik.no, Andreas.Hafslund@no.thalesgroup.com

***Abstract*— OSPF is the standard interior routing protocol widely deployed in the fixed Internet. Wireless OSPF is targeted at extending OSPF in a mobile ad hoc environment in which reducing protocol overhead is of paramount importance. So far two proposals have received most attention and are currently under active investigation within the IETF. In this paper, we present the design and implementation of one of them, referred to as WOSPF-OR in the context, that uses *overlapping relays* for reliable and efficient flooding. We adopt a plugin module concept in our implementation design and our implementation is developed based on the Quagga routing software platform.**

I. Introduction

Several routing protocols have been standardized by the IETF Mobile Ad-hoc Network (MANET) [1] working group during the past few years, where a typical category is developed based on traditional link state routing protocols. Optimized Link State Routing (OLSR) [8], for example, is the most representative routing protocol of this kind, where protocol overhead minimization is achieved through *unreliable* flooding of network topology messages via Multipoint Relays (MPRs). On the other hand, another link state routing protocol, Open Shortest Path First (OSPF), which is the current Internet standard for interior routing, has been extensively studied and widely deployed in the wired Internet. A natural question that has recently received much attention with the IETF is: Why not adopt OSPF in MANETs?

The motivation for adopting OSPF in a MANET environment is twofold: Interoperability and familiarity. With both wired and wireless support, a MANET-capable OSPF router could operate properly when plugged into a wired OSPF network. By extending and building an OSPF framework, the transition and interoperability between wired and wireless networks would become seamless. Furthermore, OSPF is already a mature routing protocol. Experience and lessons learned from OSPF can be of great help when extending OSPF to wireless networks.

However, adopting OSPF in MANETs is not an easy task since OSPF was originally designed for more or less static, wired networks. There are several reasons that the OSPF protocol cannot be deployed directly in a MANET environment. First of all, OSPF does not have a suitable interface type for a wireless broadcast environment which is characterized by a multicast-capable transmission medium and where routers do not necessarily form a full mesh. Moreover, to adapt to the unpredictable behavior of mobile nodes, OSPF will have to increase the amount of topology dissemination messages, leading to a prohibitively high routing overhead when adopted in MANETs. Consequently, OSPF does not scale well even for a quite small MANET [17]. Furthermore, since links in a wireless environment cannot be assumed to be bi-directional, the Designated Router (DR) mechanism, a common OSPF flooding optimization mechanism, will not perform correctly in MANETs as this mechanism assumes a true multi-access network.

The first proposal for wireless extension of OSPF [17] employed an unacknowledged flooding technique based on the principles of OLSR. Later in 2004, the IETF OSPF working group decided to focus on protocols using acknowledged flooding to keep consistency of OSPF which relies on *reliable* flooding. So far, two proposals, [2] and [4], have received most attention. In this study we have designed and implemented wireless extensions to the OSPF for IPv6 (usually referred to as OSPFv3) routing protocol, based on the mechanisms described in [2], referred to as WOSPF-OR in this context. This proposal is aimed at reducing the routing overhead on the network by optimizing flooding of Link State Advertisements (LSAs) using Overlapping Relays (ORs), incremental Hellos, and reducing the size of Hello messages. These mechanisms may have significant impact on OSPF performance in MANETs. The optimized flooding scheme minimizes the number of retransmissions needed to diffuse topology information throughout the network. A decrease in size of a message type that is to be transmitted quite frequently will over time benefit the network performance.

The rest of the paper is organized as follows. Section 2 gives an overview of OSPF, WOSPF-OR, and discusses some related works. The designed and implementation of WOSPF-OR are presented in Section 3, followed by discussions on other related issues in Section 4. The paper is concluded in Section 5.

II. OSPF and WOSPF-OR functionality

This section outlines the functionality of OSPF and WOSPF-OR. First we provide a high-level overview of OSPF, whereas details are further discussed when comparing OSPF with WOSPF-OR.

A. Overview of OSPF

OSPF is one of the most studied and well-known link state routing protocols to date. Being a link state protocol, every

router maintains a complete map of the network topology. As a consequence, at any given moment a router can compute the path to any destination in the network, based on its existing knowledge about the network. To ensure that every router maintains a consistent view of the network, the participating routers send out routing messages announcing their link states. This information is reliably flooded throughout the network to ensure its reception by every router in the network.

OSPF is classified as an Interior Gateway Protocol (IGP) in that it routes information within an Autonomous System (AS), often referred to as a routing domain. For scaling purposes *areas* have been defined, and a collection of areas makes up an AS. Conceptually, an area is a generalization of an IP subnetwork, grouping networks together in the same manner as one groups routers together to form a network. The topology of an area is hidden from routers outside it, thus reducing routing traffic elsewhere in the AS.

B. WOSPF-OR Functionality

The wireless extension to OSPF described below is based on the Internet-draft [2]. This draft proposes a MANET extension to OSPF, focusing on reduction of routing overhead by using ORs. The proposed protocol extensions have the following new features which are enabled on MANET interfaces:

1) The OSPF-MANET interface: OSPF has defined interface types for networks in which one cannot assume that a router has bi-directional connectivity to all the other routers in the network (for example LANs). One such interface type is the point-to-multipoint interface, in which every neighbor is described as a point-to-point neighbor. The point-to-multipoint interface has been chosen to represent the OSPF-MANET interface, as MANET nodes typically will be able to reach only a subset of its neighbors.

When a new node joins an area it must learn the link state database of that area. By the use of Database Descriptor (DD) and Link State (LS) Update packets, it updates its database by synchronizing it with a neighbor's database. When synchronization is performed, the two neighbors are said to be *adjacent*. In OSPF's point-to-multipoint networks, and thus in OSPF-MANETs, every neighbor forms an adjacency with its neighbors. The database exchange process is unmodified compared to OSPF.

2) Link Local Signaling (LLS): Customizing OSPF for MANETs calls for information exchange not supported in existing OSPFv3 messages. To address this, a mechanism for exchanging arbitrary data on a link is defined. This mechanism is called LLS [16], and is basically a piggyback mechanism for Hello and DD messages: Hello and DD messages are appended by an LLS data block containing information in the form of Type/Length/Value (TLV) data blocks. As no TLVs are defined for being appended to DD packets, DD packets are not further examined in this paper.

Figure 1 depicts the format of a TLV data block. The LLS mechanism can be used to signal arbitrary information by defining new TLVs when needed. An LLS data block may consist of any numbers of TLVs, in addition to an LLS header.

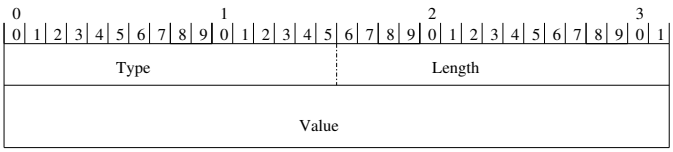


Fig. 1. The format of TLVs.

3) Neighbor Maintenance with Incremental Hellos: At regular time intervals OSPF routers emit Hello messages, which serve to maintain and acquire neighbor relationships. Due to the (possible) rapid change in the topology of MANETs, Hello messages on MANET interfaces should be emitted at a much higher frequency than in wired OSPF. This leads to extra routing overhead of an already resource constraint network.

To address this issue, a modification to the OSPF Hello protocol has been proposed: Instead of transmitting *full* state in every Hello message, only *changes* in a state are declared. If there is no changes in the state, the Hello message does not list any neighbor and thus serves only as a "keepalive"-message to its neighbors. This is contrary to legacy OSPF where the Hello message lists *all* neighbors known to the local router, and where a router not listed in a Hello message is implicitly not known.

Although the reduction in Hello message size is rather small, over time this reduction will have significant benefits on the total amount of routing overhead.

4) Optimized Flooding: Link state information is disseminated in Link State Advertisements (LSAs) at regular time intervals. To limit overhead caused by pure flooding, OSPF states that an LSA is to be retransmitted out on all interfaces except the one through which it was received. MANETs need special attention as LSAs usually are re-flooded on the same interface on which it was received.

In a typical MANET, pure flooding causes every node (except the originator) to retransmit the message, and most nodes will receive the message more than once. To reduce the number of retransmissions in a MANET environment, and hence optimize flooding, a router elects a subset of its neighbors to relay LSAs. This set is known as the *Active Overlapping Relay (AOR) set* for the local router. It is important to note that the nodes in an AOR set are merely the nodes that are to relay LSAs *first* - non-AOR nodes are required to relay LSAs if the AORs nodes do not. This might be the case in situations where the AOR set is not up to date due to for example poor link quality.

To ensure that every router within an area has identical LS databases, LSAs are disseminated in a reliable fashion by the use of Link State Acknowledgements (LS Acks). A router continues to retransmit an LSA to its neighbors until each neighbor confirms reception with an LS Ack, transitions to state "down", or implicitly signals reception of the LSA by retransmitting it on the interface.

To reach all immediate neighbors, routing protocol messages are sent to a well-known multicast address. Because AORs retransmit such messages, a node may receive more than one copy of a message. Therefore, every node maintains

a set of messages which have been received, so that duplicate messages can be detected and discarded.

C. Flooding in WOSPF-OR and OLSR

As MANETs typically consist of resource constraint nodes communicating over low capacity wireless links, some sort of flooding optimization would clearly be beneficial. The flooding scheme used in WOSPF-OR is essentially the same as the use of MPRs in the OLSR protocol, except that WOSPF-OR's use of LS Acks and backup relays (i.e., non-AORs) to ensure reliable transmissions.

D. Other Drafts on Wireless OSPF

Several solutions [3], [4] and [17] have been proposed for MANET extensions to OSPF. In particular, [4] is the other proposal currently under active discussion on the OSPF-MANET mailing list. The extension specified in this Internet-draft is called OSPF MANET Designated Router (OSPF-MDR). OSPF-MDR has some resemblance to the extensions discussed in this paper by the use of incremental Hellos and overlapping relays, but differs on whether the *Connected Dominating Set (CDS)* is dependent or independent of the source. A MANET running the OSPF-MDR routing protocol consists of several (Backup) DRs forming a source independent . As in OSPF, the (B)DRs flood LSAs, minimizing the set of flooding originators. Also, the LSAs to be flooded are relayed by overlapping relays. More recently, a flooding mechanism based on Smart Peering has also been proposed [18].

III. Design and Implementation of WOSPF-OR

The extensions to OSPF are designed to be an OSPF "module" for operating in a MANET environment, in that mechanisms such as overlapping relays and differential Hellos apply to MANET interfaces. When operating in a MANET, a different flooding and Hello scheme is utilized to minimize routing overhead.

As the overlapping relay mechanism described in this paper is essentially equal to the MPR mechanisms defined for OLSR, using already implemented OLSR code could be beneficial for implementing the overlapping relay mechanism.

Our WOSPF-OR design has therefore two considerations: Reuse of UniK's OLSR code [10], and limit the modifications to Quagga's OSPFv3 code. UniK's OLSR implementation has gained international recognition for its quality and extensiveness. The code has been thoroughly studied, documented and tested, and should thus provide a solid foundation for adaptation to some of the WOSPF-OR mechanisms. OSPFv3 is a rather extensive routing protocol, which is reflected in the complexity of the source code. To minimize the level of complexity of the extension, and to reflect the "module"-based approach of the proposed extensions, the modifications to the source code itself are kept at a minimum.

A WOSPF-OR router is a router able to utilize the OR mechanism (and optionally also the incremental Hellos mechanism) on its interfaces. A router's MANET interfaces utilize these mechanisms, while interfaces to standard OSPFv3 networks operate as described in [7]. Neighbors on MANET interfaces are referred to as *WOSPF-OR neighbors*.

A. WOSPF-OR Neighbors

The WOSPF-OR neighbor data structures located in the neighbor table include information such as whether the neighbor is chosen to act as an AOR for the local router or not. This limits the amount of different data structures needed to make calculations, as the WOSPF-OR neighbor data structure maintains information relevant to different calculations needed to operate in a MANET.

A WOSPF-OR neighbor is merely an OSPF neighbor with the ability to perform different MANET routing overhead optimizations. These optimizations require additional information to be registered on a per neighbor basis, and this is implemented by the use of *encapsulation*: Instead of extending the neighbor data structure already defined in OSPF, the existing OSPF neighbor data structures are in as many cases as possible encapsulated in new WOSPF-OR neighbor data structures in the following manner:

```
struct wospf_neighbor {
  <WOSPF-OR specific information>
  <pointer to corresponding
   OSPF neighbor data structure>
}
```

As a consequence, no modification of the existing OSPF neighbor data structure is needed, as all WOSPF-OR specific information is added to the WOSPF-OR neighbor data structure only. These data structures are organized in a *WOSPF-OR neighbor table*, equivalent to the organization of neighbor entries in [10]. This structure reflects the idea behind adding extensions to OSPF - every WOSPF-OR neighbor is an OSPF neighbor with some functionality needed in MANETs.

WOSPF-OR also maintains a table of two hop neighbors, the same as in OLSR. A two hop neighbor entry does not have a link to an OSPF neighbor data structure. The entry serves the purpose of maintaining a view of the local two-hop neighborhood in order to compute the AOR set.

B. Signaling Non-OSPF Information

Outgoing Hello packets are intercepted by a WOSPF-OR function responsible for appending an LLS data block. The format of the LLS block is defined in [2].

TLVs related to signaling of AOR-elections are constructed after looking up AORs by iterating with the neighbor table. The entries constituting the neighbor table have information on the neighboring router, so by iterating with the neighbor table the router IDs of AORs and dropped AORs can be listed in "AOR-TLVs" and "Dropped-TLVs".

C. Relaying LSAs

Nodes in the non-AOR set is required to relay information if nodes in the AOR set fail to do so. In effect, these neighbors serve as backup overlapping relays. The AOR selection algoritm is based on the algorithm used in OLSR for calculating the MPR set.

The AOR election is also registered with each WOSPF-OR neighbor data structure, utilizing the encapsulation design as described in the previous subsection.

Figure 2 illustrates part of the information flow of outgoing LS Update messages. Outgoing messages are defined as messages that have already been processed or originated by the routing system, and are to be transmitted on the interface. Received LSAs are either retransmitted immediately or delayed according to the mechanism outlined in the following paragraphs. As OSPF already implements numerous forwarding decision, the "Flood filter" box represents the additional conditions that are needed to utilize the overlapping relays mechanism.

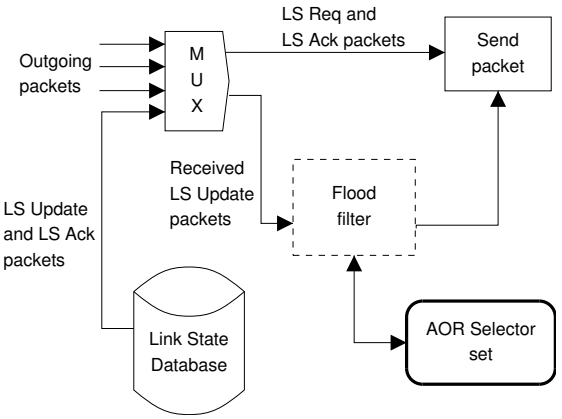


Fig. 2. The flow of outgoing link state messages. LS Update messages are retransmitted based additional criterias such as whether the node is an AOR for the transmitter.

1) The Active set: When a received LSA is to be re-flooded by the local node, a decision is made whether it should do so immediately, or back off for some time. If the local node has been elected by the transmitting node to serve as an AOR (i.e., the transmitting node is registered with the local node's AOR Selector set) the local node refloods immediately.

2) The Non-Active set: A non-AOR refloods an LSA if it decides this will not result in an redundant retransmission. Obviously, a transmission is redundant if all neighbors have already received the LSA. However, the implementation of this decision process is somewhat complicated, as the router must store and update information about which LSAs has not been received by which neighbors.

When a router decides it should not immediately re-flood an LSA (because it is not an AOR of the sender), it adds the LSA to a dedicated *pending LSA* list unique for each WOSPF-OR interface, and the LSA is scheduled to be retransmitted within a short time interval. For each pending LSA, neighbors who have not yet received this LSA (i.e., neighbor from which the router has not received an ack or heard a reflood) are added to this list. Hence, each interface has one two-dimensional list consisting of pending LSAs and their neighbors for which the LSAs are pending. This list of neighbors is referred to as the pending LSA's *backup wait list*.

When the routers register that a neighbor has received the LSA (either by receiving a multicast LS Ack or hearing a re-flood), the neighbor is removed from the pending LSA list. When the LSA is scheduled for retransmission, if any neighbors remain listed in the pending LSA's *backup wait list*, the LSA is re-flooded. Note that under normal circumstances this should not occur, as the router was not elected as an AOR for the sender in the first place.

A router may receive LS Ack messages for LSAs it has not received yet. It is even possible that every neighbor acks the LSA before the local router receives the LSA. To keep track of which LS Acks have been received by which neighbors, each WOSPF-OR neighbor data structure maintains a list of LSAs that have been acked, but which the router itself has not received. This list is referred to as the *Acked LSA list*, and is essentially an acknowledgement cache. This list makes use of the OSPF link state database data structure already implemented, and operations on the database is made through the interface implemented in the OSPFv3 source code.

D. Incremental Hellos

Routers supporting partial neighbor information (i.e., incremental Hellos) signal this by setting a bit in the Hello message's option field. Communication with such a neighbor will then utilize the incremental Hello scheme by not including the neighbor's router ID in the consecutive Hello packets. If the neighbor does not support this scheme the local router will continue listing the neighbor in its Hello messages, as specified in [7]. Thus, an incremental Hello capable router, WOSPF-OR, will be able to communicate compatibly with routers who are not capable of this.

If one or more state changes occur, the local State Check Sequence (SCS) number is incremented and the next Hello message will contain full state. This number indicates current state, and must be signaled when using the incremental Hellos scheme.

To utilize the incremental Hellos scheme some modifications to the Hello protocol are required. As soon as a router starts forming an adjacency with an incremental Hellos capable neighbor, the router removes the neighbor router ID from the neighbor list reported in the Hellos. The neighboring router, on the other hand, failing to find its router ID in the Hello messages does not interpret this as a relationship failure - such information is signaled explicitly using the LLS mechanism.

Neighbors that are no longer known are listed in a dedicated "Neighbor Drop" TLV. When a router finds that it is listed in such a TLV it declares the neighbor as down and deletes all data structures associated with that neighbor.

Figure 3 illustrates part of the information flow of outgoing Hello packets. Outgoing messages are, as for Figure 2, defined as messages already processed or originated by the routing system, and are to be transmitted on the interface. An LLS data block is appended to Hello packets. Which TLVs to be carried in the LLS block depends on changes in state or willingness to serve as an AOR. The TLV carrying the local router's SCS number is always appended to indicate current state. The

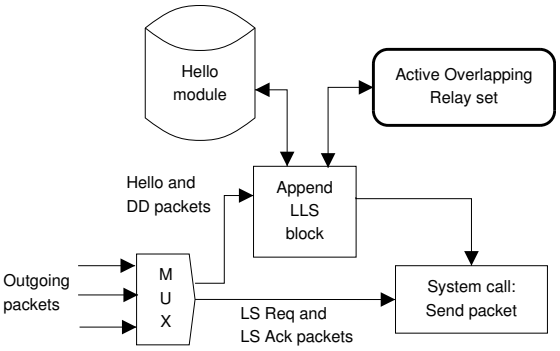


Fig. 3. The flow of outgoing Hello and DD messages. These messages are appended an LLS data block containing one or more TLVs.

Hello module seen in the figure is a conceptual data structure maintaining all information related to the incremental Hellos scheme. In practice, this module consists of lists of dropped relays, neighbors requesting full state, and so forth.

E. Discussions on the Design

Designing a software system of this size and complexity calls for manageable and smart solutions. Modifying an existing protocol implementation should be done with caution, as adding to the complexity of a protocol can result in unexpected results. Our design goal is therefore to modify the source code as less as possible, without compromising functionality. On the other hand, the extension modules themselves are designed for protocol efficiency.

As incremental Hello messages may not be listing any neighbors even though they are not dropped, these messages cannot flow through the OSPF system as usual. These messages are intercepted and processed by the Hello module, and the OSPF neighbor data structures are modified accordingly. An alternative would be to have the Hello module construct a "fake" Hello message listing *all* known neighbors, and run it through the OSPF system in the same manner as the traditional OSPF Hello messages would. For the implementation the former solution is used for efficiency reasons (the local router does not need to spend time and processor power to construct new "fake" Hello messages).

As adjacency forming and maintenance imposes relatively high routing overhead on the network (consider for example scenarios where nodes move in and out of transmission range), reducing the set of neighbors with which a node forms adjacency might be desirable. One such technique is proposed in [18], and is left for further work.

F. Plugins

Extending the functionality of a protocol might increase the complexity, in addition to adding functionality not really part of the protocol itself. Under these circumstances plugins come in handy, as they provide the flexibility to modify or add to the functionality without altering the source code. Our WOSPF-OR implementation is to support plugins by defining a plugin interface based on the one defined in [10].

G. Implementation Issues

1) Quagga: The platform upon which the implementation was built is Quagga [5]. Quagga is a GPL licensed routing software suite forked off from GNU Zebra [6], and is under active development. Because of this active developer community, Quagga was elected to be the basis for our WOSPF-OR extensions. Both Quagga and [10] are implemented in C, which allows for reuse of the source code of [10].

As of this writing Quagga is still beta software, meaning that no official versions have been released. The latest stable release is version 0.98.5.

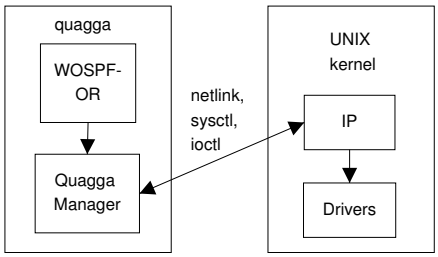


Fig. 4. The system architecture of Quagga. The Quagga Manager daemon provides the WOSPF-OR implementation with a platform independent core.

Figure 4 shows a high-level overview of the Quagga architecture with our WOSPF-OR implementation. The strict layered architecture enables the routing protocol to be implemented independent of the underlying operating system.

2) GNU/Linux: The Quagga routing software provides routing protocol implementations for Unix platforms such as FreeBSD and GNU/Linux. The testbed is made up of routers running either Debian or Ubuntu. Currently, the routers with our implementation make up a wired network connected by a hub. To emulate a mobile environment where links are created and destroyed in a more or less random fashion, a script has been developed that emulates link breakage by blocking IPv6 packets from chosen neighbors. Blocking packets is done by modifying the *ip6tables* on each machine. To emulate a link breakage between two routers A and B, A's iptables is added a rule blocking *all* packets send by B, and vica versa.

IV. Other Related Issues

There are a number of issues needed to be addressed before Wireless OSPF for a MANET can be successfully and effectively deployed. This section discusses some of these issues.

A. Addressing

WOSPF-OR is explicitly designed as an extension to OSPFv3. In a stand-alone MANET the nodes must coordinate address configuration in a distributed manner.

An IPv6 based interface may configure its own IPv6 address based on for example its MAC address and the network prefix. Ideally *all* MAC-addresses should be unique, but since this can not be assumed due to reasons such as the ability to manually manipulate a network card's MAC address, the nodes must

utilize a Duplicate Address Detection (DAD) scheme to ensure address uniqueness. This is further examined in [15].

OSPF, and hence WOSPF-OR, identify routers on their router ID. Consequently, the router ID *must* be unique, otherwise a loop would originate as a result of an OSPF reflooding what would seem to be a self-originating LSA with erroneous sequence number.

[7] proposes router ID being set to the lowest IP address associated with the router, and this mechanism is already implemented in Quagga. This, however, assumes that the IP address is unique, and the node should run a DAD scheme.

B. Global Connectivity

Gateways in IPv4-based MANETs often implement a NAT or Mobile IP mechanism to cope with the scarcity of global IPv4 addresses. This, in addition to that the use of default routes is normally allowed in IPv4 MANETs, greatly complicates global connectivity, see [11].

Using IPv6-based nodes simplifies external connectivity, especially in multi-homed MANETs. As an IPv6-based MANET node can configure a unique global address based on the prefix of one of the gateways, there is no need for a NAT or Mobile IP agent at the gateways, and no default route.

C. WOSPF-OR Areas

OSPF has very good scaling properties by the use of *areas*. As the scaling goals of the intra-area extensions described in this paper is merely 50-100 nodes [13], it is desirable to deploy the area model in MANETs. This, however, is not destined to work because of the problem of node mobility and area formation in such mobile networks. [12] proposes a two-step solution to area design in such networks, using both a classical graph partitioning algorithm as well as an ad-hoc heuristic.

Area formation in MANETs may be solved by applying the above mentioned algorithm. However, issues such as area *mobility* must also be addressed as a wandering node may find itself in a different area cut off from its home area. [14] discusses this issue, and proposes a solution, where a wandering node automatically detects and joins new areas.

D. Simplified Multicast Forwarding (SMF)

[19] specifies a generic function for basic multicast forwarding in MANETs. This function, called SMF, aims at efficiently disseminating messages by useing an (improved) flooding mechanism. The overlapping relay mechanism described in this paper implements SMF in that nodes forward LSAs only once based on neighbors' AOR calculations and signaling. The SMF designed in this paper is expected to lower routing overhead as the number of unnecessary transmissions is minimized.

WOSPF-OR assumes reliable dissemination of LSAs, enforcing reception of LSAs by *all* routers in the network. For this purpose nodes acknowledge reception of LSAs either explicitly or implicitly. As AORs implicitly ack the LSAs, the number of explicit LSAs (carried in LS Ack messages) is reduced. Although the reliable flooding scheme imposes additional routing overhead on the network, this mechanism reduces the WOSPF-OR's convergence time as routing updates consequently are reached by all neighbors.

V. Concluding remarks

This paper studies wireless extensions of OSPFv3 to support mobile ad hoc networking, with major focus on design and implementation of one of the most promising proposals, WOSPF-OR. The WOSPF-OR extensions utilizes OSPFv3's point-to-multipoint interface as its wireless interface when used in MANETs. The reliable and optimized flooding is achieved through the use of ORs and an optimized flooding scheme with reduced Hello message size.

Using the Quagga routing software suite as the platform, we have implemented the WOSPF-OR extensions based on GNU/Linux. No test performance results are reported in this submission since this work is still undergoing as of this writing.

References

[1] J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC 2501, January 1999

[2] M. Chandra, "Extension to OSPF to Support Mobile Ad Hoc Networking", Internet draft, draft-chandra-ospf-manet-ext-03.txt (expired), April 2005

[3] J. Ahrenholz, T. Henderson, P. Spagnolo, E. Baccelli, T. Clausen and P. Jacquet, "OSPFv2 Wireless Interface Type", Internet draft, draft-spagnolo-manet-ospf-wireless-interface-01.txt (expired), May 2004

[4] R. Ogier, and P. Spagnolo, "MANET Extension to OSPF using CDS Flooding", Internet draft, draft-ogier-manet-ospf-extension-06.txt, December 2005

[5] The Quagga Routing Suite, http://www.quagga.net

[6] The Zebra Routing Suite, http://www.zebra.net

[7] R. Coltun, D. Ferguson, and J. Moy, "OSPF for IPv6", RFC 2740, December 1999

[8] T. Clausen, and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October, 2003

[9] C. E. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing Protocol", RFC 3561, July 2003

[10] A. Tønnesen, Implementation of OLSR spesification as specified in RFC 3626. Source code can be downloaded from http://www.olsr.org

[11] P. E. Engelstad, A. Tønnesen, A. Hafslund, and G. Egeland, "Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks", Proceedings of IEEE ICC'04, Paris, France, June, 2004

[12] S. Galli, H. Luss, J. Sucec, A. McAuley, S. Samtani, D. Dubois, K. DeTerry, R. Stewart, and B. Kelley, "A Novel Approach to OSPF-Area Design for Large Wireless Ad-Hoc Networks", Proceedings of IEEE ICC'05, Seoul, Korea, 2005

[13] OSPF and MANET WG meetings, Proceedings of IETF 64, November 2005

[14] F. Baker, "An Outsider's View of MANET", Internet draft, draft-baker-manet-review-01.txt (expired), March 2002

[15] S. Nesargi, and R. Prakash, "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network", Proceedings of IEEE INFOCOM'02, New York, USA, June, 2002

[16] A. Zinin, B. Friedman, A. Roy, L. Nguyen, and D. Yeung, "OSPF Link-local Signaling", draft-nguyen-ospf-lls-05.txt (expired), September 2004

[17] T. R. Henderson, P. A. Spagnolo, and J. H. Kim, "A Wireless Interface Type for OSPF", Proceedings of IEEE MILCOM'03, Boston, USA, October 2003

[18] A. Roy, "Adjacency Reduction in OSPF using SPT Reachability", Internet draft, draft-roy-ospf-smart-peering-01.txt, November 2005

[19] J. P. Macker, J. Dean, and W. Chao, "Simplified Multicast Forwarding for MANET", Internet draft, draft-ietf-manet-smf-01.txt, June 2005