IT 09 030
Examensarbete 30 hp
Juli 2009

Mobile Groups in Ad-hoc Networks

Kristoffer Kobosko
Henrik Wallentin

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Mobile Groups in Ad-hoc Networks

Kristoffer Kobosko & Henrik Wallentin

Most current research on Mobile Ad-hoc NETworks (MANETs) concerns one ad-hoc network at a time. A common topic is how to handle the merging of new nodes appearing in range of an existing MANET into a new, larger network. Current MANET implementations do not define the handling of policies for controlling membership or routing messages, neither do they support policy-based inter-MANET routing.

In this thesis, we present an implementation of Mobile Group Protocol (MGP) which enhances the reactive routing protocol AODV with an administrative framework in which Mobile Ad-hoc NETwork (MANET) policies are defined. Such policies are used to control membership and signaling in a MANET. We define new roles for the MANET members such as nodes, leaders and gateways. We discuss the message exchange taking place during forming of Mobile Groups (MGs) and study the the particular implementation by simulating a group of simple MGP scenarios using Network Simulator 2 (NS-2).

We conclude our work by discussing the using of MGP as a base for an inter-MANET routing protocol providing policy-based routing in MANETs.

Handledare: Johan Nielsen
Ämnesgranskare: Erik Nordström
Examinator: Anders Jansson
IT 09 030
Sponsor: Ericsson Research

Tryckt av: Reprocentralen ITC

Sammanfattning

Huvuddelen av dagens forskning inom trådlösa ad-hocnätverk, s k MANETs, fokuserar på mekanismer inom nätet. Vanligtvis diskuteras problem som har att göra med nätverkskomposition, routing, eller mobilitet. Vårt arbete ämnar tillföra en inledande studie av implementationen av policy-baserad routing ovanpå ett befintligt MANET-routingprotokoll. Dagens MANET-implementationer stödjer typiskt inte inter-nätverksrouting m h a routingpolicies.

I detta arbete presenteras MGP, en utökning av det reaktiva routingprotokollet AODV, tillförande ett administrativt ramverk i vilket MANET-policies kan uttryckas. Sådana policies kan användas t.ex för att kontrollera medlemskap och signallering i MANETs. Vi definierar också nya roller i MANETs : medlemmar (members), gateways och ledare (leaders).

Vi diskuterar meddelandeutbytet som äger rum i den utökade implementationen, och studerar detta m h a en grupp simuleringar av enkla MGP-scenarion.

Arbetet avslutas med en diskussion om MGPs möjlighet att agera bas i en utökning av MANET till ett policy-drivet inter-MANET, ett steg i riktningen mot ett trådlöst Internet.

Contents

Acknowledgments

This is a Dissertation submitted for the Degree of Master of Science in Computer Science at the Information Technology Computing Science Department at Uppsala University, Sweden, August 2005 and the work has been carried out at Ericsson AB, Stockholm.

We would like to thank Erik Nordström for helping and supporting us with the AODV-UU implementation.

At Ericsson we would like to thank Johan Nielsen and Lars Westerberg for helping with the thesis description and participating in discussions regarding questions emerged during our work at Ericsson in Kista.

Henrik Wallentin and Kristoffer Kobosko
Stockholm
2005-08-19

Chapter 1

Background

1.1 Introduction

Most current research in wireless ad-hoc networking concerns routing that takes place in one ad-hoc network at time. Additional nodes or networks that come in range integrate with the existing network by definition. However, during real-world deployments of *Mobile Ad-hoc NETworks (MANETs)*, scenarios may require some degree of control over the composition of the network. Network composition should allow nodes to control the resources to share with other participants. For example, an administrating entity may want to hide the internal topology of the network, including addresses and services, from other networks it composes with for a longer or shorter period.

The Internet's core routing protocol *Border Gateway Protocol (BGP)*, [5],[14] maintains tables of network prefixes and exchanges network reachability information between *Autonomous Systems (AS)* , which are classes of routers under the same technical administration. Using BGP as an inter-AS routing protocol, routing in, out and through an AS can be done without the complete internal topology of an AS being revealed. Each organization (telecommunications operator, ISP, etc) controlling any specific AS has made agreements with others how to allow traffic to flow through its AS and where to send outbound traffic destined for specific networks. The routing rules set up in routers running BGP reflecting the above agreements, are called *routing policies* in BGP.

BGP, while being a verified technology in the context of fixed large-scale IP-networks with hierarchical IP-addressing, is a poorer choice in context of MANETs. The MANETs considered, are temporary mobile networks, each network being under control of the carrier of the equipment, instead of a larger organization accustomed to the negotiation of routing agreements. Therefore, they are better suited for a *bring-your-own-IP-address* flat addressing scheme, than of the prefix-based hierarchical routing model of the Internet. Based on the expected high mobility / high package loss reality, we expect scenarios where knowledge about the neighbors of a node is relevant for a maximum of a couple of seconds at a time. BGP peers, on the other hand, are established by manual configuration between routers to create a TCP session, with a typical period of 60 seconds between updates. We assume that goals similar to those of BGP can be achieved by building upon a reactive MANET routing protocol instead.

However, MANETs can benefit from policy-based routing as well. The amount of information regarding a specific ad-hoc network being shared should be possible to restrict, shared information should be possible to control and the negotiation for sharing this information should follow well-defined rules. In this thesis we devise and investigate a network protocol for administrative domains, and we provide a proof-of-concept implementation in order to study its properties and limitations. This is a first look at MGP as a component in a larger routing protocol suite together with some inter-domain border routing protocol. Our implementation of MGP is a modification of the *Uppsala University implementation of AODV (AODV-UU)* for the Linux 2.6 kernel and *Network Simulator 2 (NS-2)* [6].

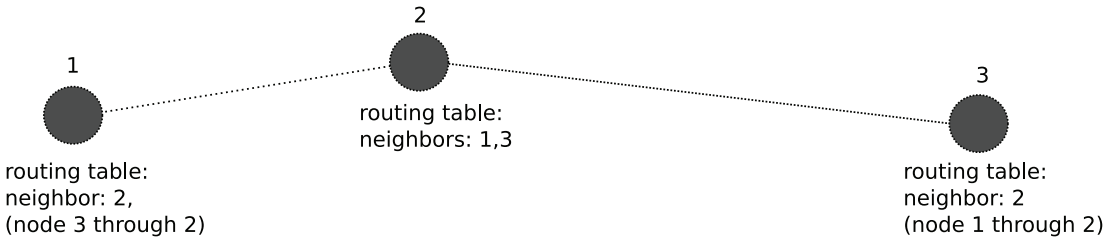

Fig. 1.1: A simple MANET with three nodes and corresponding routing tables.

1.2 Mobile Ad-Hoc Networks

Traditional networks are said to be *infrastructure based*, that is, they employ a fixed infrastructure containing networking devices such as hosts and routers that are predominantly stationary. Such networks typically have a hierarchical topology where a large network is split up into many networks or subnets connected via gateways. Routing between these parts is done by the infrastructure and, at a minimum, every connected end-user device only has to know the address of the default gateway to be able to send data packets to a destination. If data packets only travel within a network, the gateway is not involved in the decision making. In contrast to fixed infrastructure networks, an *infrastructure-less* MANET is a collection of mobile wireless network nodes or devices which simultaneously function as both hosts and routers. These nodes may move around arbitrarily and create dynamic network topologies based on proximity. Typically there are no direct communication links between all nodes at all times. Therefore, the nodes forming the MANET cooperate in relaying traffic between each other. Fig. 1.1 shows three nodes and their typical routing tables. In all kinds of routing, paths are created, updated and terminated using static routing tables, or some kind of automatic process such as one defined by a routing protocol. A MANET runs one of several ad-hoc routing protocols. The MANET working group within the *Internet Engineering Task Force (IETF)* concentrates on standardization in the field of mobile ad-hoc routing, publishing drafts and *IETF Request for Comments (RFC)* documents concerning these.

1.2.1 Reactive Routing Protocols

A reactive routing protocol works on-demand and only initiates route discovery when there is actual data to be sent and no route to the destination is currently known. The result is a protocol with a small amount of traffic overhead, but may instead incur delays when the route to a destination is not known beforehand. Fixed periodical signaling, if any, is limited to the closest neighboring nodes. Route dropping is implemented using timers that mark routes for deletion if no data passes during a certain time.

Popular reactive routing protocols include *Ad-Hoc On-demand Distance Vector Protocol (AODV)* [1] and *Dynamic Source Routing (DSR)* [3]

1.2.2 Proactive Routing Protocols

Proactive routing protocols maintain a complete distributed routing table by using periodical messaging. It is a protocol type with smaller or no route discovery delays. Proactive routing protocols can suffer in scenarios with high node mobility since they only update their routes periodically. On the other hand,

On the other hand, in difficult environments [1] some research shows that proactive protocols can be more stable due to their slower convergence [9]. A popular proactive ad-hoc routing protocol is the *Optimized Link State Routing Protocol (OLSR)* [8].

[1] Environments with a high number of noise, with temporary obstacles, or in situations relating to the Gray Zone problem

1.2.3 Hybrid Routing Protocols

Hybrid routing protocols combine the reactive and proactive models. The idea behind hybrid protocols is that it is considered advantageous to accurately know only the neighbors of any mobile terminal (i.e., mobile terminals that are accessible in a fixed number of hops). Since they are close to the terminal, communicating with neighbors is less expensive and neighbors are most likely to take part in the routing of the messages sent from the terminal. Based on this, a hybrid protocol implements a proactive protocol for communication with mobile terminals in the neighborhood and a reactive protocol for communication with the other terminals.

Chapter 2

Problem Description

2.1 Introduction

In recent years, handheld terminals such as *Personal Digital Assistants (PDA)*, mobile cellular phones and notebook computers using either *Bluetooth* or *WiFi* have become ubiquitous. Public IP-services offered can be delivered to anyone using a wireless-enabled device such as a laptop computer, wireless IP telephony terminal, PDA and so forth. Today, such usage typically requires infrastructure like Access points and Internet gateways. Therefore, hardware manufacturers have focused on the development of infrastructure-oriented products and, in comparison, little work has been done to bring MANET technology into the public. In public WiFi *hotspots*, technologies for controlling access to the network, for example to paying customers only, are readily available. When introducing MANET technology as a concept for public wireless networks, each participant can function as a router for other participants. This increases the range, the flexibility, the ease of configuration, and also lowers the cost of network deployment compared to fixed infrastructure networks. However, this type of altruistic networking is not very attractive to wireless Internet service providers or non-altruistic users due to the lack of control. Policy control can be one way to alleviate the control problem, as well as a way to provide incentives for non-altruistic users.

2.2 Scenarios

2.2.1 Generic Scenarios

Policy control in MANETs can, for example, be used when:

- A MANET owner wants to permit or deny access to its resources by other parties.
- An access network operator wants to limit the access to some or all its services to paying customers only.
- A government entity wants to completely deny access to all of its wireless network infrastructure.
- Wireless backbone networks of different service providers want to transit some or all of the network traffic based on agreements made between the companies, in a manner similar to how BGP is used in the Internet.

2.2.2 Real-World Scenario

An example where policy control in MANETs can be applied is in a public environment, such as a commuter train wagon, with a number of *Personal Area Networks (PAN)* consisting of, for example, PDAs equipped with WiFi-connections hooked up to cellular phones. Some PANs

can have public Internet access through their cellular data providers. Other PANs on the train may lack Internet connection. If configured by generous Internet-accessing users to share the connection over a local MANET, this "train back-bone" handles routing for interconnected systems under separate management by its owners. Enabling policy controlled resource sharing between administrative domains, each user extends the Internet-connected wireless domain. Lack of altruism by the owners of the individual PANs can be alleviated by providing fixed, benevolent Internet-connected MANET nodes. These nodes participate in the network as separate administrative domains supplying Internet access to all PANs in the train. By using policy control, such fixed nodes can be configured to give traffic priority to packets originated in beneficial PANs.

2.3 Scope

We concentrate on the definition of an administrative domain containing entities such as **nodes**, **gateways**, and the **central authority** maintained by a MANET routing protocol extension. Our implementation of this is, as described in section 1.1, a protocol extension to the AODV-UU implementation. The scope of this work is to define the roles and mechanisms in a single domain and is not intended to cover the complete implementation of the real-world scenario in Example 2.2.2, which includes inter-domain routing. We define our implementation as a first attempt at Internet-wide policy-based routing in MANETs, and an extensible platform for future experiments.

Chapter 3

Protocol Description

3.1 Introduction

In this chapter we describe a protocol for the administrative domain, the MGP. We present the key elements and discuss the properties and actions of its routing protocol. We also introduce the classification of nodes as a **node**, a **gateway** and a **central authority**. We define the messages and the conversations required for the creation and maintenance of an administrative domain. Throughout the chapter, we use a generic naming convention which is further specialized into an implementation-specific convention, in chapter 4. The chapter is concluded with a description of how nodes maintain domain membership, how internal routing messages are handled within a domain, and how routing message dissemination is limited to a specific domain.

3.2 Prerequisites

The protocol for administrative domains is designed as an add-on to an existing reactive MANET routing protocol, such as AODV, which is proven to work in small scenarios with simulation times ranging from tens of seconds to a couple of minutes.

3.3 Requirements and Assumptions

The administrative domain is defined as zero or more *nodes* **and** one, statically configured, non-migrating, administrative *central authority*. As such, the central authority is the single point-of-failure for the domain (see section 3.9.2 for a discussion of this design choice). Further, we require that:

1. The existence of a central authority has the main purpose of controlling all routing within a domain. The authority should ideally also control inter domain routing, but this is outside the scope of this work.

2. The administrative authority is a static role. It does not seek membership in any other domain, and does not seek to change its role. Election, re-election or creation of further central authorities should be possible, but is outside the scope of this thesis.

3. The administrative authority is not to be dedicated to management exclusively. It can be a normal device exchanging application-layer packet data.

4. Inter-group routing should be achievable, for example by encapsulating external routing messages in internal ones on arrival at a gateway.

5. A *non-member node*, whether being a node with a domain identity not equal to the domain identity of the authority (an *external node*), or a node of *empty identity*, not being member of any domain, can not participate in normal message relay. It may however request membership in a domain and upon doing so, may be rejected by the authority of the domain, or be granted membership, or, if it is a member in another domain, interact with a specially chosen *gateway*.

6. Routing messages from one domain should not leak into other domains. Routing isolation is solved by cooperation.

7. Membership in a domain is defined by the existence of a *fresh enough* route to the central authority in each node as defined by its routing table.

8. The fundamental behavior of the protocol is described as *default policies*. User-defined behavior is also possible, through the use of *custom policies*.

9. Hardening, for example, requirements 1, 5, and 6, from a security point-of-view, is not the scope of this work.

10. All policies in the administrative domain are enforced through cooperation.

11. The Protocol for the administrative domain is based on a reactive MANET routing protocol, containing a flooding, broadcasted *Routing Request* routing message, and a unicast *Routing Reply* routing message, both being extensible.

3.4 Traffic

We divide data traffic in the domain into three categories. The first consists of signaling according to the protocol, and is hereafter referred to as *messages*. The second category consists of *routing messages*, which belong to the underlying *routing protocol*. All other data, such as application-layer data is hereafter referred to as plain *data*.

3.5 Identity

Typically, MANETs employ a flat address space where any address is permitted, in contrast to the Internet, where a hierarchical network topology based on prefixes is used. In the administrative domain, each node has a dynamic *group identity*, indicating which administrative domain it belongs to (i.e. indicating *location*), and a static *node identity* which is a globally unique, permanent identifier (i.e. indicating *identity*). This partition of the identity permits an inter-domain routing scenario, where a border protocol could operate only on the Group identity level, abstracting internal compositions of individual domains.

An alternative implementation of the protocol for administrative domains could use the IP Address *host number* and *network number* to represent Group identity and node identity, but for the purpose of this thesis, we implement the concept of identity explicitly within the messages and states of our protocol.

3.6 Elements

3.6.1 The Central Authority

The central authority enforces routing policies in the administrative domain. The main responsibility is granting or denying member candidate nodes access to the domain. As declared in section 3.3, all nodes must cooperate in transferring policy decisions concerning the whole domain to the central authority. The central authority, being a regular MANET node, as well as a node in the administrative domain context, can send arbitrary data and participate in message

and routing message dissemination. A simplification in this work, discussed in section 3.9, is that the central authority role is not to be dynamic. Therefore, a node is configured to be the central authority when the protocol starts. Just as any other node, each central authority may define node-specific custom policies apart from the default policies enforcing the administrative domain.

3.6.2 Nodes

A node is a generic device in the administrative domain which participates in message processing and relaying and routing as well as the sending and receiving of application layer data. A Non-Member node has an Group identity indicating no administrative domain membership and is passive until hearing a *Membership Beacon (MB)*, on which it triggers the request mechanism and may, based on policies in the given domain, be granted membership.

3.6.3 Gateways

Each member node, hearing a MB, can apply for the right to become a gateway. The role of the gateway is to interface with other administrative domains.

The gateway property is a dynamic one and is granted and revoked by the central authority followed by a gateway Request from a node which has set up a neighbor route between itself, the *interior gateway*, and an *exterior gateway*, which is an opposite counterpart in another domain. As described in chapter 4, the handling of gateway status by the central authority can be an custom policy implemented using *Event Messages (EMs)* (see section 3.7.4).

3.7 Message Types

3.7.1 Membership Beacon

All members of an administrative domain periodically broadcast a domain advertisement beacon, the *Membership Beacon (MB)*. As explained in chapter 4, the beacons can be piggy-backed on the routing protocol's own beacon messages. It consists of a solicitation for new potential members. A domain identification included in the beacon tells the surrounding world about the existence of the particular domain and the identity of its central authority. This message is not relayed and, as such, is only detectable by immediate neighbors. A node can react on this solicit by applying for membership. nodes not belonging to any domain, shut down the MB until the next time the node is granted membership in a domain. Upon being granted membership, the node forces the routing protocol to send a first MB immediately, disregarding timers. See section 3.9 for an explanation of this behavior.

3.7.2 Membership Request Message

The *Membership Request (MR)* is defined as an extension to the broadcasted, MANET flooding, routing request message of the routing protocol and contains the identity of the requesting node. However, instead of being attached to every routing request, it is explicitly triggered when hearing a MB while in *non-member* state. By not permitting unsolicited membership request messages, we attempt to limit flooding of the MANET to situations where there is an actual administrative domain present.

Upon arriving at its destination, the central authority executes its default policy, which is to welcome the new node into the administrative domain. It replies with a *Membership Welcome (MW)* as a unicast to the requester.

3.7.3 Membership Welcome Message

The MW message is an extension to the unicast Routing Reply message in the routing protocol, sent by the central authority upon granting a node entry into the administrative domain. The

routing reply has the side-effect of creating routes back to the requesting node, and thus ensures that a new member in the administrative domain has a fresh route to its central authority. Upon receiving a MW, the requesting node updates its identity information to reflect that it is now a member of the administrative domain. The address of the central authority and its own, new identity its state to become a *member node*.

3.7.4 Event Message

All built-in or user provided policies based on unicast conversations are implemented using EMs, which can carry any type of payload. Using conversations based on these messages, *custom policies* can be implemented. Any member node is required to forward an *Event Message (EM)* to the destination by unicast to the next-hop node along the path. Relayed messages can trigger events in the relaying nodes as the messages are being forwarded. There are three built-in EMs: the *Route Discovery Permission Request (RDPR)*, the *Route Discovery Permission (RDP)*, and the *Route Discovery Denial (RDD)*.

3.7.5 Route Discovery Permission Request

Controlling routing within the domain is done in a cooperative manner, as described in section 3.3, requirement 10. A node asks the central authority whether it has permission to initiate a route discovery using a RDPR, an EM that will periodically be re-sent until the node gets an explicit permission or denial to initialize route discovery.

3.7.6 Route Discovery Permission

The RDP is the positive answer to a RDPR, i.e. an EM sent upon granting a node the right to initiate the route discovery process.

3.7.7 Route Discovery Denial

This message is the negative answer to a RDPR. A node receiving it will terminate its permission process and generate a *no-route-to-host* message to the upper layers of the network protocol stack. The lack of an explicit denial is handled using a timeout which eventually produces the same effect.

3.7.8 Gatewaying Request

Gatewaying Requests (GR) are sent by nodes to the central authority when interaction with a node of another administrative domain has taken place, usually when hearing a MB with foreign group identity (see section 3.5).

3.7.9 Gatewaying Granted

The central authority responds to the GR using an EM of type *Gatewaying Granted (GG)*. The receiving node will change its state and become a gateway when it processes this message. As there is no explicit gatewaying Denial message, nodes will resend GRs to the central authority at every heard MB, originating at a foreign node. Since this is a unicast message, signaling overhead is not considerable given the small topologies being the context of this thesis. However, a Denial message can be added as an extension, by using EMs which trigger a request fall-back behavior, without disrupting the existing protocol.

3.8 States

3.8.1 The Central Authority State

A simplification for the purpose of this thesis, presented in requirement 2 of section 3.3 states that the central authority role is permanent. Therefore, the central authority has only one state (Fig. 3.1), in which it:

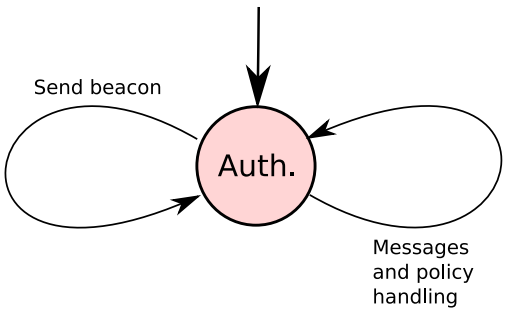

Fig. 3.1: The central authority state

- Sends MBs.
- Handles incoming MRs and RDPRs.
- Processes messages using policies and replies using MWs, RDPs and RDDs or other, custom EMs.
- Forwards RDPRs and EMs.

Being a regular device, as described in requirement 3 of section 3.3, the central authority is also able to exchange any application layer data, whenever its administrative domain contains other suitable nodes to take part in this exchange.

3.8.2 Node States

Figure 3.2 displays a state machine for the *non-member*, *node* and *gateway* states which a node typically visits during its life-time.

Non-Member

A node having its group identity (see section 3.5) field set to the non-member value will not participate in any message dissemination, and will only react to MBs. The default policies will demand that such a node asks for membership in every administrative domain it hears MBs from, using a MR. A node in member state will become a non-member when it invalidates its last known route towards the central authority.

Member

In the member state, a node is a proper member of an administrative domain and will:

- Participate in Route Discovery (send and forward RDPRs, receive RDDs, send and forward routing messages, e.g. RDPRs, see requirement 11, section 3.3).
- Participate in EM exchange (send, forward and receive EMs).
- Exchange upper-layer protocol data.

- Send MBs.
- Send GRs to the central authority when hearing a MB from a foreign Member.
- Become a gateway when receiving a GG from the central authority.

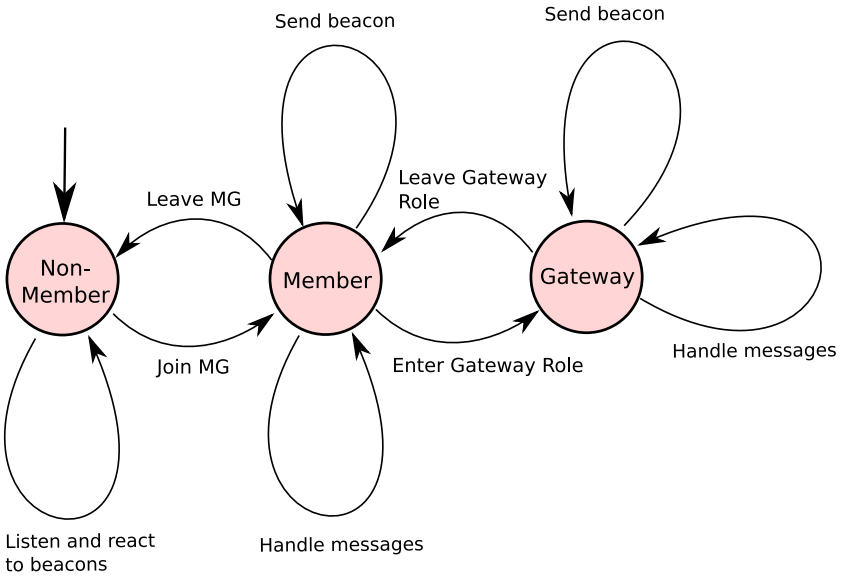

Fig. 3.2: node states

Gateway

A node in gateway state (see section 3.6.3) inherits all properties of the member node, and works identically as the member node, except, it does not send GRs. It becomes a regular member node again, as soon as the last known route to its exterior gateway is invalidated. We want to study the basic node - gateway state transitioning, but since a full implementation of a border routing protocol is not presented in this thesis, we use the gateway state as a placeholder.

3.9 Simplifications

The protocol presented in this chapter was developed in a empirical way, during a series of modifications to a reactive MANET routing protocol and is thus not a complete design. It was developed in an effort to study the basics of administrative domains in MANETs, and therefore contains simplifications.

3.9.1 Non-Generic nodes

One simplification in our design is the splitting of the Membership states between two types of dedicated nodes: nodes and central authorities. In a more generic approach, nodes would always be members of a domain, and the smallest domain would be one node only. In this case, a node is the central authority of a group with itself as the only member.

This approach, however, imposes an extra level of complexity, since one way to interpret it is that the central authority state is not permanent. A node would have to react to MBs from other domains and then have policies to cover prioritization between staying with its current administrative domain, and leaving its central authority role, becoming a Member node in another domain.

An alternative way of interpreting this, would be allowing multiple memberships per node. A node could have different roles in different administrative domains. Both these approaches would add complexity and implementation time, without providing more answers to our problem, which consists of studying the forming of an administrative domain in small and simple simulation topologies.

3.9.2 A Single Point of Failure

A major simplification in our work is the assumption of the central authority being a unique and constantly present node during the whole lifetime of the administrative domain. One can argue that, in spontaneous networks, it should not be required to configure a single node to be the central authority statically. Due to the mobility expected, it is highly probable that the central authority will be out of touch with the rest of the nodes during periods of time. In this case, all routes towards the central authority invalidate, cancelling all membership and rendering the administrative domain useless.

Once again, a simplification was chosen because the complete solution would include the central authority state to be dynamic, generating the same logical complexity as in section 3.9.1. It would also contain the added complexity of a central authority re-election mechanism. Both mechanisms would, however, take place outside of the chosen time scope of our simulations, where we represent time segments when a central authority actually exists, and is in reach of a series of nodes. Therefore, the chosen subset design presented in this chapter suffices for evaluating the dynamics of the administrative domain.

3.10 Design Caveats

The design of the Protocol for administrative domains is to be as much aligned with the reactive routing qualities of the underlying routing protocol as possible. The reasoning behind this decision was formed during an attempt to introduce periodic ”membership updating” signaling into the protocol. This was done due to observations of long membership setup times in *sparse domains* (domains requiring many hops on average between a requester and the central authority and where routes would have a higher frequency of invalidation).

A pro-active, periodic updating of node membership was proposed and implemented using EMs. This would alleviate the need of constant Membership re-requests. However, this approach added to the signaling in the domain and negatively affected the reactive properties of the protocol, contradicting the choice of the underlying reactive routing protocol. More effort was instead put into studying the long Membership set-up times.

We found that MBs, being extensions to a current routing protocol beacon, were only triggered using a timer mechanism in the routing protocol, and that new Member nodes would not immediately signal their Membership to neighbor nodes on entry into the administrative domain. This had the effect of the administrative domain size spreading very slowly, as the radius of the administrative domain could only rise one step, each time the synchronized MBs fired. This problem was solved by triggering an ”early MB” directly when a node becomes Member in the administrative domain (see section 3.7.1).

Chapter 4

Implementation

4.1 Introduction

In this chapter, we present the implementation of MGP, which is based on the protocol description defined in chapter 3. MGP was implemented as an extension to *Uppsala University implementation of AODV (AODV-UU)*, a well-known, reactive MANET routing protocol implementation.

4.2 AODV

AODV [1] is a routing protocol which handles the maintenance of routes between mobile computers. AODV allows mobile computers not in direct contact with each other to communicate by discovering routes via a series of intermediate neighboring nodes. Out of a number of possible routes to a destination, the route with the shortest successful routing message roundtrip time is chosen, since the first valid routing reply that reaches the route seeking node is assumed to have traveled the shortest or "best" way. Neighboring nodes monitor the link connectivity between each other either by link layer information or, like in AODV-UU, by the use of application layer *Hello Beacon (HELLO)* messages.

Each node broadcasts a periodical HELLO message with a TTL of 1, which is not forwarded by receiving nodes but only added to a neighbor list. The actual routing takes place when a node wants to send application-layer data to a remote node to which there is no route. Route discovery is initiated by the broadcast of a *Route Request Message (RREQ)* (figure 4.1).

Every node receiving this message decides whether to forward this routing message or to reply, something a node can do only if it has a fresh enough route to the destination or if it actually is the destination.

Nodes which neither have fresh enough routes, nor are destinations for the RREQ will forward the message to neighbors by broadcasting on all interfaces. Each time a RREQ is forwarded a reverse route is built. This reverse route is used when the destination, or a node with fresh enough route towards the destination, unicasts the *Route Reply Message (RREP)* back to the RREQ-originating node. At the end of the routing message roundtrip, next-hops have been built up all the way between the source and the destination node.

Figure 4.1 shows an AODV Routing Request flood. Each node along the way towards the destination, can answer the RREQ if it has a fresh enough route to it. Freshness is decided by means of sequence numbers of routes, where higher number is newer. RREQs are broadcasted on all node interfaces to flood the MANET. Routing loops are avoided by AODV sequence numbers which ensure the "freshness" of routing message information. Every time a routing message regarding a specific destination is overheard in the network, its destination sequence number is compared to the number stored for that particular destination in the routing table. If the number in the message is at least as high, then the information in it is considered new and valid. The stored sequence number is updated to match the newest "heard" number at all times.

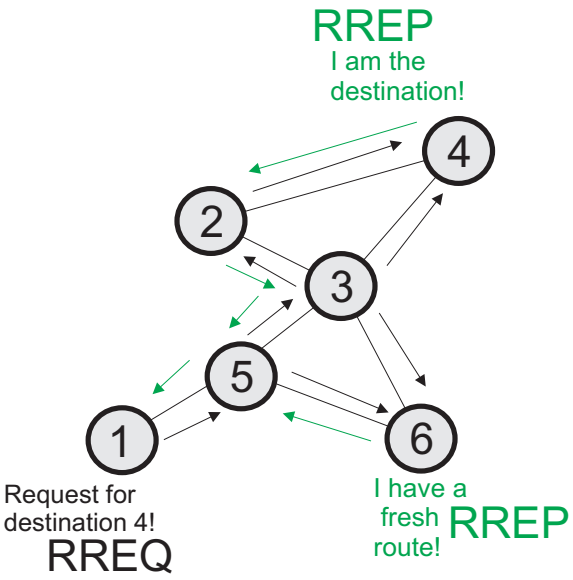

Fig. 4.1: The AODV Route Discovery process

On routing reply, the RREP message is configured with the maximum possible sequence number for the destination by choosing the maximum of the sequence number in the routing table and in the RREQ message from the RREQ originator. RREP messages with lower destination sequence numbers than the stored ones are considered old and are discarded. AODV uses a *Route Error Message (RERR)* to signal the loss of a route to neighbors marked as potential users of the route, so called precursors.

4.3 AODV-UU

4.3.1 Overview

AODV-UU [11] is the Uppsala University reference implementation of the AODV protocol. The implementation can be compiled to run native on a linux 2.4 and 2.6 kernel, or to run as a routing agent in NS-2. AODV-UU consists of a user-space routing daemon and *kaodv*, a linux kernel module passing messages based on kernel network events to the routing daemon using the *NETLINK* [7] protocol (Figure 4.2). The AODV-UU source tree (summarized in Table 4.1) is split into code running in kernel space (the kaodv module) and code running in user-space (the aodvuu daemon) as well as some files specific to compiling the protocol implementation for use with NS-2. In simplification, the kernel module sends events to the user-space routing daemon requesting route discovery and the daemon reacts, sending information about new routes found, for insertion into the kernel routing table.

4.3.2 Native Implementation

The AODV-UU routing daemon is started using the main binary, supplying the appropriate arguments on the command line. Invocation of the daemon automatically loads the kaodv kernel module and forks the process into the background. Any debug messages (output is enabled using the Makefile) are displayed directly on the terminal.

4.3.3 NS Implementation

AODV-UU can run in NS-2 [6], a network simulator to which the simulated protocol implementation is linked prior to running. A developer of a routing protocol is required to provide

Table 4.1: Subset of the file structure of AODV-UU

File name	Description
aodv_hello.c	Code for sending, receiving and processing HELLO beacons.
aodv_rerr.c	Code for sending, receiving and processing RERR messages.
aodv_rrep.c	Code for sending, receiving and processing RREP messages.
aodv_rreq.c	Code for sending, receiving and processing RREQ messages.
aodv_socket.c	Socket used for sending and receiving all AODV routing messages
aodv_neighbor.c	Code for interacting with immediate neighbors: link breaks, etc.
aodv_timeout.c	Code called using timers, such as time-outs for Route Discovery, HELLO Beacon timeouts, etc.
timer_queue.c	The timer implementation.
lnx/kaodv-debug.c lnx/kaodv-expl.c lnx/kaodv-mod.c lnx/kaodv-queue.c lnx/kaodv-netlink.c	AODV kernel-module (Kernel Space) code hooking routing events in the kernel. Kaodv-netlink.c is the kernel side of a NETLINK interface connecting the AODV routing daemon with the kernel.
nl.c	The User Space end of the NETLINK interface connecting the AODV routing daemon with the kernel.
main.c	Main entry point into the AODV routing daemon binary when run native, outside of NS-2.
ns/aodv-uu.cc	NS-2 wrapper, providing AODV as a NS-2 routing protocol. Main entry point (protocol class constructor etc.) to the protocol — when run in NS-2.
ns/packet_input.cc	NS-2 class providing a simulated packet input when running AODV-UU inside NS-2.
routing_table.c	AODV routing table, consisting of a copy of the kernel routing table, extended with AODV-specific fields.
seek_list.c	Code implementing the seek-list, a mechanism keeping state about the routing messages involved in the route discovery process.
debug.c	Debug logger for AODV

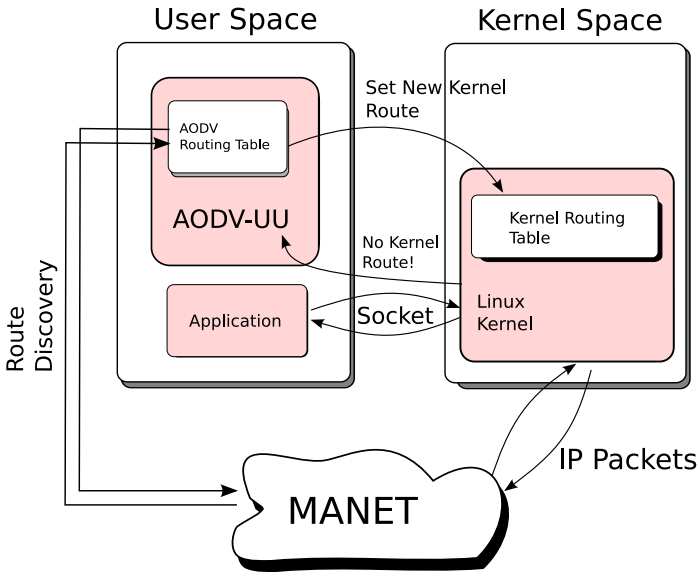

Fig. 4.2: The architecture of the AODV-UU implementation

calls into the simulator in each place where the routing protocol would normally communicate with lower layers of the network stack or the operating system. Simulations are described using OTcl language scripts. Protocol properties are exposed as OTcl Objects and some customization code must therefore be written to allow the setting of parameters in the OTcl environment. The standard AODV-UU distribution comes with the required customization for NS-2. This is installed using a source code patch, which is applied to NS-2 prior to building AODV-UU for use with NS-2. When adapting code to run in NS-2, any interactions with lower layers of the network stack must be rewritten specifically for the simulator. In AODV-UU, these are activated using compile-time arguments (in the Makefile) enabling the proper code blocks during the pre-processor step. This approach makes it possible to switch back and forth between testing AODV-UU in the simulator and on a native Linux installation, which only requires recompiling. The output data set consists of time-stamped log files describing packet data, node movement and debug output. The different files involved are described in section 4.10.

4.4 Implementation Overview

In chapter 3, we present a protocol for the administrative domain. We use generic and descriptive naming of the roles and message types involved. To separate our MGP implementation from the Protocol Description, we introduce a new naming convention for MGP. Table 4.2 presents concepts of the protocol description in the left column. The right column contains the corresponding MGP implementation names. Table 4.4 describes new files contributed to the AODV-UU tree by our MGP implementation, and table 4.3 presents the files which have been modified by us.

4.5 The Mobile Group

The *Mobile Group Leader (MGL)* is the MGP implementation of the central authority (section 3.6.1). The *Mobile Group Node (MGN)* represents the node (section 3.6.2) and the *Mobile Group Gateway (MGG)* is the MGP implementation of the gateway (section 3.6.3). Each node carries a *Mobile Group Host Info Block (MGHI)*, which implements the identity tag described in section 3.5. The MGHI is attached to the message header of all MGP messages. The internal AODV-UU routing table is extended with a MGHI entry for each route.

Table 4.2: Concepts in the Protocol Description mapped to the MGP protocol.

Protocol Description	Protocol Implementation
administrative domain	*Mobile Group (MG)*
central authority	*Mobile Group Leader (MGL)*
node	*Mobile Group Node (MGN)*
gateway	*Mobile Group Gateway (MGG)*
Membership Beacon (MB)	*MGP-enhanced HELLO beacon (MGP_HELLO)*
Membership Request (MR)	*MGP-enhanced Route Request Message (MGP_RREQ)*
Membership Welcome (MW)	*MGP-enhanced Route Reply Message (MGP_RREP)*
Event Message (EM)	MGP_MESSAGE
Route Discovery Permission Request (RDPR)	RREQ_PERMISSION_REQUEST
Route Discovery Permission (RDP)	RREQ_PERMISSION_GRANTED
Route Discovery Denial (RDD)	RREQ_PERMISSION_DENIED
Gatewaying Request (GR)	MGP_MGG_APPLICATION
Gatewaying Granted (GG)	MGP_MGG_GRANTED
routing message extension	MGP_CONTROL_MESSAGE

Table 4.3: AODV-UU files modified by the MGP extension

File name
aodv_rreq.c
aodv_rrep.c
aodv_hello.c
debug.c
routing_table.c
main.c
aodv_timeout.c
aodv_neighbor.c

Table 4.4: Files added by the MGP extension (header files left out).

File name
default_policies.c
aodv_mgp_mgg.c
aodv_mgp_policy.c
aodv_mgp.c
aodv_mgp_timerlist.c

Table 4.5: The common MGP message header

1																													32
MGN ID																													
MGN TYPE																													
MGL ID																													
MG ID																													
UNUSED																													
UNUSED																													

4.6 Membership

AODV-UU contains an extension mechanism making it easy to add custom content to the headers of its routing messages. The Membership process consists of extensions to the RREQ and RREP messages, adding the new message formats MGP_RREQ and MGP_RREP. It is implemented by adding code to the AODV_RREQ.C and AODV_RREP.C files. (Table 4.4). Both extensions contain the common MGP header (table 4.5), which is added to the end of the AODV-UU header by "piggy-backing".

MGP_RREQ and MGP_RREP messages are used for the Membership Request process only. Subsequent routing uses regular AODV-UU routing messages. Thanks to the transparent extension mechanism, the AODV-UU implementation experiences the new message types as plain RREQ and RREP messages. This makes it possible to maintain the broadcast style of the RREQ in the MGP_RREQ and the reverse-route building functionality of the RREP in MGP_RREP.

The membership process starts by a non-member node hearing a MGP_HELLO. The default policy is to broadcast a MGP_RREQ using the MGL address extracted from the MGHI of the MGP_HELLO. Upon receiving the MGP_RREQ, the MGL consults its *Policy Engine (PE)*. Based on the decision taken, it can answer with a positive, by issuing a MGP_RREP.

The membership process is proactive in the sense that it is triggered by the receiving of a beacon and not by route discovery. A future enhancement would be to initiate membership signaling in the reactive route discovery process, making the MGP implementation purely reactive, but also more complex.

4.7 Routing Control

Control of intra-MG routing is introduced into the start of the route discovery process. The three message types described in this section are not extensions of current AODV-UU message types, but instead implemented using MGP_MESSAGEs (EMs), and all functionality reacting to the receiving of these message types is handled by providing call-back functions to the PE (section 4.8). The route discovery process is modified to be started on the return of a RDPR (figure 4.4) instead of a kernel message saying that there is no known route to the destination

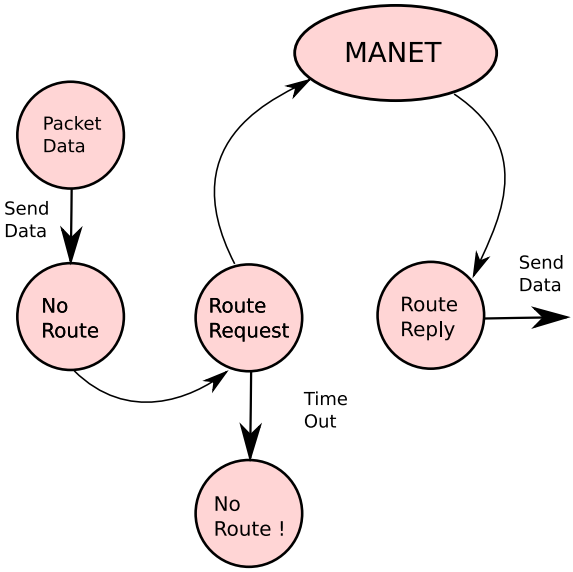

Fig. 4.3: The AODV-UU route discovery process

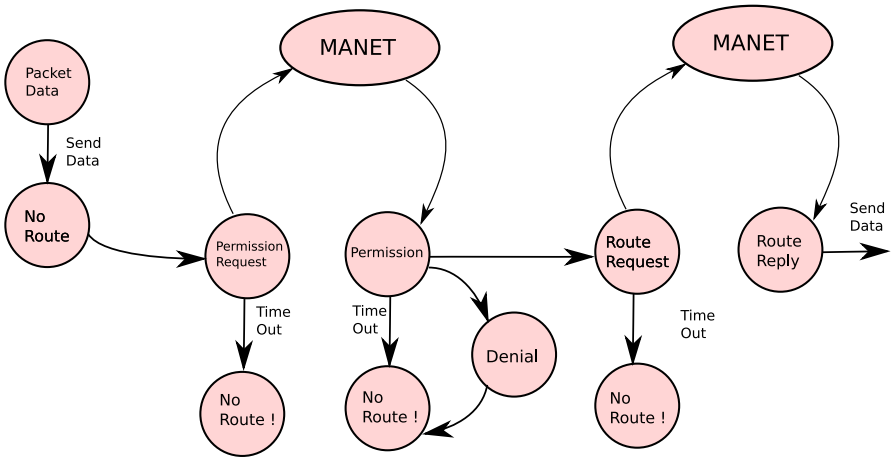

Fig. 4.4: The controlled MGP route discovery process

(figure 4.3). This kernel event instead triggers the sending of a RDPR to the MGL. The process is asynchronous and a timer ensures that the lack of a timely RDP or the receiving of a RDD will be reported as a no-route-to-host to the upper layers of the protocol stack.

4.8 The Policy Engine

4.8.1 Policy Engine API

We have implemented the PE as a layer abstracting the internal workings of the routing protocol in order to facilitate extensibility of the routing protocol, and as a design principle making the MGP code more readable.

The policy decision logic is broken out into call-back functions registered into the PE using the PE API.

The prototype of a PE call-back function,

```
typedef void (*callback_function_t)(void *, void *);
```

specifies two arbitrarily typed references, one for supplying an argument pointer and one for supplying a response pointer. Using this method, the only modification to the original MGP code that has to be done when implementing policy-based decision making, is a initialization call to the PE, supplying a callback function and an event type under which it is registered. The PE provides a simple API, making MGP act as a framework for adding custom decision logic to AODV-UU. A developer can design a custom message format based on *MGP Message (MGP_MESSAGE)* with a custom event type number and register a handler for this event type number in the PE. Within the provided call-back function, the API user adds some code which when executed is able to interact using messages and/or change some state in AODV-UU. Using AODV_MGP_TIMERLIST.C, a user can register an event into a *timer list*, to be executed when a timer times out. This makes it possible to implement, for example, the re-sending of custom messages in specified intervals, or to time out custom MGP state.

We have used MGP_MESSAGEs to implement the Routing Control mechanism, as well as the gateway Permission Request mechanism in the default MGP implementation, registering their decision-making as PE call-back functions.

4.8.2 Default Policies

For practical reasons, we have hard-coded the default MGP policy decisions using regular functions accessible in DEFAULT_POLICIES.C. These functions generally return an integer value describing the policy decision made based on the input, but since they are internally executed in various places of the MGP code, they can differ in number and type of input arguments. These are:

- HANDLE_MEMBERSHIP_REQUEST,
  executed by the MGL when receiving a MGP_RREQ.
- HANDLE_RREQ_REQUEST,
  executed by the MGL when receiving a RDPR.
- HANDLE_MGG_APPLICATION,
  executed by the MGL when receiving a *Gatewaying Request (GR)*.
- HANDLE_WELCOME,
  executed by a MGN when receiving a MGP_RREP.
- HANDLE_NEW_MG_FOUND,
  executed by a MGN or MGG when hearing a MGP_HELLO from an exterior node.
- HANDLE_LOST_MG,
  executed when a MGN or MGG looses the last route to its MGL

- HANDLE_LOST_MGG_MG,
  executed when a MGN looses the last route to an exterior MG.

- HANDLE_FORWARD,
  executed before forwarding a MGP_MESSAGE.

4.9 Testing the Native Implementation

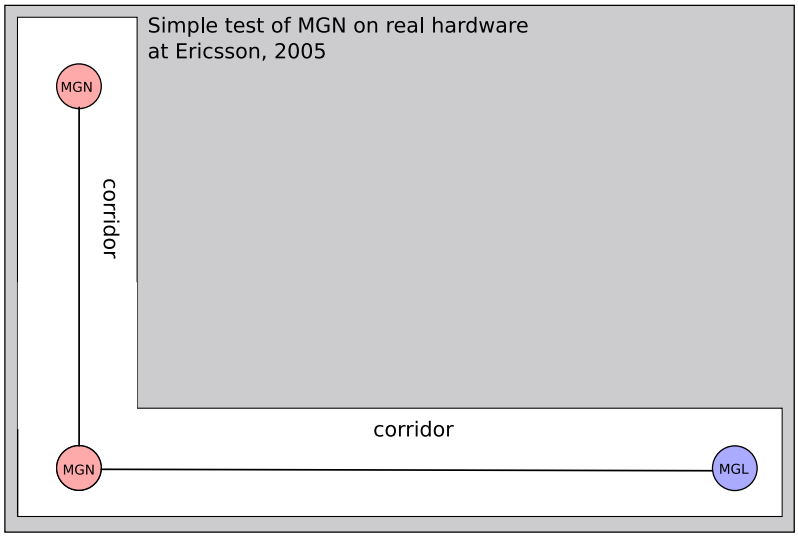

Fig. 4.5: Real-world MGP testing

Using three Linux laptops equipped with Orinoco and Centrino wireless adapters, we were able to successfully run MGP using one laptop as MGL and two MGN laptops. A MGN was used to relay packets between another MGN and the MGL. Figure 4.5 gives a sketch of the simulation topology. The scenario was useful for verifying basic group formation and dissmissal on real hardware but too small for collecting more detailed data. Each laptop displayed the MGP logs and a constantly refreshing routing table. Membership was successfully aquired and lost depending on the existence of a working route between MGN and MGL.

4.10 Logging

When running AODV-UU in NS-2, every node places debug data in a separate AODV-UU.LOG text file marked with the NS-2 node number. If the AODV-UU routing daemon runs stand-alone, debug logging in AODV-UU is written to the terminal. To make the analysis of MGP simulations easier, we extract all MGP specific debug logging into equally numbered MGP.LOG-files, containing time stamped information about membership, policy decisions, MGP EMs and MGP events. During analysis of the simulation output, MGP.LOG-files are cross-referenced with the corresponding AODV-UU.LOG files and *trace files*, which contain dumps of the actual data flowing in the simulated network. The trace file writer CMU-TRACE.CC has been extended by adding output of MGP-specific packet attributes.

4.11 Visualization

In addition to the output files described in section 4.10, protocols implemented for running in NS-2 typically supply code for generating *Network Animator (NAM)* animations. The NAM

animator is a very simple tool, reading a .NAM text file, providing a visual history of network events regarding packets, node topology changes, router queues etc. and renders an animation with a time line of the simulation. nodes are typically depicted as black circles, network media (typically consisting of fixed wiring) is represented using black edges between circles, and data packets flowing through he network are illustrated using small flat boxes, or in the case of wireless networks, as expanding circles with the wireless nodes at their centers.

When running MGP in NS-2, we want to visualize the MGHI, as well as distinguish between message types sent in the network. Therefore, MGP contributes code to print node identity and node type above nodes. The coloring of nodes is introduced to distinguish the nodes based on group identity. Since no fixed network media exists in the simulation scenarios, the edge line of NAM is "hijacked" by us to illustrate sent messages between nodes. Edges are color coded based on the event type of messages sent.

Bugs and general instability of NAM (including random redraws of already expired message packets etc) ultimately discouraged us from using the NAM graphs directly in this thesis. Instead, we used it as a debugging tool for simulation scripts. All topography charts in this thesis are drawn based on the original NAM graphs. Ping traffic graphs where charted using bash/awk scripts and Xgraph, parsing NS-2 trace and combining them with events from the MGP logs.

Chapter 5

Simulation 1

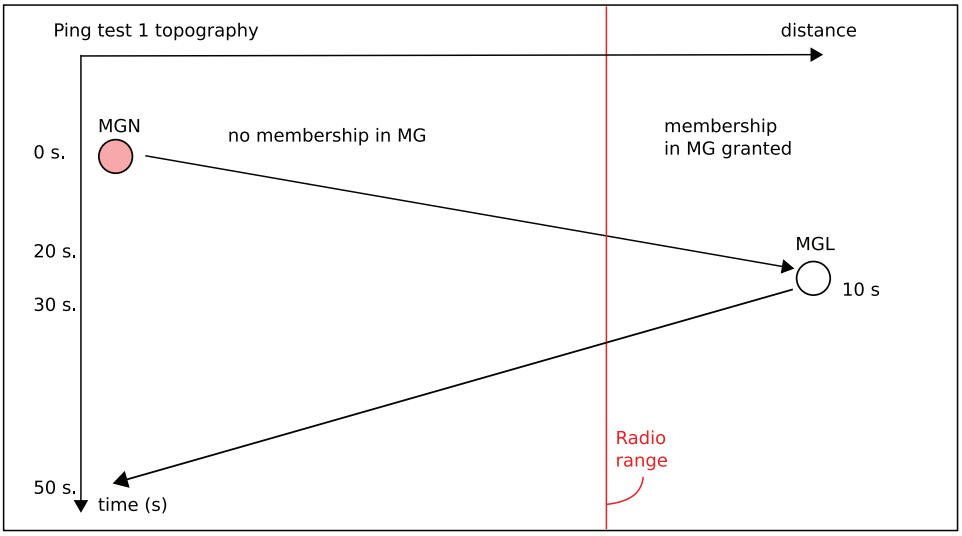

Fig. 5.1: Scenario 1

The purpose of the first simulation, which is illustrated in fig. 5.1, is to study if the basic membership mechanism works. The scenario is one of the simplest possible and consists of one mobile non-member MGN (marked as _0_ in the traffic logs) moving towards an active MGL (marked as _1_ in the logs). When the MGN reaches the MGL, it stops and waits 10 seconds and then returns to its original position during 20 seconds. During the whole experiment the MGN will try to ping the MGL at a rate of 4 Echo Requests per second.

5.1 Applied Policies

In a real-life situation, the MGN would probably not know the MGL address beforehand, and would therefore not know what it is supposed to ping. Since this is a simulation and we are only interested in whether the MG is formed correctly we leave out the process where the MGN first learns the identity of a possible MGL from a MGHI block in a MGP_HELLO beacon. Instead we hard code its address into the simulator script and use the simulator to schedule the MGN to ping the MGL four times per second. Pings are only transmitted when there is an active route to the destination. During the experiment, the MGL will continously send MGP_HELLO beacons which will be heard by no one at first. As the MGN gets closer to the MGL, it eventually hears the MGP_HELLO beacon and its default policy will be to try to acquire membership by sending a MGP_RREQ to the MGL. The MGL should then

immediately consult its PE which has a default policy of accepting all membership requests. This should result in a MGP_RREP from MGL to MGN which grants MG membership. The MGN now reconfigures the MGHI according to its new identity as a MG member. The result of the membership process should be a new MG consisting of two member nodes. The MGN will stay at the position of the MGL for 10 seconds, after which it will start moving back to its point of origin during 20 seconds. At approximately the same point (noted with "radio range" in figure 5.1) where it first heard the MGL, the MGN should now stop getting replies for its ping echo requests. Shortly after, the route towards the MGL should time out and ping messages stop being transmitted by the interface on the MGN. As soon as the MGL route disappears, the MGN will by default change its MG membership state to unknown and stop sending MGP_HELLO messages.

5.2 Results

The NS-2 trace file has been filtered to show selected Ping and MGP_HELLO traffic in listing 5.1 and listing 6.2 Both listings contain rows prefixed with line numbers from the complete trace file which is to large to include in this document. Figure 5.2 shows a plot of the ping

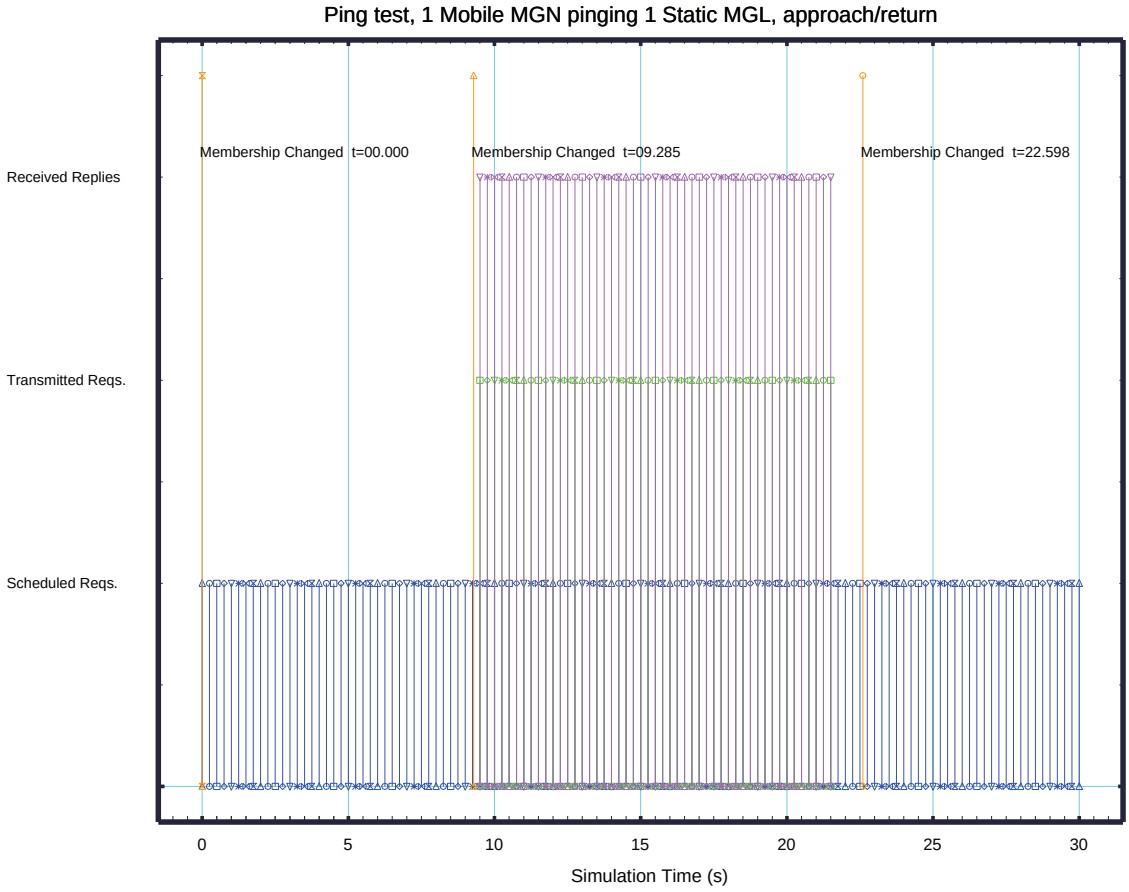

Fig. 5.2: Ping results, sim 1

traffic during Simulation 1. The graph is divided into three vertical sections. The bottom

one shows the four per second scheduled echo requests to be sent from the MGN and replied for by the MGL during the whole 30 second experiment. This section verifies the frequency of echo requests and the duration of the experiment. The middle section represents all ping echo requests actually transmitted from the network interface and verify that a route existed at that time, since without a route, no echo request is sent at all. The top section represents all echo replies at the source, that is, concludes that there are working round-trip routes: from the MGN to the MGL and then back again. The X-axis of the graph represents the time-line of the experiment. At four occasions during

```
s 9.501051034 _0_ MAC --- 50 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 9.750991017 _0_ MAC --- 52 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 10.001211001 _0_ MAC --- 54 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 10.251091001 _0_ MAC --- 56 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 10.501091001 _0_ MAC --- 60 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 10.750931001 _0_ MAC --- 62 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 11.000871001 _0_ MAC --- 64 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 11.250891001 _0_ MAC --- 66 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 11.501271001 _0_ MAC --- 70 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 11.750771001 _0_ MAC --- 72 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 12.000771001 _0_ MAC --- 74 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 12.251231001 _0_ MAC --- 76 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 12.501271001 _0_ MAC --- 80 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 12.750851001 _0_ MAC --- 82 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 13.001251001 _0_ MAC --- 84 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 13.251111001 _0_ MAC --- 86 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 13.501051001 _0_ MAC --- 90 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 13.750791001 _0_ MAC --- 92 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 14.001351001 _0_ MAC --- 94 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 14.251131001 _0_ MAC --- 96 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 14.501111001 _0_ MAC --- 100 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 14.750791001 _0_ MAC --- 102 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 15.001271001 _0_ MAC --- 104 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 15.251011001 _0_ MAC --- 106 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 15.501171001 _0_ MAC --- 110 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 15.750851001 _0_ MAC --- 112 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 16.001231001 _0_ MAC --- 114 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 16.250771001 _0_ MAC --- 116 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 16.501191001 _0_ MAC --- 120 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 16.751291001 _0_ MAC --- 122 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 17.000971001 _0_ MAC --- 124 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 17.250751001 _0_ MAC --- 126 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 17.501945265 _0_ MAC --- 130 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 17.751091001 _0_ MAC --- 132 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 18.001131001 _0_ MAC --- 134 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 18.251271001 _0_ MAC --- 136 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 18.501351001 _0_ MAC --- 139 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 18.751351001 _0_ MAC --- 142 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 19.001031001 _0_ MAC --- 144 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 19.250871001 _0_ MAC --- 146 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 19.501331001 _0_ MAC --- 148 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 19.751131001 _0_ MAC --- 152 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 20.000791001 _0_ MAC --- 154 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 20.250891018 _0_ MAC --- 156 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 20.500751034 _0_ MAC --- 158 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 20.750831051 _0_ MAC --- 162 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 21.001131068 _0_ MAC --- 164 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 21.251031084 _0_ MAC --- 166 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
s 21.501031101 _0_ MAC --- 168 ping 116 [13a 1 0 800] ------- [0:0 1:0 32 1]
```

Listing 5.1: ICMP Echo Requests (PINGs) actually transmitted during Simulation 1

the simulation, group membership is changed. The first is implicit, during the setup phase of the simulation when both nodes get their initial group identities: MGN none (0) and the MGL, MG 5. At roughly t=9.2667 s, the MGN is close enough to be able to hear its first MGP_HELLO (Marked line in Listing 6.2), ask for membership and receive a positive MGP_RREP. It changes its MG identity to 5. At roughly t=9.501 s, the first actual ping is sent(the first line of listing 5.1) which means that an actual route is up. According to the graph, a reply is also received for this ping. Just after t=21.501, the last ping is successfully sent (and a reply is received). At t=22.598 the group membership is cancelled at the MGN which indicates that the MGL route has been lost. In listing 6.2, we can see that HELLO messages set a neighbor route time-out of 2000 ms. Since the last MGP_HELLO from the MGL is heard at the MGN at t= 20.548853019, which is almost exactly 2 seconds earlier we can conclude that the MGN forces the loss of its own membership (a built-in policy) due to the time-out of a neighbor route, more specifically: The route to the MGL itself.

Chapter 6

Simulation 2

6.1 Scenario Overview

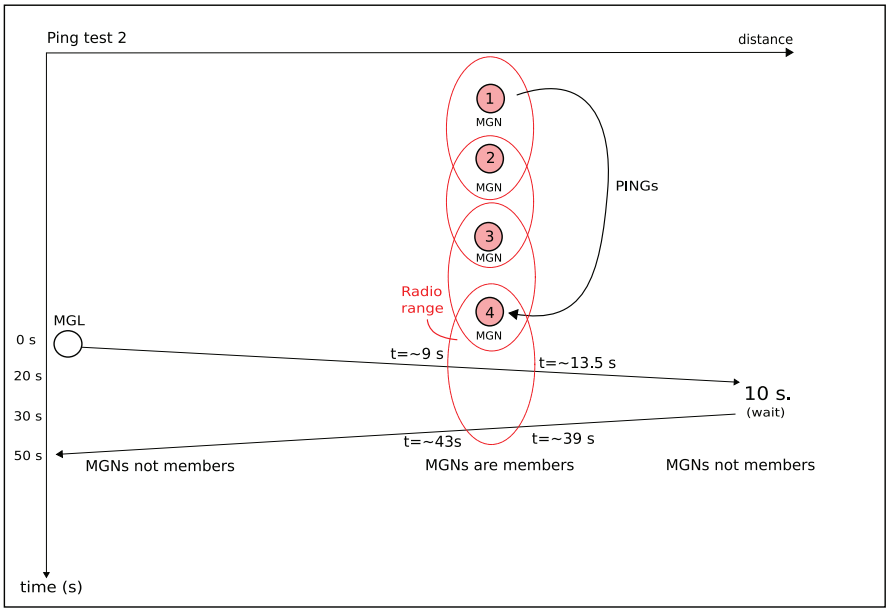

Fig. 6.1: Scenario 2

The second simulation is an extension of the first one. Figure 6.1 is an image of the scenario. The MGL (marked _0_ in the trace log) is mobile and travels into and through a group of MGN candidates on its way to a distant point on the other side, and past radio range. The trip takes 20 seconds. It waits at the destination for 10 seconds only to return to its origin on the same path and with the same speed. The return trip takes 20 seconds. During two short periods of time, the MGL will be in radio range with the MG candidates. Only node MGN 4 fig. 6.1 (marked _4_ in the traces) will be in range to hear its MGP_HELLO message and will be able to apply for membership.

```
s 1.007157453 _1_ MAC --- 5 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 2.049112303 _1_ MAC --- 10 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 3.082236353 _1_ MAC --- 15 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 4.118414571 _1_ MAC --- 20 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 5.117500137 _1_ MAC --- 25 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 6.148396131 _1_ MAC --- 30 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 7.173457969 _1_ MAC --- 35 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 8.227480956 _1_ MAC --- 40 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 9.265474548 _1_ MAC --- 46 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 9.266578573 _0_ MAC --- 46 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)}
s 9.293854627 _0_ MAC --- 49 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 9.294958651 _1_ MAC --- 49 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
```

```
s 10.300795082 _1_ MAC --- 58 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 10.301899082 _0_ MAC --- 58 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 10.329910601 _0_ MAC --- 59 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 10.331014601 _1_ MAC --- 59 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 11.338338620 _1_ MAC --- 68 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 11.339442620 _0_ MAC --- 68 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 11.353365849 _0_ MAC --- 69 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 11.354469850 _1_ MAC --- 69 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 12.357910027 _0_ MAC --- 78 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 12.359014028 _1_ MAC --- 78 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 12.360867986 _1_ MAC --- 79 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 12.361971986 _0_ MAC --- 79 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 13.389637220 _0_ MAC --- 88 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 13.390741221 _1_ MAC --- 88 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 13.405701772 _1_ MAC --- 89 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 13.406805772 _0_ MAC --- 89 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 14.414947229 _0_ MAC --- 98 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 14.416051229 _1_ MAC --- 98 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 14.427051060 _1_ MAC --- 99 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 14.428155060 _0_ MAC --- 99 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 15.437977712 _1_ MAC --- 109 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 15.439081712 _0_ MAC --- 109 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 15.440398148 _0_ MAC --- 108 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 15.441502148 _1_ MAC --- 108 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 16.446377063 _1_ MAC --- 118 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 16.447481063 _0_ MAC --- 118 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 16.465373152 _0_ MAC --- 119 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 16.466477152 _1_ MAC --- 119 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 17.473049158 _1_ MAC --- 128 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 17.474153158 _0_ MAC --- 128 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 17.500035264 _0_ MAC --- 129 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 17.501139265 _1_ MAC --- 129 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 18.492377222 _1_ MAC --- 138 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 18.493481222 _0_ MAC --- 138 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 18.515249209 _0_ MAC --- 141 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 18.516353209 _1_ MAC --- 141 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 19.547423341 _1_ MAC --- 150 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 19.548527342 _0_ MAC --- 150 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 19.555979218 _0_ MAC --- 151 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 19.557083218 _1_ MAC --- 151 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 20.547749001 _1_ MAC --- 160 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
r 20.548853019 _0_ MAC --- 160 AODVUU 86 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 20.557129181 _0_ MAC --- 161 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
r 20.558233200 _1_ MAC --- 161 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 21.568590142 _0_ MAC --- 170 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 21.605276142 _1_ MAC --- 171 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 22.604800815 _0_ MAC --- 176 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 2] [0] 2000.000000] (HELLO)
s 22.625753994 _1_ MAC --- 177 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 23.662147468 _1_ MAC --- 182 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 24.712517065 _1_ MAC --- 187 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 25.722619786 _1_ MAC --- 192 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 26.765456283 _1_ MAC --- 198 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 27.771062425 _1_ MAC --- 203 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 28.814700431 _1_ MAC --- 208 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
s 29.841257754 _1_ MAC --- 213 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 1] [1] 2000.000000] (HELLO)
```

Listing 6.2: HELLO traffic sent/received during Simulation 1

The group of MGN candidates are all placed so that they have a maximum radio range of one neighbor. This way we can observe a sequential group formation where a node must become member in order for its neighbor to try to aquire membership and so on, all during which the MGL is moving through the group. The purpose is to study how traffic starts to flow in a newly formed MG and how it stops when the MGL disapears. Are routes and membership set up and timed out correctly during this short existence of a MG?

6.2 Applied Policies

In order for the Ping traffic between the two nodes at the far ends of the formation (MGN 1 and MGN 4) to work there have to be active routes between all of the nodes on the path between MGN 1 and MGN 4. Since default policies in MGP only permit the forming of routes between members of the same MG, we will only see successful round-trip Ping traffic when all of our nodes have become members. Each node sends a MGP_HELLO message as soon as it is allowed into the MG without waiting for a AODV hello timer. Therefore we should typically see a rather fast dissemination of membership state.

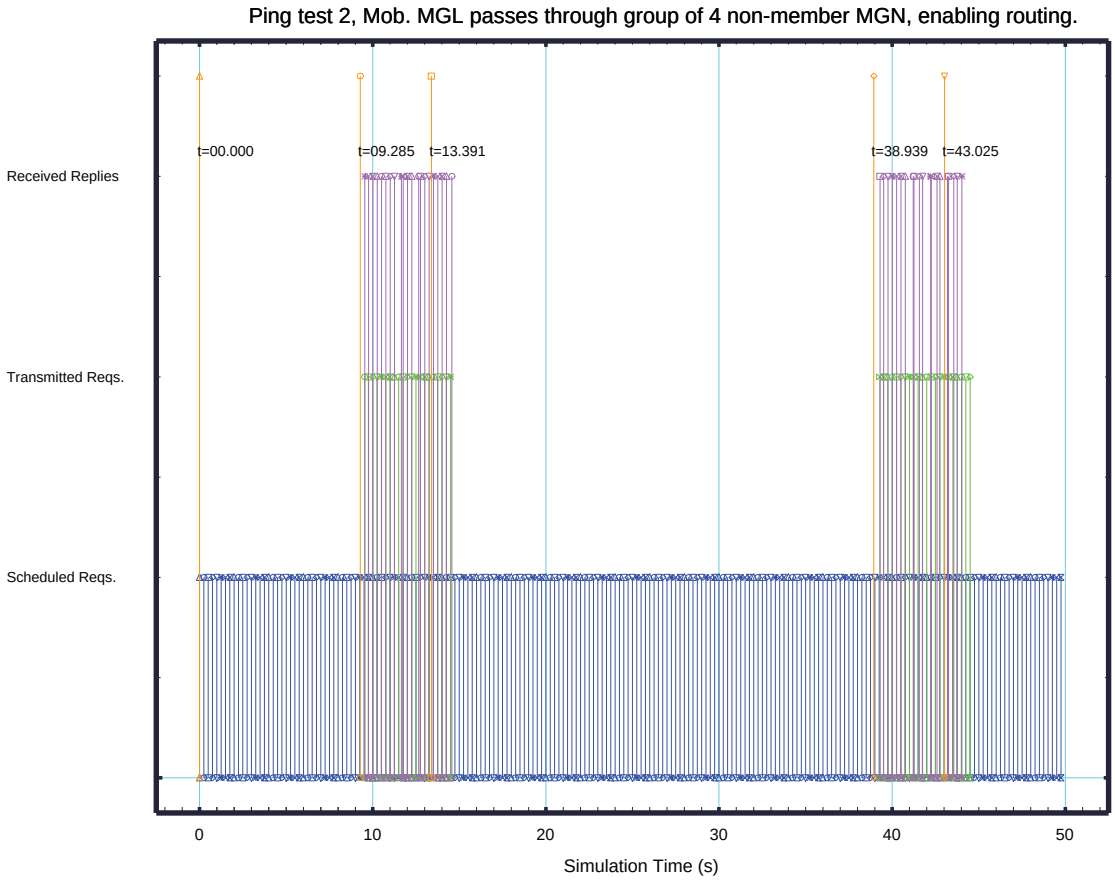

Fig. 6.2: Ping results, sim. 2

6.3 Results

6.3.1 The forming of a MG

Just as in fig. 5.2, the new Ping traffic graph (6.2) is split into three sections: Scheduled Pings, Transmitted Pings and Received Replies. The vertical bars ("t=...") respresent membership changes of MGN 4, the node closest to the trajectory of the mobile MGL. Listing 6.3 shows the sent and received HELLO messages during the first MG formation (line numbers in column 1 of the listing). At roughly t=9.265 (line 1) a MGP_HELLO is sent by the MGL (marked as node _0_ in the trace logs). Since the nodes have carefully been placed so as to not hear any other traffic than that of the closest neighbors, this message is heard only by MGN 4 (line 2). In line 10 of listing 6.3 we see that the first MGP_HELLO reaches MGN 1 (the most distant node in the system) at approximately t=9.39 s which is less than 121 ms. later than the first MGP_HELLO message was received from the MGL (line 2). Finally we can observe that a complete MG has been formed at roughly t=9.45 s (line 12 of listing 6.3) when MGN 1 sends its first MGP_HELLO. The first Ping Echo request is sent already at t=9.54. (listing 6.4). The listing shows the first round-trip Ping conversation proving that the required routes have been set up and that this is an effect of a complete MG.

6.3.2 Membership lost

The Ping traffic continues at the 4 per second interval with the last successful reply received at MGN 1 at roughly t=13.50 s. (listing 6.5). Figure 6.2 tells us that additional pings are transmitted for a while longer, suggesting that this is probably until MGN 1 looses MG membership and/or a route towards MGN 4. Actually, by studying the AODV-UU log file for MGN 1 around that time (listing 6.6), we learn that the neighbor MGN 2 sends an AODV RRER message about the MGL not being reachable

```
1 s 9.265474548 _0_ MAC --- 46 AODVUU 138 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 1] [0] 2000.000000] (HELLO)
2 r 9.266578572 _4_ MAC --- 46 AODVUU 86 [0 ffffffff 0 800] ------- [0:255 -1:255 1 0] [0x2 0 [0 1] [0] 2000.000000] (HELLO)
3 s 9.293854623 _4_ MAC --- 49 AODVUU 138 [0 ffffffff 4 800] ------- [4:255 -1:255 1 0] [0x2 0 [4 2] [4] 2000.000000] (HELLO)
4 r 9.294958647 _0_ MAC --- 49 AODVUU 86 [0 ffffffff 4 800] ------- [4:255 -1:255 1 0] [0x2 0 [4 2] [4] 2000.000000] (HELLO)
5 r 9.294958671 _3_ MAC --- 49 AODVUU 86 [0 ffffffff 4 800] ------- [4:255 -1:255 1 0] [0x2 0 [4 2] [4] 2000.000000] (HELLO)
6 s 9.337963762 _3_ MAC --- 54 AODVUU 138 [0 ffffffff 3 800] ------- [3:255 -1:255 1 0] [0x2 0 [3 2] [3] 2000.000000] (HELLO)
7 r 9.339067810 _4_ MAC --- 54 AODVUU 86 [0 ffffffff 3 800] ------- [3:255 -1:255 1 0] [0x2 0 [3 2] [3] 2000.000000] (HELLO)
8 r 9.339067812 _2_ MAC --- 54 AODVUU 86 [0 ffffffff 3 800] ------- [3:255 -1:255 1 0] [0x2 0 [3 2] [3] 2000.000000] (HELLO)
9 s 9.386107863 _2_ MAC --- 61 AODVUU 138 [0 ffffffff 2 800] ------- [2:255 -1:255 1 0] [0x2 0 [2 2] [2] 2000.000000] (HELLO)
10 r 9.387211913 _1_ MAC --- 61 AODVUU 86 [0 ffffffff 2 800] ------- [2:255 -1:255 1 0] [0x2 0 [2 2] [2] 2000.000000] (HELLO)
11 r 9.387211913 _3_ MAC --- 61 AODVUU 86 [0 ffffffff 2 800] ------- [2:255 -1:255 1 0] [0x2 0 [2 2] [2] 2000.000000] (HELLO)
12 s 9.452966221 _1_ MAC --- 69 AODVUU 138 [0 ffffffff 1 800] ------- [1:255 -1:255 1 0] [0x2 0 [1 2] [1] 2000.000000] (HELLO)
```

Listing 6.3: Initial HELLO traffic received during Simulation 2

anymore which is received at t=12.343. The MGL route (0.0.0.0) is immediately invalidated and after 2 additional seconds the rest of the routing table is invalidated (routes towards MGN 2 and MGN 4) because of time-outs. In the two last lines of the listing, we can study how the sending of Ping messages starts getting dropped at t=14.5 s. Listing 6.7 shows what happens on the MGP layer at the same time. The MGL route invalidation is detected and the default policy of membership cancelling is executed at t=12.343, immediately after the loss of the MGL route. The Ping Echo-Request scheduled for sending at t=14.5 is denied the triggering of an AODV RREQ since the local policy is to not allow this when the node has no MG membership (last line of the listing). However, between the time of the loss of membership at t=12.343 and the last successfully sent ping at t=14.25 s, 8 Ping Echo Requests (Listing 6.8) are sent between MGN 1 and MGN 4 even though no MG membership exists at that point in time. This is a good example of how MGP works. The policies implemented by us only control permissions for setting up new routes in the MG. Because neighbor links still exist between MGN 1 and MGN 2 during approx. 2 seconds, Ping traffic from MGN 1 will continue to flow until AODV/MGP HELLO beacons stop being received which causes these neighbor links to time out at t=14.5 s.

```
s 9.539653299 _1_ MAC --- 70 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 9.539912258 _2_ MAC --- 70 ping 64 [13a 2 1 800] ------- [1:0 4:0 32 2]
s 9.540952358 _2_ MAC --- 70 ping 116 [13a 3 2 800] ------- [1:0 4:0 32 3]
r 9.541211317 _3_ MAC --- 70 ping 64 [13a 3 2 800] ------- [1:0 4:0 32 3]
s 9.542371413 _3_ MAC --- 70 ping 116 [13a 4 3 800] ------- [1:0 4:0 32 4]
r 9.542630370 _4_ MAC --- 70 ping 64 [13a 4 3 800] ------- [1:0 4:0 32 4]
```

```
s 9.543830466 _4_ MAC --- 81 ping 116 [13a 3 4 800] ------- [4:0 1:0 32 3]
r 9.544089423 _3_ MAC --- 81 ping 64 [13a 3 4 800] ------- [4:0 1:0 32 3]
s 9.545489523 _3_ MAC --- 81 ping 116 [13a 2 3 800] ------- [4:0 1:0 32 2]
r 9.545748482 _2_ MAC --- 81 ping 64 [13a 2 3 800] ------- [4:0 1:0 32 2]
s 9.546928582 _2_ MAC --- 81 ping 116 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 9.547187541 _1_ MAC --- 81 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
```

Listing 6.4: The first PING round-trip

```
r 9.547187541 _1_ MAC --- 81 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 9.759405343 _1_ MAC --- 83 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 10.009025343 _1_ MAC --- 85 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 10.259005343 _1_ MAC --- 87 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 10.509445343 _1_ MAC --- 92 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 10.759205343 _1_ MAC --- 96 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 11.010285343 _1_ MAC --- 98 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 11.258865343 _1_ MAC --- 100 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 11.510045343 _1_ MAC --- 105 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 11.759745343 _1_ MAC --- 109 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 12.008765343 _1_ MAC --- 111 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 12.259745343 _1_ MAC --- 113 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 12.510799456 _1_ MAC --- 121 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 12.759065343 _1_ MAC --- 123 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 13.009505343 _1_ MAC --- 125 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 13.259165343 _1_ MAC --- 127 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 13.509485343 _1_ MAC --- 131 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 39.057047671 _1_ MAC --- 306 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 39.260045343 _1_ MAC --- 308 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 39.508965343 _1_ MAC --- 310 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 39.759365343 _1_ MAC --- 312 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 40.010479439 _1_ MAC --- 317 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 40.259565343 _1_ MAC --- 321 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 40.509785343 _1_ MAC --- 323 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 40.759005343 _1_ MAC --- 325 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 41.009065343 _1_ MAC --- 329 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 41.259625343 _1_ MAC --- 334 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 41.509725343 _1_ MAC --- 336 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 41.758905343 _1_ MAC --- 338 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 42.009245343 _1_ MAC --- 343 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 42.259685343 _1_ MAC --- 348 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 42.509445343 _1_ MAC --- 350 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 42.758845343 _1_ MAC --- 352 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
r 43.009225343 _1_ MAC --- 356 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
```

Listing 6.5: All received Ping Replies during Simulation 2.

```
0.0.0.1: 00:00:12.343 aodv_socket_process_packet: Received RERR
0.0.0.1: 00:00:12.343 rerr_process: ip_src=0.0.0.2
0.0.0.1: 00:00:12.343 log_pkt_fields: rerr->dest_count:1 rerr->flags=-
0.0.0.1: 00:00:12.343 rerr_process: unreachable dest=0.0.0.0 seqno=5
0.0.0.1: 00:00:12.343 rerr_process: removing rte 0.0.0.0 - WAS IN RERR!!
0.0.0.1: 00:00:12.343 interfaceQueue: Dropping pkts by dest for 0.0.0.0 queue_len=0
0.0.0.1: 00:00:12.343 hello_stop: No active forwarding routes - stopped sending HELLOs!
0.0.0.1: 00:00:12.343 rt_table_invalidate: 0.0.0.0 removed in 15000 msecs
0.0.0.1: 00:00:12.343 rerr_process: Not sending RERR, no precursors or route in RT_REPAIR
0.0.0.1: 00:00:14.393 hello_timeout: LINK/HELLO FAILURE 0.0.0.2 last HELLO: 2049
0.0.0.1: 00:00:14.393 neighbor_link_break: Link 0.0.0.2 down!
0.0.0.1: 00:00:14.393 rt_table_invalidate: 0.0.0.2 removed in 15000 msecs
0.0.0.1: 00:00:14.393 rt_table_invalidate: 0.0.0.4 removed in 15000 msecs
0.0.0.1: 00:00:14.500 packet_queue_add: buffered pkt to 0.0.0.4 uid=142 qlen=1
0.0.0.1: 00:00:14.500 packet_queue_set_verdict: DROPPED 1 packets for 0.0.0.4!
```

Listing 6.6: AODVUU log snippet about the loss of the MGL route at MGN 1

```
0.0.0.1: [g 5;id 1;t 1,l 0.0.0.0,d 0,mggs 0] 00:00:12.343 rt_table_invalidate: MGL route invalidated!
0.0.0.1: [g 0;id 1;t 1,l 0.0.0.0,d 0,mggs 0] 00:00:12.343 MGP_cancel_membership: Cancelled membership, will re-
aply for membership at next MGPHELLO heard
0.0.0.1: [g 0;id 1;t 1,l 0.0.0.0,d 0,mggs 0] 00:00:14.500 processPacket: No NS route for 0.0.0.4, Initiating
RREQ permission request.
0.0.0.1: [g 0;id 1;t 1,l 0.0.0.0,d 0,mggs 0] 00:00:14.500 MGP_route_discovery_request: Not member of a MG!
Cancelled sending RREQ permission request.
```

Listing 6.7: MGP log snippet about the loss of the MGL route at MGN 1

```
s 12.502125213 _1_ MAC --- 119 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 12.510799456 _1_ MAC --- 121 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
s 12.751051100 _1_ MAC --- 122 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 12.759065343 _1_ MAC --- 123 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
s 13.000811100 _1_ MAC --- 124 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 13.009505343 _1_ MAC --- 125 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
s 13.250911100 _1_ MAC --- 126 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 13.259165343 _1_ MAC --- 127 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
s 13.501291100 _1_ MAC --- 130 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
r 13.509485343 _1_ MAC --- 131 ping 64 [13a 1 2 800] ------- [4:0 1:0 32 1]
s 13.750851100 _1_ MAC --- 135 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
s 14.001111100 _1_ MAC --- 137 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
s 14.250811100 _1_ MAC --- 139 ping 116 [13a 2 1 800] ------- [1:0 4:0 32 2]
```

Listing 6.8: Ping traffic at MGN 1 after its loss of MG membership.

Chapter 7

Simulation 3

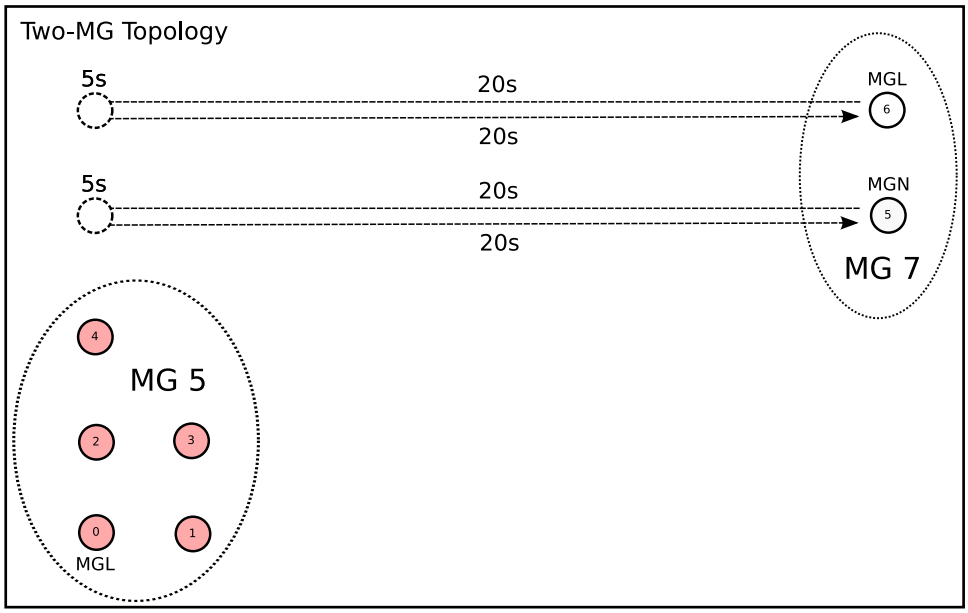

Fig. 7.1: Simulation 3

7.1 Scenario Overview

The third scenario is more complex, containing two MGs. The whole simulation script is supplied in Appendix 1, Chapter 11. Our goal is to evaluate the ability of confining routing information to the own MG while being able to form controlled gateway relationships with external MGs. We do this by studying the messaging occurring when a MGN negotiates the right to become MGG and by studying routing and traffic logs to ensure that no routes between nodes in different MGs can be formed without explicit permission.

The scenario (fig. 7.1) consists of one mobile MG (7) moving into reach of the static one (MG 5) during a 20 second voyage, followed by a 5 second rest and a 20 second return by MG 7 to the origin. MG 7 consists of one MGN, (5) and one MGL (6), whereas MG 5 consists of one MGL (0) and four MGN (1-4). At some point in time, MGN 5 and at least one MGN in MG 5 should acquire MGG status. Route establishment shall, however, not take place between MGs, permitting, for example, data packets from MGN 5 to reach a node in MG 5. An exception is the forming of a neighbor route between the interior and exterior gateways. Throughout the

whole 45 second simulation, we generate 4 ping packets per second with the source, MGN 5, and the target MGL 0, and expect that no packet ever reaches its destination, since no explicit custom policy has yet been defined by us to handle inter-MG routes.

7.2 Results

7.2.1 Membership

All MGNs establish initial MG membership successfully at first MGP_HELLO heard (MGN 1 at t=1.052s, MGN 2 at t=1.057, MGN 3 at t=1.076 and MGN 4 at t=1.106s).

MGN 5, which belongs to MG 7, has its MGL as neighbor, and acquires membership already at t=1.033 (Listing. 7.9). Since the default policy is to not change membership as long as the node has an active membership, and because MGN 5 is in touch with its MGL 6 at all times, MGN 5 stays member of MG 7 throughout the simulation.

```
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:00.750 MGP_route_discovery_request: Not member of a MG! Cancelled sending RREQ permission request.
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.000 processPacket: No NS-2 route for 0.0.0.0, Initiating RREQ permission request.
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.000 MGP_route_discovery_request: Not member of a MG! Cancelled sending RREQ permission request.
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.022 MGP_control_msg_process: MGP_CONTROL_MSG (msg->msg_type=1)
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.022 MGP_control_msg_process: Parsing MGP_HELLO message
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.022 MGP_process_hello_message: Received HELLO from [mgn_id=6,mgn_type=2,mg_id=7,mgl_addr=0.0.0.6]
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.022 handle_new_mg_found: Received HELLO-MGP that we can use for applying for membership
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.0,d 0,mggs 0] 00:00:01.022 MGP_process_hello_message: Requesting for membership to MGL=0.0.0.6
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.022 MGP_request_membership: Creating MGP_request_membership message
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.022 MGP_request_membership: Configuring MGP_request_membership message
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.022 MGP_request_membership: Sending MGP_REQUEST_MEMBERSHIP:
                                                          {socket_dest=255.255.255.255,ttl=35,src interface=0.0.0.5,size=70}
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.022 MGP_request_membership: Adding broadcast to MGL=0.0.0.6 ttl=35 in seek-list
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.022 hello_process: Received HELLO-MGP from 0.0.0.6. Adding neighbor route
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.033 MGP_control_msg_process: MGP_CONTROL_MSG (msg->msg_type=7)
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.033 MGP_control_msg_process: Parsing MGP_WELCOME
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.033 MGP_process_welcome_message: MGP_WELCOME message received =
                                                          {mgn_id=6,mgl_id=6,mgn_type=2,mg_id=7}
0.0.0.5: [g 0;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.033 MGP_process_welcome_message: Setting up new values for MG_ID, MGL_ID
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.033 MGP_NAM_trace_node: [NAM] Changing membership to MG, new value=7
```

Listing 7.9: MGN 5 Acquiring membership in MG 7

7.2.2 RREQ Permission

A first result of MGN 5 trying to ping MGL 0 is a large number of RREQ Permission Requests to MGL 6 (Listing. 7.11). In fact, once MGN 5 has been granted membership in MG 7 (t=1.033s), every ping packet queued to be transmitted by MGN 5 initiates one RREQ_PERMISSION_REQUEST to be sent to the MGL, resulting in one RREQ_PERMISSION returned to MGN 5. This happens 4 times per second, which is the frequency at which Ping transmissions are scheduled in this simulation (Listing. 7.10). This expected behavior is caused by the lack of caching in the RREQ Permission Request mechanism, which forces the RREQ_PERMISSION_REQUEST to be constantly re-sent in absence of a route to the destination.

```
s 0.000000000 _5_ AGT  --- 0 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 0.250000000 _5_ AGT  --- 1 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 0.500000000 _5_ AGT  --- 2 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 0.750000000 _5_ AGT  --- 3 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 1.000000000 _5_ AGT  --- 4 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 1.250000000 _5_ AGT  --- 34 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 1.500000000 _5_ AGT  --- 39 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 1.750000000 _5_ AGT  --- 44 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 2.000000000 _5_ AGT  --- 47 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 2.250000000 _5_ AGT  --- 57 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 2.500000000 _5_ AGT  --- 60 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 2.750000000 _5_ AGT  --- 65 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
s 3.000000000 _5_ AGT  --- 68 ping 64 [0 0 0 0] ------- [5:0 0:0 32 0]
```

Listing 7.10: Pings scheduled to be sent by MGN 5 to MGL 0 during the first three seconds.

```
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.250 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.500 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:01.750 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:02.000 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:02.250 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:02.500 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:02.750 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:03.000 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:03.250 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=12,event_type=5}
```

Listing 7.11: RREQ_PERMISSION_REQUESTS sent from MGN 5 to its MGL during the first 3 seconds.

7.2.3 Gatewaying

The voyage of MG 7 ends at t=20 seconds when the destination is reached. Right before that, MGN 5 starts receiving MGP_HELLO beacons from MGN 4 in MG 5. The MGP log of MGN 5 shows that the node receives a total of six MGP_HELLO beacons from a node of the foreign MG during the whole simulation (Listing. 7.12), and that this results in MGN 5 applying for gateway status for the first time at t=19.875 seconds (Listing 7.13), acquiring the status of MGG at t=19.878 seconds (Listing 7.14).

```
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:20.905 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:21.908 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:22.932 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:23.942 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:24.978 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
```

Listing 7.12: MGP_HELLOs from foreign MGNs received by MGN 5

```
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_control_msg_process: MGP_CONTROL_MSG (msg->msg_type=1)
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_control_msg_process: Parsing MGP_HELLO message
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_process_hello_message: Received HELLO from [mgn_id=4,mgn_type=1,mg_id=5,mgl_addr=0.0.0.0]
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 handle_new_mg_found: Received HELLO-MGP from MGN/MGG in neighboring MG.
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_message_send: Sending MGP_MESSAGE {dest=0.0.0.6,orig=0.0.0.5,next hop=0.0.0.6,size=96,event_type=3}
```

Listing 7.13: First MGG application sent

```
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_process_message: Got MGG_APPLICATION_GRANTED from MGL
0.0.0.5: [g 7;id 5;t 1,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_process_message: Processing MGG_APPLICATION_GRANTED from MGL
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_process_message: Adding MGG to list
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_add_MGG: Added the following MGG to list:
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_debug_MGG: Exterior MGG: (Addr=4,MGN_ID=4,MGN_type=3,MG_ID=5)
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 0] 00:00:19.878 MGP_debug_MGG: Interior MGG: (Addr=5,MGN_ID=5,MGN_type=3,MG_ID=7)
```

Listing 7.14: MGN 5 becoming MGG

```
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:27.465 MGP_remove_MGG: Removing entry:
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:27.465 MGP_debug_MGG: Exterior MGG: (Addr=4,MGN_ID=4,MGN_type=3,MG_ID=5)
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 1] 00:00:27.465 MGP_debug_MGG: Interior MGG: (Addr=5,MGN_ID=5,MGN_type=3,MG_ID=7)
0.0.0.5: [g 7;id 5;t 3,1 0.0.0.6,d 0,mggs 0] 00:00:27.465 MGP_remove_MGG: Switching back to MGN status. No MGGs left.
```

Listing 7.15: MGN 5 loosing its MGG status

```
0.0.0.5: 00:00:27.465 hello_timeout: LINK/HELLO FAILURE 0.0.0.4 last HELLO: 2050
0.0.0.5: 00:00:27.465 neighbor_link_break: Link 0.0.0.4 down!
```

Listing 7.16: AODV log showing lost MGG route

```
0.0.0.5: [g 7;id 5;t 1,l 0.0.0.6,d 0,mggs 0] 00:00:19.875 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 3,l 0.0.0.6,d 0,mggs 1] 00:00:20.905 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 3,l 0.0.0.6,d 0,mggs 1] 00:00:21.908 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 3,l 0.0.0.6,d 0,mggs 1] 00:00:22.932 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 3,l 0.0.0.6,d 0,mggs 1] 00:00:23.942 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
0.0.0.5: [g 7;id 5;t 3,l 0.0.0.6,d 0,mggs 1] 00:00:24.978 MGP_process_hello_message: Sending MGG_APPLICATION for MGG'ing to MG #5
```

Listing 7.17: MGG Applications sent by MGN 5

At t=27.465, MGN 5 finally looses its MGG status (Listing 7.15). An examination of the AODV-UU log (Listing 7.16) from the same instance in time, shows that this is, as expected, caused by the loss of the MGN 4 route, which is MGN 5's exterior MGG.

An unexpected behavior is observed in the MGP logs of MGN 5 (Listing 7.17). During the time between t=19.878 and t=27.465, that is, between the points in time when MGN 5 first becomes MGG and MGN 5 terminates its MGG state, a total of 6 MGG_APPLICATIONs are sent, 5 of which are sent during a period of time when MGN 5 reports that it already has MGG status. The anticipated behavior of MGN 5 in this situation is that it does not initiate MGG_APPLICATIONs for MG's it has already gained MGG status for. The explanation is found in a faulty implementation of the default policy for sending MGG_APPLICATIONs, executed whenever a MGP_HELLO message is heard containing an MG ID other than the MGN's own. The if/else-statement on line (1) and (6) in listing 7.18 correctly distinguishes between two cases: (1) There is no current MGG_APPLICATION conversation, in which case we can initiate a new one (3-4) and remember this conversation (5) or: (6) There is already a current active MGG_APPLICATION conversation going on, in which case no new MGG_APPLICATION is to be initiated. However, the obvious exception to (1), which is the case of MGN 5 already being MGG for the particular MG, in which case no MGG_APPLICATION should be sent is not covered in the implementation of this default policy, and is thus the reason of the unexpected behavior. The flaw is easily fixed by introducing an extra check between (1) and (2) to verify if the current MGG_APPLICATION is not in regard to an exterior MGG with which this node already has a gateway relationship.

```
1 if (MGP_timerlist_find(mgn_addr.s_addr,MGG_APPLICATION)==NULL)
   {
2       DEBUG_MGP(LOG_DEBUG,0,"Sending MGG_APPLICATION for MGG'ing to MG #%d",
                msg->mgn_hostinfo.mg_id);
3       mgg_application = MGP_message_create(mgp_mgl_addr,
                                             sizeof(mgg_list_t),
                                             MGG_APPLICATION,
                                             (char *)&mgg_attachment);

         //send the MGG application once:
4        MGP_message_send(mgg_application);

         // schedule 0 more re-sends (message 1/1 sent), but insert MGG_APPLICATION into list
         // of active MGP_MESSAGE conversations:
5        MGP_timerlist_insert(mgg_application,1,1,500,MGG_APPLICATION,mgn_addr.s_addr);
   }

6 else
   {
7       DEBUG_MGP(LOG_DEBUG,0,"Already applied for MGG'ing to MG #%d. Not sending MGG_APPLICATION",
                                msg->mgn_hostinfo.mg_id);
   }
```

Listing 7.18: Default policy for sending the MGG_APPLICATION, line numbers added.

7.2.4 Ping traffic

Another unanticipated behavior is shown in listing 7.19, which displays a large number of ping packets from MGN 5 of MG 7 reaching MGL 0 of MG 5. The time segment during which these packets are received match the time during which MG 7 is in reach of MG 5 almost exactly. Listing 7.20 shows three snapshots of the MGN 5 routing table taken at t=18.5, t=19.0 and t=19.5 seconds. We see that a route towards 0.0.0.0 (MGL 0) is formed at MGN 5 somewhere between t=19.0 seconds and t=19.5 seconds which coincides with the time segment of listing 7.19. Additionally, the routing table log shows that routes are formed to MGN 4 and MGN 3. MGN 4 is MGN 5's exterior MGG in MG 7, so this route is necessary, but at t=19.0 seconds, a route to MGN 3 is also formed, one that is not to be allowed by our default MGP policies. This violates the default MGP policy ensuring that routes may not be built between nodes in different MGs without an explicit policy allowing this.

```
r 19.204921639 _0_ MAC  --- 376 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.211169257 _0_ MAC  --- 386 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.217698674 _0_ MAC  --- 391 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.224333206 _0_ MAC  --- 395 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.229021552 _0_ MAC  --- 400 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.234042873 _0_ MAC  --- 409 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.258390364 _0_ MAC  --- 426 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.504848172 _0_ MAC  --- 428 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 19.754908171 _0_ MAC  --- 430 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 20.004748171 _0_ MAC  --- 436 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 20.257858871 _0_ MAC  --- 443 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 20.504388171 _0_ MAC  --- 450 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 20.754588171 _0_ MAC  --- 452 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 21.004368171 _0_ MAC  --- 460 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 21.259949455 _0_ MAC  --- 465 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 21.504348171 _0_ MAC  --- 474 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 21.754428171 _0_ MAC  --- 476 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 22.004608171 _0_ MAC  --- 484 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 22.254548171 _0_ MAC  --- 486 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 22.505048171 _0_ MAC  --- 498 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 22.754568171 _0_ MAC  --- 500 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 23.004848171 _0_ MAC  --- 508 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 23.254808171 _0_ MAC  --- 510 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 23.504548171 _0_ MAC  --- 522 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 23.754228171 _0_ MAC  --- 524 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 24.004628171 _0_ MAC  --- 532 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 24.254388171 _0_ MAC  --- 534 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 24.504508171 _0_ MAC  --- 546 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 24.753948171 _0_ MAC  --- 548 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 25.004348171 _0_ MAC  --- 555 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 25.254188171 _0_ MAC  --- 557 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 25.504988172 _0_ MAC  --- 586 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 25.7545881733 _0_ MAC  --- 588 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 26.004728174 _0_ MAC  --- 592 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
r 26.254508176 _0_ MAC  --- 594 ping 64 [13a 0 2 800] ------- [5:0 0:0 32 0]
```

Listing 7.19: Ping packets from MGN 5 reaching MGL 0

```
# Time: 00:00:18.500 IP: 0.0.0.5, seqno: 16 entries/active: 1/1
Destination     Next hop        HC  St. Seqno Expire Flags Iface MGN type   MGN ID     MG ID      Precursors
0.0.0.6         0.0.0.6         1   VAL 1     2752         nsif  2          6          7

# Time: 00:00:19.000 IP: 0.0.0.5, seqno: 17 entries/active: 3/3
Destination     Next hop        HC  St. Seqno Expire Flags Iface MGN type   MGN ID     MG ID      Precursors
0.0.0.3         0.0.0.4         3   VAL 9     5207         nsif
0.0.0.4         0.0.0.4         1   VAL -     2847         nsif
0.0.0.6         0.0.0.6         1   VAL 1     2868         nsif  2          6          7

# Time: 00:00:19.500 IP: 0.0.0.5, seqno: 18 entries/active: 4/4
Destination     Next hop        HC  St. Seqno Expire Flags Iface MGN type   MGN ID     MG ID      Precursors
0.0.0.0         0.0.0.4         3   VAL 8     3000         nsif
0.0.0.3         0.0.0.4         3   VAL 9     4707         nsif
0.0.0.4         0.0.0.4         1   VAL -     2700         nsif
0.0.0.6         0.0.0.6         1   VAL 1     2709         nsif  2          6          7
```

Listing 7.20: MGN 5 incorrectly acquiring routing information about MG 7

Looking at the AODV log from the same instant in time (Listing 7.21), we note that, at

t=18.847 seconds, a new neighbor is discovered by AODV-UU (1), which is then inserted into the routing table at line (2) and (3). This neighbor, 0.0.0.4 (MGN 4) is a direct neighbor, which by default MGP policies is desirable as an exterior MGG. Therefore, a route to this MGN is wanted.

```
1 0.0.0.5: 00:00:18.847 neighbor_add: 0.0.0.4 new NEIGHBOR!
2 0.0.0.5: 00:00:18.847 rt_table_insert: Inserting 0.0.0.4 (bucket 4) next hop 0.0.0.4
3 0.0.0.5: 00:00:18.847 rt_table_insert: New timer for 0.0.0.4, life=3000
4 0.0.0.5: 00:00:18.847 rreq_process: ip_src=0.0.0.4 rreq_orig=0.0.0.3 rreq_dest=0.0.0.0
5 0.0.0.5: 00:00:18.847 rreq_record_insert: Buffering RREQ 0.0.0.3 rreq_id=7 time=5600
6 0.0.0.5: 00:00:18.847 rreq_process: rreqlen - extlen=24, RREQ_SIZE=24
7 0.0.0.5: 00:00:18.847 log_pkt_fields: rreq->flags:GD rreq->hopcount=2 rreq->rreq_id=7
8 0.0.0.5: 00:00:18.847 log_pkt_fields: rreq->dest_addr:0.0.0.0 rreq->dest_seqno=9
9 0.0.0.5: 00:00:18.847 log_pkt_fields: rreq->orig_addr:0.0.0.3 rreq->orig_seqno=9
10 0.0.0.5: 00:00:18.847 rreq_process: Creating REVERSE route entry, RREQ orig: 0.0.0.3
11 0.0.0.5: 00:00:18.847 rt_table_insert: Inserting 0.0.0.3 (bucket 3) next hop 0.0.0.4
12 0.0.0.5: 00:00:18.847 rt_table_insert: New timer for 0.0.0.3, life=5360
13 0.0.0.5: 00:00:18.847 rreq_forward: forwarding RREQ src=0.0.0.3, rreq_id=7
14 0.0.0.5: 00:00:18.847 aodv_socket_send: AODV msg to 255.255.255.255 ttl=32 size=24
```

Listing 7.21: MGN 5 acquiring route to MGN 3 through its neighbor (line numbers added)

The problem in this case, is that the route to MGN 4 is created during the processing of a RREQ, not a MGP_HELLO, in which case extra information in the RREQ is used by AODV-UU to create further routes, something that has not been foreseen by our MGP implementation.

By looking at rows (7-9) in listing (Listing 7.21), we see that the incoming packet is a RREQ with ID 7, sent my MGN 3 (0.0.0.3) in order to find a route to MGL 0 (0.0.0.0). The default behavior of AODV-UU is thus to create a *reverse* route to the originator of the RREQ, if the originator is previously unknown by setting the address of the neighbor through which this routing message was received as next hop. Consequently, one desired, and one undesired route is set up by hearing this RREQ.

```
0.0.0.5: 00:00:19.192 route_discovery_timeout: Seeking 0.0.0.0 ttl=8 wait=640
0.0.0.5: 00:00:19.192 rreq_create: Assembled RREQ 0.0.0.0
0.0.0.5: 00:00:19.192 log_pkt_fields: rreq->flags: rreq->hopcount=0 rreq->rreq_id=16
0.0.0.5: 00:00:19.192 log_pkt_fields: rreq->dest_addr:0.0.0.0 rreq->dest_seqno=0
0.0.0.5: 00:00:19.192 log_pkt_fields: rreq->orig_addr:0.0.0.5 rreq->orig_seqno=18
0.0.0.5: 00:00:19.192 aodv_socket_send: AODV msg to 255.255.255.255 ttl=8 size=24
0.0.0.5: 00:00:19.200 aodv_socket_process_packet: Received RREP
0.0.0.5: 00:00:19.200 rrep_process: from 0.0.0.4 about 0.0.0.5->0.0.0.0
0.0.0.5: 00:00:19.200 log_pkt_fields: rrep->flags: rrep->hcnt=2
0.0.0.5: 00:00:19.200 log_pkt_fields: rrep->dest_addr:0.0.0.0 rrep->dest_seqno=8
0.0.0.5: 00:00:19.200 log_pkt_fields: rrep->orig_addr:0.0.0.5 rrep->lifetime=974
0.0.0.5: 00:00:19.200 rt_table_insert: Inserting 0.0.0.0 (bucket 0) next hop 0.0.0.4
```

Listing 7.22: Route to MGL 0 0.0.0.0 is created (wrongfully) in MGN 5

At t=19.192, the next RREQ is assembled, triggered by a Route Discovery time-out in AODV-UU. By that time, a neighbor route to MGN 4 already exists, and since MGN 4 has a fresh enough route to MGL 0, a RREP is immediately returned to MGN 5 which causes MGN 5 to create a route to destination MGL 0 with MGN 4 as next hop, at t=19.2 seconds. As we can see from listing 7.19, shortly afterwards, at approx. t=19.205, the first ping packet from MGN 5 is received at MGL 0. We can conclude that this routing "leak" is introduced not because of a pure bug in the implementation of a default policy of MGP, but rather because of the incomplete knowledge about the underlying routing protocol.

Solving this issue, as well as similar policy routing issues requires a deeper investigation of all possible interactions between nodes in AODV-UU. This is further discussed in chapter 10. Some enhancements to MGP, addressing these issues, are presented in chapter 8.

Chapter 8

Future Work

8.1 General MGP Improvements

As seen in the simulation chapters, some immediate improvements of the default policy mechanisms could be done in the short term. Some are pure fixes, like the problem described in section 7.2.3 where MGG_APPLICATIONS are sent, even after the node has obtained MGG status. Some improvements consist of further development of the existing policy code, to solve problems with routing data containment as seen in section 7.2.4. Other work could be focused on testing MGP in massive real-world simulations, running the protocol in the APE [4] test bed, an environment for producing repeatable simulations on real hardware, and in real topologies. Based on results from this, some further work could focus on tuning the MGP signaling, for example modifying the default number of re-sends of certain Event Messages if packet loss frequently causes disruptions in policy mechanisms. Finally, some pure experimental modifications could be interesting to investigate, like the possibility of making the membership process purely reactive. This could, for instance, be implemented as a single message combining an MGP_RREQ, a RREQ_PERMISSION_REQUEST and a RREQ all in one, making the membership process independent of beacons.

8.2 Dynamic Node Types

A single non-migrating MGL entity is a single point of failure. As mentioned in chapter 2, this is a major limitation. MGP would benefit from some work to investigate a dynamic node type property where a node could have multiple memberships, potentially having different node types in different MGs, as well as the possibility to migrate the MGL state and have multiple MGL per MG. All of these additions would enhance the usability of MGP in real world scenarios where nodes are expected to be highly mobile, and packet loss is expected to be higher, a scenario requesting a more dynamic approach to MGP roles, in order to maintain a high frequency of membership. A dynamic MGL role also demands the investigation of a voting algorithm used to pick a new MGL when the old one disappears.

8.3 MG Partitioning and MG Composition

Section 8.2 presents enhancements to MGP which generate new problems. One obvious problem is: What happens when a lost route splits an MG into two separate units, each perhaps having one MGL, which then merge when new routes are formed later. This is only one example of the many scenarios in which allowing the use of the dynamic node type opens possibilities of partitioning and composition of the domain, and where work could be done to investigate new policy-controllable mechanisms for how to handle these scenarios.

8.4 Inter-MG Routing

The next natural step towards supporting scenarios such as proposed in section 2.2.2 is a border routing protocol, providing the glue which interconnects MGs. One possible implementation would be a hierarchical MGP, where MGP could be run on more than one abstraction layer at once, having the MG implemented on the lowest layer, and then running a full MGP multi-MG domain, where each node is actually a whole MG running its own MGP. The advantage of this would be a flexible solution where each network brought into a multi-MG world could specify its own sub-MG:s with its own sets of rules. The obvious disadvantage of such an implementation is its higher complexity. A simpler mechanism would consist of a series of custom policies implemented using the PE to support inter-MG routing for MG *entry* (routing into a MG), *exit* (routing out of an MG), and *transit* (routing through an MG) routing.

8.5 User Interface

In scenarios such as described in section 2.2.2, real-world usage of MGP would need a way of actually communicating with the user to present available resources and let the user decide about policies. If targeting an implementation like MGP towards regular users of laptop computers or PDAs, the governing of resources such as a PAN or an internet connection is only usable if the user has a way of answering policy questions like ”Do you allow user X to transit through your internet connection?”. The current PE implementation is only usable to API programmers, so a investigation of the types of resources available to the PE together with an implementation of an interactive MGP user interface is a valuable step towards bringing MGP into a real-world scenario.

Chapter 9

Related Work

9.1 Ambient Networks

Our thesis was inspired by ideas present within The *Ambient Networks (AN)* [12] project. AN was a cooperation of more than 50 partners in the telecommunications industry lead by Ericsson under the 6th European Framework program. An AN can be of any size. Some are small PAN and some are large access networks. The idea is that any network, fixed or mobile, should be able to provide services such as access to media resources or internet connection to any other AN based on agreements made between the different AN operators. The AN project is not specifically related to IP routing as is our work, but the concept of policy based routing in MANETS comes from the AN project.

9.2 Zone-based Routing Protocol

Zone Routing Protocol (ZRP), is a hybrid routing protocol suitable for a wide variety of mobile ad-hoc networks, especially those with large network spans and diverse mobility patterns. Each node proactively maintains routes within a local region (referred to as the routing zone). Knowledge of the routing zone topology is leveraged by the ZRP to improve the efficiency of a reactive route query/reply mechanism. The proactive maintenance of routing zones also helps improve the quality of discovered routes, by making them more robust to changes in network topology. The ZRP can be configured for a particular network by proper selection of a single parameter, the routing zone radius. The partitioning idea behind this type of routing protocol is similar to MGP, but ZRP focuses on group or ”zone” formation from a routing efficiency point of view, as opposed to MGP where the maintenance of groups of nodes is static and concentrated on administrative policy decisions.

9.3 Clustering Protocols

Another field of research relates to different partitioning schemes for MANETS. Most notable is *Clusterhead Gateway Switch Routing protocol (CGSR)* [13][2], which employs a very similar approach to ours when splitting a network topology into parts, but not for the purpose of introducing Administrative Domains. CGSR is a distance vector-based routing protocol, partitioning the network into clusters, where each cluster has a cluster head. The head is the cluster's central point of connection to the outside world. The clustering method provides an effective way to allocate wireless channels among different clusters. Advantages of the protocol compared to other reactive routing protocols are that routing tables get smaller as does the range of broadcast floods for the purpose of route discovery, since cluster heads do all the routing and define boundaries of the broadcast domains. If completing MGP with a full implementation of inter-MG routing, concepts from the above papers should be researched. A very similar idea is

the Cluster Based Routing Protocol (CGSR) [10]. Interesting is that the author defines a node identifying string and a cluster head identifying string the same way as we define MG_ID and MGN_ID strings of the MGHI block. However, this protocol focuses solely on clustering in the routing process and thus, as with the above protocols doesn't address administrative domains as does our work.

Chapter 10

Conclusions

10.1 Implementing MGP

The three simulations presented in this thesis are only three representative samples for a large number of tests executed by us during the iterative work involved in the design of the MGP implementation. Each simulation has opened new doors into our understanding of MANET routing concepts and created new problems for MGP to solve. The current MGP implementation represents our conclusive knowledge of AODV-UU and reactive MANET routing protocols. As noted in chapter 7, MGP is not complete in the sense that it does not have fully functional routing information containment, as required in section 3.3. Additionally, the gateway mechanism has flaws in default policies and the implementation of all added signaling is preliminary, since we have not evaluated if it is feasible in larger topologies and under heavier traffic load.

Nevertheless, we have shown that, using a relatively small number of modifications to a reactive MANET routing protocol, we have been able to introduce the administrative domain into the MANET and implement a policy based routing framework for it.

The work involved in implementing MGP and describing the administrative domain has been an iterative process. During our earliest efforts, we used NAM to learn how to script NS-2 and our simulations only served the purpose of being basic tests. As we progressed in our understanding of AODV-UU, we moved from purely studying NAM animations to writing our own MGP debug output, to studying the AODV-UU logs, giving us a reverse-engineering understanding of AODV-UU, to finally, the study of the packet trace data cross-referencing it with AODV-UU and MGP logs, which gave us detailed information about problems in our MGP implementation. This way of working ultimately proved to be superior to step debugging a routing protocol, since routing protocol state is distributed in the network topology.

10.2 The Membership Mechanism

All our simulations are conclusive in that we see membership state spread almost instantly after the first MGP_HELLO heard in the MG, which is typically at around t=1s. This is a result of an addition that we call ”early MGP_HELLO send”, added by us to solve a problem experienced in an early MGP implementation attempt. Each new member MGN would send its first MGP_HELLO at even AODV-UU beacon intervals, which meant that the membership state would take up to one second before it could propagate to neighbors. In low packet data scenarios, frequent time-out of routes lead to frequent membership termination. In these scenarios, membership would be intermittent, and membership state would oscillate with lower frequencies in MGNs with longer hop-distance to the MGL. By forcing each new member MGN to directly send a beacon on entry into the MG, the oscillation effect was almost completely removed.

10.3 Integrating MGP into AODV-UU

As noted in section 7.2.4, a complete understanding of AODV-UU, by mapping all of the possible ways a node can interact with other nodes, is necessary to prevent the type of problems we experienced with routing information leaking between MGs. This problem is present in implementations like MGP, where added functionality is implemented on the same abstraction layer as the current functionality as opposed to governing routing with some kind of overlay network solution, where the current implementation is unaware of the extensions added to it (as it is being used by- and not integrated with the extension). There seems to be a design trade-off, where integrating a mechanism such as MGP into a routing protocol needs a deeper understanding, whereas putting a layer on top of it may cause a different set of design problems, notably with the need of encapsulation of the overlay network data. A possible advantage with an overlay network solution is perhaps that control of routing may not need to be implemented in a cooperative manner. In this case, the possibility of the encryption of administrative domain signaling means that even non-altruistic nodes could participate without disrupting the domain.

Bibliography

[1] E. Belding-Royer C. Perkins and S. Das. Ad-hoc on-demand distance-vector routing protocol (aodv), July 2003. http://www.ietf.org/rfc/rfc3561.txt.

[2] Ching-Chuan Chiang, Hsiao-Kuang Wu, Winston Liu, and Mario Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *IEEE Singapore International Conference on Networks, SICON'97, April 16-17, 1997, Singapore*, pages 197–211. IEEE, IEEE, April 1997.

[3] D. Maltz D. Johnson, Y. Hu. The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4, February 2007. http://tools.ietf.org/html/rfc4728.

[4] Lundgren H. Nordstrom E. Nielsen J. et. al. The ape protocol evaluation testbed. http://apetestbed.sourceforge.net.

[5] Rekhter et al. Border gateway protocol 4 (bgp-4), 2006. http://www.ietf.org/rfc/rfc4271.txt.

[6] A. Legout et.al. The network simulator 2 (ns-2). http://nsnam.isi.edu/nsnam/.

[7] A. Kleen A. Kuznetsov J. Salim H. Khosravi. Netlink, ietf rfc 3549. http://www.ietf.org/rfc/rfc3549.txt.

[8] Jacquet JP and Clausen T. Optimized link state routing protocol (olsr). ietf rfc 3626, October 2003. http://www.ietf.org/rfc/rfc3626.txt.

[9] H. Lundgren. Implementation and experimental evaluation of wireless ad-hoc routing protocols. In *Digital Comprehensive Summaries of Uppsala Dissertations from the faculty of Science and Technology*, pages 17. xi+41pp. Uppsala University, Acta Universatis Upsaliensis., 2005.

[10] Y.C Tay Mingliang Jiang, Jinyang Li. Cluster based routing protocol, ietf draft, August 1999. http://tools.ietf.org/wg/manet/draft-ietf-manet-cbrp-spec/draft-ietfmanet-cbrp-spec-01.txt.

[11] E Nordstrom. Ad-hoc on-demand distance vector uppsala university (aodvuu) implementation. http://core.it.uu.se/AdHoc/AodvUUImpl.

[12] The Ambient Networks project. Ambient networks web page. http://www.ambient-networks.org.

[13] University of luxenbourg SECAN-lab. Clusterhead gateway switching protocol. http://wiki.uni.lu/secan-lab/Cluster-Head+Gateway+Switch+Routing+Protocol.html.

[14] Wikipedia.org. Border gateway protocol (bgp), 2006. http://en.wikipedia.org/wiki/bgp.

Chapter 11

Appendix 1: Simulation 3 Tcl Script

```
#############################################################################################################
#  Simulation 3: One moving MGG, its moving MGL, Three static MGN. One Static MGL. Ping test. MG 7 moves in and out of reach of MG 5
#############################################################################################################


set node_stepping_x 10

set num_nodes 7
set node_size 0.5

set dimension_x 60
set dimension_y 60

set group_leader_node 1

# Create simulator instance
set ns_ [new Simulator]

# Setup topography object
set t [new Topography]

# Create instance of GOD object
set god_ [create-god $num_nodes]

#file handles for trace files for ns and nam:
set nstrace  [open 3.tr w]
set namtrace [open 3.nam w]

#turn on logging:
$ns_  trace-all $nstrace
$ns_  namtrace-all-wireless $namtrace $dimension_x $dimension_y

# Set up the topology:
$t load_flatgrid [expr $dimension_x+1.0] [expr $dimension_y+1.0]

# Make queue prioritize routing packets
Queue/DropTail/PriQueue set Prefer_Routing_Protocols 1


#Set up properties of wireless interface and medium:
#(Numbers supplied in test scripts developed by Eric Nordstr?m & co at
#Uppsala University)

Phy/WirelessPhy set Pt_ 0.015        ;# Tx power (W),
#Phy/WirelessPhy set Pt_ 0.031622777        ;# Tx power (W),
;# 15 dBm
Phy/WirelessPhy set bandwidth_ 11Mb        ;# 11 Mbps bandwidth
Mac/802_11 set dataRate_ 11Mb              ;# 11 Mbps for data
Mac/802_11 set basicRate_ 1Mb              ;# 1 Mbps for broadcasts
Phy/WirelessPhy set freq_ 2.472e9          ;# Europe, Channel 13, 2.472 GHz
Phy/WirelessPhy set CPThresh_ 10.0         ;# Capture threshold (dB)
Phy/WirelessPhy set CSThresh_ 5.011872e-12 ;# Carrier sense threshold (W),
;# receiver sensitivity -83 dBm
Phy/WirelessPhy set L_ 1.0                 ;# System loss
Phy/WirelessPhy set RXThresh_ 5.82587e-09  ;# Receive threshold (W),
# from threshold utility (22.5 m)
                                           # given the other parameters

# create new channel as in ns-2.28/tcl/ex/wireless-mitf.tcl:
set chan [new Channel/WirelessChannel]

#set mobile node properties:
$ns_  node-config -adhocRouting AODVUU \
    -llType LL \
    -macType Mac/802_11 \
    -ifqType Queue/DropTail/PriQueue \
    -ifqLen 50 \
    -antType Antenna/OmniAntenna \
    -propType Propagation/TwoRayGround \
    -phyType  Phy/WirelessPhy \
    -topoInstance $t \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace ON \
```

```
    -channel $chan \
    -movementTrace ON

#create MGL:

set MGL [$ns_  node]

#create MGN1:
set MGN1 [$ns_  node]

#create MGN2:
set MGN2 [$ns_  node]

#create MGN3:
set MGN3 [$ns_  node]

#create MGN4:
set MGN4 [$ns_  node]

#create MGG70:
set MGG70 [$ns_  node]

#create MGL71:
set MGL71 [$ns_  node]


#disable node random motion:
$MGN1 random-motion 0

#disable node random motion:
$MGN2 random-motion 0

#disable node random motion:
$MGN3 random-motion 0

#disable node random motion:
$MGN4 random-motion 0

#disable node random motion:
$MGG70 random-motion 0

#disable node random motion:
$MGL71 random-motion 0



#get hold of routing agent:
set r_MGN1 [$MGN1 set ragent_]

#set routing agent properties:
$r_MGN1 set debug_ 1
$r_MGN1 set log_to_file_ 1
$r_MGN1 set rt_log_interval_ 500
$r_MGN1 set groupleader_ 0
$r_MGN1 set hello_jittering_ 1
$r_MGN1 set mg_id_ 0
$r_MGN1 set mgn_id_ 1
$r_MGN1 set llfeedback_ 0

#get hold of routing agent:
set r_MGN2 [$MGN2 set ragent_]

#set routing agent properties:
$r_MGN2 set debug_ 1
$r_MGN2 set log_to_file_ 1
$r_MGN2 set rt_log_interval_ 500
$r_MGN2 set groupleader_ 0
$r_MGN2 set hello_jittering_ 1
$r_MGN2 set mg_id_ 0
$r_MGN2 set mgn_id_ 2
$r_MGN2 set llfeedback_ 0

#get hold of routing agent:
set r_MGN3 [$MGN3 set ragent_]

#set routing agent properties:
$r_MGN3 set debug_ 1
$r_MGN3 set log_to_file_ 1
$r_MGN3 set rt_log_interval_ 500
$r_MGN3 set groupleader_ 0
$r_MGN3 set hello_jittering_ 1
$r_MGN3 set mg_id_ 0
$r_MGN3 set mgn_id_ 3
$r_MGN3 set llfeedback_ 0

#get hold of routing agent:
set r_MGN4 [$MGN4 set ragent_]

#set routing agent properties:
$r_MGN4 set debug_ 1
$r_MGN4 set log_to_file_ 1
$r_MGN4 set rt_log_interval_ 500
$r_MGN4 set groupleader_ 0
$r_MGN4 set hello_jittering_ 1
$r_MGN4 set mg_id_ 0
$r_MGN4 set mgn_id_ 4
$r_MGN4 set llfeedback_ 0

#get hold of routing agent:
set r_MGG70 [$MGG70 set ragent_]

#set routing agent properties:
$r_MGG70 set debug_ 1
$r_MGG70 set log_to_file_ 1
$r_MGG70 set rt_log_interval_ 500
$r_MGG70 set groupleader_ 0
$r_MGG70 set hello_jittering_ 1
$r_MGG70 set mg_id_ 0
```

```
$r_MGG70 set mgn_id_ 5
$r_MGG70 set llfeedback_ 0

#get hold of routing agent:
set r_MGL71 [$MGL71 set ragent_]

#set routing agent properties:
$r_MGL71 set debug_ 1
$r_MGL71 set log_to_file_ 1
$r_MGL71 set rt_log_interval_ 500
$r_MGL71 set groupleader_ 1
$r_MGL71 set hello_jittering_ 1
$r_MGL71 set mg_id_ 7
$r_MGL71 set mgn_id_ 6
$r_MGL71 set llfeedback_ 0

#disable node random motion:
$MGL random-motion 0

#get hold of routing agent:
set r_mgl [$MGL set ragent_]

#set routing agent properties:
$r_mgl set debug_ 1
$r_mgl set log_to_file_ 1
$r_mgl set rt_log_interval_ 500
$r_mgl set groupleader_ 1
$r_mgl set hello_jittering_ 1
$r_mgl set mg_id_ 5
$r_mgl set mgn_id_ 50
$r_mgl set llfeedback_ 0


#set MGL transport properties:

set MGL_start_x [expr $dimension_x * 0.01]
set MGL_start_y [expr $dimension_y * 0.01]


$MGL set X_ $MGL_start_x
$MGL set Y_ $MGL_start_y
$ns_ initial_node_pos $MGL $node_size

#set MGN1 transport properties:

set MGN1_start_x [expr $dimension_x * 0.25]
set MGN1_start_y [expr $dimension_y * 0.01]


$MGN1 set X_ $MGN1_start_x
$MGN1 set Y_ $MGN1_start_y

$ns_ initial_node_pos $MGN1 $node_size

#set MGN2 transport properties:


set MGN2_start_x [expr $dimension_x * 0.01]
set MGN2_start_y [expr $dimension_y * 0.25]

$MGN2 set X_ $MGN2_start_x
$MGN2 set Y_ $MGN2_start_y

$ns_ initial_node_pos $MGN2 $node_size

#set MGN3 transport properties:


set MGN3_start_x [expr $dimension_x * 0.25]
set MGN3_start_y [expr $dimension_y * 0.25]


$MGN3 set X_ $MGN3_start_x
$MGN3 set Y_ $MGN3_start_y

$ns_ initial_node_pos $MGN3 $node_size

#set MGN4 transport properties:


set MGN4_start_x [expr $dimension_x * 0.01]
set MGN4_start_y [expr $dimension_y * 0.5]

$MGN4 set X_ $MGN4_start_x
$MGN4 set Y_ $MGN4_start_y

$ns_ initial_node_pos $MGN4 $node_size

##############################################################################################
#set MGG70 transport properties:

set MGG70_start_x [expr $dimension_x*0.99]
set MGG70_start_y [expr $dimension_y * 0.75]
set MGG70_stop_x [expr $dimension_x*0.01]
set MGG70_stop_y $MGG70_start_y
set MGG70_speed_x [expr [expr $MGG70_stop_x - $MGG70_start_x] / -40]
set MGG70_speed_y 0

$MGG70 set X_ $MGG70_start_x
$MGG70 set Y_ $MGG70_start_y

$ns_ initial_node_pos $MGG70 $node_size

#set MGL71 transport properties:

set MGL71_start_x [expr $dimension_x*0.99]
set MGL71_start_y [expr $dimension_y * 0.99]
set MGL71_stop_x [expr $dimension_x*0.01]
```

```
set MGL71_stop_y $MGL71_start_y
set MGL71_speed_x [expr [expr $MGL71_stop_x - $MGL71_start_x] / -40]
set MGL71_speed_y 0

$MGL71 set X_ $MGL71_start_x
$MGL71 set Y_ $MGL71_start_y

$ns_ initial_node_pos $MGL71 $node_size
##########################################################################################

#define stop procedure:
proc stop {} {
    global ns_
    $ns_ flush-trace
}

#ping

# Ping agent recv instance procedure, required for Ping agent to work
Agent/Ping instproc recv {from rtt} {
    $self instvar node_ fd
    set ns_ [Simulator instance]
    set now [$ns_ now]
    puts $fd "r $now PING-REP _[$node_ id]_ $from $rtt"}


# Create Ping agents at 1 and 4
set srcping [new Agent/Ping]
$srcping set packetSize_ 64
$ns_ attach-agent $MGG70 $srcping
$srcping set fd $nstrace

set destping [new Agent/Ping]
$destping set packetSize_ 64
$ns_ attach-agent $MGL $destping

# Ping client at mobile node should generate 4 ping-packets/second
set pingrate 4

# Schedule sending of Ping packets from 1 to 4
for {set i 0} {$i < [expr 40*$pingrate]} {incr i} {
$ns_ at [expr $i * 1.0/$pingrate] "$srcping send"
}

# Connect Ping agents
$ns_ connect $srcping $destping

#schedule simulation events:
$ns_ at 0.0 "$MGL setdest $MGL_start_x $MGL_start_y 0"
$ns_ at 0.0 "$MGN1 setdest $MGN1_start_x $MGN1_start_y 0"
$ns_ at 0.0 "$MGN2 setdest $MGN2_start_x $MGN2_start_y 0"
$ns_ at 0.0 "$MGN3 setdest $MGN3_start_x $MGN3_start_y 0"
$ns_ at 0.0 "$MGN4 setdest $MGN4_start_x $MGN4_start_y 0"
$ns_ at 0.0 "$MGG70 setdest $MGG70_stop_x $MGG70_stop_y $MGG70_speed_x"
$ns_ at 0.0 "$MGL71 setdest $MGL71_stop_x $MGL71_stop_y $MGL71_speed_x"

$ns_ at 40.0 "$MGL setdest $MGL_start_x $MGL_start_y 0"
$ns_ at 40.0 "$MGN1 setdest $MGN1_start_x $MGN1_start_y 0"
$ns_ at 40.0 "$MGN2 setdest $MGN2_start_x $MGN2_start_y 0"
$ns_ at 40.0 "$MGN3 setdest $MGN3_start_x $MGN3_start_y 0"
$ns_ at 40.0 "$MGN4 setdest $MGN4_start_x $MGN4_start_y 0"
$ns_ at 40.0 "$MGG70 setdest $MGG70_stop_x $MGG70_start_y 0"
$ns_ at 40.0 "$MGL71 setdest $MGL71_stop_x $MGL71_start_y 0"


$ns_ at 40.01 "$MGL reset"
$ns_ at 40.01 "$MGN1 reset"
$ns_ at 40.01 "$MGN2 reset"
$ns_ at 40.01 "$MGN3 reset"
$ns_ at 40.01 "$MGN4 reset"
$ns_ at 40.01 "$MGG70 reset"
$ns_ at 40.01 "$MGL71 reset"

$ns_ at  40.02 "stop"
$ns_ at  40.03 "$ns_ halt"


#finally, after all events are set, let's start our simulation!
puts "Starting Simulation!"
$ns_ run
```

Listing 11.23: