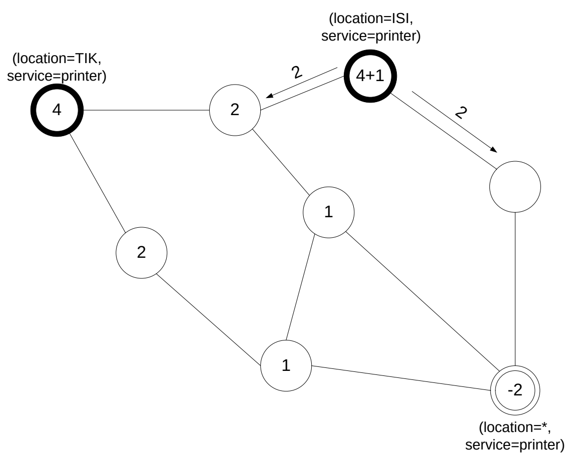
Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Departement Elektrotechnik
Professur für Technische Informatik
Professor Dr. Bernhard Plattner

Lorenz Bührer, D-ITET/DS

Simulation of a Data Centric Routing Protocol for Mobile Ad Hoc Networks

Diploma Thesis DA-2003.07
Winter 2002/2003



Advisor:
Vincent Lenders

Supervisor:
Prof. Dr. B. Plattner

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Advisor
Vincent Lenders, lenders@tik.ee.ethz.ch

Supervisor
Prof. Dr. Bernhard Plattner, plattner@tik.ee.ethz.ch

Student
Lorenz Bührer, lbuehrer@ee.ethz.ch

Acknowledgments

This documentation was written at the *Computer Engineering and Networks Laboratory (TIK)* [1] of the *Swiss Federal Institute of Technology (ETH) Zurich* [2] during the winter semester 2002/2003.

The author would like to thank his tutors Vincent Lenders, PhD students at TIK, for his dedicated and competent help in various aspects. Furthermore, I would like to thank Prof. Dr. Plattner for his assistant inputs.

I could benefit from a excellent infrastructure provided by TIK. As a consequence, it was possible to concentrate from the beginning on the important tasks.

Zurich, 4th March 2003

Lorenz Bührer

Abstract

The goal of this diploma thesis was to prove the concept of Magnetic Routing. Therefore, the following steps had to be accomplished.

- Get familiar with mobile ad hoc routing and the concept of data-centric routing.
- Learn how to use the network simulator *ns-2* developed at the University of Southern California (USC) by the Information Science Institute (ISI).
- Implement the Magnetic Routing algorithm in the network simulator.
- Evaluate the implemented routing algorithm with ns-2 for different application scenarios.

Several different algorithms such as DSDV, AODV, DSR, ZRP and Direct Diffusion were analyzed. Most of these protocols are already implemented in ns-2 and were used as references for the implementation of Magnetic Routing.

A printing service was selected to evaluate the protocol. Both, Magnetic Routing and AODV were simulated and the results were compared.

To sum up, the concept of Magnetic Routing works even though the performance of the current implementation is not overwhelming. Further development should increase the performance to an adequate standard that is needed for secure and reliable data transmission in mobile ad hoc networks.

Kurzfassung

Das Ziel dieser Diplomarbeit war es, das Konzept von Magnetic Routing zu überprüfen. Dafür mussten die folgenden Schritte erledigt werden.

- Einarbeiten in Mobile Ad Hoc Routing und das Konzept von datacentric Routing.
- Erlernen der Bedienung des Netzwerk simulator *ns-2*, entwickelt an der University of Southern California (USC) vom Information Science Institute (ISI).
- Implementation von Magnetic Routing im Netzwerk Simulator.
- Evaluation des Algorithmus mit ns-2 für verschiedenen Applikationsszenarien.

Mehrere verschiedene Algorithmen wie DSDV, AODV, DSR, ZRP und Direct Diffusion wurden analysiert. Die meisten dieser Protokolle sind bereits implementiert in ns-2 und wurden als Referenzen für die Implementierung von Magnetic Routing genutzt.

Ein Printer Service wurde ausgewählt zur Evaluation des Protokolls. Sowohl Magnetic Routing, als auch AODV wurden simuliert und die Resultate verglichen.

Zusammenfassend kann gesagt werden, dass das Konzept von Magnetic Routing funktioniert, auch wenn die Performance der aktuellen Implementation nicht überwältigend ist. Weiterentwicklungen sollten die Performance auf einen adäquaten Standard erhöhen, der benötigt wird um sichere und zuverlässige Datenübertragung gewährleisten zu können.

Timeschedule

	Activity	Meetings	Milestones
Week 1	installing software	administration	software installed
Week 2	searching for papers about manet routing and ns-2		
Week 3	searching for papers about manet routing and ns-2		tutorials finished, report started
Week 4	looking for related stuff		search complete
Week 5	analysis of related work		analysis complete
Week 6	feasibility study of different approaches		
Week 7	choosing appropriate approach		first version report
Week 8	designing		design complete
Week 9	implementation, configuration	Prof. Dr. Plattner	
Week 10	implementation, configuration		
Week 11	implementation, configuration		
Week 12	implementation, configuration		
Week 13	implementation, configuration		
Week 14	evaluation		
Week 15	evaluation		evaluation finished
Week 16	evaluation	presentation	
Week 17	working on report		
Week 18	working on report		terminating all

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past, mobile computing has become more and more popular. Charles E. Perkins, an expert in ad hoc networking at Nokia Research Center and editor of the book *Ad Hoc Networking* [3] postulated in 2001 that by the year 2002, more than a billion wireless communication devices will be in use. Now, in the year 2003 even more mobile devices are in use.

In areas where there is little or no communication infrastructure at all, mobile users may still want to be able to communicate. This can be done with the formation of an ad hoc network, where mobile nodes can not just act as host but also as routers.

Conventional routing protocols with a network stack that is divided into several layers providing discovery mechanism at each layer, are not suitable for such type of networks. Due to the dynamic nature of such networks and the limited resources that are provided by mobile devices that participate in such a type of network it is necessary that new routing protocols are developed.

Therefore a new class of routing algorithms was developed, the *ad hoc routing algorithms*. The characteristics of this class of routing algorithms are:

self-organisation In areas where no central server is available devices may still want to communicate. Therefore, the network has to be configured autonomously.

resource saving The performance of mobile devices as well as the battery power is limited. It is necessary that the protocol considers this fact. Therefore, the protocol has to be simple and energy conserving.

scalability The routing protocol has to provide secure routing even for large network with high dynamics.

Why Magnetic Routing ? Current ad hoc routing protocols do not scale for large networks with high dynamics. The new concept that is inspired from the electromagnetic field theory solves this problem by aggregating state information. This aggregation minimises routing overhead and therefore ensures scalability even for large networks with high dynamics.

A description of different routing concepts in ad hoc routing (proactive vs. reactive) can be found in Chapter 2. Moreover, the ad hoc routing protocols *DSDV* (see 2.2), *AODV* (see 2.3), *DSR* (see 2.4), *ZRP* (see 2.5) and *Direct Diffusion* (see 2.6) are described briefly.

The *Magnetic Routing* concept is introduced in chapter 3. The key features *Route Setup* and *Data Dissemination* are explained in detail.

The implementation of the AODV protocol was used as a basis for the implementation of the Magnetic Routing protocol. Both protocols are simulated with the network simulator *ns-2* (see section 4.1) with the same simulation setup (see Chapter 4). Finally, chapter 5 summarises the results and gives a outlook on further work.

Chapter 2

Ad Hoc Routing Protocols

Routing protocols that are used in ad hoc networks can be divided into three categories based on when and how the routes are discovered. One category of protocols, the *proactive* or *table driven* protocols, collect routing information without an actual need for an arbitrary route. This information is stored in route tables and has to be maintained. On the other hand, *reactive* or *on-demand* routing protocols build up a route to a destination node when a source demands for it. The third category, the *hybrid* routing protocols are a combination of proactive and reactive routing protocols. Some information about the network is proactively stored at each node, whereas other information is gathered on-demand.

An overview of these three concepts with advantages and disadvantages can be found in the following section. Furthermore, the protocols *DSDV*, *AODV*, *DSR*, *ZRP* and *Direct Diffusion* are presented briefly.

2.1 Routing Concepts

2.1.1 Proactive or Table Driven Protocols

Each node maintains one or more tables containing routing information to every other node in the network. This information about the network is gathered proactively. In other words, each node attempts to update his routing tables after a given time period in order to ensure consistent routes from each node to every other node in the network. There are miscellaneous strategies what is stored in these routing tables, e.g. the complete route to a desired destination or just the next hop in the route as in DSDV (see section 2.2), an example for a proactive protocol.

- Advantages:
  Communication with arbitrary destinations experience minimal delay ”from the point of view of an application”. In other words, if

a route to a desired destination exists, the application can immediately select the route from a well-maintained route table.

- Disadvantages:
  Ad hoc networks are presumed to contain numerous mobile nodes. Therefore, links are likely to be broken frequently and thus, any route that depended on this link can no longer be supported by the two nodes that had established the link. If the broken route has to be repaired even though no application is using it, the repair effort can be considered wasted. This additional control traffic that is needed to continually update stale route entries, can cause scarce bandwidth resources and limited battery power to be wasted. Moreover, control packets occupy valuable queue space at intermediate network points and cause further congestion. Since control packets are often put at the head of a queue, the likely result will be data loss at congested network points. Since data loss in networks generally can not be tolerated, retransmission is necessary which translates to even more delays and further congestion.

2.1.2 Reactive or On-Demand Protocols

Routing information is acquainted only when it is actually needed. Therefore, an application looking for a route to an arbitrary node initiates a *route discovery* procedure. In this case the network is flooded with an appropriate route discovery message. Depending on the respective protocol either the destination node or any node knowing a stable route to the destination can answer with a *route reply* message. This message is returned to the node that had initiated the route discovery. The information acquainted with a route discovery is usually cached in a route table, similar to the route tables in proactive protocols. This cache is purged when a *route error* occurs, when the route expires or when the route is not needed anymore. Examples for this routing concept are AODV (see section 2.3) and DSR (see section 2.4).

- Advantages:
  Reactive routing protocols may often use far less bandwidth for maintaining the route tables and thus, providing stable routes. Basically control traffic appears when it is needed to discover a route to an arbitrary node or when a route has to be repaired. Hence, routing overhead can be minimized and scarce bandwidth can be saved. Moreover, battery power that is a limited resource in mobile devices can be saved if network activity can be kept low or can even be prevented when it is not used.

- Disadvantages:
  Route information may not be available before a route reply is received. Thus, the delay to determine a route can be quite significant. Hence strict proactive routing protocols are not suitable for real time communication since the delay time is not adequate.

2.1.3 Hybrid Protocols

To avoid congestion and data loss as a result of the routing overhead as well as long delay times as a result of the route discovery procedure, the hybrid protocols have been developed. This type of protocol is a combination of proactive as well as reactive protocol concepts. Some information is periodically updated whereas other information is just acquainted when an application asks for it. Proactive protocols tend to distribute information about topology changes widely in the network, incurring large cost. Hybrid protocols may limit the propagation of such information on topology changes to the neighbourhood of the node, thereby limiting the cost. E.g. ZRP limits the scope of the proactive procedure to the node's local neighbourhood, the routing zone.

- Advantages:
  Routing overhead is minimized to avoid congestion and data loss and preserve scarce bandwidth. On the other hand, enough routing information about the network is gathered proactively. This ensures that the delay time can be kept in an adequate scope to suffice real time application requirements.

2.2 DSDV

The Destination-Sequenced Distance Vector (DSDV) [4] routing protocol uses routing tables listing all possible destinations within the network and the number of hops to each destination. Every node maintains such a routing table. The routing table entries in all nodes for a specific destination specify a *virtual destination-based tree* to send packets to that destination. In other words, each node maintains a routing table listing the "next hop" for all reachable destinations. To avoid routing loops DSDV each route is tagged with an even monotonically growing sequence number advertised by the destination node. A route is considered more favourable if it has a greater sequence number. If the sequence numbers of two routes are equal, the route with the lower metric, i.e. the route with less hops, is preferred. To maintain the consistency of route tables in a dynamically varying topology, each node periodically transmits updates once in every 30 - 90 seconds or when significant new information is available. More precisely, when the routing table has changed largely. As a consequence, DSDV is shown to have a very high routing overhead compared to other reactive routing protocols, such as AODV or DSR.

2.3 AODV

The Ad Hoc On-Demand Distance-Vector (AODV) [5] routing protocol is specially designed for ad hoc wireless networks. It provides quick and efficient route establishment with minimal control overhead and minimal route latency. AODV is an improvement of DSDV with the goal to minimize the number of required system-wide broadcasts. DSDV issues broadcasts to announce every change in the overall connectivity of the ad hoc network. Since AODV is a pure reactive routing protocol and thus, nodes that are not on a selected path do not maintain routing information or participate in route table exchanges, it is no longer required that topology changes initiate system-wide broadcasts.

When a node determines that it needs a route to a destination node and does not have one available, it initiates a *route discovery* procedure by broadcasting a *RREQ* packet. A node receiving such a packet may answer it either if it is the destination node or if it has a valid route to the destination, by replying a *RREP* packet. Any node forwarding a reply updates it's own route table with a pointer to the destination. As long as the route is used, it will be continuously maintained. Unused routes time out after a certain period and are consequently discarded. This reduces the effect of stale routes as well as the need for route maintenance for unused routes. AODV supports unicast as well as multicast routes that are set up in a similar manner.

2.4 DSR

The Dynamic Source Routing (DSR) [6] protocol is designed to allow nodes to dynamically discover a source route across multiple hops to any destination in the network. Comparing to DSDV and AODV where a node just knows the ”next hop” to a destination, a sender of a packet in DSR determines the complete sequence of nodes through which a packet must be routed from the source to the destination node. This complete sequence is listed in the header of a data packet. Thus, intermediate hops do not need to maintain any routing information in order to route a received packet. The packets themselves contain all necessary routing information. This concept is known as source routing in contrast to hop-by-hop routing.

DSR can be divided into three fundamental mechanism which operate entirely on-demand: *routing*, *route discovery* and *route maintenance*. Routing is relatively trivial as briefly described above. A source wishing to transmit a packet to a destination but not knowing the path, performs a route discovery. Therefore, the node initiates a route discovery by flooding the network with a route request packet. Any node knowing a path to the requested destination can reply by sending a route reply packet. Whenever a node receives such a route reply packet it stores the useful routing information in a routing table. Route maintenance is the mechanism by which a sender can detect if the topology of the network has changed. If a next hop from a packet header is not in range anymore the route to the destination is no longer valid and a new route discovery has to be initiated by the source node.

DSR allows nodes to snoop packets sent and received by their neighbours. Since complete paths are listed in the header of these packets, this information can be used to update cache and thus, reduce the need for route requests. Moreover, a ring search strategy can further reduce the cost of route discovery, in other words, route request within the neighbours and if no answer is received in the entire network.

2.5 ZRP

The Zone Routing Protocol (ZRP) [7] is a hybrid protocol. It combines features from proactive as well as from reactive concepts. This combination allows efficient and fast route discovery in ad hoc networks. Just a small amount of routing information has to be maintained at each node. More precisely, each node proactively maintains just a *routing zone* around itself using a protocol similar to DSDV. Thus, the cost in wireless resource for the maintenance of proactive gathered routing information can be limited. The routing zone consists of all nodes within a certain *zone radius*, i.e. within a certain number of hops from the node. In this zone each node knows the optimal path to every other node. If a packet has to be routed, ZRP verifies if the destination node is within the zone. If so, the optimal path can be selected from the proactively acquainted routing table. Otherwise, a protocol similar to DSR is employed. Routing within the own zone is known as *Intrazone Routing* whereas routing outside of this zone is known as *Interzone Routing*. However, to reduce routing overhead from redundant route discovery broadcasts, a node transmits route query packets only to the nodes at the border of the routing zone. This feature is known as *bordercasting*. The border nodes start the whole procedure all over again. First, they verify if the destination is within their own routing zone. If the destination is not within this zone a route query is sent to the border nodes.

This strategy combined with the limited proactively gathered routing information reduces routing overhead significantly.

2.6 Direct Diffusion

Direct Diffusion [8] is a data-centric routing protocol that was developed for distributed sensor networks. This concept is significantly different from IP-style communication where nodes are identified by their end-points and inter-node communication is layered on an end-to-end delivery service provided within the network. More precisely, data generated in sensors is named and routed by attribute-value pairs towards a node that formerly propagated an *interest* for the corresponding data. Such a node that asks for data generated at sensor nodes is denoted as a *sink*. Interests for data have to be renewed periodically. Therefore, an *interest message* for each active task is disseminated to the network. A node receiving such a message caches the type of interest in the *interest cache* and forwards the message. Moreover, a pointer, known as the *gradient*, which specifies a direction and a value, is set to the node the packet was received from. Cache entries just contain information about this immediately previous hop towards the sink and not about the sink itself. This previous node appears to its neighbours as the initiator of the interest, although it might have come from a distant sink.

Interests can be aggregated, i.e. a node that receives an equal interest messages from different neighbours registers just one entry in the interest cache but one corresponding gradient for each neighbour. There are different strategies how these interests are disseminated through the network. The easiest method is flooding where an interest message is forwarded to all neighbours. More efficient but in addition more complicated is *constrained or directional flooding based on location* or *directional propagation based on previously cached data*.

Finally, the gradient field is set up ”pulling-down” data towards a sink, i.e. data packets can be routed through the network from sensors to sinks. Therefore, data matching an interest registered in the *interest cache* is disseminated to the node the related gradient is pointing to. If there is more than one gradient for a specific interest type, the data packet is forwarded to all nodes these gradients are pointing to. Effectively, there is a unicast to the relevant neighbour nodes.

2.7 Summary

From all ad hoc routing protocols described above, Direct Diffusion has most in common with the Magnetic Routing protocol that is introduced in chapter 3. Both protocols proactively set up a magnetic-like gradient field for a sink respectively for a service. Direct Diffusion establishes gradients by caching the node and setting a gradient pointer to this node where the interest message came from whereas Magnetic Routing reactively discovers the neighbour with the highest field entry and thus, sets the gradient pointer to this node.

Additionally, Magnetic Routing also sets up a field for service consumers. In contrast to the positive field of services, consumers set up a negative field. The goal is to ensure that nodes that consume a specific service are less suitable to forward data packets of the specific service type than just relaying nodes because they disseminate data for this service itself.

Chapter 3

Magnetic Routing

The question about the need for a new routing algorithm is answered in this chapter. Furthermore, the following sections give an overview of the key concept of Magnetic Routing. *Route Setup* and *Data Dissemination* are described in detail.

3.1 Motivation

Routing algorithms can more or less be differed by the total amount of data that is needed to provide stable paths and how this data is acquainted. This information is also known as *state information* and includes routes, route updates, routing tables, etc. A tradeoff has to be made between *state information* and *routing overhead*. In other words, the more control traffic the more scarce bandwidth is wasted. A parameter that affects this tradeoff is the *network dynamics*. The protocols introduced in Chapter 2 except Direct Diffusion that is a special case do not support partial state information exchange for route setup as required by networks with high dynamics. This fact is a possible answer to the question *Why Magnetic Routing ?*

Moreover, proactive protocols suffer from a routing overhead that increases dramatically for dynamic networks, whereas the delay in reactive protocols increases caused by the fact that every route has to be rediscovered as a consequence of rapid network changes. However, hybrid protocols do not scale for large networks with high dynamics.

The new concept of Magnetic Routing supports partial state information even for networks with high dynamics. Furthermore, it scales for large networks with high dynamics.

3.2 Concept

Magnetic Routing is a flexible routing method. It is often referred as data-centric, content-based or semantic routing. The concept is inspired from the electromagnetic field theory in physics. Charged particles influence the surrounding space by forming fields. An other charged particle getting in contact with this field experiences a force. Either if the polarity of the two particles is the same they repel each other or if the polarity is opposite they attract themselves. Magnetic Routing works in a similar manner. A service that is always provided by a single node injects a positive magnetic field to the network. The farther away or more precisely, the more hops away from the node, the lower the field force at that specific node. The field of two nodes offering the same service type can be aggregated. On the other hand, a service consumer referred as the complement of a service provider injects a negative field to the network. Finally, a data packet is assigned with a negative charge. Thus, data packets are repelled by service consumers and attracted by service providers. As a consequence, data packets diffuse through the network, from the source to a certain destination.

Magnetic Routing is a hybrid routing protocol that includes both proactive and reactive routing concepts. The *magnetic field* is built up proactively whereas requests for the magnetic entry of neighbour nodes are acquainted on-demand. This information about the neighbour with the highest field force is used because data packets are forwarded to this node.

Typically, three node types are distinguished in magnetic routing: *service providers*, *service consumers* and *relaying nodes*. These nodes are differentiated by a unique identifier. In addition, service descriptions and magnetic entries are assigned to nodes, which are used to route data packets through the network. Service descriptions, as the name implies, describe what type of service has initiated the magnetic field, e.g. printer, scanner, etc. Moreover, attributes specify a certain service type. Thus, a service description is a compound of the form:

```
(location=ETZ(room=G61.3),service=printer(form=A4))
```

In the example a printer is specified that is located at ETZ in room G61.3. Furthermore, it is able to print papers of the format A4.

3.3 Route Setup

A node that offers a service periodically broadcasts its service description with a positive magnetic entry. Relaying nodes that forward such packets divide the magnetic entry by two at each hop until a network specific threshold is reached. Nodes that consume a specific type of service initiate a field as well. In contrast to the positive field originated by services the field injected by consumers is negative. Positive and negative values are aggregated. In special cases, field forces compensate each other to a value of `mag = 0`.

This section outlines the main function that is involved in the procedure of *Route Setup*. Figure 3.1 illustrates the procedure in a flowchart. First of all, it is verified whether the *field update packet* was originated by the node itself. If so, the packet is dropped. Otherwise, it is checked whether the service that originated the field update packet is already registered. If not, the field table is updated with the service description and the magnetic entry. If the service is already in the table, the sequence number of the packet and of the entry in the table is compared. If the sequence number of the packet is older the packet is dropped. If it is newer the table is updated with the new magnetic entry, in other words, with the field force from the packet. If the sequence numbers are equal the field forces are compared. If the field force is equal or lower the packet is dropped. Otherwise, if the packet has a higher field force i.e. the packet passed less hops and therefore a better route the table is updated with the value. Moreover, the field force in the packet is divided by two and the packet is forwarded. The corresponding code can be found in Appendix C.

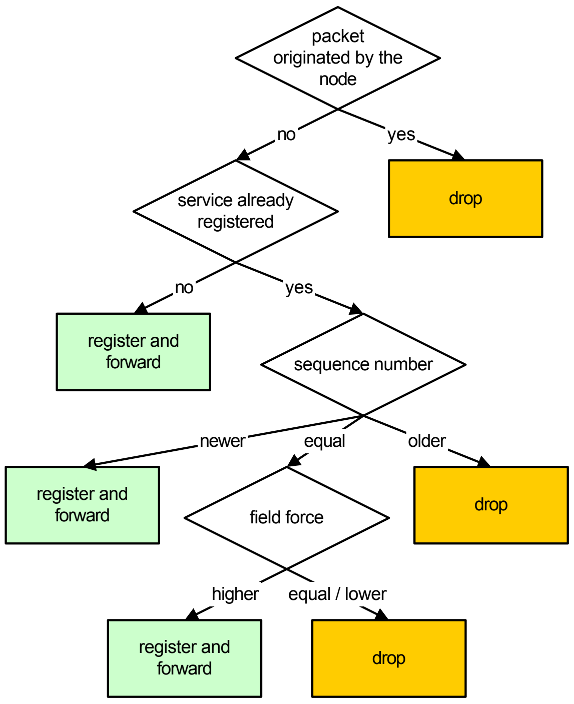


Figure 3.1: Magnetic Routing: Flow Chart of Field Update

Figure 3.2 shows an example of a *Route Setup* with eight nodes. Two of them are service providers, one is a service consumer. The residual nodes are just relaying nodes. Both nodes offer the same service type, in this case a printer. Their service description can only be differed by the location attribute (TIK, ISI). Both services start with an initial field force of `mag = 4` whereas the consumer with the initial value of `mag = -2`. These values are chosen randomly. A case for further investigation is to determine how these initial values are defined and whether they are static or dynamic.

1. The service with the location TIK divides its field force by two and broadcasts a packet with this value. In the example the value is `mag = 2`.
2. Every node that receives this packet registers the field force in a routing table. Moreover, the packet is forwarded after the value was divided by two again.
3. The printer with the attribute `location = ISI` aggregates its own force with the value received from the other service. Thus, the node has a field force `mag = 4 + 1`. The threshold in the example has a value of `mag = 1`. Therefore, packets with a value equal or below that threshold are not forwarded. Furthermore, the printer initiates its own magnetic field with a value of `mag = 2`.
4. As in step 2 nodes that receive packets register the field force in a routing table and forward the packets. Since the two provided services in this example are equal values are aggregated.
5. All nodes forward the packets with a value divided by two. If services are equal aggregation is applied.
6. The service consumer initiates its own field with a value of `mag = -1`.
7. Finally, the field is setup and data packets can be routed through the network.

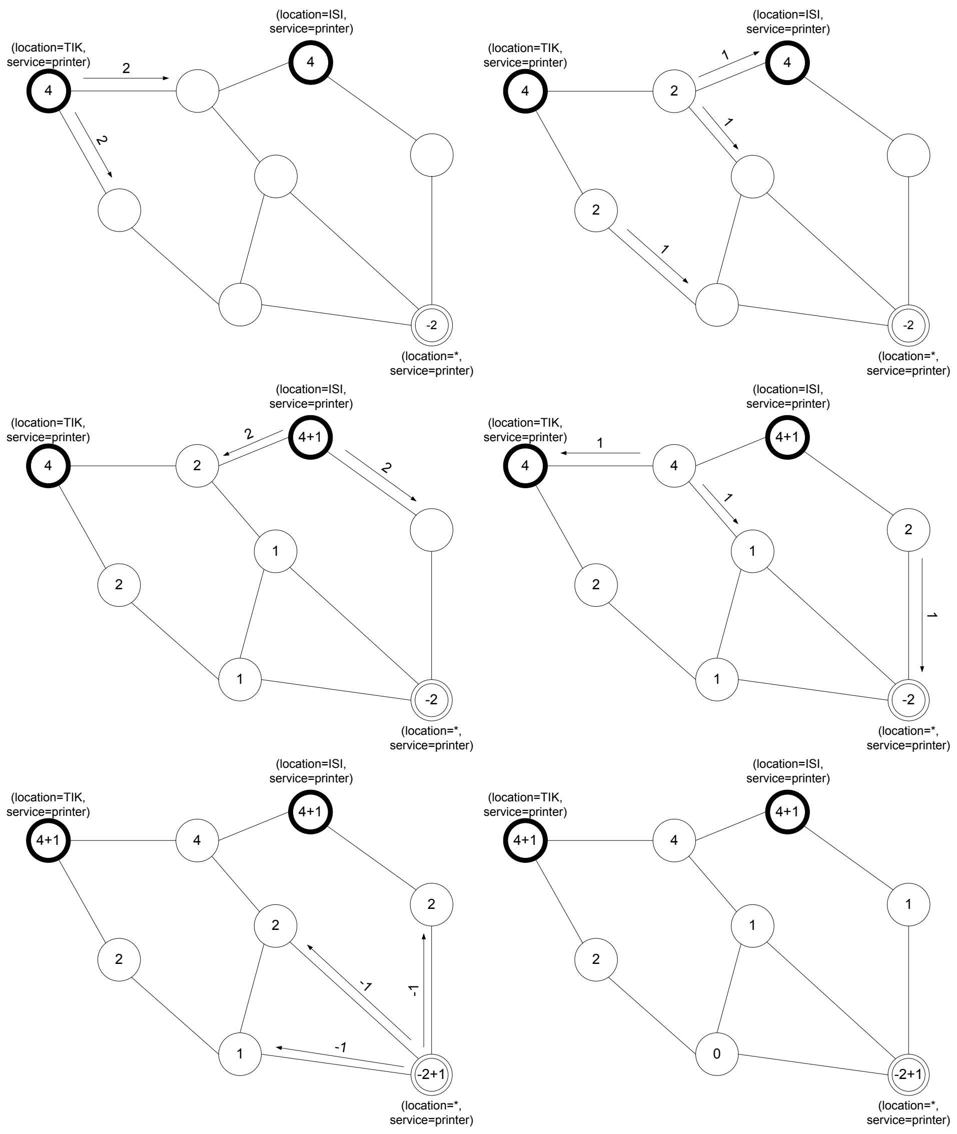

(location=TIK, service=printer)
(location=ISI, service=printer)
(location=*, service=printer)
4
4+1
-2
-2+1
2
1
0
-1

Figure 3.2: Magnetic Routing: Route Setup

3.4 Data Dissemination

Since Magnetic Routing is a data-centric routing method and data packets are identified by a service description no destination address has to be provided in the header of a data packet.

When the field is set up a service consumer can start sending data packets. Therefore, the node aggregates all magnetic entries of the desired service type. The data packet header is equipped with this value and with the service description. This information suffices to route data packets through the network.

The routing procedure in the implementation of Magnetic Routing consists of one central function that is illustrated in Figure 3.3 as a flowchart. When a node receives a data packet it verifies if it has already received the packet. In this case the packet is dropped. Otherwise, it is verified whether the node is the destination, in other words, if the node provides the service that is requested by the data packet. If so, routing was successful. Finally, if the node does not provide the service it is decided whether the data packet is forwarded. If the current node has a higher field force than the previous node the header of the packet is updated with the field force of the current node and the packet is forwarded. Otherwise, if the field force is equal or lower the packet is dropped. The corresponding code can be found in Appendix C.

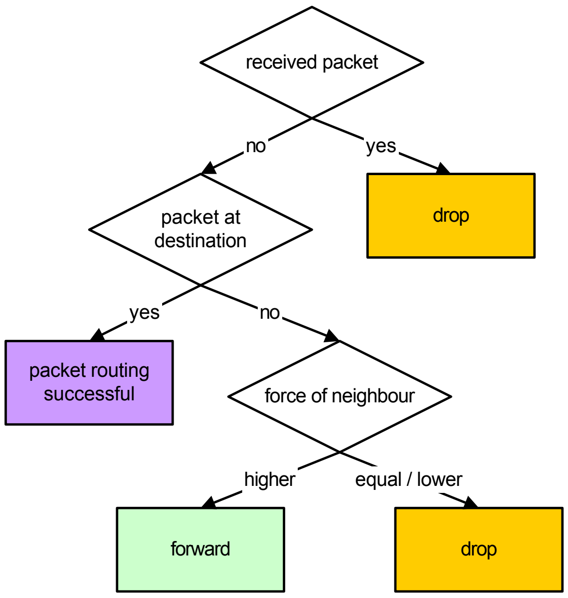


Figure 3.3: Magnetic Routing: Flow Chart of Data Dissemination

Figure 3.4 shows how *Dissemination of a Data Packet* is managed. The network with the field configuration is taken from the example above. The service consumer transmits one data packet destined for a service of the type printer at any location.

1. The service consumer assigns the data packet with a negative charge. In the example a value of `mag = -1`. Furthermore, the neighbour node with the highest field force for the specific service type is discovered. Since there are two nodes with an equal magnetic entry, one of them is selected randomly. Finally, the data packet is sent to this node. If no better node exists, the packet is dropped.

2. A relaying node applies the same strategy as the node that initiated data transmission. At first, the node has to determine which of its neighbour has the best magnetic entry for the requested service type. If two neighbours are equal one is chosen randomly. The data packet is forwarded to this node.

3. The next node repeats the previous step and chooses randomly the printer service with the attribute `location = TIK`. Finally, the packet reaches the destination and routing was successful.

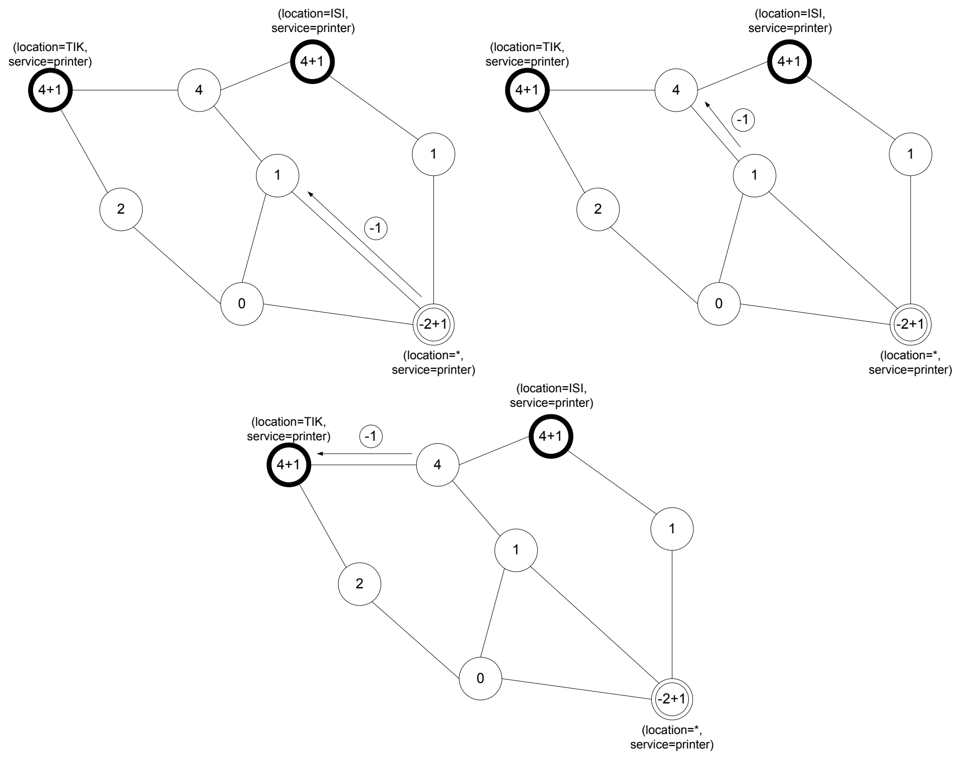

(location=ISI, service=printer)
(location=TIK, service=printer)
4+1
4
4+1
1
1
-1
2
0
-2+1
(location=*, service=printer)

Figure 3.4: Magnetic Routing: Dissemination of a Data Packet

Chapter 4

Simulation & Evaluation

This chapter discusses the simulation of the Magnetic Routing protocol. The network simulator *ns-2* (see 4.1) developed at *University of Southern California (USC)* [9] by the *Information Science Institute (ISI)* [10] was selected as the simulation environment. The following section gives an overview of the simulator. Moreover, two useful tools that are provided with the simulation package are described. Furthermore, the simulation setup and the simulated application is introduced. Finally, the simulation results and the evaluation is outlined.

4.1 ns-2

ns-2 [11] is a discrete event, object oriented network simulator. It provides a simulation environment for wired networks, e.g. *point-to-point link*, *LAN*, *unicast/multicast routing*, *transport* or *application layer*. Moreover, wireless networks can be simulated, such as *Mobile IP*, *ad hoc routing* or *satellite network*. The simulator is available for several platforms, Sun Solaris [12], Linux [13] and Windows [14] are supported.

For further information about ns consult *The ns Manual* [15]

4.1.1 Installation

ns-2 is open source and can be downloaded from the ns download page [16]. There are two different ways to install ns-2. Either download and install the packages separately or download the *all-in-one* version where all required packages come in a single file. For ”ns-newbies” it is recommended to download the *all-in-one* version. First, expand the file. There is a install script which executes all further steps, that are needed to configure and compile the source of all packages.

The *all-in-one* version 2.1b9a was used to simulate Magnetic Routing. The ns package was updated on January 16th 2003 using CVS [17].

4.1.2 C++ and OTcl

The simulator is written in *C++* [18], and uses an *OTcl* [19] interpreter as a command and configuration interface.

On one hand, detailed simulation of protocols requires a programming language which can efficiently manipulate bytes, packet headers, and implement algorithm that run over large data sets. Therefore, run-time speed is essential whereas turn-around-time (run simulation, find bug, fix bug, recompile, re-run) is less important. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation.

On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. Thus, iteration time (change the topology and re-run) is more important. In contrast, run-time of this part of the task is less important. OTcl runs slowly but can be changed easily, making it ideal for simulation configuration.

4.1.3 Scenario Files

Topologies containing a large number of nodes are usually specified in a separate file for better convenience. Therefore, ns-2 provides a scenario generator called *setdest* that can be used to create such random node-movement scenario files. The package consisting of *setdest.h*, *setdest.cc* and a *Makefile* and can be found in the ns directory under *indep-utils/cmu-scen-gen/setdest*. Originally, the tool was developed at *Carnegie Mellon University (CMU)* [20] and used a system dependent function to generate the random numbers. Further development replaced this system dependent function with a more portable random number generator (class RNG) available in ns. To use setdest, the package has to be compiled first. In order to do this follow the steps below.

1. Go to the ns directory and run *configure*. This creates the *Makefile* for the setdest package.

2. Got to indep-utils/cmu-scen-gen/setdest directory. Run *make*, which first create a stand-alone object for tools/rng.cc and then creates the executable setdest.

In order to run setdest properly, use the arguments passed to setdest as shown below.

```
./setdest [-n nodes] [-p pausetime] [-s maxspeed]
          [-t simtime][-x maxx] [-y maxy]
          > [outdir/movement-file]
```

nodes specifies the number of nodes in the network.

pausetime specifies the time that a node waits when it has finished a movement before it starts a new movement.

maxspeed is the maximum speed of node-movements, the average movement speed is the half of the maximum speed.

simtime is the duration of the simulation.

maxx and **maxy** define the dimension in which nodes are randomly placed.

outdir is the directory where the **movement-file** is written into.

Scenario files consist of the three types of lines shown below.

- define the initial position of the node:
  `$node_(0) set X_ 742.47`
  the initial x-coordinate of the node_(0) is 742.47.

- define node movements:
  `$ns_ at 24.12 "$node_(0) setdest 468.89 20.83 7.46"`
  node_(0) starts to move at the time 24.12 towards the destination 468.89 20.83 with a speed of 7.46 m/s.

- define optimal path information:
  `$ns_ at 152.77 "$god_ set-dist 0 1 2"`
  The *General Operations Director* (GOD) object is used to store global information about the network, that a ”god-like” observer of the network would have. Any participant in the simulation does not have this information. The current version of ns just uses the god object to store optimal path information. This information is not calculated during the simulation and has to be defined in the scenario file. The above line tells the simulator that at the time 152.77 the optimal path between node_(0) and node_(1) is two hops. During a node movement these distances can change significantly. For large, mobile networks optimal path calculation can become a very computation intensive task. Thus, a scenario file with 150 nodes, a pause time of 0 s, a simulation time of 900 s, a maximal speed of 20 m/s and a maximal x and y value of 1800 takes several hours to be generated on a *Sunfire 280R* [21] with 2x 900 MHz processors and 2048 MB RAM.

4.1.4 Traffic Files

Traffic files for large network should be specified in a separate file, similar to large scenario files. It is not recommended to create large traffic files by hand. Therefore, a traffic generator comparable to the scenario generator comes with the ns package. Comparing to the executable setdest, a tcl script called *cbrgen.tcl* that is located in the ns directory under *indep-utils/cmu-scen-gen* has to be executed with ns. This script generates connections with a random start time between 0 s and 180 s. The script is used as shown below.

```
ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed]
              [-mc connections] [-rate rate]
              > [outdir/traffic-file]
```

type specifies the traffic type, either *cbr* for Constant Bit Rate traffic or *tcp* for Transmission Control Protocol traffic.

nodes specifies the number of nodes in the network.

seed specifies the seed value.

connections specifies the number of connections that are generated.

rate specifies the rate of the connections.

outdir is the directory where the **traffic-file** is written into.

A traffic connection generated by the script is shown below. The *UDP Agent* is attached to the source (node_(19)) whereas the *Null Agent* is attached to the destination (node_(0)). These two agents are connected together. Moreover, the *CBR Traffic Application* is attached to the *UDP Agent*. Finally, the traffic is started at the time 102.55 s.

```
#
# 19 connecting to 0 at time 102.55
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(19) $udp_(0)

set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(0) $null_(0)

set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
```

```
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)

$ns_ connect $udp_(0) $null_(0)
$ns_ at 102.55 "$cbr_(0) start"
```

This traffic setup works fine for the simulation of the AODV protocol whereas for the simulation of Magnetic Routing protocol the *UDP Agent* and the *Null Agent* have to be replaced with a *STREAM Agent* and a *SERVICE Agent* that has to be started with two arguments. The first argument declares the type of service, the second argument the initial field force. Communication is between a service and a consumer of the same service type and not explicitly between the nodes the STREAM and SERVICE Agent are attached to. Thus, the two agents are not connected in contrast to UDP and Null Agent. In the example below, a *printer* is started at the time 0.01 s with an initial field force of 256. The argument passed to the *STREAM Agent* specifies for which type of service data is sent to. Below, data is intended for a printer.

```
#
# 19 connecting to 0 at time 102.55
#
set service_(0) [new Agent/SERVICE]
$ns_ attach-agent $node_(0) $service_(0)
$ns_ at 0.01 "$service_(0) start printer 256"

set stream_(0) [new Agent/STREAM]
$ns_ attach-agent $node_(19) $stream_(0)
$stream_(0) service printer

set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $stream_(0)

$ns_ at 102.55 "$cbr_(0) start"
```

4.2 Simulation

Two protocols are simulated. On the one hand the *Magnetic Routing* protocol to determine whether the concept does work or not, on the other hand the *AODV* routing protocol to compare the performance of the new concept of Magnetic Routing with an approved concept such as AODV.

4.2.1 Setup

Two different sized scenarios are simulated. One with 20 nodes on an quadrat of 750 m and one with 50 nodes on a square of 1000 m. The maximal speed is 20 m/s. To simulate varying mobility, scenarios with different pause times as shown in Table 4.1 are generated. A *random-waypoint* model was used as the movement model. Thereby, a node starts to move at a specific time with a specific speed towards a destination (see Section 4.1.3). Four scenarios for each parameter settings are created. Thus, the results could be averaged for a more reliable conclusion. To sum up, 48 scenarios are used. A shell script simplifies this task, it can be found in Appendix C in section C.3. Multiplied by two protocols a total of 96 simulations are made.

Speed	20	20	20	20	20	20
Pause	900	600	300	100	30	0
20 nodes 750 m x 750 m	4 runs	4 runs	4 runs	4 runs	4 runs	4 runs
50 nodes 1000 m x 1000 m	4 runs	4 runs	4 runs	4 runs	4 runs	4 runs

Table 4.1: Simulation Setup

4.2.2 Printing Service

A ”printing-like” service was chosen as network traffic. Therefore, every tenth node was assigned as a printing service, i.e. every tenth node is a destination for data. Every fifth node acts as a service consumer that transmits data. A Constant Bit Rate (CBR) stream was selected as the data stream. Table 4.2 and 4.3 illustrate how the nodes are connected in scenarios with 20 and 50 nodes. In AODV two nodes are connected explicitly, whereas in Magnetic Routing a service consumer generates data for any of the printing services. Nodes are numbered from 0 to 19 resp. 49. The first node is connected to the last two nodes etc.

Service Provider	0		1	
Service Consumer	19	18	17	16

Table 4.2: Traffic Connection 20 Nodes

Service Provider	0		1		2		3		4	
Service Consumer	49	48	47	46	45	44	43	42	41	40

Table 4.3: Traffic Connection 50 Nodes

4.3 Simulation Results

During the simulation ns-2 reports all actions in a *trace file*. Depending on the protocol the file can look slightly different. The upper example is a part of an AODV trace file, whereas the lower example is a part of a Magnetic Routing trace file.

- AODV trace file:

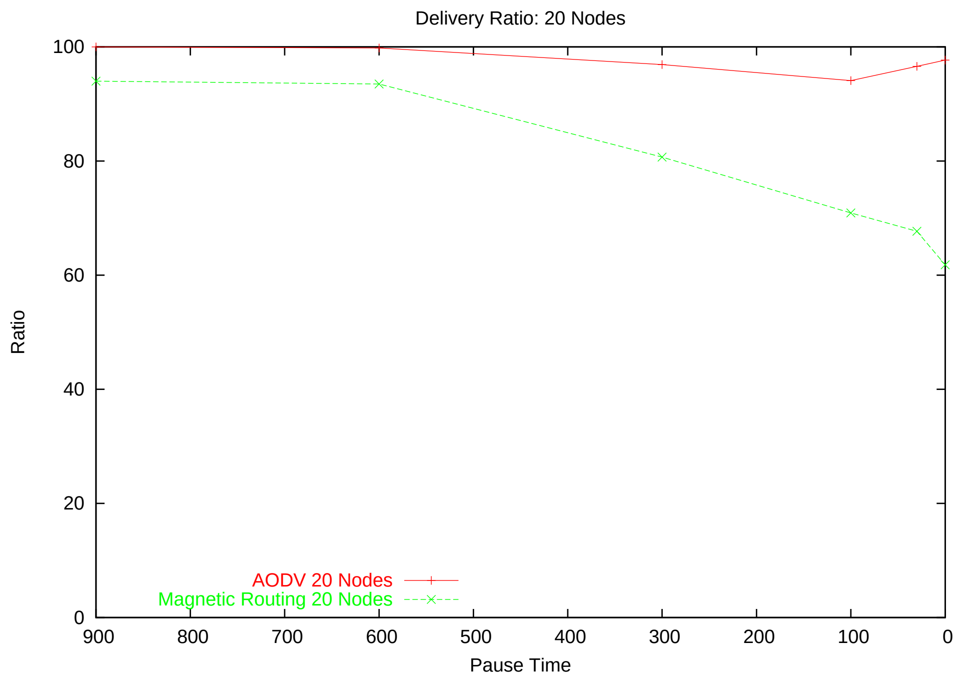
```
s 102.556838879 _19_ AGT  --- 0 cbr 512 [0 0 0 0] ------- [19:0 0:0 32 0] [0] 0 2
r 102.556838879 _19_ RTR  --- 0 cbr 512 [0 0 0 0] ------- [19:0 0:0 32 0] [0] 0 2
s 102.556838879 _19_ RTR  --- 0 AODV 48 [0 0 0 0] ------- [...] [...] (REQUEST)
r 102.557779351 _4_ RTR  --- 0 AODV 48 [0 ffffffff 13 800] ------- [...] [...] (REQUEST)
r 102.557779367 _2_ RTR  --- 0 AODV 48 [0 ffffffff 13 800] ------- [...] [...] (REQUEST)
r 102.557779612 _14_ RTR  --- 0 AODV 48 [0 ffffffff 13 800] ------- [...] [...] (REQUEST)
r 102.559801340 _0_ RTR  --- 0 AODV 48 [0 ffffffff 4 800] ------- [...] (REQUEST)
s 102.559801340 _0_ RTR  --- 0 AODV 44 [0 0 0 0] ------- [...] [...] (REPLY)
```

- Magnetic Routing trace file:

```
s 102.600386898 _19_ RTR  --- 0 MAGN 43 [0 ffffffff 2 800] ------- [19:255 2:255 1 2]
r 102.601769003 _2_ RTR  --- 0 MAGN 43 [13a 2 13 800] ------- [19:255 2:255 1 2]
r 102.603364458 _2_ RTR  --- 0 MAGN 43 [13a 2 1 800] ------- [1:255 2:255 1 2]
s 102.638762075 _2_ RTR  --- 0 MAGN 512 [13a 2 13 800] ------- [2:255 1:255 1 1] STREAM
r 102.643234119 _1_ RTR  --- 0 MAGN 512 [13a 1 2 800] ------- [2:255 1:255 1 1] STREAM
r 102.643234119 _1_ AGT  --- 0 MAGN 512 [0 0 13a 1] ------- [2:2048 2:255 1 255] STREAM
s 102.843891553 _19_ AGT  --- 1 MAGN 512 [0 0 0 0] ------- [19:255 -1:255 30 0] STREAM
r 102.843891553 _19_ RTR  --- 1 MAGN 512 [0 0 0 0] ------- [19:255 -1:255 30 0] STREAM
```

Information about *delivery ratio*, *routing overhead* and *average hops* can be extracted from the trace file. The trace file for a scenario with 50 nodes and a pausetime of 100 s is about 50 MB large and has more than 500000 lines. Therefore, it was inevitable to automatise information extraction. Five Perl [22] scripts (delivery.pl, overheadmagn.pl, overheadaodv.pl, hopsmagn.pl and hopsaodv.pl, the corresponding code can be found in Appendix C in Section C.3) accomplished the task. The scripts basically analyses each line and count the specific lines. E.g. delivery.pl, the script that analyses the delivery ratio, counts the lines where a packet is sent by an agent and received by an agent and forms the relation between the two counts.

delivery ratio, *routing overhead* and *average hops* for the scenarios with 20 respectively 50 nodes for Magnetic Routing as well as for AODV are illustrated in the Figures 4.1, 4.2 and 4.3. The current implementation of the Magnetic Routing algorithm needs further development. Hence, the results for Magnetic Routing are not overwhelming. Especially, the aggregation of field update packets that is not implemented in the current version results in explicit routing overhead that limits the performance massively. Furthermore, a *Bus Error* occurred in the simulation of Magnetic Routing. It was not found out during the thesis what the source of this error is. It is possible that the implementation is the source and thus, has a bug or that ns-2 is the source.



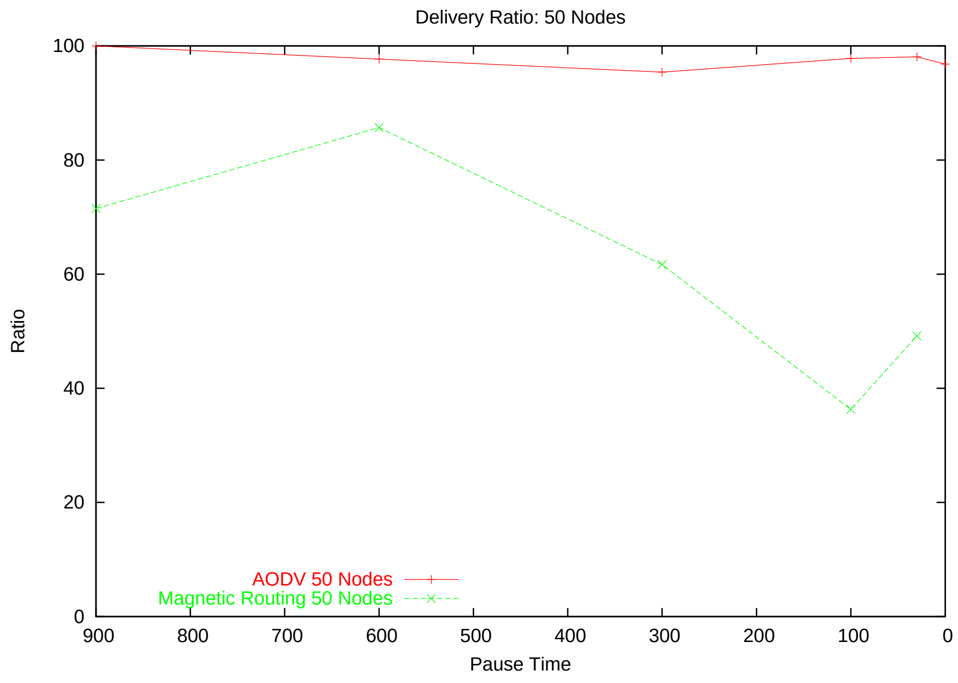


Figure 4.1: Simulation: Delivery Ratio, 20 and 50 Nodes

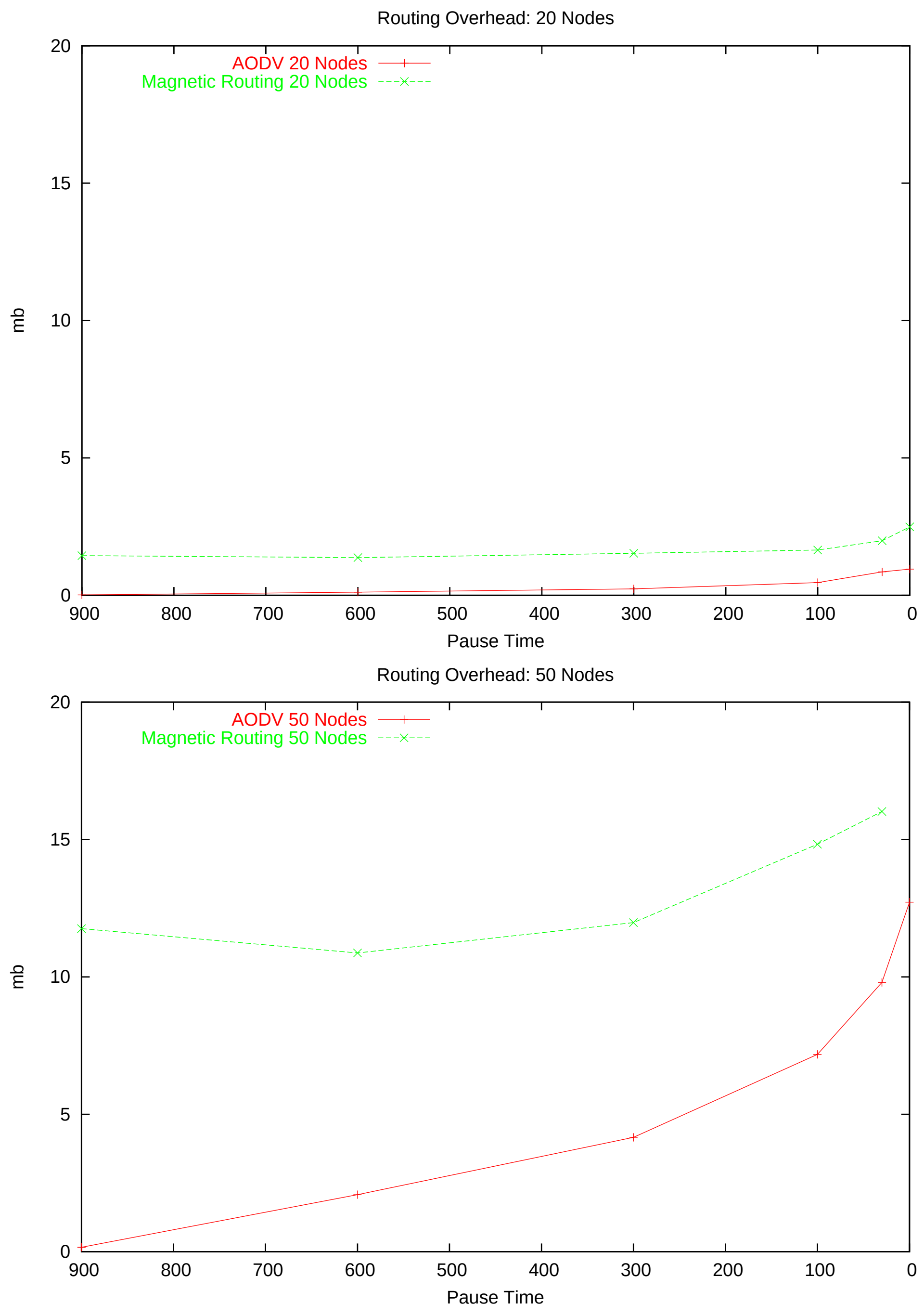
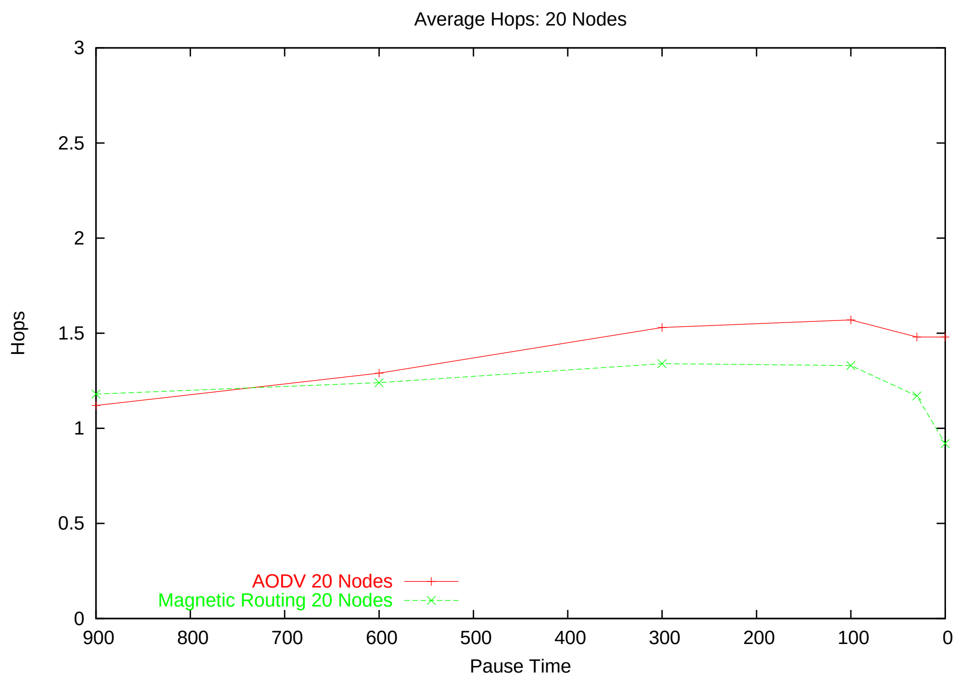


Figure 4.2: Simulation: Routing Overhead, 20 and 50 Nodes



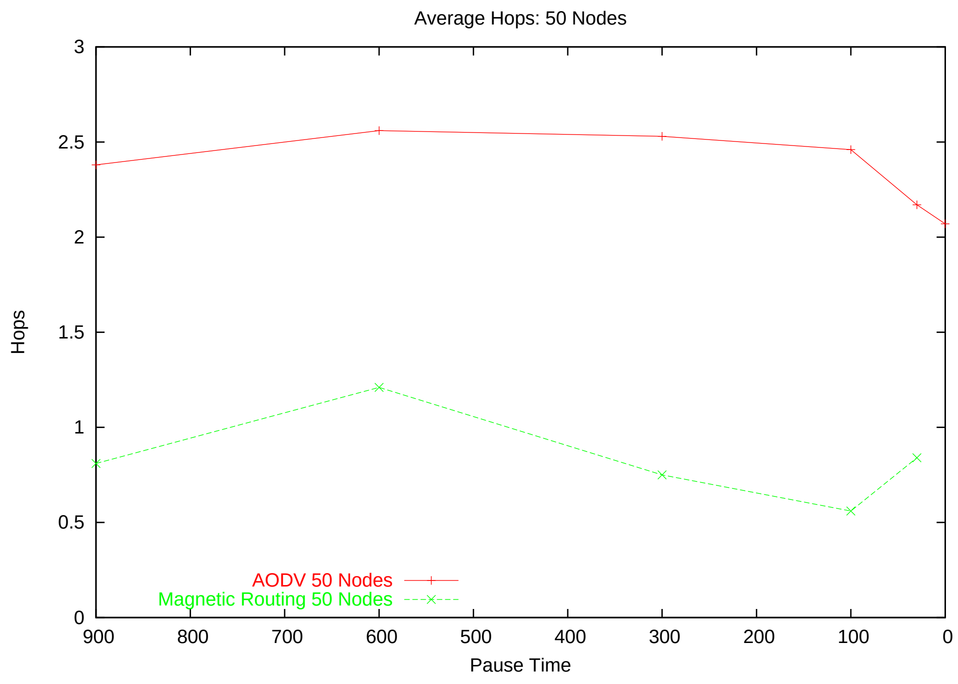


Figure 4.3: Simulation: Average Hops, 20 and 50 Nodes

4.4 Evaluation

There are a several parameter that have an influence on the performance of Magnetic Routing.

FIELD_INTERVAL defines the time interval between two route updates that are initiated by a service node. There is a tradeoff between routing overhead and the confidence on secure routes in a high dynamic network. In the implementation the value is set at 10 s.

ALLOWED_FIELD_LOSS defines the number of packets the can be lost before a route expires. This parameter was set at 1.

FIELD_EXPIRE = ALLOWED_FIELD_LOSS * FIELD_INTERVAL defines how long a magnetic entry is stored in the field table. On the one hand, if the value is too high the node can have an entry in the field table even if there is no service available. On the other hand, if the value is too small the routing overhead increases dramatically.

CACHE_INTERVAL defines the time interval between two cache purge procedures. It is checked whether an expired cache entry is still registered. Simulations with different values indicated that a value of 1 s is adequate.

ALLOWED_CACHE_LOSS defines the number packets that can be lost before the cache expires.

CACHE_EXPIRE = CACHE_INTERVAL * ALLOWED_CACHE_LOSS defines how long field forces of neighbour nodes are cached. The scenarios used for the simulation showed that a value of 1 s was optimal.

Further development of the implementation makes it inevitable to adjust the parameters described above. Especially, the aggregation of field update packets has a drastic influence to the values of the parameters.

Chapter 5

Conclusions

5.1 Results

This diploma thesis consists of four main tasks. Below, the result of every task is summarised.

- Get familiar with mobile ad hoc routing and the concept of data-centric routing:
  Several different ad hoc routing algorithm such as DSDV, AODV, DSR, ZRP, Direct Diffusion were analysed. It was found that Direct Diffusion has most in common with Magnetic Routing. Both protocols are data-centric and build up a magnetic-like gradient field. Data packets are routed with the information provided by this field.

- Learn how to use the network simulator ns-2:
  At the beginning a tutorial [23] originally developed by Marc Greis and now maintained by VINT group was worked through. Additionally, the ad hoc routing protocols presented in Chapter 2 that are implemented in ns-2 were analysed.

- Implement the proposed algorithm in ns-2:
  The implementation of AODV in ns-2 was used as a basis for the implementation of Magnetic Routing. Since the concept of these two ad hoc routing algorithms is significantly different Magnetic Routing had to be implemented almost from scratch. In Magnetic Routing data packets are routed by a service description, whereas in AODV data packets are routed by a conventional address.

- Evaluate Magnetic Routing with ns-2 for different application scenarios:
  The protocol Magnetic Routing was simulated for two different

sized scenarios 20 and 50 nodes. A printing service traffic scenario was selected. Thereby, a source node transmits data to a destination node with a constant bit rate as if a user transmits a file to a printer. The results of the simulation were compared with the results from corresponding simulations with the AODV protocol. This comparison shows that the current implementation of Magnetic Routing does work, but needs further development.

5.2 Outlook

This thesis can be used as a basis for future work. Some possible improvements are outlined in this section.

- Upgrade the implementation:
  The performance of the current implementation of Magnetic Routing is not overwhelming. Since, field update packets are not aggregated the routing overhead is significant comparing to the routing overhead in AODV. The performance of Magnetic Routing will improve if aggregation of field update packets is implemented. Furthermore, the queue management at nodes has to be improved. The current version does not support more than one service type in one queue. Further development of the implementation of Magnetic Routing involve a new evaluation for the parameters (time intervals) that influence the performance of the protocol.

- Modification to ns-2:
  The *mobilenode* class in ns-2 provides a node that is identified by an address. Since nodes in Magnetic Routing are not identified by addresses but by a service description a new class *magneticnode* is proposed where nodes do not need to have an address. This modification requires extensive changes to ns-2, since ns-2 is more or less based on the fact that nodes are identified by an address.

- Simulation environment:
  ns-2 is too slow for *large-scale* simulation. Thus, it is not suitable to evaluate the scalability of an ad hoc routing algorithm for networks with several 1000 nodes. As a consequence, a different simulation environment must be chosen. ns-2 simulates the MAC layer. This is a very computation intensive task and is not explicitly needed to analyse the scalability of a routing protocol. Therefore, it is recommended to select a simulation environment that does not simulate the MAC layer for better simulation performance in large networks.

- Bus Error:
  Under several scenarios a *Bus Error* occurred. The error occurred at a line in the traffic file similar to the one shown below.
  ```
  $stream_(3) service printer
  ```
  The error did not always appear at the same line. Some scenarios worked just fine, whereas others did not work at all. During the thesis there was not enough time to locate the source of this error. On the one hand, it is possible that the implementation of Magnetic Routing is the source and thus, has a bug. On the other hand, it is possible that ns-2 itself is the source of the error. It is even possible that there was a problem with the computer where the simulation was made. What was very confusing is the fact that a scenario resulted in an error when logged in from a Windows computer, whereas the same scenario worked just fine when logged in from a Linux computer. A possible strategy to determine the source of the error is by debugging ns-2 with a project debugger e.g. *gdb* [24]. Thus, the source of the error can be located.

Appendix A

Used Software Tools

ns-2

ns-2 2.1b9a for Solaris (see [11])
The ns package was updated on January 16th using CVS.

Application

LaTeX(see [25])
GNU Emacs 21.1.2 (see [26])
Gnuplot (see [27])
Microsoft Visio 2000 (see [28])
Adobe Acrobat 5.0 (see [29])
WinEdt 5.3 (see [30])
MiKTeX (see [31])

Appendix B

Howtos

Install ns-2

ns-2 can be downloaded from the ns download page [16]. There are two different ways to install ns-2.

- Download all packages separately. This is recommended for advanced ns-2 users who want to be up to date with a specific packages.
- The download page provides a single file that includes all required packages for ns-2. An *install script* is provided with the *all-in-one* version that can be executed after the downloaded file is expanded. This script executes all steps to install the needed packages.

Recompile ns-2

Changes that were made to a routing protocol must be compiled before an effect takes place. Therefore, the *configure* script has to be executed and results in the creation of the *Makefile*. This is normally already done when ns-2 is installed. Moreover, the Makefile has to executed with the command *make* in the ns directory.

Changes to ns-2 Files

Several files from the ns package have to be modified to add a new routing protocol to ns-2. Section C.2 in Appendix C shows the part of every file that was modified for Magnetic Routing.

Appendix C

Implemented and Modified Software

C.1 Magnetic Protocol

magnetic.h

```
/*
 * magnetic.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __magnetic_h__
#define __magnetic_h__

#include "priqueue.h"
#include "cmu-trace.h"
#include "random.h"
#include "magnetic/magnetic_ftable.h"
#include "magnetic/magnetic_packet.h"
#include "magnetic/magnetic_queue.h"

class MagneticAgent;

#define CURRENT_TIME    Scheduler::instance().clock()
#define CLIENT_FORCE -8
#define NETWORK_DIAMETER 30  // [hops]
#define FORWARD_NUMBER 1     // 0 = Broadcast, 1 = to the best node
                             // 2 = to the 2 best nodes ...

#define FIELD_INTERVAL 10      // [s]
#define ALLOWED_FIELD_LOSS 1   // [packets]
#define FIELD_EXPIRE ALLOWED_FIELD_LOSS * FIELD_INTERVAL  // [s]

#define CACHE_INTERVAL 1       // [s]
#define ALLOWED_CACHE_LOSS 1   // [packets]
#define CACHE_EXPIRE CACHE_INTERVAL * ALLOWED_CACHE_LOSS  // [s]

/*
 * Timers
 */

class TimerField : public Handler {
public:
    TimerField(MagneticAgent* a) : agent(a) {}
    void handle(Event*);
private:
    MagneticAgent *agent;
    Event intr;
};

class TimerPurge : public Handler {
public:
    TimerPurge(MagneticAgent* a) : agent(a) {}
    void handle(Event*);
private:
    MagneticAgent *agent;
    Event intr;
};

class SendTimer : public TimerHandler{
public:
    SendTimer(MagneticAgent* a) : TimerHandler(){agent = a;}
protected:
    virtual void expire(Event* e);
    MagneticAgent* agent;
};

/*
```

```
  The Routing Agent
*/

class MagneticAgent : public Agent {
    friend class TimerHello;
    friend class TimerField;
    friend class TimerPurge;
    friend class SendTimer;
public:
    MagneticAgent(nsaddr_t id);
    int command(int argc, const char*const* argv);
    void recv(Packet*, Handler*);
protected:
    nsaddr_t      index;          // Address of this node
    u_int32_t     seqno;          // Sequence Number
    bool          clientregister;
    bool          sending;
    bool          requesting;
    Trace         *logtarget;

    magnetic_ftable   ftable;         // FieldforceTable
    service_cache     schead;         // Service Cache
    service_entry     sehead;         // Service Entries
    receive_entry     rehead;         // Receive List

    // for the future: a queue to store packets if no field
    // for a service is present

    // this queue is just to store a packet dureing force request
    magnetic_queue magqueue;

    /*
     * A pointer to the network interface queue that sits
     * between the "classifier" and the "link layer"
     */
    PriQueue *priqueue;

    int initialized() { return 1 && target_; }
    void trace(char* fmt, ...);

    /*
     * Timers
     */
    TimerField ftimer;
    TimerPurge ptimer;
    SendTimer  stimer;

    /*
     * Packet Transmission Routines
     */
    void sendForceRequest(char service[]);
    void initiateField();
    void forwardPacket(Packet *p);
    void forwardMulticast();

    /*
     * Packet TX Routines
     */
    void recvMagnetic(Packet *p);
    void recvHello(Packet *p);
    void recvForceRequest(Packet *p);
    void recvField(Packet *p);
    void recvPacket(Packet *p);
    void recvService(Packet *p);

    /*
     * Service Management
     */
    bool isService(char service[]);
    bool receivedPacket(nsaddr_t src, u_int32_t src_seqno);

    /*
     * Cache Management
     */
    void updateCache(nsaddr_t id, char service[], int force);
    void deleteCache();

};

#endif /* __magnetic_h__ */
```

magnetic.cc

```
/*
 * magnetic.cc
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#include "magnetic/magnetic.h"
#include "cmu-trace.h"
#include "mac-802_11.h"

/*
 * TCL linkage
 */

int hdr_magnetic::offset_;
```

```
static class MagneticHeaderClass : public PacketHeaderClass {
public:
    MagneticHeaderClass() : PacketHeaderClass("PacketHeader/MAGN",
                          sizeof(hdr_all_magnetic)) {
        bind_offset(&hdr_magnetic::offset_);
    }
} class_magnetic_hdr;

static class MagneticClass : public TclClass {
public:
    MagneticClass() : TclClass("Agent/MAGN") {}
    TclObject* create(int argc, const char*const* argv){
        assert(argc==5);
        return (new MagneticAgent((nsaddr_t) atoi(argv[4])));
    }
}class_magnetic;


/*
 *Timers
 */

void TimerField::handle(Event*) {
    agent->initiateField();
    Scheduler::instance().schedule(this, &intr, FIELD_INTERVAL);
}

void TimerPurge::handle(Event*) {
    agent->ftable.purgeService();
    agent->deleteCache();
    Scheduler::instance().schedule(this, &intr, CACHE_INTERVAL);
}

void SendTimer::expire(Event*) {
    agent->forwardMulticast();
}

/*
 * Constructor
 */

MagneticAgent::MagneticAgent(nsaddr_t id) : Agent(PT_MAGN),
                        ftimer(this), ptimer(this),
                        stimer(this), magqueue() {
    index = id;
    seqno = 2;
    priqueue = 0;
    logtarget = 0;
    clientregister = false;
    sending = false;
    requesting = false;
    LIST_INIT(&schead);
    LIST_INIT(&sehead);
    LIST_INIT(&rehead);
}

/*
 * Packet Transmission Routines
 */

int MagneticAgent::command(int argc, const char*const* argv) {
    if(argc==2) {
        Tcl& tcl = Tcl::instance();

        if(strncasecmp(argv[1], "id", 2) == 0) {
            tcl.resultf("%d", index);
            return TCL_OK;
        }

        if (strncasecmp(argv[1], "start", 2) == 0 ) {
            // start the timers
            ftimer.handle((Event*) 0 );
            ptimer.handle((Event*) 0 );

            return TCL_OK;
        }
    }else if (argc == 3) {
        if (strcmp(argv[1], "index") == 0) {
            index = atoi(argv[2]);
            return TCL_OK;
        }else if(strcmp(argv[1], "log-target") == 0 ||
             strcmp(argv[1], "tracetarget") == 0) {
            logtarget = (Trace*) TclObject::lookup(argv[2]);
            return TCL_OK;
        }else if(strcmp(argv[1], "drop-target") == 0) {
            return Agent::command(argc, argv);
        }else if(strcmp(argv[1], "if-queue") == 0) {
            priqueue = (PriQueue*) TclObject::lookup(argv[2]);
            if(priqueue == 0)
                return TCL_ERROR;
            return TCL_OK;
        }
    }
    return Agent::command(argc, argv);
}

void MagneticAgent::initiateField(){
    // for all services in service table
    Service_Entry *se = sehead.lh_first;
```

```
        for (; se; se = se->se_link.le_next) {

        Packet *p = Packet::alloc();
        struct hdr_cmn *ch = HDR_CMN(p);
        struct hdr_ip *ih = HDR_IP(p);
        struct hdr_magnetic_field *mf = HDR_MAGNETIC_FIELD(p);

        mf->fi_type = MAGNETICTYPE_FIELD;
        strcpy(mf->fi_service, se->se_service);
        mf->fi_force = se->se_force / 2;
        mf->fi_src = index;
        mf->fi_src_seqno = seqno;

        //printf("ini:%i\n",se->se_force);

        ch->ptype() = PT_MAGN;
        ch->size() = IP_HDR_LEN + mf->size();
        ch->iface() = -2;
        ch->error() = 0;
        ch->addr_type() = NS_AF_NONE;
        ch->prev_hop_ = index;

        ih->saddr() = index;
        ih->daddr() = IP_BROADCAST;
        ih->sport() = RT_PORT;
        ih->dport() = RT_PORT;
        ih->ttl_    = NETWORK_DIAMETER;

        Scheduler::instance().schedule(target_, p, 0.0);

        seqno+=2;
    }
}

void MagneticAgent::sendForceRequest(char service[]){
    requesting = true;

    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_force *mf = HDR_MAGNETIC_FORCE(p);

    mf->fo_type = MAGNETICTYPE_FORCE;
    strcpy(mf->fo_service, service);
    mf->fo_src = index;
    mf->fo_force = 0;

    ch->ptype() = PT_MAGN;
    ch->size() = IP_HDR_LEN + mf->size();
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_NONE;
    ch->prev_hop_ = index;

    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;
    ih->ttl_    = 1;

    Scheduler::instance().schedule(target_, p, 0.0);
}

void MagneticAgent::forwardPacket(Packet *p) {
    struct hdr_magnetic_packet *mp = HDR_MAGNETIC_PACKET(p);
    if(FORWARD_NUMBER == 0){
        // broadcast
        Scheduler::instance().schedule(target_, p, 0.0);
    }else{
        // unicast/mulitcast
        Service_Cache *sc = schead.lh_first;
        if(sc != NULL && !strcmp(sc->sc_service, mp->pa_service)){
            // cache for the requested service
            magqueue.enque(p);
            sending = true;
            forwardMulticast();
        }else{
            magqueue.enque(p);
            // no cache for the requested service
            sending = true;
            if(!requesting){
                deleteCache();
                sendForceRequest(mp->pa_service);
                stimer.sched(0.04); // wait 4 [ms] to forward the packet
            }
        }
    }
}

void MagneticAgent::forwardMulticast() {
    Packet *p;
    requesting = false;

    while(p = magqueue.deque()){
        struct hdr_magnetic_packet *mp = HDR_MAGNETIC_PACKET(p);
        int actualforce = ftable.serviceForce(mp->pa_service);

        Service_Cache *sc = schead.lh_first;

        for(int i = 0; i < FORWARD_NUMBER; i++){
            // Send to n better nodes as long as the force is greater
            if(sc == NULL){
                drop(p,DROP_RTR_NO_ROUTE);
                return;
            }

            if(sc->sc_force < actualforce) return;

            Packet* pnew = p->copy();

            struct hdr_cmn *ch = HDR_CMN(pnew);
            struct hdr_ip *ih = HDR_IP(pnew);

            ch->prev_hop_ = index;
            ch->next_hop_ = sc->sc_addr;

            ih->saddr() = index;
            ih->daddr() = sc->sc_addr;
            ih->sport() = RT_PORT;
            ih->dport() = RT_PORT;
            ih->ttl_    = 1;

            Scheduler::instance().schedule(target_, pnew, 0.0);

            sc = sc->sc_link.le_next;
        }
        Packet::free(p);
    }
    sending = false;
}


/*
 * Packet Reception Routines
 */

void MagneticAgent::recv(Packet *p, Handler*) {
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);

    assert(initialized());

    // Check the TTL. If zero then discard
    if(ih->ttl_ == 0) {
        drop(p, DROP_RTR_TTL);
        return;
    }

    if(ch->ptype() == PT_MAGN) {
        ih->ttl_ -=1;
```

```
        recvMagnetic(p);
        return;
    }
}


void MagneticAgent::recvMagnetic(Packet *p) {
    struct hdr_magnetic *ma = HDR_MAGNETIC(p);
    struct hdr_ip *ip = HDR_IP(p);

    /*
      Incoming packets
    */
    switch(ma->ma_type) {

    case MAGNETICTYPE_FORCE:
        recvForceRequest(p);
        break;
    case MAGNETICTYPE_FIELD:
        recvField(p);
        break;
    case MAGNETICTYPE_PACKET:
        recvPacket(p);
        break;
    case MAGNETICTYPE_SERVICE:
        recvService(p);
        break;
    default:
        fprintf(stderr, "Invalid Magnetic type (%x)\n", ma->ma_type);
        exit(1);
    }
}

void MagneticAgent::recvForceRequest(Packet *p) {
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_force *mf = HDR_MAGNETIC_FORCE(p);

    if(mf->fo_force == 0){

        mf->fo_dest = index;
        mf->fo_force = ftable.serviceForce(mf->fo_service);

        ch->prev_hop_ = index;
        ch->direction() = hdr_cmn::DOWN;
        ch->next_hop_ = mf->fo_src;

        ih->saddr() = index;
        ih->daddr() = mf->fo_src;
        ih->sport() = RT_PORT;
        ih->dport() = RT_PORT;
        ih->ttl_    = 1;

        Scheduler::instance().schedule(target_, p, 0.0);
    }else{
        // cache the service, the best at the head of the list
        updateCache(mf->fo_dest, mf->fo_service, mf->fo_force);
    }
}

void MagneticAgent::recvField(Packet *p) {
    // forward "good" field packets, drop "bad" ones
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_field *mf = HDR_MAGNETIC_FIELD(p);

    if(mf->fi_src != index && mf->fi_force != 0){
        if(ftable.updateService(mf->fi_src, mf->fi_src_seqno,
                    mf->fi_service, mf->fi_force)){
            // forward if fieldtable was updated

            ch->direction() = hdr_cmn::DOWN;
            ch->prev_hop_ = index;
            mf->fi_force = mf->fi_force / 2;
            Scheduler::instance().schedule(target_, p, 0.0);
        }else{
            Packet::free(p);
        }
    }
}

void MagneticAgent::recvPacket(Packet *p) {
    // forward "good" packets, drop "bad" ones
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_packet *mp = HDR_MAGNETIC_PACKET(p);
    struct hdr_mac802_11 *mh = HDR_MAC802_11(p);

    if(receivedPacket(mp->pa_src, mp->pa_src_seqno)){
        // already received this packet
        drop(p,"AREC");      }else{
        if(mp->pa_src == index && mp->pa_force == 0){
            // packet is from the stream agent
            // add the negative field force for a client
            if(!clientregister){
                // register a client only once
                ftable.registerService(index, mp->pa_service, CLIENT_FORCE);
                Service_Entry *senew =
                    new Service_Entry(mp->pa_service, CLIENT_FORCE);
                LIST_INSERT_HEAD(&sehead, senew, se_link);
                clientregister = true;
            }
            int force = ftable.serviceForce(mp->pa_service);
            // forward packet
            mp->pa_force = force;
            forwardPacket(p);

        }else if(isService(mp->pa_service)){
            // this node is the service

            int src = Address::instance().get_nodeaddr(ih->saddr());
            int dst = Address::instance().get_nodeaddr(ih->daddr());
            trace("r %.9f _%i_ AGT  --- %i MAGN %i [%x %x %x %x]
                  ------- [%d:%d %d:%d %d %d] STREAM",
                  CURRENT_TIME,
                  index,
                  mp->pa_src_seqno,
                  ch->size(),
                  ih->flowid(),
                  ch->uid(),
                  mh->dh_duration,
                  ETHER_ADDR(mh->dh_da),
                  ETHER_ADDR(mh->dh_sa),
                  GET_ETHER_TYPE(mh->dh_body),
                  src, ih->sport(),
                  dst, ih->dport(),
                  ih->ttl_, (ch->next_hop_ < 0) ? 0 : ch->next_hop_);
        }else{
            // calculate if the current node is better
            if(ftable.serviceForce(mp->pa_service) > mp->pa_force){
                // better node -> forward
                mp->pa_force = ftable.serviceForce(mp->pa_service);
                ch->direction() = hdr_cmn::DOWN;
                forwardPacket(p);
            }else if(ftable.serviceForce(mp->pa_service) == mp->pa_force){
                // euqal node -> there must be a better node
                Packet::free(p);
            }else {
                // worse node -> drop
                Packet::free(p);
            }
        }
    }
}

void MagneticAgent::recvService(Packet *p) {
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_service *ms = HDR_MAGNETIC_SERVICE(p);
    ftable.registerService(index, ms->se_service, ms->se_force);
    Service_Entry *senew = new Service_Entry(ms->se_service, ms->se_force);
    LIST_INSERT_HEAD(&sehead, senew, se_link);
    Packet::free(p);
}

/*
 * Service Management
 */

bool MagneticAgent::isService(char service[]) {
    Service_Entry *se = sehead.lh_first;
    for (; se; se = se->se_link.le_next) {
        if(!strcmp(se->se_service, service) && se->se_force > 0){
            return true;
        }
    }
    return false;
}

bool MagneticAgent::receivedPacket(nsaddr_t src, u_int32_t src_seqno){
    // return true if packet was received already
    Receive_Entry *re = rehead.lh_first;
    for (; re; re = re->re_link.le_next) {
        if(re->re_src == src && re->re_src_seqno == src_seqno){
            return true;
        }else if(re->re_src == src && re->re_src_seqno < src_seqno){
            re->re_src_seqno = src_seqno;
            return false;
        }
    }
    Receive_Entry *renew = new Receive_Entry(src, src_seqno);
    LIST_INSERT_HEAD(&rehead, renew, re_link);      return false;
}

void MagneticAgent::updateCache(nsaddr_t id, char service[], int force) {
    // service cache is a sorted list of service forces
    Service_Cache *sc = schead.lh_first;
    Service_Cache *scn;

    if(sc == NULL || force >= sc->sc_force){
        Service_Cache *scnew = new Service_Cache;
        scnew->sc_addr = id;
        scnew->sc_force = force;
        strcpy(scnew->sc_service, service);
        scnew->sc_expire = CURRENT_TIME + FIELD_EXPIRE;
        LIST_INSERT_HEAD(&schead, scnew, sc_link);
        return;
    }else{
        for (; sc; sc = sc->sc_link.le_next){
            scn = sc->sc_link.le_next;
            if(scn == NULL || force >= scn->sc_force){
                Service_Cache *scnew = new Service_Cache;
                scnew->sc_addr = id;
                scnew->sc_force = force;
                strcpy(scnew->sc_service, service);
                scnew->sc_expire = CURRENT_TIME + CACHE_EXPIRE;

                sc->sc_link.le_next = scnew;
                scnew->sc_link.le_next = scn;

                return;
            }
        }
    }
}

void MagneticAgent::deleteCache(){
    // delete the cache
    if(!sending){
        double now = CURRENT_TIME;
        Service_Cache *sc = schead.lh_first;
        if(sc != NULL){
            if(sc->sc_expire < now){
                LIST_INIT(&schead);
            }
        }
    }
}

/*
 * Trace
 */

void MagneticAgent::trace(char *fmt, ...){
    va_list ap;
    if (!logtarget) return;
    va_start(ap, fmt);
    vsprintf(logtarget->pt_->buffer(), fmt, ap);
    logtarget->pt_->dump();
    va_end(ap);
}
```

magnetic_packet.h

```
/*
 * magnetic_packet.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __magnetic_packet_h__
#define __magnetic_packet_h__

/*
 * Packet Formats
 */

#include "magnetic/magnetic_ftable.h"

#define MAGNETICTYPE_PACKET     0x01
#define MAGNETICTYPE_HELLO      0x02
#define MAGNETICTYPE_FORCE      0x04
#define MAGNETICTYPE_FIELD      0x08
#define MAGNETICTYPE_SERVICE    0x10

/*
 * MAGN Routing Protocol Header Macros
 */
#define HDR_MAGNETIC(p)
    ((struct hdr_magnetic*)hdr_magnetic::access(p))
#define HDR_MAGNETIC_PACKET(p)
    ((struct hdr_magnetic_packet*)hdr_magnetic::access(p))
#define HDR_MAGNETIC_FORCE(p)
    ((struct hdr_magnetic_force*)hdr_magnetic::access(p))
#define HDR_MAGNETIC_FIELD(p)
    ((struct hdr_magnetic_field*)hdr_magnetic::access(p))
#define HDR_MAGNETIC_SERVICE(p)
    ((struct hdr_magnetic_service*)hdr_magnetic::access(p))

struct hdr_magnetic {
    u_int8_t     ma_type;

    static int offset_; // required by PacketHeaderManager
    inline static int& offset() { return offset_; }
    inline static hdr_magnetic* access(const Packet* p) {
        return (hdr_magnetic*) p->access(offset_);
    }
};

struct hdr_magnetic_packet {
    u_int8_t    pa_type;                // Type
    char        pa_service[STLEN];      // Service
    int         pa_force;               // Last Force
    nsaddr_t    pa_src;                 // Source Address
    u_int32_t       pa_src_seqno;       // Source Sequence Number

    inline int size() {
        int sz = 0;

        sz =      sizeof(u_int8_t)      // pa_type
            + sizeof(char[STLEN])       // pa_service
            + sizeof(int)               // pa_force
            + sizeof(nsaddr_t)          // pa_src
            + sizeof(u_int32_t);        // pa_src_seqno

        assert (sz >= 0);
        return sz;
    }
};

struct hdr_magnetic_force {
    u_int8_t    fo_type;                // Type
    char        fo_service[STLEN];      // Service
    nsaddr_t    fo_src;                 // Source Address
    nsaddr_t    fo_dest;                // Destination Address
    int         fo_force;               // Force

    inline int size() {
        int sz = 0;

        sz =      sizeof(u_int8_t)      // fo_type
            + sizeof(char[STLEN])       // fo_service
            + sizeof(nsaddr_t)          // fo_src
            + sizeof(nsaddr_t)          // fo_dest
            + sizeof(int);              // fi_force

        assert (sz >= 0);
        return sz;
    }
};

struct hdr_magnetic_service {
    u_int8_t    se_type;                // Type
    char        se_service[STLEN];      // Service
    int         se_force;               // Force

    inline int size() {
        int sz = 0;

        sz =      sizeof(u_int8_t)      // se_type
            + sizeof(char[STLEN])       // se_service
            + sizeof(int);              // se_force
        assert (sz >= 0);
        return sz;
    }
};
```

```
struct hdr_magnetic_field {
    u_int8_t    fi_type;                // Type
    char        fi_service[STLEN];      // Service Type
    int         fi_force;               // FieldForce
    nsaddr_t    fi_src;                 // Source Address
    u_int32_t       fi_src_seqno;       // Source Sequence Number

    inline int size() {
        int sz = 0;

        sz =      sizeof(u_int8_t)      // fi_type
            + sizeof(char[STLEN])       // fi_service
            + sizeof(int)               // fi_force
            + sizeof(nsaddr_t)          // fi_src
            + sizeof(u_int32_t);        // fi_src_seqno

        assert (sz >= 0);
        return sz;
    }
};

// for size calculation of header-space reservation
union hdr_all_magnetic {
    hdr_magnetic      ma;
    hdr_magnetic_packet   pa;
    hdr_magnetic_service  se;
    hdr_magnetic_force    fo;
    hdr_magnetic_field    fi;
};

#endif /*__magnetic_packet_h__ */
```

magnetic_ftable.h

```
/*
 * magnetic_ftable.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __magnetic_ftable_h__
#define __magnetic_ftable_h__

#include <assert.h>
#include <sys/types.h>
#include "config.h"
#include "lib/bsd-list.h"
#include "scheduler.h"

#define STLEN 10

/*
 *   Magnetic Service Cache
 */

class Service_Cache {
    friend class MagneticAgent;
public:
    Service_Cache(){}
protected:
    LIST_ENTRY(Service_Cache) sc_link;
    nsaddr_t   sc_addr;
    char       sc_service[STLEN];
    int        sc_force;
    double     sc_expire;
};

LIST_HEAD(service_cache, Service_Cache);

/*
 *   Magnetic Service Entry
 */

class Service_Entry {
    friend class MagneticAgent;
public:
    Service_Entry(char service[], int force) {
        strcpy(se_service, service);
        se_force = force;
    }

protected:
    LIST_ENTRY(Service_Entry) se_link;
    char    se_service[STLEN];
    int     se_force;
};

LIST_HEAD(service_entry, Service_Entry);

/*
 * Packet receive List
 */

class Receive_Entry {
    friend class MagneticAgent;
public:
    Receive_Entry(nsaddr_t src, u_int32_t seqno) {
        re_src = src;
        re_src_seqno = seqno;
    }
protected:
    LIST_ENTRY(Receive_Entry) re_link;
    nsaddr_t   re_src;
    u_int32_t  re_src_seqno;
};

LIST_HEAD(receive_entry, Receive_Entry);

/*
 * Magnetic ft entry
 */

class magnetic_ft_entry {
    friend class magnetic_ftable;
    friend class MagneticAgent;
public:
    magnetic_ft_entry(){}
protected:
    LIST_ENTRY(magnetic_ft_entry) ft_link;
    nsaddr_t       ft_src;
    u_int32_t      ft_src_seqno;
    char           ft_service[STLEN];
    int            ft_force;
    double         ft_expire;

};

class magnetic_ftable {
    friend class MagneticAgent;
public:
    magnetic_ftable() { LIST_INIT(&fthead); }
    magnetic_ft_entry* head() { return fthead.lh_first; }

    bool updateService(nsaddr_t, u_int32_t, char[], int );
    void registerService(nsaddr_t, char[], int );
    void purgeService(void);
    int serviceForce(char[]);
private:
    LIST_HEAD(magnetic_fthead, magnetic_ft_entry) fthead;
};

#endif /* _magnetic__ftable_h__ */
```

magnetic_ftable.cc

```
/*
 * magnetic_ftable.cc
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#include "magnetic/magnetic_ftable.h"
#include "magnetic/magnetic.h"

bool magnetic_ftable::updateService(nsaddr_t id, u_int32_t seqno,
                                    char service[], int force){
    // return true wenn fieldtable nicht up to date
    magnetic_ft_entry *ft = fthead.lh_first;

    for(;ft; ft = ft->ft_link.le_next){
        if(ft->ft_src == id && !strcmp(ft->ft_service, service)){
            // service already in fieldtable
            if(seqno > ft->ft_src_seqno){
                // newer magnetic packet
                ft->ft_src_seqno = seqno;
                ft->ft_force = force;
                return true;
            }else if(seqno == ft->ft_src_seqno){
                if(force > 0 && force > ft->ft_force){
                    // equal new packet from a service
                    // with higher fieldforce
                    ft->ft_force = force;
                    return true;
                }else if(force < 0 && force < ft->ft_force) {
                    // equal new packet from a client
                    // with lower fieldforce
                    ft->ft_force = force;
                    return true;
                }else{
                    return false;
                }
            }
        }
    }

    // service not in fieldtable
    magnetic_ft_entry *ftnew = new magnetic_ft_entry;
    ftnew->ft_src = id;
    ftnew->ft_src_seqno = seqno;
    strcpy(ftnew->ft_service, service);
    ftnew->ft_force = force;
    ftnew->ft_expire = CURRENT_TIME + FIELD_EXPIRE;
    LIST_INSERT_HEAD(&fthead, ftnew, ft_link);
    return true;
}
```

```
void magnetic_ftable::registerService(nsaddr_t id, char service[], int force){
    // enter the service to the fieldtable of the own node
    magnetic_ft_entry *ftnew = new magnetic_ft_entry;
    ftnew->ft_src = id;
    ftnew->ft_src_seqno = 0; // with seqno = 0 it is the own service
    strcpy(ftnew->ft_service, service);
    ftnew->ft_force = force;
    ftnew->ft_expire = 0; // with expire = 0 it is the own service
    LIST_INSERT_HEAD(&fthead, ftnew, ft_link);
}

void magnetic_ftable::purgeService() {
    // purge all expired field entries
    magnetic_ft_entry *ft = fthead.lh_first;
    magnetic_ft_entry *ftn;
    double now = CURRENT_TIME;
    for(;ft; ft = ftn){
        ftn = ft->ft_link.le_next;
        if(ft->ft_expire < now && ft->ft_expire != 0){
            LIST_REMOVE(ft,ft_link);
        }
    }
}

int magnetic_ftable::serviceForce(char service[]){
    // return the fieldforce for a given service
    int force = 0;
    magnetic_ft_entry *ft = fthead.lh_first;

    for(;ft; ft = ft->ft_link.le_next){
        if(!strcmp(ft->ft_service, service)){
            force += ft->ft_force;
        }
    }
    return force;
}
```

magnetic_queue.h

```
/*
 * magnetic_queue.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __magnetic_queue_h__
#define __magnetic_queue_h__

#include "agent.h"

class magnetic_queue : public Connector {
public:
    magnetic_queue();
    void enque(Packet *p);
    Packet* deque();
private:
    Packet *head_;
    Packet *tail_;
    int len_;
};

#endif /* _magnetic__queue_h__ */
```

magnetic_queue.cc

```
/*
 * magnetic_queue.cc
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#include "magnetic/magnetic_queue.h"

magnetic_queue::magnetic_queue(){
    head_ = tail_ = 0;
    len_ = 0;
}

void magnetic_queue::enque(Packet *p){
    if(head_ == 0){
        head_ = tail_ =  p;
        len_++;
    }else{
        tail_->next_ = p;
        tail_ = p;
        len_++;
    }
}

Packet* magnetic_queue::deque(){
    Packet *p = head_;
    if(head_ == tail_){
        head_ = tail_ = 0;
        len_--;
    }else{
        head_ = head_->next_;
        len_--;
    }
    return p;
}
```

stream.h

```
/*
 * stream.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __stream_h__
#define __stream_h__

#include "magnetic/magnetic.h"

class StreamAgent : public Agent {
public:
    StreamAgent();
    virtual void sendmsg(int nbytes, const char *flags =0){
        sendmsg(nbytes, NULL, flags);
    }
    virtual void sendmsg(int nbytes, AppData* data, const char *flags =0);
    virtual void recv(Packet* pkt, Handler*);
    virtual int command(int argc, const char*const* argv);
    void wait(float seconds);
protected:
    int seqno_;
    char service_[];
};

#endif /* __stream_h__ */
```

stream.cc

```
/*
 * stream.cc
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#include "magnetic/stream.h"

static class StreamClass: public TclClass {
public:
    StreamClass() : TclClass("Agent/STREAM") {}
    TclObject* create(int, const char*const*) {
        return (new StreamAgent());
    }
}stream_agent;

StreamAgent::StreamAgent() : Agent(PT_MAGN), seqno_(-1){
    bind("packetSize_", &size_);

}

void StreamAgent::sendmsg(int nbytes, AppData* data, const char* flags){
    // send a packet to the routing agent
    Packet *p;

    p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_packet *mp = HDR_MAGNETIC_PACKET(p);

    mp->pa_type = MAGNETICTYPE_PACKET;
    strcpy(mp->pa_service, service_);
    mp->pa_force = 0;
    mp->pa_src = addr();
    mp->pa_src_seqno = ++seqno_;

    ch->ptype() = PT_MAGN;
    ch->size() = size_;
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_NONE;
    ch->prev_hop_ = addr();  // Hack
    ch->uid() = seqno_;

    ih->saddr() = addr();
    ih->daddr() = IP_BROADCAST;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;
    ih->ttl_    = NETWORK_DIAMETER;

    target_->recv(p);
    idle();
}
```

```
void StreamAgent::recv(Packet* pkt, Handler*){
        }

int StreamAgent::command(int argc, const char*const* argv){
    if(!strcmp(argv[1], "service")){
        strcpy(service_,(char[])argv[2]);
        return (TCL_OK);
    }
    if(argc == 4) {
        if (strcmp(argv[1], "send") == 0) {
            PacketData* data = new PacketData(1 + strlen(argv[3]));
            strcpy((char*)data->data(), argv[3]);
            sendmsg(atoi(argv[2]), data);
            return (TCL_OK);
        }
    }else if(argc == 5) {
        if(strcmp(argv[1], "sendmsg") == 0) {
            PacketData* data = new PacketData(1 + strlen(argv[3]));
            strcpy((char*)data->data(), argv[3]);
            sendmsg(atoi(argv[2]), data, argv[4]);
            return (TCL_OK);
        }
    }
    return (Agent::command(argc, argv));
}
```

service.h

```
/*
 * service.h
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#ifndef __service_h__
#define __service_h__

#include "magnetic/magnetic.h"

class ServiceAgent : public Agent{
public:
    ServiceAgent();
    void sendService(char *service, int force);
    virtual int command(int argc, const char*const* argv);
protected:
    int seqno_;
};

#endif /* __service_h__ */
```

service.cc

```
/*
 * service.cc
 *
 * authors: Lorenz Buehrer
 * created: November 02 - March 03
 *
 */

#include "magnetic/service.h"

static class ServiceClass: public TclClass {
public:
    ServiceClass() : TclClass("Agent/SERVICE") {}
    TclObject* create(int, const char*const*) {
    return (new ServiceAgent());
    }
}service_agent;

ServiceAgent::ServiceAgent() : Agent(PT_MAGN){
}

int ServiceAgent::command(int argc, const char*const* argv){
    if (!strcmp(argv[1], "start")){
        char service[10];
        strcpy(service, argv[2]);
        int force = atoi(argv[3]);
        sendService(service, force);
        return (TCL_OK);
    }
    return (Agent::command(argc, argv));
}

void ServiceAgent::sendService(char service[], int force){
    // send a packet to the routing agent containg
    // the information on the service
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_magnetic_service *ms = HDR_MAGNETIC_SERVICE(p);

    ms->se_type = MAGNETICTYPE_SERVICE;
    strcpy(ms->se_service, service);
    ms->se_force = force;

    ch->ptype() = PT_MAGN;
    ch->size() = IP_HDR_LEN + ms->size();
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_NONE;
    ch->prev_hop_ = addr();  // Hack

    ih->saddr() = addr();
    ih->daddr() = IP_BROADCAST;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;
    ih->ttl_    = NETWORK_DIAMETER;

    target_->recv(p);
}
```

C.2 ns-2 Files

Makefile

```
.
.
.
dsr/simplecache.o dsr/sr_forwarder.o \
aodv/aodv_logs.o aodv/aodv.o \
aodv/aodv_rtable.o aodv/aodv_rqueue.o \
magnetic/magnetic.o magnetic/magnetic_ftable.o \
magnetic/magnetic_queue.o \
magnetic/service.o magnetic/stream.o magnetic/client.o \
common/ns-process.o \
satellite/satgeometry.o satellite/sathandoff.o \
satellite/satlink.o satellite/satnode.o \
.
.
.
```

tcl/lib/ns-agent.tcl

```
.
.
.

Agent/AODV set sport_   0
Agent/AODV set dport_   0

Agent/MAGN instproc init args {

         $self next $args
}

Agent/MAGN set sport_ 0
Agent/MAGN set dport_ 0
```

tcl/lib/ns-default.tcl

```
.
.
.

Agent/Ping set packetSize_ 64

#added by buehrer
Agent/STREAM set packetSize_ 512


Agent/UDP set packetSize_ 1000
Agent/UDP instproc done {} { }
Agent/UDP instproc process_data {from data} { }

.
.
.
```

tcl/lib/ns-lib.tcl

```
            .
            .
            .
    AODV {
    set ragent [$self create-aodv-agent $node]
    }
    #added by buehrer
    MAGN {
    set ragent [$self create-magn-agent $node]
    }
    TORA {
    Simulator set IMEPFlag_ ON
    set ragent [$self create-tora-agent $node]
    }

.
.
.

Simulator instproc create-aodv-agent { node } {
        #  Create AODV routing agent
set ragent [new Agent/AODV [$node node-addr]]
        $self at 0.0 "$ragent start"     ;# start BEACON/HELLO Messages
        $node set ragent_ $ragent
        return $ragent
}

#added by buehrer
Simulator instproc create-magn-agent { node } {
set ragent [new Agent/MAGN [$node node-addr]]
        $self at 0.0 "$ragent start"     ;# start BEACON/HELLO Messages
        $node set ragent_ $ragent
        return $ragent
}


Simulator instproc use-newtrace {} {
Simulator set WirelessNewTrace_ 1
}

.
.
.
```

tcl/lib/ns-packet.tcl

```
.
.
.

foreach prot {
AODV
MAGN
ARP
   aSRM
Common
.
.
.
```

common/packet.h

```
.
.
.
PT_AODV,
PT_IMEP,
// added by buehrer
PT_MAGN,

// RAP packets
PT_RAP_DATA,

.
.
.
name_[PT_DSR]= "DSR";
name_[PT_AODV]= "AODV";
name_[PT_IMEP]= "IMEP";

                // added by buehrer
name_[PT_MAGN]= "MAGN";

name_[PT_RAP_DATA] = "rap_data";
name_[PT_RAP_ACK] = "rap_ack";

                .
.
.
```

C.3 Scripts

makescen.sh

```
#!/usr/bin/sh
# makescen.sh
#
# authors: Lorenz Buehrer
# created: November 02 - March 03
#
# Shell script to automatically generate scenario files with
# the scenario generator provided by ns-2

nodes='50' #'20'
speed='20'
time='900'
X='1000'   #'750'
Y='1000'   #'750'

for pause in 900 600 300 100 30 0
  do
  for i in 1 2 3 4;
do echo n$nodes/scen-n$nodes-p$pause-s$speed-t$time-x$X-y$Y-$i
setdest -n $nodes -p $pause -s $speed -t $time -x $X -y $Y > \
n$nodes/scen-n$nodes-p$pause-s$speed-t$time-x$X-y$Y-$i;
```

```
  done
  echo ------------------;
done
```

maketraf.sh

```
#!/usr/bin/sh
# maketraf.sh
#
# authors: Lorenz Buehrer
# created: November 02 - March 03
#
# Shell script to automatically generate traffic files with
# the traffic generator provided by ns-2

type='cbr'
nodes='50' #'20'
seed='1'
# 1/5 of nodes
mc='10'    #'4'
rate='4.0'

ns /home/lbuehrer/ns-allinone-2.1b9a/ns-2.1b9a/
    indep-utils/cmu-scen-gen/cbrgen.tcl \
-type $type -nn $nodes -seed $seed -mc $mc -rate $rate \
> $type-n$nodes-s$seed-mc$mc
```

delivery.pl

```
#!/usr/bin/perl
# delivery.pl:
# Perlscript to analyzed the delivery ratio
# in a trace file
use strict;
my($is, $ir,$ratio, $file, $dataline, @filelist, @line);

@filelist=`ls -1|grep MAGN`; #.tr`;

foreach $file(@filelist){
  if($file !~ /~/i){
$is = 0;
$ir = 0;
chomp ($file);
print $file;
if(open(FILE,$file)){
  while($dataline=<FILE>){
@line = split(/ /,$dataline);
if(@line[1] > 1){
  # all services have to be installed before 1 s
  if($dataline =~ /AGT/) {
if (@line[0] =~ /s/) {
  $is++;
}elsif(@line[0] =~ /r/) {
  $ir++;
}
  }
}
  }
  if ($is == 0) {
print "\tError ! No packets sent\n"
  }else {
$ratio = 100.0 * $ir / $is;
print "\tS: ",$is," R: ",$ir," Ratio: ",$ratio," %\n";
  }
}else{
  die "Error while opening trace file.\n";
}
  }
}
```

overheadmagn.pl

```
#!/usr/bin/perl
# overheadmagn.pl:
# Perlscript to analyzed the routing overhead
# in a MAGN trace file
use strict;
my($ir,$overhead, $file, $dataline, @filelist, @line);

@filelist=`ls -1|grep MAGN`;

foreach $file(@filelist){
  if($file !~ /~/i){
$ir = 0;
$overhead = 0;
chomp ($file);
print $file;
if(open(FILE,$file)){
  while($dataline=<FILE>){
if($dataline !~ /STREAM/){
  @line = split(/ /,$dataline);
  if(@line[1] > 1) {
# all services have to be installed before 1 s
if (@line[0] =~ /r/ && @line[3] =~ /RTR/ && @line[7] =~ /MAGN/){
  $ir++;
  $overhead += @line[8];
}
  }
}
  }
  print "\tR: ",$ir," Overhead : ",$overhead, "\n";
}else{
  die "Error while opening trace file.\n";
}
  }
}
```

overheadaodv.pl

```
#!/usr/bin/perl
# overheadaodv.pl:
# Perlscript to analyzed the routing overhead
# in a AODV trace file
use strict;
my($ir,$overhead, $file, $dataline, @filelist, @line);

@filelist=‘ls -1|grep AODV‘;

foreach $file(@filelist){
  if($file !~ /~/i){
$ir = 0;
$overhead = 0;
chomp ($file);
print $file;
if(open(FILE,$file)){
  while($dataline=<FILE>){
@line = split(/ /,$dataline);
if(@line[0] =~ /r/ && @line[3] =~ /RTR/ && @line[7] =~ /AODV/){
  $ir++;
  $overhead += @line[8];
}
  }
  print "\tR: ",$ir," Overhead : ",$overhead, "\n";
}else{
  die "Error while opening trace file.\n";
}
  }
}
```

hopsmagn.pl

```
#!/usr/bin/perl
# hopsmagn.pl:
# Perlscript to analyzed the average hops
# in a MAGN trace file
use strict;
my($is, $if, $ir, $hops, $file, $dataline, @filelist, @line);

@filelist=‘ls -1|grep MAGN‘;

foreach $file(@filelist){
  if($file !~ /~/i){
$is = 0;
$if = 0;
$ir = 0;
chomp ($file);
print $file;
if(open(FILE,$file)){
  while($dataline=<FILE>){
@line = split(/ /,$dataline);
if(@line[1] > 1) {
  # all services have to be installed before 1 s
  if($dataline =~ /STREAM/) {
if (@line[0] =~ /s/ && @line[3] =~ /AGT/){
  $is++;
}elsif(@line[0] =~ /r/ && @line[3] =~ /AGT/){
  $ir++;
}elsif(@line[0] =~ /s/ && @line[3] =~ /RTR/){
  $if++;
}
  }
}
  }
  if ($is == 0) {
print "\tError ! No packets sent\n"
  }else {
#$if -= $is; # all sents from the source node are counted too
$hops = $if / $is;
print "\tS: ",$is," R: ",$ir, " F: ",$if," AV. Hops: ",$hops,"\n";
  }
}else{
  die "Error while opening trace file.\n";
}
  }
}
```

hopsaodv.pl

```
#!/usr/bin/perl
# hopsaodv.pl:
# Perlscript to analyzed the average hops
# in a AODV trace file
use strict;
my($is, $if,$ir, $hops, $file, $dataline, @filelist, @line);

@filelist=‘ls -1|grep AODV‘;
```

```
foreach $file(@filelist){
  if($file !~ /~/i){
$is = 0;
$if = 0;
$ir = 0;
chomp ($file);
print $file;
if(open(FILE,$file)){
  while($dataline=<FILE>){
@line = split(/ /,$dataline);
if(@line[1] > 1) {
  if($dataline =~ /cbr/){
if (@line[0] =~ /s/ && @line[3] =~ /AGT/){
  $is++;
}elsif(@line[0] =~ /r/ && @line[3] =~ /AGT/){
  $ir++;
}elsif(@line[0] =~ /f/) {
  $if++;
}
  }
}
  }
  if ($is == 0) {
print "\tError ! No packets sent\n"
  }else {
$hops = $if / $is;
print "\tS: ",$is," R: ",$ir," F: ",$if," AV. Hops: ",$hops,"\n";
  }
}else{
  die "Error while opening trace file.\n";
}
  }
}
```

Bibliography

[1] TIK Homepage:
http://www.tik.ee.ethz.ch

[2] ETH Homepage:
http://www.ethz.ch

[3] Charles E. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001.

[4] Charles E. Perkins, Pravin Bhagwat. *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. SIGCOMM August 1994, London.

[5] Charles E. Perkins, Elisabeth M. Belding-Royer and Samir R. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-13.txt, February 2003

[6] David B. Johnson, David A. Maltz andYih-Chun Hu. *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)*. IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-08.txt, February 2003

[7] Zygmunt J. Haas, Marc R. Pearlman and Prince Samar.*The Zone Routing Protocol (ZRP) for Ad Hoc Networks*. IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-zone-zrp-04.txt, July 2002

[8] Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin. *Direct Diffusion: A Scalable and Robust Communication Paradigm for Sensor Network*. MOBICOM 2000, Boston.

[9] USC Homepage:
http://www.usc.edu

[10] ISI Homepage:
http://www.isi.edu

[11] The Network Simulator - ns-2:
http://www.isi.edu/nsnam/ns

[12] Solaris Operating System:
http://wwws.sun.com/software/solaris/index.html

[13] Linux:
http://www.linux.org

[14] Microsoft Windows:
http://www.microsoft.com/windows/

[15] The *ns* Manual:
http://www.isi.edu/nsnam/ns/ns-documentation.html

[16] ns-2 Download Page:
http://www.isi.edu/nsnam/ns/ns-build.html

[17] Concurrent Versions System (CVS):
http://www.cvshome.org

[18] C++ Reference Page:
http://www.cplusplus.com

[19] OTcl Tutorial:
http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/tutorial.html

[20] CMU Homepage:
http://www.cmu.edu

[21] SunFire 280R:
http://www.sun.com/servers/entry/280r/index.html

[22] Perl Mongers:
http://www.perl.org/

[23] ns-2 Tutorial:
http://www.isi.edu/nsnam/ns/tutorial/index.html

[24] GDB: The Gnu Project Debugger:
http://sources.redhat.com/gdb/

[25] LaTeXTypesetting System:
http://www.latex-project.org

[26] Emacs Editor:
http://www.gnu.org/software/emacs/emacs.html

[27] Gnuplot Central:
http://www.gnuplot.info/

[28] Microsoft Visio 2000:
http://www.visio.som

[29] Acrobat 5.0:
http://www.acrobat.com

[30] WinEdt 5.3:
http://www.winedt.com

[31] MikTex:
http://www.miktex.org