UNIVERSITÄT MANNHEIM

Lehrstuhl für Elektrotechnik – Institut für Technische Informatik

MIRP
Map Information Routing Protocol for Mobile Ad-Hoc Networks

Diplomarbeit

von

Arndt Oberhöffken

Betreuer:	Dipl.-Wirtsch.-Inf. Holger Füßler
Hochschullehrer:	Professor Dr.-Ing. Dr.-Ing. E.h. N. Fliege
	Professor Dr. W. Effelsberg
Ausgabetermin:	02. Juni 2003
Abgabetermin:	15. April 2004
Unterbrechung:	18 Wochen

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, den 15. April 2004 Arndt Oberhöffken

Acknowledgements

I would like to thank Holger Füßler and Mathias Transier for their help concerning questions about the ns-2 simulator.
Furthermore Dipl.-Inf. Dirk Merettig has my gratitude.

Abstract

Development of an algorithm to route packets through a city scenario. Vehicular nodes in range of junctions gather information about the 'states' of adjacent streets and so act as superior routing stations.
Previous work indicates that greedy position-based forwarding works very well in one-dimensional street configurations. Thus an effort is made to separate the problem into greedy routing to the next junction and the selection of the street to use. The object of this thesis will be to understand streets in a city and the vehicles travelling upon them as communication links between junctions. These links can be considered up or down with respect whether data packets reach the neighbouring junction.

Contents

Abbreviations

AODV	Ad-Hoc On-Demand Distance Vector Routing
ARP	Address Resolution Protocol
CBF	Contention Based Forwarding
CCA	Clear Channel Assessment
CSMA/CD	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear To Send
DCF	Distributed Coordination Function
DFIR	Diffused Infrared
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
FHSS	Frequency Hopping Spread Spectrum
GOD	General Operations Director
GPS	Global Positioning System
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing

GSM	Global System for Mobile Communication
HLS	Hierarchical Location Service
JRTE	Junction Routing Table Entries
LMNM_Timer	Leaving Master New Master Timer
LSFV	Link State Flag Vector
LSRC_Timer	Link State Refresh Cycle Timer
MANET	Mobile Ad-Hoc Networks
MatJTimer	Master at Junction Timer
MDB	Map Database
MFR	Most Forward Within Transmission Range
MID	Map ID
MIRP	Map Information Routing Protocol
MIRP_DATA	Standard Data Packet
MIRP_LMACK	Leaving Master Acknowledge
MIRP_LMADV	Leaving Master Advertisement
MIRP_LMAXP	Leaving Master Exchange Packet
MIRP_LSREP	Link State Reply
MIRP_LSREQ	Link State Request
MIRP_MatJ	Master at Junction
MIRP_PMRP	Present Master Recall Packet
MIRP_RADV	Receive Advertisement
OMNILOC	Omniscient Location Service

PCF	Point Coordination Function
PLCP	Physical Layer Convergence Protocol
RTS	Request To Send
TTL	Time To Live
UMTS	Universal Mobile Telecommunications System
VANET	Vehicular Ad-Hoc Networks
WLAN	Wireless Local Area Network

List of Figures

Chapter 1

Introduction

Today's technical progress is astounding.

The rise in information and knowledge exchange began in 1876 when Bell developed the first telephone. Nowadays nearly every teenager has a mobile phone enabling them to communicate wirelessly with others. The availability of Wireless Local Area Network (WLAN) adapters in even such small dimensions as the Secure Data (SD) Card slots in state-of-the-art cellular phones enables the creation of local WLAN in combination with GSM (Global System for Mobile Communication), Bluetooth and UMTS (Universal Mobile Telecommunications System).

was sind das

Mobile Ad-Hoc Networks (MANET) form an interesting area of reseach due to their challenge in developing suitable algorithms for the communication of their participants. An article covering embedding of MANET in the real world is [14].

Whereas the earlier mentioned cellular networks are based on partly wired layout in between different base transceiving spelling stations and on the other hand wireless communication between the cellular phones and these stations, MANET normally dispense with any wired and fixed parts giving them great mobility. Thereby they are most valuable in areas where either infrastructure is missing or an existing infrastructure has been destroyed by war or natural disaster.

Because of the range restrictions of wireless transmissions the nodes have to be able to forward the information. As a result each participant in a MANET is consumer and distributor at the same time. The nodes can be both stationary or mobile. Each node has to meet the same routing standards, and the problem of a constantly changing network must be solved.

MANET could be interesting in any kind of inter-vehicular communication which leads over to Vehicular Ad-Hoc Networks (VANET) . These could be propagating automated traffic information as well as street surface inadequacy warnings or basically any security issue concerning the traffic following behind, namely the on-board active safety systems. VANET could even have been a possible solution to the government's (ministery of transport in Germany in spring 2004) toll collecting projects including some major representatives of the German industry.

VANET feature certain differences from original MANET, which need to be explained. The embedding of wireless network devices in automobiles solved the problem of the limited power supply, which is a drawback of almost all mobile devices, in addition VANET are able to marshal relatively large computational resources. As the vehicles are bound to driving speed restrictions and usually constrained by roads, VANET constitute high resource/performance wireless technology with an extraordinary research challenge.

1.1 Problem statement

The Department of Computer Science IV at the University of Mannheim is proud of its developments concerning MANET. One of the research fields consists in the FleetNet Project which deals with VANET [4, 5]. Position-based routing algorithms for VANET are especially interesting as they form a different approach to packet routing than the existing topology-based routing algorithms. Due to the availability of GPS (Global Positioning System)[1] in modern vehicles positional information[1] can be easily acquired and compared

[1]concerning the own position

with the cartographic information from a navigation system . This offers interesting possibilities for research in intervehicular city scenarios.

Beside many of its aspects the theme of this thesis is focused on an algorithm for city scenarios. City scenarios are of special interest beacuse of the difficulties of buildings interfering with the radio transmissions as well as the possibilities to use certain specific and characteristic information. Valuable attributes can be for example a higher node density than in rural areas and an accumulation of nodes in junction areas with a possible delay, either to give right of passage or because of traffic lights. Other reasons like train or river crossings or as in New York toll checkpoints are rather specific. A Routing Strategy for Vehicular Ad Hoc Networks in City Environments [10] has been developed by the Department of Computer Science IV which models obstacles into the ns-2 simulations to account for the stated transmission restrictions in city environments. Still a better usage of the provided information could be emphasised. A street to link abstraction is therefore the main task of this thesis. The idea is that nodes residing in the junction areas acquire information about the link states of the adjacent streets. The street links are classified in up or down states, referring to a response of the junction at the other side of the street link. An algorithm will be presented to meet these objectives.

1.2 Structure

Whereas Mobile Ad-Hoc Network basics have been covered and explained in the main part of the second chapter, the end of the second chapter describes the Contention-Based-Forwarding protocol which is a fundament of this work. The third chapter gives a general overview of the MIRP algorithm on the basis of the defined possible node states. While the fourth chapter introduces the ns-2 network simulation environment and describes the assembly of the required input files, the fifth chapter deals with the implemented algorithm in detail introducing a UML diagram overview and an explicit data dictionary for virtually every used routine. The sixth chapter presents the results, partly on basis of the FleetNet simulation data, mostly on the network simulation.

The thesis concludes with an outlook and some prospectus.

Chapter 2

Mobile Ad-Hoc Networks

2.1 IEEE 802.11

2.1.1 Physical Layer

The IEEE 802.11 standard supports three different types of physical layer management. These are:

- FHSS - Frequency Hopping Spread Spectrum
- DSSS - Direct Sequence Spread Spectrum
- DFIR - Diffused Infrared

The most common of these, DSSS, uses the 2.4 GHz frequency wave spectrum and signals the occupancy of the medium by the Clear Channel Assessment signal CCA. If a receiving node can not deduce the allocation of the medium despite being in range, a problem called "hidden terminal" [2] arises. The detection is then prevented by a signal collision of at least one other sending node (outside the radio range of the first - as it would have sensed the unavailability of the medium, if not affected by a hidden terminal appearance itself). For this purpose the Physical Layer Convergence Protocol (PLCP) adds a preamble (144 bit) and header (48 bit) to all outgoing packets resulting in a constant delay for all transmissions. Node A cannot sense that node C is transmitting information to node B as it is outside the radio range of C

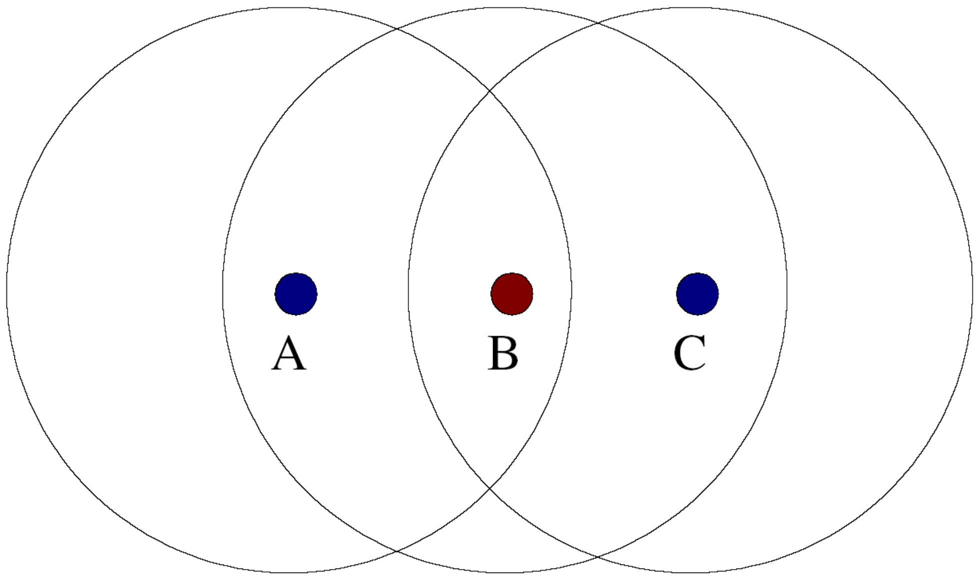

Figure 2.1: Hidden terminal

(as indicated by the circle surrounding C as centre). A therefore may start sending packets which results in a packet collision at B, which, indicated by the interconnection of the circles, is in the intersection of both transmission ranges. Here C is a hidden terminal for A and vice versa.

Because of the limitations of air as the medium the handling of packets and amount of redundancy added is a difficult tradeoff. Too much redundancy results in less amount of data transmitted, whereas too few in form of recovery information could result in severe data loss since the medium is prone to collisions resulting in packet drops or losses.

2.1.2 Media Access Control

Of the two basic medium access mechanisms the MAC layer provides, the Point Coordination Function (PCF) is unconvertible in MANET owing to their infrastructural nature which is conflictive with the very idea of MANET. The second, the Distributed Coordination Function (DCF), consists of two schemes, basic CSMA/CA - Carrier Sense Multiple Access with Collision Avoidance - and extended CSMA/CA with Request To Send (RTS) and

Clear To Send (CTS) as a handover policy. DCF introduces priority based medium access in combination with contention to avoid packet collision. The main idea, a contention method, is of special interest as it is one of the major concepts for packet propagation in the MIRP algorithm. In basic CSMA/CA ordinary packets (with no priority) listen for a free medium. When this idle state is detected, the rivalling nodes enter contention phase. They each generate a random delay and wait for its expiration. The first node with an expired timer wins the contest and suppresses the other nodes by starting to send. These are hereby informed and keep their remaining timer duration for the next contention period. This keeping of the unexpired timer rudiment results in a fair sharing of the medium. The longer the node waits for a successful possibility to send, the shorter is its remaining timer, which results in a higher probability to be the next contention winner.

In the extended CSMA/CA version with RTS/CTS packets a handover is realized to avoid the hidden node problem. The procedure is quite simple: The sending node initiates the transmission with a small RTS packet which contains information about the data transfer it wants to perform. Included in the packet are the receiving nodes ID, the amount it wants to transmit as well as the probable duration of the transfer including the request, data and ACK. It is read by the destination node as well as by all the others in radio range. The destination node then prepares a CTS return packet also including the duration of the transfer and this packet is heard by possible interfering nodes on the opposite side of the sender, which would not be suppressed by the request packet and could not detect the sending of the source. This CTS packet initiates the transfer and additionally hampers possible other senders in range of the destination for the given time period.

2.2 Routing Protocols

The existing protocols for MANET can be generally partitioned into two classes: Topology-based and position-based routing protocols.

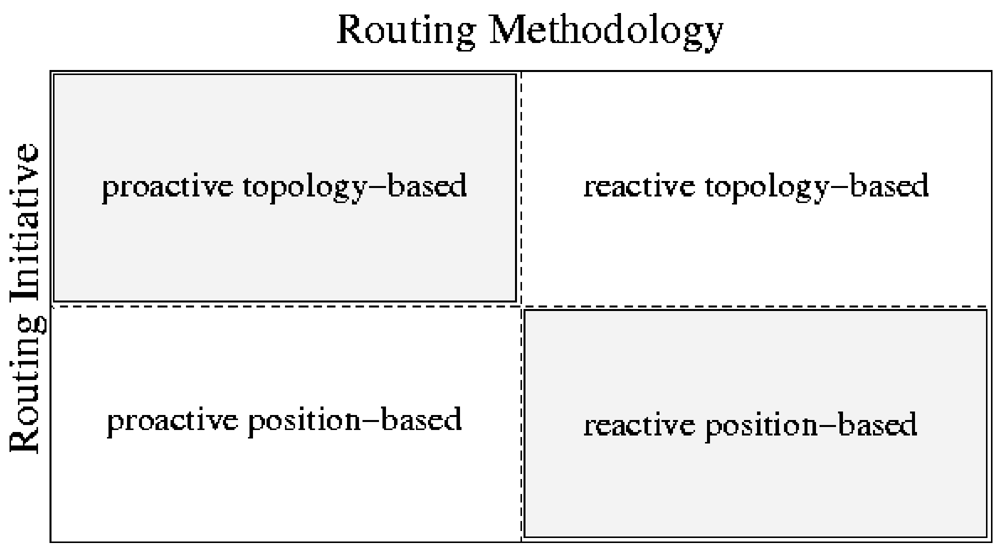

Figure 2.2: Types of routing protocols

2.2.1 Topology-based routing

Topology-based protocols make their routing decisions on the logical layout of the surrounding network. Their routing algorithms are akin to the wired routing protocols with the extra accounting for randomly changing routes in the wireless scenario. Topology-based protocols thus perform a two step approach: At first the composition of the route to the destination and then the transmission of the data to the destination via this route.

2.2.2 Position-based routing

The availability of cheap GPS (Global Positioning System)[1] receivers paved the way for position-based protocols. These use a totally different approach. Their routing decisions are calculated on the geographical information of the destination node. The routes taken are dependent on the last hop position as well as on the coordinates of the next forwarding node. Each packet sent off needs to find its own way to the destination. To know the position of the destination a service is needed. In unicast networks this is achieved via a location service. Hierarchical Location Service (HLS)[9] and Homezone are two representatives.

2.2.3 Proactive and reactive approaches

Proactive and reactive approaches exist in these two protocol classes. Proactive would mean preventative tracing of routes and their upkeep, reactive acts *on demand* only. Obviously the proactive concept is very resource consuming and dealing with only the air as a very limited medium reactive handling is the more promising concept.

Examples

A well known topology-based proactive protocol is *Destination-Sequenced Distance-Vector Routing* (DSDV)[11]. It maintains routing tables with all available paths in the network. As these paths are represented by wirelessly linked moving nodes the topology is due to change permanently. The result is a constant bandwidth usage to update the routing table. Representatives of a reactive approach are *Ad-Hoc On-Demand Distance Vector Routing* (AODV)[12] and *Dynamic Source Routing* (DSR)[13]. After an initial discovery phase only active routes are stored and every reconnect of an expired route needs to trigger a rediscovery.

Greedy Perimeter Stateless Routing (GPSR)[7] on the other hand is a popular position based routing protocol. Greedy routing refers to the 'Most Forward Within Transmission Range' (MFR) heuristic, the effort to make the most progress towards the destination, i.e. taking huge steps towards it. In GPSR the sending node decides which node has made the most progress. To be able to do so, each node has to know the positions of the nodes in radio range, the neighbouring nodes. This information is exchanged via special beacon packets which carry information like ID (IP-address) and position of the sender and in some cases even advanced information like speed and driving direction to enable a better choice for the suitable next hop. As an alternative to permanent beacon packet exchange nodes could use their network interfaces promiscuously, listening to all passing packets, which enables piggybacked beaconing. Thus no more beacons have to be spread but instead every data packet carries the position of its source as additional information. The drawback is the extra payload for every data packet sent off.

But greedy forwarding bears some problems with local minima (areas where no greedy node closer to the destination exists). A possible solution to cope with minima is the right hand rule. The area of the minimum represents a wall and - similar to a human in a simple labyrinth, who touches the wall with his right hand and moves on - the packets move around the area till eventually reaching a position being closer to the destination than the minimum. To perform this, a planarisation of the graph of routes needs to be calculated. Once moved around the area, greedy routing can be applied again.

A different concept represents *Contention Based Forwarding* (CBF)[6, 3, 8], which has been developed at the Chair of Computer Science IV at the University of Mannheim. Its basic version is used as basis for MIRP and will therefore be presented in more detail.

2.3 Contention Based Forwarding

The CBF protocol, as a position-based routing protocol, renounces the beacon exchange by assigning the task of selecting the next hop to the possible forwarders. The sending node broadcasts the packet. Hearing it the neighbouring nodes calculate their suitability to be the next hop. Their progress towards the packet destination is transformed into a specific delay for this packet on the basis of the formula:

$$t_{backoff}(p) = T \cdot (1 - p) \qquad p \epsilon [0, 1],$$

where p denotes the forwarding progress and T is the maximum forwarding delay. The progress is calculated from:

$$F(x) = F(\vec{n}) = \begin{cases} \frac{\|\vec{d}-\vec{s}\ \| - \|\vec{d}-\vec{n}\ \|}{r} & if\ \|\vec{d} - \vec{s}\| \geq \|\vec{d} - \vec{n}\| \\ 0 & else, \end{cases}$$

where $\vec{d}$ is the position of the destination node, $\vec{s}$ is the position of the forwarding node, $\vec{n}$ is the position of the neighbour node and r is the radio range. As negative delays are impossible in the simulation environment the special case $\|\vec{d} - \vec{s}\| - \|\vec{d} - \vec{n}\| > r \Rightarrow p = 1$ (*which can only happen in the simulation*) is needed. All contestants activate a timer for the packet ID. The node with the best progress has the shortest delay and thus forwards the packet first. This forwarded packet is then recognised by the competitors as a signal of successful relay and they all cancel their pending timers and discard their version of the packet.

CBF, as a greedy forwarding algorithm, contains a fallback strategy in case of a found local minimum for greedy packet forwarding. The mode is switched, and the earlier mentioned right hand rule strategy is used to find a new forwarding node, closer to the destination then the excluded node, as a result of the local minimum. Basic CBF is affected by packet duplication and packet collision due to the spread of its packets. Hidden terminal is only a special form which can appear. Straighter forward is simply the fact that two possible contenders for a packet from the same source are just outside the range to receive transmissions from each other. They have no means to detect the forwarding of their pending packet owing to a simple range problem. The consequence is a duplicate packet and could even result in a collision at the location of a sole next hop in range of both contestants.

Hence CBF also offers more advanced features for packet forwarding. A Reuleaux Triangle is used for a more restrictive forwarding strategy. Only forwarding contenders localized in this special Reuleaux area are allowed to forward the packet. The triangle is drawn from the source node in direction towards the destination. If no candidate is found, the packet is retransmitted with a 60° rotated area to find a contender. If still no node is detected the next retransmission rotates the area -60° from the original position. This sequence is repeated till the packet expires or a forwarding node is found. The dimension of these Reuleaux Triangles ensures that all possible competitors are in radio range to each other and therefore packet duplication (with just one sender) is avoided.

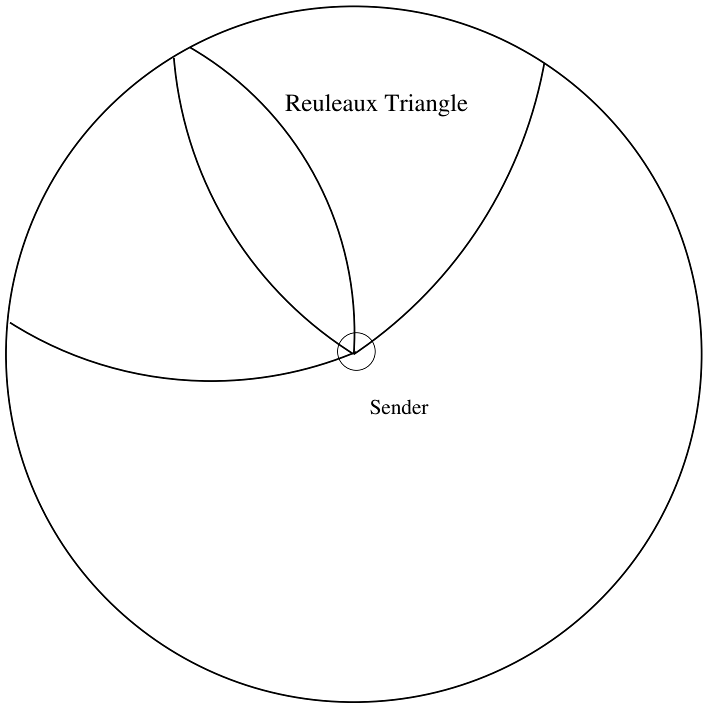

Figure 2.3: Reuleaux Triangle

Chapter 3

The Map Information Routing Protocol

3.0.1 Motivation

Another challenge to position based routing lies in the area of city environments. These scenarios harbour the difficulty with buildings or other obstacles disabling the communication in between different streets, as they are impenetrable by radio transmission. Even worse, the reflections on most surfaces add extra noise to the existing transmissions with a disturbing delay. On the other hand city layouts are, looked at from bird's eye view, a multi meshed network of connecting streets. This induces an interesting thought:

Why not use streets as links and junctions as hubs, collecting linkstates and doing a routing akin to wired routing?

This is precisely the main idea of MIRP.

3.0.2 Setup

Now a modelling environment is needed to set up a simulation. The ns-2 network simulator originates from the ns simulator which started in 1989 as a variant of the REAL network simulator. Its origins base on wired network simulations which over the years, and the project becoming more and more popular, evolved into ns-2, a discret event simulator for network research and development. In 1998 wireless networking support was integrated by the Monarch Group at the Carnegie Mellon University. The sources are publicly available and consist of C++ for the core engine and OTcl as frontend. For the development of the MIRP routing algorithm version 2.1b8a of the ns-2 network simulator was used, including many modifications over the past years in the context of the FleetNet project in cooperation with the network laboratories of NEC. Results produced by the ns-2 simulator should not only be considered from the qualitative point of view. Real-world tests with Smart cars equipped with wireless network devices and GPS have been undertaken by DaimlerChrysler in Ulm to some extend. DaimlerChrysler has also gathered streetscenario data with a flow of traffic simulator named *Videlio* which will be a basis for the simulations and will be mentioned in detail later on.

3.0.3 Prerequisite:

- a routing agent has to be implemented in ns-2
- a close-to-reality street scenario file for ns-2
- a close-to-reality implementation of node (car) movement
- a randomised communication pattern file

3.1 Algorithm

All participating nodes imbibe one of three definite states, depending on their location:

- 0 = junction-node slave
- 1 = street-node
- 2 = junction-node master

A node is considered to be member of the junction if its distance drops below a certain value. This value is directly accountable for the amount of nodes considered to belong to a junction. A junction may only have a single master node at the same time. All other nodes in range are set to slave mode. Every master queries the state of his adjacent links (streets) to learn about its status. This query is a special link state packet destined for the junction on the other side of the street. A link is considered up or working if a link state reply packet from the queried junction is returned, bearing the same packet ID as the query. Since MIRP is based on the CBF concept it is designed to enable packet forwarding without any non-local information with the exception of the master nodes residing at the junctions. All forwarding packets are broadcasts and analysed by all nodes in radio range with the exceptional case of the junction database exchange packet, which is transmitted as unicast to the receiving slave.
MIRP uses greedy forwarding with contention in between junctions. The problem of running into a local minimum does not exist since all packets are bound to a certain street and the radio range is always greater than the street width. It is possible that no further forwarding node for a street is available. In that case the link is broken. The master node administers a junction database in which it stores information about the link states. Based on this database a suitable next junction is chosen for each data packet.

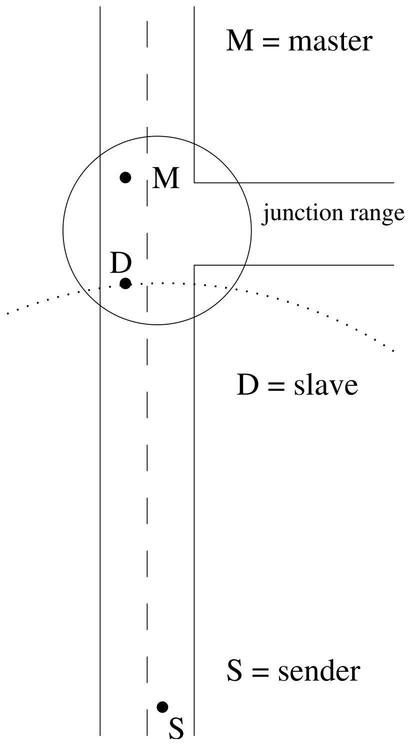

Figure 3.1: The delayed slave

3.1.1 The Junction Slave (0)

All nodes sojourning in the range of a junction are considered slaves. They are not directly taking part in any relaying of packets except when originating data packets (CBR) themselves .

- As a minor functionality slave nodes participate in the contention for packets destined for their junction, but with a small extra increase in the contention timer value to eliminate the possibility of swooping packets from the master, if their distance to the sender results in a minimum timer delay. This antagonises the possibility of a packet loss for the junction if the master is either just outside of the radio range the slave node might still reside in or could not receive the packet due to a packet collision his slave was not affected by. The figure 3.1 shows an example: The master hears the packet, even if he is beyond radio range limits due to a good transmission. The junction slave (D) would calculate his delay to be 0 as he is in optimum range. The result is a renewed delay

$$d = \frac{r_j \cdot 2}{r_{rr}} \qquad d = delay, r_j = junctionradius, r_{rr} = radiorange$$

- All arriving nodes at the junction are operating in slave mode. They wind up a timer during which period they are listening for a master being present at their junction. If a master is recognized, the timer is rescheduled using a fixed increment and a variable part (*this could be derived from the nodes sojourn time*). Otherwise the node assumes that no master is present and broadcasts a master-at-junction-advertisement-packet to inform all other slaves at this junction of the now existing master. The other slave nodes thereupon reschedule their timer and resume in waiting and contending slave state.

- Another task of the present slaves is listening for an advertisement of a master leaving their junction. Upon reception the slave nodes then apply to become the new master. To contest they activate another timer (`LMNM`) with a value corresponding to their sojourn time $d = \frac{t_{current} - t_{entry}}{100}$. The node with the smallest timer value wins the contention and answers with the broadcast of an acknowledge packet destined for the junction it resides in. This ACK packet causes all other slaves to be suppressed, resulting in a cancellation of their pending timers and the master knowing his successor. The master database is now exchanged in unicast mode and the old master is a street member again.

3.1.2 The Street Node (1)

All nodes outside the junction perimeters are street nodes. These nodes employ with standard CBF mechanisms with the sole extension of discarding any packets that are off link, i.e. packets which are not destined for the street they are received on. The procedure upon arrival of a packet runs as follows: (*viewpoint of the deciding street node*)

- ensure that the packet is for my link
- check if a timer for this packet ID is running. If it is, then another node (a competitor) won the contention and this packet and the pending one (same ID) can be dropped, as this packet is from the winner of the contention. It's the next forwarding action.
- if no timer is pending, then calculate the distances from myself to the destination, from the last hop to the destination and from myself to the last hop (*in a close-to-reality scenario it is virtually impossible that two distances are equal*)
  - if my distance to the destination is greater than the distance of the last hop towards the destination, then I am an unsuitable forwarder and should ignore (*drop*) the packet
  - if my distance to the destination is smaller than the distance of the last hop node towards it and my distance from the last hop is inside the radio range, then I am a suitable contender and my progress is weighted, forming my timer delay
  - depending on the handling rules of the algorithm the nodes whose distance is huge enough to be outside radio transmission[1] either take part in the contention taking the risk that their relay does not reach the predecessor leaving him uninformed of the successful transmission or this distance results in the packet being dropped by the node deeming itself as an inappropriate contender
- *as each data packet can have its very own one hop retry counter* - if the retry timer has triggered, decrement the retries, drop the packet if it was the last chance or else measure the progress again and perform the appropriate action

[1]there is no really fixed length for the distance the signal can travel and the range fluctuates

3.1.3 The Junction Master (2)

In a normal traffic scenario with a decent number of participating nodes, the junction master state is the rarest. These nodes are the backbone of the network, they make the decisions about the larger scale routing. The master nodes bear the information about the surrounding neighbourhood of junctions. Because of the complexity of operations to perform, these nodes use a variety of different packets to exert influence and direct the processes in their immediate surrounding. Virtually all packet types have just one hop as their propagation range. Only data packets and link state information packets travel further (*to the next junction or explicit destination*).
A subdivision might be useful to explain the functionality :

- Close range packet types manage procedures with the surrounding slaves
- Junction to junction link state packet types gather information from the neighbouring masters abiding in their junctions
- Data packets travel from master to master till they approach a distance of one junction to their destined destination

Close Range Packets

This group can be subdivided in two categories:

- Junction Advertisement Packets
  These packets are broadcasts originated by the current junction master to inform all possible slaves at the junction of its presence. They are small packets broadcasted when either a slave just became master or when a certain amount of time has passed after the last transmission of a packet, which would reveal the master's presence at this node. This task could as well be handled by a link state request packet, but every link state request packet could entail a quite expensive (*in terms of*

bandwidth usage due to the size of the packet header including numerous link state information from the queried junction) link state reply by the answering node. Link state requests have a special routine to its propagation cycle, which will be professed in detail later. The repetition cycle of the junction advertisement packet ought to be shorter than the linkstate cycle. The danger of consuming too much bandwidth with its propagation is unsubstantiated as the packet is only deployed when no other type of packet that is capable of revealing the master's presence is dispatched. This means that, if no data packet, link state request or link state reply packet is conveyed by the master, there is not really much traffic at all, so triggering a junction advertisement packet doesn't do any harm.
But what is the reason for the need of a small delay for this kind of packet? Any time a slave receives a junction advertisement packet the special timer for this event is rescheduled again. If this timer expires the slave deems itself next qualified master and broadcasts a junction advertisement packet itself. Thus a small delay results in a quick seizure of the junction as master by a slave. This is of particular importance in junctions with less through traffic, which results in only a few nodes sojourning overall. A long wait to recognize that no master is present until the seizure is unwanted, since no effective packet forwarding can take place.
A junction takeover even though a valid master is present may happen if a slave could not evaluate the junction advertisement or any other awareness packet as a result of a packet collision. In this particular case the original master becomes a slave again. *A quick unicast packet exchange requesting the timestamp of the promotion and a resulting agreement on a future master could be contemplated.*
I have recently tried to put this thought to practice and had to realize that this unawareness of a present master by a slave is a major problem if a huge radio range (500m) is used which spans over to another junction. Simply triggering a unicast packet exchanging the age since is has become master (which would result in the old master winning) is

not as easy, because the new pert slave (now master) is quite busy with junction advertisement, and link state packets and will most probably not hear the first request to exchange age information. A way has to be found that not every of these awareness packets from the new master trigger a new request for age comparison. In an awkward case even more than one slave seize the master's throne, resulting in even more packet collisions and difficulties solving the situation. As a possible solution a close range packet has been introduced carrying the master's timestamp[2] in the packet's timestamp field. The receiving other master (a possible pert slave) compares the timestamps and subdues into slave status, if his timestamp is newer. If his timestamp is of older status a return packet delivers his master timestamp. (*It still needs to be discussed if this packet should be send unicast, as it is at the moment, or as a broadcast. The advantage of a broadcast is that all possible masters are questioned, the drawback is a needed partitioning for the answer if many masters are older.*)

- Leaving Master Packets
  This group of packets all involve the order of events if a master should leave the junction. They are occurring in the following order:

  1. A master detects that he left the junction perimeter and has a junction database to hand over back to the junction. The node hence broadcasts a leaving-master-advertisement packet destined for the junction (*the exact strategy concerning packet fields and target handling will be discussed subsequently*) and waits for the resulting contention winner (a slave at the targeted junction) to answer.
  2. This answer packet is the only explicit close-range packet a slave may propagate. The leaving-master-acknowledge packet is a broadcast again to inform all other contending slaves of the ceased contest, which thereupon cancel their pending timers.

[2]timestamp of becoming master at this junction

3. In the last step the master now knows the new master of the junction he just left and transmits a unicast leaving-master-exchange packet to the ID of the promoted node containing all collected link state information including the unchanged timestamps of the entries.

After the transfer the old master deletes his database as his last official act (*his affiliation has been of street node kind already after the transgression of the junction boundaries*)

Link State Packets

MIRP uses two kinds of link state packets, one to query and one for the reply. The request packet is straightforward and consists of the street ID and the destination ID (of the junction) of the target. The creation of both kinds of link state packets is solely in the master's responsibility. The creator's ID is not stored in the MIRP packet header, only the source ID with respect to the junction ID. This way the reply packet can still be used, even if the master at the junction changes till the reply arrives. Once instantiated the packet travels along the street it was destined for. It propagates via the street CBF protocol in greedy progress until it is evaluated at the destination junction. This junction then generates a one-hop database entry for the successful link and creates a link state reply packet. This packet contains the gathered database entries of this node up to n hops. It also stores the corresponding timestamps with the entries and returns the packet with the same MIRP header packet ID as the query. The broadcast of the link state packet also causes the master node to reschedule its advertisement timer, as this packet bears the junction ID as source in the MIRP packet header making the master known to all surrounding slaves, which on their part reschedule the timer. The link state reply packet now finds its way back to the junction on the same way the link state request came from. Upon arrival the retry timer for the link state request is rescheduled and the one hop link state entry is updated in the database with the new timestamp. For the database entries of the neighbouring junction the following procedure is run:

- Find a matching entry for the destination
  - if no entry is found, add the entry to the database keeping the timestamp
  - if a match is found and the timestamp of the existing entry is newer than the received one, ignore the entry
- If the timestamp of the received matching entry is newer and the path length is longer than the existing path length replace the entry only if the existing entry is invalid due to expiration
- If the matching entry is newer and shorter or of same length replace the old entry

Received Data Packets

There are two slightly different kinds of received data packets: Packets which have been handled by a master before and packets which have previously not reached a master node.

- Data packets first arriving at a master node have no valid ID in the last junction ID field. Therefore the master checks other header fields which might help him in his forwarding decision. If the source is a street node, the master remembers not to route the packet back on the link it came from, because either the junction is closer to the destination, in which case the street node made the right decision, or the junction is further away than the source, then this packet was a last chance packet and could not be delivered up the other end of the street anyway. Now the master searches its database for the junction entry being closest to the destination. The path towards this junction is embedded into the MIRP header and the last junction ID field is set to the value of the junction the master abides in.
- As recently described, data packets from another junction have a valid MIRP header field for the source junction containing the junction ID they came from. This information is now used to update the junction

database with the one-hop entry referring to the last junction. Then the rest of the header fields are inspected and depending on the algorithm setup either an existing suggested next path is extracted from the MIRP header field, the last junction ID updated and the packet forwarded again, or otherwise a new search for a destination junction is initiated.

3.1.4 Different Sources of Data Packets

Every node in the scenario can originate data packets while only the master nodes have a superior knowledge of the network. Therefore, if a non-master node generates data packets it needs to come to a decision concerning the routing. A simple flooding till the packet reaches the first master would be easiest but inefficient. Routing the fresh data packets to all neighbouring junctions using CBF was pondered on but rejected. Because the generation of data packets has not yet been discussed, let's take a closer look at the handling in general:

Street Node

Suppose the node is a street node. It knows (via the Omniscient Location Service (OMNILOC) - by asking the General Operations Director (GOD), a uniquely running ns-2 object) the position of the destination. Since it has no database about link states it can only use its knowledge of the map. The node therefore calculates the distances between the two ends of the street it is currently on using the junction coordinates and simply picks the one closer to the destination as next hop with respect to junctions. As the probability of a street node originating data packets in a evenly distributed scenario is higher for the very reason that most of the nodes will be street nodes, a street node as source of data packets is very likely. Regrettably this is the situation where the packets have their most perilous forthcoming apart from the street needed to be taken to reach the destination, as only these two streets are certain. For all the other possible dead-link situations in between, the master junctions could be prepared. Nevertheless the street node made a decision and the packet was sent off. To give some more safety to the delivery the packet's next-hop junction is rerouted to the other connected junction if the packet has its last retry. Possibly nodes exist in this direction which dropped earlier transmissions due to misfit and can now relay the packet to the opposite junction.

Slave Node

In case the node is a slave node at a junction it simply hands the packet over to the master. It could use the same approach as the street node, but on the one hand the junction has probably more connected other junctions to pick from than the two possibilities of the street. On the other hand why should the probably better knowledge of the topology of the master in just one hop range be ignored?

Master Node

The most improbable case is the master node itself originating data packets. After acquiring the destination's positional information the master searches its database for the junction ID closest to these coordinates and stores the path to the chosen junction as well as his junction ID in the MIRP header's corresponding fields. All in all this routine is virtually identical to a master's action when receiving a data packet which has never been handled by a master before.

Selected Packet Header Fields

pmi_ Packet map information field carries the street ID and is used by the street nodes in comparison with their `MID` (Map ID).

pmdst_ Packet map destination is the next junction ID the packet should reach

pmsrc_ Packet map source is the last junction ID the packet came from

src/dst/last Header fields all of type `node_posinfo` carrying information about the nodes on the packet's way

Chapter 4

The Network Simulation Environment

4.1 ns-2

As mentioned earlier the network simulator ns-2 is taken for simulation purposes. The simulator is deduced from the ISO/OSI model. It works on packet level basis, i.e. it creates an object for each packet that is simulated making it ill suited for large scale simulations but very accurate for smaller simulation layouts. Several tutorials exist to guide a novice into the complex structure of ns-2. Two to mention are Marc Greiss' tutorial, an excellent entry into the matter, as well as "NS by Example" by Jae Chung and Mark Claypool.
The ns-2 manual is a useful description for the main functionalities of the simulator. Especially the inner structure of the mobile node with its interconnecting agents, multiplexer and where appropriate demultiplexer is explained.

The figure 4.1 shows the schematic of the inner structure of a mobile node. As all wireless networks share the air as their common medium, the *Address Resolution Protocol* (ARP) as a part of the mobile node in the ns-2 simulation is deactivated, because MIRP is based on broadcasts, as CBF is, too.
(ARP is used to acquire the MAC addresses of comunication partners.)
To set up a routing agent several steps have to be accomplished.

4.1.1 Implementation of a new Routing Agent

First of all ns-2 has to be made aware of the new agent. One of the files that need to be changed is the `cmu-trace.cc` file. This file is responsible for the output of the simulation, the trace file. Its general structure as well as certain output formats are specified here. The programmer may define which information, i.e. in which format, has to be written to the tracefile.
The needed changes are:

- An inclusion of the header file of the new agent, to make the datastructures for the protocol known.
- `CMUTrace::format:` In this method the call to the tracing function is made on the basis of the packet type `ch->ptype()`
- In the `switch` statement an entry needs to be created to call the new tracing function for the agent

The `packet.h` file needs to be edited in the following way:

- In `enum packet_t` a `"PT_`*protocol tag*`"` has to be added, `PT_NTYPE` has to remain the tail entry
- The `p_info class` must be extended by `name_[PT_MIRP]= "`*protocol tag*`";`

The next file that needs adjustment is `ns-packet.tcl`. It can be found in the `/tcl/lib` directory. The *protocol tag* needs to be added to the `foreach prot` loop. To be able to trace the involved packet types in a MAC collision,

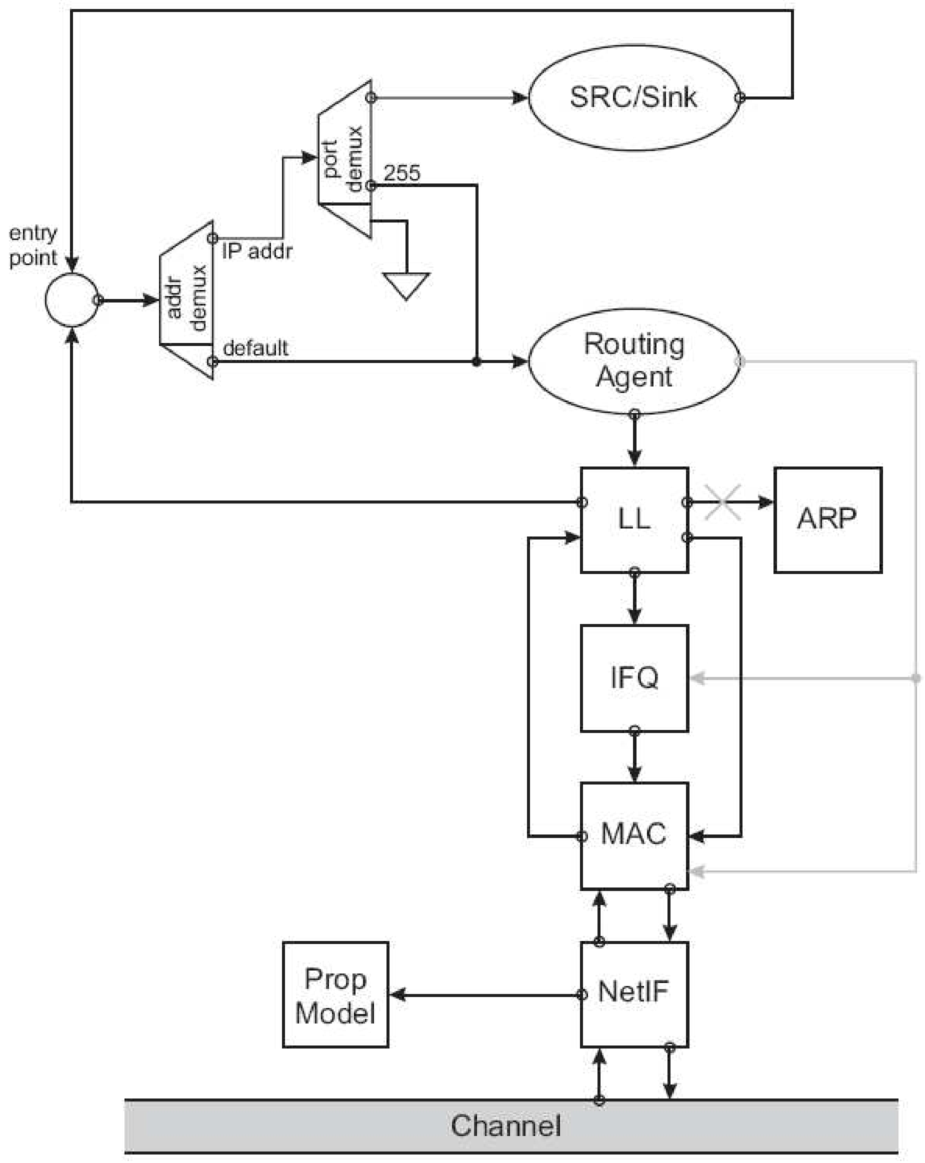

Figure 4.1: Design of a mobile node - taken from CBF by M. Käsemann

`mac-802_11.cc` has to be edited in the `MAC_VERBOSE` section according to the existing entries. To configure ns-2 with the new agent, `Makefile.in` has to be extended with the new agent files.
The main control file for ns-2 is the `run.tcl` file. It provides access to the main simulation variables. To set up a simulation input files consisting of the participating nodes, including their position, their communication patterns and to account for *mobile* ad-hoc networks their movement patterns have to be generated. For better analysis of the algorithm a simple test environment is useful.

4.1.2 The FleetNet Simulation Files

For a promising analysis of a routing agent designed for city scenarios some criteria have to be met. Apart from a suitable scenario layout, referring to the streets where the vehicular nodes travel upon, and a close-to-reality movement of these nodes, it is important to account for possible transmission interferences. The MIRP algorithm ensures that only packets belonging to the street the node is currently moving on are processed by the node. Concerning the scenario layout a district of Berlin, the capital of Germany, has been chosen. The map shows the part of Berlin-Neukölln which was used as a basis for the simulation survey. Only streets with a decent traffic flow have been extracted and processed by the traffic flow simulator Videlio at the DaimlerChrysler AG. The resulting data represent the raw node movement data for my simulations. These files had to be edited and processed further to be compatible with the ns-2 simulation environment, which will be explained in the following section. More information about the origin of the simulator data can be obtained from the diploma thesis of Christian Lochert[10], who acquired these scenario data for his own thesis.

4.1.3 Conversion and Adaption

Of the different simulation files 3 types were processed:

- A numeration of all lanes. A lane is one direction of travel.

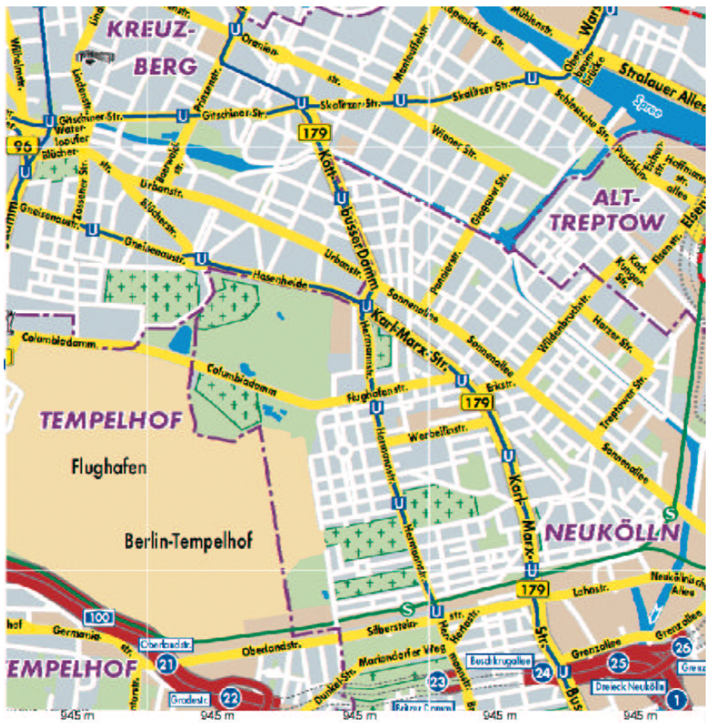

Figure 4.2: Neukölln

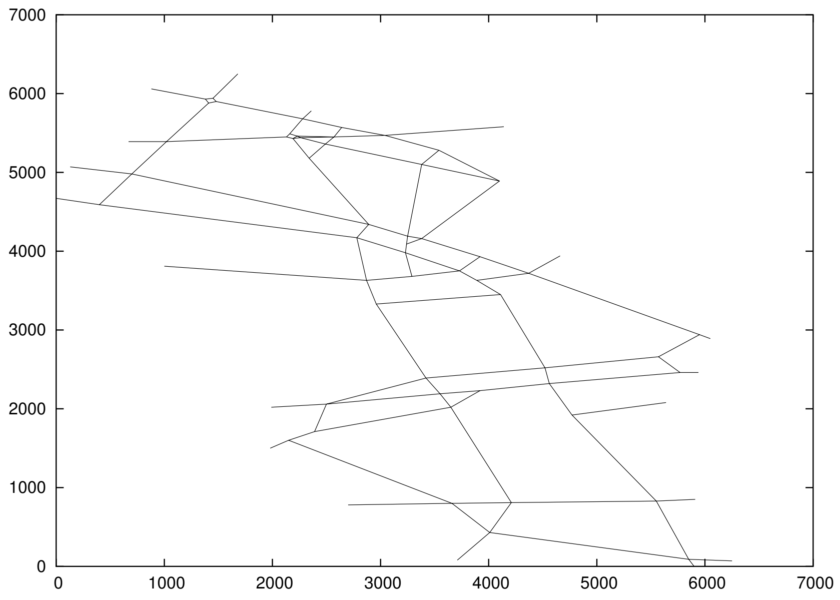

Figure 4.3: Complete streetlayout

ID = 1	lane ID
VMAX = 13.888889	speed limit for the lane
KNOTEN = 1 4	connected junction IDs
ANZSP = 3	number of tracs
LAENGE = 105.000000	length of the lane
ZEIT = 7	unused
AWINKEL = 113	unused
EWINKEL = 113	unused
KLASSE = 1	unused
ENDEKANTE = Kante 1	Lane x end tag

Variables titled unused are of no particular interest but shown for completeness.

- A collection of node (vehicle) movement files numbered from 61-240 and representing the timeslot from 4 o'clock PM to 6 o'clock PM. As an example a short extraction is given to illustrate.

Fleetnet-Simulation 16: 0: 0 bis 16: 0:59 DaimlerChrysler AG 2002					
Uhrzeit	Fzg-ID	K-ID	X-Koordinate	Y-Koordinate	V[km/h]
16: 0: 0	16913	122	13.42710	52.48924	10.0
16: 0: 0	16239	122	13.42562	52.48968	50.0
16: 0: 0	16945	128	13.42610	52.48965	50.0
16: 0: 0	17426	128	13.42507	52.48993	50.0
16: 0: 1	16913	122	13.42701	52.48927	28.0
Time	Veh-ID	K-ID	x coordinate	y coordinate	velocity

- A listing of all junctions in the simulation

ID = 1	ID of the junction
XKOORD = 13457.200000	x coordinate
YKOORD = 52445.900000	y coordinate
TYP = RVL	right of way type: right before left
ANFKANT = 1: 1	number of starting lanes and ID
ENDKANT = 1: 4	number of ending lanes and ID
VORFAHRT = 1 : 0	unused
ABBIEG = 1 : 13	unused
EINBIEG = 1 : 13	unused
ENDE = Knoten 1	ID end tag

ID = 4	ID of the junction
XKOORD = 13456.700000	x coordinate
YKOORD = 52446.800000	y coordinate
TYP = LSA	right of way type: traffic lights
LSANR = 1	number of traffic lights
ANFKANT = 4: 4 5 6 7	number of starting lanes and ID
ENDKANT = 4: 1 2 9 21	number of ending lanes and ID
VORFAHRT = 1 : 0 0 0 0	unused
VORFAHRT = 2 : 0 0 0 0	unused
VORFAHRT = 3 : 0 0 0 0	unused
VORFAHRT = 4 : 0 0 0 0	unused
ABBIEG = 1 : 33 11 33 12	unused
ABBIEG = 2 : 11 11 11 11	unused
ABBIEG = 3 : 11 22 22 22	unused
ABBIEG = 4 : 23 44 11 44	unused
EINBIEG = 1 : 33 11 22 12	unused
EINBIEG = 2 : 33 11 22 11	unused
EINBIEG = 3 : 11 11 22 22	unused
EINBIEG = 4 : 12 11 11 22	unused
AMPEL = 1 : 3 3 3 3	unused
AMPEL = 2 : 2 2 2 2	unused
AMPEL = 3 : 4 4 4 4	unused
AMPEL = 4 : 1 1 1 1	unused
ENDE = Knoten 4	Junction x end tag

Every FleetNet simulation file contains the node movements for a period of 60 seconds.

The first task was to find a suitable way to create streets out of the discrete lanes. An easy way of transformation would be to marry the lane IDs with a certain offset, this was done using perl. But albeit the data at hand does not consist of a one-way street, its occurance is surely possible. The chosen way combines the IDs of the connected junctions with an offset of 100 (the simulation has a maximum of 72 junctions), ensuring the smaller ID always to be the shifted one. Another perl program (`calc.pl`) had to convert the FleetNet simulation files for the simulator which uses OTcl code as input. The nodes (car IDs) were sorted and renumbered to start with 0 and the position information needed to be transformed so that the layout began with the origin.

As it is vital for MIRP to know which node is in which map state, either street or junction (master or slave is decided by the protocol dynamically), the distance of every node to every junction was compared. The chosen junction radius, i.e. the distance in which a node is counted as junction member), is of great importance, since it defines the number of possible contestants to be junction master. If the value of this variable is chosen too small some junctions may be vacant for a longer period of time, disabling any packet routing across it. If on the other hand this variable is set too high many nodes rival to become master, which could result in a higher collision probability, as well as there are less street nodes to relay the packet and finally, the special case of a junction slave receiving a packet while the master is out of range has a higher probability. Worst of all is the effect that the possible master's distance from the junction's centre could be so great that the master is unsuitable to act as a relaying station.

Another factor is the sojourn time of the nodes as members of the junction. A larger value in the radius of the junction directly results in a higher period of association. Although this effect is small, it might result in less junction databases to be transmitted. The `calc.pl` programme therefore calculates the maximum as well as the median sojourn time of all nodes. A statistics of the average number of nodes being junction members is to

be prepared. For each node its activation period is processed, i.e. the time during which it takes part in packet handling and with the aid of the node positions at every second the median velocities for all nodes for each interval are calculated, as these data are needed by ns-2 for the node-movement files. Every node's starting location is set in the scenario file, which furthermore contains a new destination position in x/y coordinates with the calculated velocity in discrete time intervals of one second. Every five seconds this is extended by the coordinates from the FleetNet simulation file to minimize deviation from the source.
The second file created is the active pattern file which contains the commandlines to change the nodes' active states, triggered by the `run.tcl` file. The nodes' active times are also stored in a special `nodes-`*fs-filenumber*`_onoff.txt` file which is evaluated by the `miro.pl` program to find random communication partners. Every time a node enters or leaves a junction is stored in the `mem-`*fs-filenumber*`.tcl` file. The exact times have previously been calculated on the basis of the discrete velocities to get exact entry and exit times. This preparation of the scenario file (setup and movement of nodes) and the active pattern file only needs to be done once.
Here are some snippets of the 3 files:
A movement entry in the `sc-61.tcl` file looks like this:

```
$node_(0) set X_ 2890.0000
$node_(0) set Y_ 4334.0000
$ns_ at 0.0002 "$node_(0) setdest 2881 4337 9.48683"
$ns_ at 1.00000 "$node_(0) setdest 2866 4342 15.81139"
```

The first and second line place the node in the simulator at position 2890x / 4334y. The third line gives the node an initial direction towards the position 2881/4337 and speed of 9.48683 metres per second, about 34 km/h. The last entry refreshes this directional guideline.

```
$ns_ at 0 "changeActiveState 0 0"
$ns_ at 0.0001 "changeActiveState 0 1"
$ns_ at 11.9999 "changeActiveState 0 0"
```

This is an extract from the `on_off-61.tcl` file, representing the activity mapping. The first line is the initialization of the node 0 with 0, i.e. switching it off, followed by the reactivation in line two. Node 0 takes part in the simulation right from the start. Line 3 shows the deactivation of node 0 at the timestamp of 11.9999. From this point onward the nodes do not take part in any more communication in the simulation.
As all occurrences have to be scheduled in the simulator, Christian Lochert took 0.0001 as the increment until the node status was set, 0.0002 for the movement entry and I selected 0.00015 as the increment for the map information setting, just before the movement is triggered. The following lines are from the file `cp-mem-61-0.25.tcl`. It is a merged file from the comunication pattern file, its generation will be explained next, and the membership of the nodes as calculated by the `calc.pl` program.

```
$ns_ at 0.00015 "[$node_(0) set ragent_] junc-member 43"
$ns_ at 0.73787 "[$node_(0) set ragent_] street-member 4346"
```

The first line informs node 0 that it is member of junction 43 from this time onward. From the second line it can be deduced that node 0 changes to the street 4346 at the timestamp 0.73787. It moves for about 11.2 seconds on this street in the direction of junction 46. This member file is now extended by a leading communication pattern.

4.1.4 The MIRO Perl Program

This communication pattern is calculated by the `miro.pl` [1]file. It manages all variable simulation parameters, for example the interval at which the CBR packets are transmitted by their sources. Either a single interval can be assigned by the command line, or the array in the beginning of the file can be edited to accomplish different automated runs with these preset intervals.

Other parameters are the number of runs to be performed, the different routines to process the output tracefile or the number of nodes taking part

[1]As this is the last step before the MIRP computation miro was chosen as the letter o is the direct predecessor of the letter p

in the communication. Certain timeslots where communication should take place or be interrupted can also be assigned in the preset array at the beginning of the perl file.
The possible usage of the NULL-MAC is also implemented and can be triggered via the command line. The NULL-MAC is an implementation by Michael Käsemann in which all packet deliveries are simulated to be transported through wired channels, i.e. undisturbed delivery is assured.

Chapter 5

Implementation of MIRP in ns-2

After the preparations to set up a routing agent which have been explained earlier have been finished, the possible approach to an implementation has to be sophisticated. Somehow the mobile nodes in the simulation have to be alerted to a change in their map status. The three predefined states, which have been introduced before, are street node (1), junction node being slave (0) and junction node being master (2).

5.1 General Operations Director (GOD)

The ns-2 network simulator provides means to inform the nodes whenever they enter or leave a junction. GOD is a unique object in the ns-2 hierarchy. It may only run in a single instance and is thereby perfect for all simulation specific operations. GOD will be used to signal the nodes of their map status changes and inform a querying node of the positional information of itself and acting as omniscient location service distributing the coordinates of a requested destination node. Apart from the geographical information the MIRP algorithm needs some knowledge about the map. The nodes need layout information of the streets and junctions.
For this purpose a database has been built up containing the map based

neighbourhood information listed below. Starting with the junction IDs, each junction entry looks like this:

junction ID	X	Y	connected street ID	leading to junc. ID	X	Y

Any additional street - junction pairs are simply appended. For easier access each entry is 4 digits long with a blank in between.
The street entries follow after the last junction entry. All street entries have a fixed length of 7. Their IDs are by setup all greater than 100 and are arranged in the following order:

street ID	junction ID	X	Y	second junction ID	X	Y

This `MDB.txt` file is read by a function implemented in GOD during the setup process of the simulation. The database is stored in arrays, these are accessed with a street or junction ID named Map ID (MID) in `god.cc` which returns the corresponding data (in form of a pointer), the first entry `MDB[MID][0]` containing the number of entries (for every MID >100 this [0] == 6, as it belongs to a street). The junction entries consist of `6 + (x * 4)` entries, x being any extra trailing street. This grants all mobile nodes access to the map-based information.
An example is a street node receiving data packets from a connected CBR Agent. Thus the street node needs to know the IDs of the junctions connected to the street it is moving on as well as the corresponding coordinates to be able to calculate the distances of each junction to the destination the CBR packets need to be routed to. The junction closer to the destination is the logical first choice.

```
void MIRPAgent::genlaunchDpkt(Packet *p) {
  int* DBptr;
  hdr_ip *iph = hdr_ip::access(p);
  hdr_cmn *cmnh = hdr_cmn::access(p);
  hdr_mirp *mirph = hdr_mirp::access(p);
  \\ getting x,y coordinates of the destination node
  mn_ = God::instance()->nodelist[mirph->dst.id];
  mn_->getLoc(&mirph->dst.x,&mirph->dst.y,&mirph->dst.z);
  mn_ = God::instance()->nodelist[mirph->src.id];

  DBptr = God::instance()->getDBentry(mid);
  int items = DBptr[0]; // entry point to the junction database
  junc_info dsta, dstb;
  int ja,jb, jx;
  double disA, disB;
  if (ms_ == 2){ // the master queries his junction DB
    findJdst(p); // check JDBv to find a suitable junctiondst
    pdr.sent ++; // allow a simple pdr output at sim end
    return;
  }
  else if (mid < 100) { // node is in junction(slave)
   //  send to its master (better knowledge!)
    mirph->pmdst_ = mid;
  }
  else { // node is on street find junction closer to dest
    ja = DBptr[1];
    jb = DBptr[4];
    dsta.id = ja;
    dsta.x = DBptr[2];
    dsta.y = DBptr[3];
    dstb.id = jb;
    dstb.x = DBptr[5];
    dstb.y = DBptr[6];
    disA = Distance(mirph->dst, dsta);
    disB = Distance(mirph->dst, dstb);
    if (disA < disB) {
      mirph->pmdst_ = ja;
    } else {
      mirph->pmdst_ = jb;
    }
  }
  iph->sport() = RT_PORT;
  iph->dport() = RT_PORT;
  cmnh->next_hop_ = MAC_BROADCAST;
  cmnh->direction() = hdr_cmn::DOWN;
  double aj = MIRP_JITTER;
  Scheduler::instance().schedule(target_,p,aj);
  pdr.sent ++;
  // Remember the Packet as processed
  struct seqnoentry tmpfrw = {
    cmnh->uid(), Scheduler::instance().clock(), cmnh->uid()
  };
  processingcache->add(&tmpfrw);
  return;
}
```

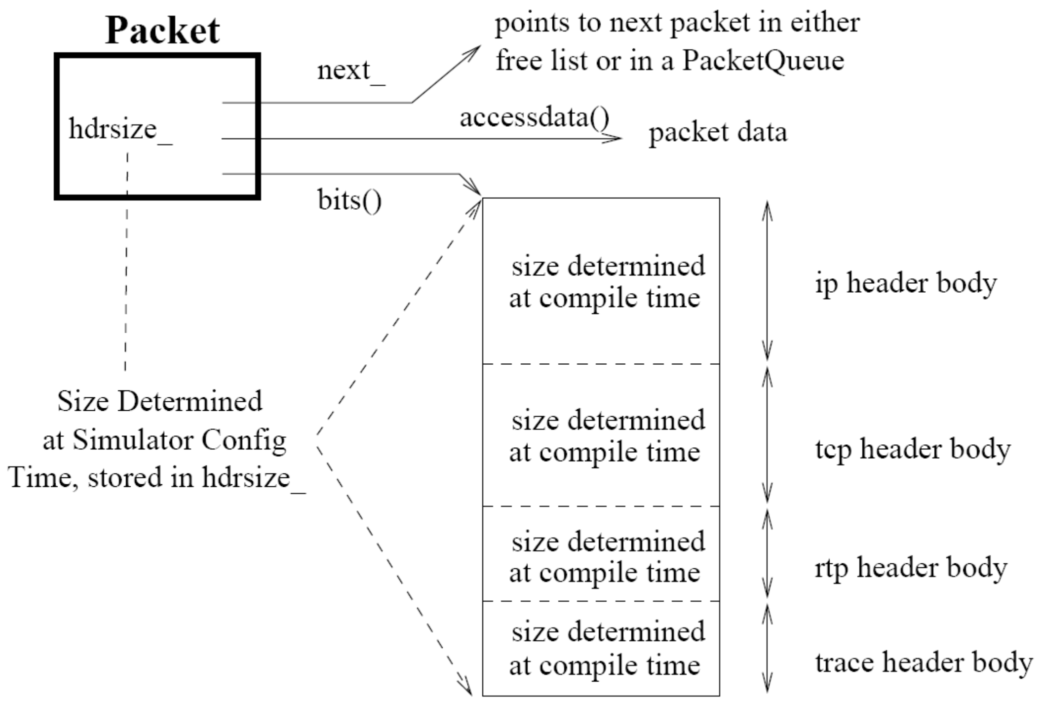
Packet
hdrsize_
next_
points to next packet in either free list or in a PacketQueue
accessdata()
packet data
bits()
size determined at compile time
ip header body
size determined at compile time
tcp header body
size determined at compile time
rtp header body
size determined at compile time
trace header body
Size Determined at Simulator Config Time, stored in hdrsize_

Figure 5.1: ns-2 Header Assembly

5.2 The MIRP Header

After the implementation of the street and junction database has been explained, it is important to understand the different header fields MIRP uses. As this figure from the ns-2 documentation shows the header size of all packets is calculated during simulation configuration time. The simulater is told to remove all possible header fields in order to reduce overhead and is then configured with only the few header fields needed for a certain simulation.

5.2.1 Packet Types

The MIRP header itself consists of several possible types of packets which are listed below for a better survey:

ID	**MIRP**	**Header Fields**
0	MIRP_DATA	Standard Data Packet
1	MIRP_LSREQ	Link State Request (M to Link)
2	MIRP_LSREP	Link State Reply (M to Link)
3	MIRP_MatJ	Propagating present Master at Junction (M to Junction)
4	MIRP_LMADV	Leaving Master Advertisement (M to new M / M broadcast)
5	MIRP_LMACK	L M Acknowledge (new M to leaving M broadcast!)
6	MIRP_LMXP	L M Exchange Packet (leaving M unicast to new M)
7	MIRP_RADV	Receive Advertisement (Destination to contenders)
8	MIRP_PMRP	Present Master Recall Packet (MwDB to nM (pert S))

0 `MIRP_DATA` Data Packets - This type contains only agent originated packets. They are routed partly by the source node, which chooses the first master for further relaying of the packet. The main routing is then done by the master nodes in the junctions, until finally the packet reaches the street or junction the destination node resides in. Data packets can be considered as awareness packets if relayed by a master. Only a master sets the `pmsrc_` entry in the header field to signal that and where this packet was touched by a master. Any receiving slave node in the same junction as `pmsrc_`, the masters junction, reschedules the MatJTimer.

1 `MIRP_LSREQ` Link state request packets are only sent by the masters residing in the junction. They are set to a certain link (street) by the pmi_ (packet map information) field. Their job is to detect the state of the link and trigger a link state reply packet if reaching the opposite junction. Link state requests are awareness packets due to the usage of the `pmsrc_` field.

2 `MIRP_LSREP` Link state reply packets are direct answers to link state request packets sent from the receiving master. They contain the junction database entries stored in the `jdbv` vector database. Entries with a path length equal to the maximum path length are not included, since these would be ignored at the receiving destination due to excess length. Because the `pmsrc_` header field is used to indicate the source junction of the link state reply, the packet is an awareness packet to surrounding slaves.

3 `MIRP_MatJ` Master at Junction type packets are the original awareness packets for the slaves residing in the junctions. This packet type is triggered in a fixed interval after the last other packet which indicates the master's presence has been transmitted.

4 `MIRP_LMADV` Leaving Master Advertisement packets are transmitted by a junction master, which has just been informed from OTcl that the node is now a street member. The intention is to perform the handover of the junction database which the old master is still carrying to the new master. The residing slaves therefore contend with each other, initiating a timer with the delay corresponding to their sojourn time so far. This method has been chosen as the probability is higher that a newly arrived slave has a longer future sojourn than an *older* one.

5 `MIRP_LMACK` The Leaving Master Acknowledge packet is sent by the slave node which has won the contention and has therefore had the smallest sojourn time. It is a broadcast transmission to inform all other contenders of the end of competition and the new master at the junction.

6 `MIRP_LMXP` The Leaving Master Exchange packet finally is a unicast packet containing the junction database routing table entries. It is transmitted from the old master to the new master upon receiving of the `MIRP_LMACK` packet.

7 `MIRP_RADV` Receive Advertisement packets are in general the confirmation packets of the last hop at a contention/suppression queue. Because this last hop receipt has not been made aware, either due to a destination reached or a master node picking a new link for further propagation, the `MIRP_RADV` packet is broadcasted. All nodes who are still contending will now end their running timers and drop the packets.

8 `MIRP_PMRP` The Present Master Recall Packet is an effort to subdue any pert slave nodes which have been unable to register the master's presence owing to packet collisions. As described above, this part is not an easy task. Several attempts have been initiated to prevent the old master from sending lots of these `MIRP_PMRP` packets, one for every awareness packet, which can quickly sum up to a decent amount. Due to the many tasks a new master has to do it is virtually impossible to get its attention. Sending a packet unicast to the master probably results in a packet collision, triggering a retry soon. A preventative measure was the embedding of the junction node master's timestamp (*the time since it became master*) into the `MIRP_MatJ` packet. Any pert slave which became master could detect its inadequacy this way and step down, leaving the older ones with the difficult task of informing the active but newer master. That means a veteran master will not be reduced but has problems informing the eager upstart to subdue.

5.2.2 Important MIRP Header fields

As some general packet header fields have already been introduced, the other variables worth mentioning are the following:

MIRP_TTL The range of the maximum hop count resembles the *Time To Live*(TTL), which is a value decremented at each hop and compared to zero. If the hop count equals the TTL the packet is discarded. It is not a real MIRP Header field, as it is taken from the IP header.

pid_ *Packet ID* used for certain return detections

src/dst/last These entries are all of the `node_posinfo` structure and like these have a node ID of address type, x,y and z position as well as a MID.

pmdst_ This field holds the *packet map destination*, which is the ID of a destination junction.

pmsrc_ The *packet map source field* to the last entry, containing a junction ID, too.

pmi_ The *packet map information* refers to the street ID on the city map.

pts_ The *packet time stamp* is used for transmission time calculations and misused by the PMRP type to transport the master timestamps in order to distinguish between the master age or valence.

jrte_c The *junction routing table entry count* refers to the number of entries contained in the `jrte[]`.

jrte[ ] The *junction routing table entry* contains an array which is of the structure `lshe` and consists of the junction ID, the path length, the timestamp of the entry and the path in form of an array.

5.3 The MIRP Timers

Just before plotting the source code in UML form the timer handlers will be explained.

MatJTimer The *Master at Junction Timer* is a timer concerning all junction nodes. At the instance of a `junction-member` OTcl trigger the addressed node is initiating the MatJTimer with a delay named *Master at Junction Propagation Time* `MIRP_MatJ_PT`. Its duration is composed of a fixed value and a randomly chosen part. Every received awareness packet triggers a reschedule of this timer for the slave. The master does not reschedule the timer but every time it expires, it is reinitiated with a fixed reduced delay to favour the master node.

LMNM_Timer The *Leaving Master New Master Timer* is responsible for selecting a new suitable master when the recent one moved on. Upon hearing the `MIRP_LMADV` packet all contending slaves in the targeted junction schedule this timer with a delay corresponding to their sojourn time. The latest arrival wins the contest and broadcasts a `MIRP_LMACK` packet.

LSRC_Timer The *Link State Refresh Cycle Timer* is a timer born of the attempt to reduce bandwidth usage. Formerly every MajJ Timer expiry triggered an advertisement packet as well as a link state request for each trailing street. The `LSRC_Timer` in cooperation with the *Link State Flag Vector* `LSFV` reduces this bandwidth usage. The `LSFV` remembers each link which could not be confirmed to be **up** and after a `MatJTimer` timeout only each unconfirmed link is queried by a new link state request packet. The `LSRC_Timer` ensures a *forced* link state request for every street to receive a fresh `MIRP_LSREP` packet containing further pursued link states.

contentiontimer The *contentiontimer* postpones the sending of a packet according to its delay. The handler is memorized to be able to delete the packet if the contention has been won by someone else.

dataretrytimer The *dataretrytimer* delays a copy of the transmitted packet until either a received confirmation causes the deletion of the copy via the stored handle, or the copy is sent off as a retry upon expiry. A retry counter is incremented till the value of the `MIRP_MAX_RETRY_ATTEMPTS` has been reached.

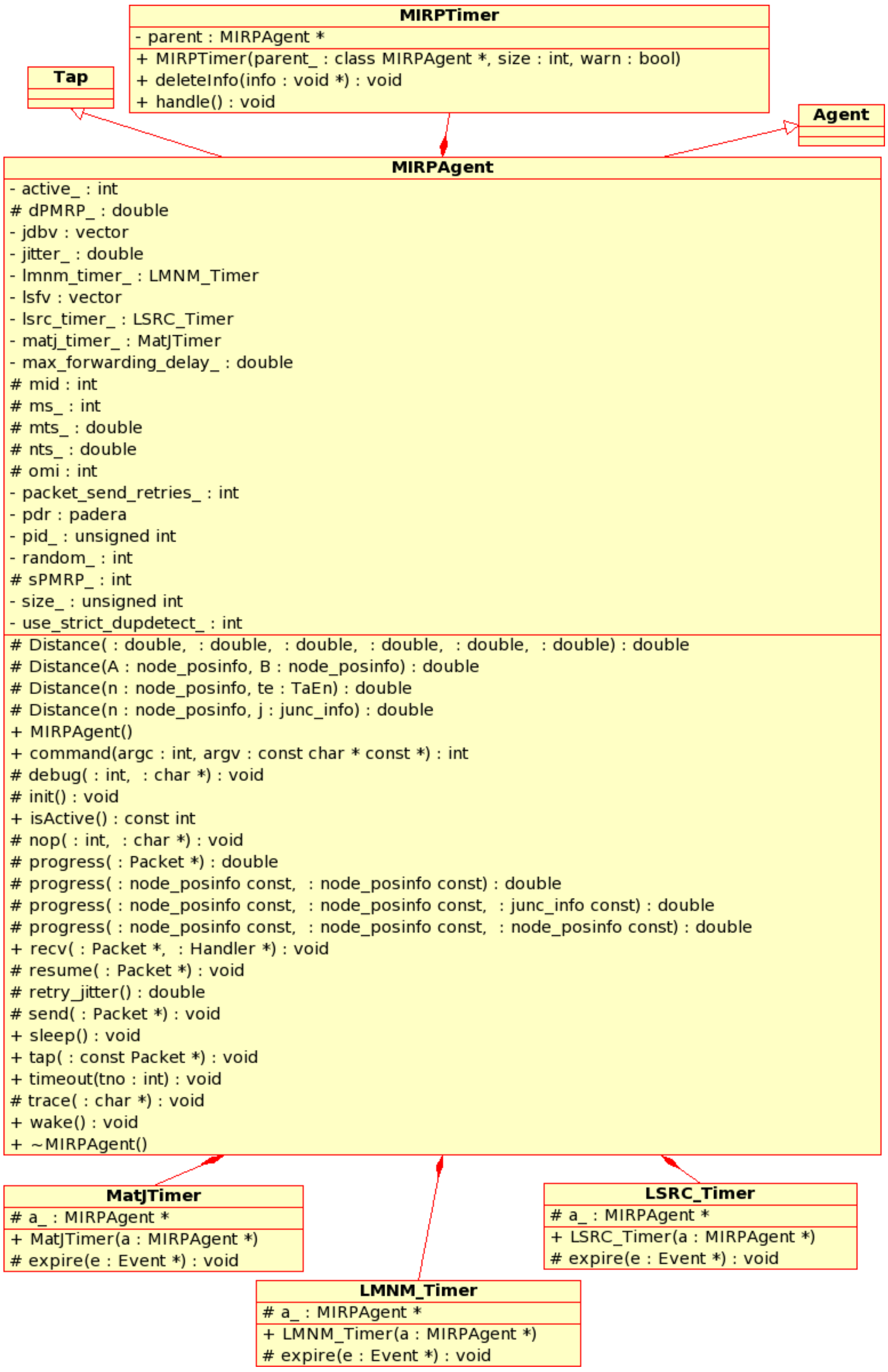
MIRPTimer
- parent : MIRPAgent *
+ MIRPTimer(parent_ : class MIRPAgent *, size : int, warn : bool)
+ deleteInfo(info : void *) : void
+ handle() : void
Tap
Agent
MIRPAgent
- active_ : int
# dPMRP_ : double
- jdbv : vector
- jitter_ : double
- lmnm_timer_ : LMNM_Timer
- lsfv : vector
- lsrc_timer_ : LSRC_Timer
- matj_timer_ : MatJTimer
- max_forwarding_delay_ : double
# mid : int
# ms_ : int
# mts_ : double
# nts_ : double
# omi : int
- packet_send_retries_ : int
- pdr : padera
- pid_ : unsigned int
- random_ : int
# sPMRP_ : int
- size_ : unsigned int
- use_strict_dupdetect_ : int
# Distance(: double, : double, : double, : double, : double, : double) : double
# Distance(A : node_posinfo, B : node_posinfo) : double
# Distance(n : node_posinfo, te : TaEn) : double
# Distance(n : node_posinfo, j : junc_info) : double
+ MIRPAgent()
+ command(argc : int, argv : const char * const *) : int
# debug(: int, : char *) : void
# init() : void
+ isActive() : const int
# nop(: int, : char *) : void
# progress(: Packet *) : double
# progress(: node_posinfo const, : node_posinfo const) : double
# progress(: node_posinfo const, : node_posinfo const, : junc_info const) : double
# progress(: node_posinfo const, : node_posinfo const, : node_posinfo const) : double
+ recv(: Packet *, : Handler *) : void
# resume(: Packet *) : void
# retry_jitter() : double
# send(: Packet *) : void
+ sleep() : void
+ tap(: const Packet *) : void
+ timeout(tno : int) : void
# trace(: char *) : void
+ wake() : void
+ ~MIRPAgent()
MatJTimer
# a_ : MIRPAgent *
+ MatJTimer(a : MIRPAgent *)
# expire(e : Event *) : void
LSRC_Timer
# a_ : MIRPAgent *
+ LSRC_Timer(a : MIRPAgent *)
# expire(e : Event *) : void
LMNM_Timer
# a_ : MIRPAgent *
+ LMNM_Timer(a : MIRPAgent *)
# expire(e : Event *) : void

Figure 5.2: UML-Class diagram

5.3.1 Data-Dictionary

Class:*MIRPTimer*
The MIRPTimer class is a queued timer. Two instances, contentiontimer and dataretrytimer, belong to this class. Their tasks have just been explained.

attribute	**description**
MIRPAgent *parent_;	pointer to the MIRPAgent to which the timer belongs

Class:*MatJTimer, LMNM_Timer, LSRC_Timer*
The MatJTimer, LMNM_Timer and LSRC_Timer classes inherit from `TimerHandler`. Their sole function *expire* is triggered and hands over a certain flag to identify the type of timeout in the corresponding timeout function.

attribute	**description**
MIRPAgent *a_;	pointer to the MIRPAgent to which the timer belongs
function	**description**
void expire(Event *e)	event handling, further details are given in time-out function

Class:*MIRPAgent*
The class MIRPAgent is the main class of the agent. The most important entries will be described below.

attribute	**description**
vector <TaEn>jbdv;	container for the junction database entries
vector <TaEn>lsfv;	container for the link state flag vector
padera pdr;	packet delivery analysis for possible output at simulation end
int active_;	node status (wake / sleep) flag - an inactive node does not take part in the routing process
unsigned int pid_;	the packet identifier for connections
double nts_;	node time stamp at junction to calculate the sojourn
double mts_;	master time stamp at junction to decide which master is older
int omi;	old map info, i.e. the last map ID the node came from
int mid;	map ID i.e. the current map ID of the node lately used for streets only
int ms_;	the map status - a slave node has the value 0, the street nodes value is 1 and the master uses the 2
int sPMRP;	sent *present master recall packet* - this flag is only used to prevent the receiving master from replying to every awareness packet, which a pert master in his junction sent out and thus reduce collisions
double dPMRP;	delay *present master recall packet* - in combination it blocks the transmission of `PMRP` packets for a certain amount of time
int packet_send_retries_;	this retry value is set from the command line and has a default entry
double max_forwarding_delay_;	this delay value is set from the command line and has a default entry
int use_strict_dupdetect_;	this boolean value is set from the command line and controls the handling of a duplicate detection

function	**description**
int command(int argc, const char* const* argv);	the command function represents the interface to the OTcl environment. Apart from the initialization of the nodes and the padera trigger for the packet delivery statistic in the end the `MIRPAgent` mainly uses the street-member and junc-member parts of the command function to inform the nodes of a change in the MID, i.e. when a node moves from street to junction or vice versa
void wake();	this function wakes the node whenever the OTcl procedure *proc changeActiveState* {*nId on*} is called
void sleep();	this function puts the node to sleep whenever the OTcl procedure *proc changeActiveState* {*nId off*} is called
void tap(const Packet*);	this function taps the passing traffic and is used by the MIRPAgent only to trigger a `MIRP_RADV` packet for each received packet destined for this node
void init();	this is the initialization function which sets most variables to starting values
void resume(Packet*)	the resume function handles expired contention or retry timer events. In case of a contention packet the actual progress made is recalculated
void send(Packet*);	the send function takes care of the relayed packet from the resume function. It either discards the packet after checking the number of retries used or relays the packet incrementing the retry counter and memorizing the processing of the packet

void timeout(int tno);	the timeout function provides procedures if one of the three timer handler expires. A `MatJ` timeout leads to an immediately transmitted `MIRP_MatJ` type packet to the junction ID the node is member of. Thereafter link state packets are relayed to all connected links (this information is extracted from the Map Database (MDB) `GOD` administers). The LSRC_Timer is started while the lsfv is filled with entries for all existing adjacent streets and the MatJ-Timer is rescheduled with a certain benefit. A `LMNM_Timer` timeout crowns the node to be new master. A `MIRP_LMACK` packet is assembled and dispatched. Finally the `LSRC` timeout triggers a forced link state query of all connected streets to accumulate new link state information of distant junctions. For this purpose the `MatJ` function is called and the `LSRC_Timer` rescheduled.
void Streetchange(int);	the Streetchange function cancels all pending timers (the MatJTimer and in case of a master node the LSRC_Timer as well), hands over the new MID, sets the map status (ms_) and if the node was a master it transmits a `MIRP_LMADV` packet to the old junction ID and invalidates the master timestamp (mts_)
void Junchange(int);	the Junchange function initializes the `MatJ_Timer` and the node timestamp (nts_) and sets the map status (ms_) to slave (0)
void recvLMADV(Packet *p);	the reception of a leaving master advertisement reschedules the MatJTimer and starts the `LMNM_Timer` with a delay corresponding to $$delay = \frac{actualtime - junctionentrytime}{100} \quad (5.1)$$

void recvLMXP(Packet *p);	the reception of a leaving master exchange packet is always unicast from the old master. The new master calls the getjuncDB function.
void recvLMACK(Packet *p);	the receiving of a leaving master acknowledge packet cancels the `LMNM_Timer` that is pending in every slave in this junction and prompts the old master to generate a `MIRP_LMXP` packet containing the junction database and transmit it in unicast fashion to the sender of the `LMACK` packet
void recvMatJ(Packet *p);	the reception of a master at junction packet by a slave in the junction causes the `MatJTimer` to be rescheduled. A master receiving this kind of packet becomes aware of another master in his junction. As a reaction the master generates a `MIRP_PMRP` packet, puts his `mts_` in the `pts_` field of the header and transmits it to the originator of the `MatJ` packet as unicast. In the new version of the `MatJ` packet the `pts_` header field correlates with the `mts_` of the sender, thus before sending the `MIRP_PMRP` packet, the master checks if the sender has an older timestamp and is therefore more privileged foregoing the `PMRP` transmission.
void recvPMRP[Packet *p);	the arrival of a present master recall packet prompts the receiving master (unicast) to compare the own `mts_` with the received `mts_` from the `mirph->pts_` field. If the originating master has a smaller timestamp he is the regular master and the receiver will cancel the `LSRC_Timer`, reschedule the `MatJTimer` and set the `ms_` to slave (0). If the receiver has an older timestamp he will retransmit a `MIRP_PMRP` type packet via unicast to the sender with his own `mts_` in the header.

void sendRADV(const Packet *p);	this function is called by the tap function upon discovery of a data packet reaching the final destination. As the packet is consumed a notification of the receipt has to be transmitted to the sender and all possible contenders. This is done by the receive advertisement packet. It is a normal broadcast packet and the MIRP header packet ID entry is set to the packet ID of the received common header.
void newMatJ(void);	the new master at junction function sends a link state request to all adjacent streets
void findJdst(Packet *p);	the find junction destination function is only invoked by a master. It searches the master's junction database for the junction entry which is closest to the coordinates of the data packet's destination. The result is stored in the header and the packet dispatched.
void getjuncDB(Packet *p);	the get junction database routine is called from the `recvLMXP` function. The receiving master reads all Junction Routing Table Entries (JRTE) and stores them in the newly created junction database.
void genlsreq(int link, junc_info const dstj, junc_info const srcj);	the generate link state request function is a helping routine to the newMatJ function. It just creates link state requests for the entries which have been handed over.

void genlaunchDpkt(Packet *p);	the generate launch data packet function uses a sort of omniloc by questioning GOD for the positional information of the destination. If the actual node is a slave, the packet is directed to its junction master. If it is a master the function findJdst(p) is invoked and if the node is a street node, the MDB is questioned and of the two junctions at the ends of the street the one closer to the destination is selected (by putting the junction ID into the `mirph->pmdst_`s field) and the packet is launched.

void recv(Packet *p, Handler*);	The receive function is the main element of the MIRPAgent. All incoming packets are introduced here. The very first task is to verify the right to take part in the communication and analysis. If the node is not awake the packet is dropped here. The next step is to access the packet header entries. This is done by the following code:
	
	 `struct hdr_mirp* mirph = HDR_MIRP(p);`
	 `struct hdr_ip* iph = HDR_IP(p);`
	 `struct hdr_cmn* cmnh = HDR_CMN(p);`
	
	 The IP header source and destination address are now stored and the common header transmission failure fields evaluated with respect to unicast one hop transmission errors. The first check is for a newly arrived agent packet from the upper layer. In the simulation a CBR agent is generating the data traffic with a 100 bit packet size. The CBR agent is connected to an UDP agent which is relaying the packets to a NULL agent connected at the destination node.

void recv(Packet *p, Handler*); *continued*	Upon arrival of a new data packet the MIRP header fields have to be initialized, the TTL in the IP header field is set to the `MIRP_TTL` which is 32 at the moment. The `mirph->src` header entries are filled with the mobile node address as well as the MID and the routing port to the UDP agent is stored in the appropriate field `mirph->origport`. As this is unusual it will be explained now. Normally the IP header destination port stays unchanged, which results in a direct delivery to the UDP agent once the packet reaches its destined location. The MIRPAgent at the destination node is not getting access to this packet unless the tap is used. The current implementation is a workaround enabling the MIRPAgent to completely renounce any usage of the tap. The momentary usage to trigger the `MIRP_RADV` packets in the tap function could likewise be done in the `recv()` function but is kept this way to show the feasibility. Next the `mirph->dst.id` is filled with the `iph->daddr()`, the MIRP packet ID is matched with the common header unique ID, the packet map information is set to match the MID of the current node and the current simulation time is stored into the MIRP header packet time stamp `mirph->pts_` field. Finally the packet is handed over to the generate launch data packet function `genlaunchDpkt(p)` which has already been explained in detail. The next action, done with the left over packet possibilities, is to decrement the TTL and drop any expired packets. Now the packet retries are reset to 0 and a switch statement delivers most of the packet types to special functions which all have been explained earlier on.

void
recv(Packet *p, Handler*);
continued

The only routine not yet explained is the handling of the `MIRP_RADV` packet reception which is still inside the main receive function and will now be discussed shortly. Upon arrival of a `MIRP_RADV` packet the data retry timer and in succession the contention timer will be searched for an entry of the MIRP header `pid_` which is equivalent to the common header unique ID (`cmnh->uid()`) of the data packet. Possible contending entries are deleted and the packet is marked as being processed.
At this point it is verified if the packet is a data-packet and if it is destined for this node. In this case a duplicate detection on the basis of the common header `uid_` is made. A possible duplicate is discarded, otherwise the `uid_` is added to the received cache to detect trailing duplicates, the `pdr.recv` counter is incremented and the data packet dispatched to the UDP agent via the IP destination port, the entry taken from the `mirph->origport` field.
All remaining packets are either link state packets or data packets for a different destination node.

ms_ == 1

Assuming the receiving node is a street node the receive procedure drops every packet that is not destined for the street ID the node is currently travelling upon. It is a simple match of the MID with the `mirph->pmi_` header field. If the packet is not dropped the CBF contention or suppression scheme begins. If the contention timer for this packet ID is not pending a weight calculation is made based on the distances of the source to the destination, the node to the source and the node to the destination.

void recv(Packet *p, Handler*); *continued*	If the node is a valid contender, which means it makes positive progress in the direction of the destination, the contention timer is activated with the delay corresponding to the progress. This has been explained earlier. In case a contention timer is already running the receiving of this packet indicates a better suited contender has forwarded this packet. This results in a removal of the `uid_` from the contention timer, an added entry of this value to the processing-cache and a discard of the packet.
ms_ == 0	In case of the receiving node being a slave the handling of the packet is rather short. If the packet map destination (`mirph->pmdst_`) matches the junction the slave resides in and no contention timer active it is added to the contention timer with the delay equal to the diameter of the junction[1] divided by the radio range. Thus a possible master in the junction is not disturbed in the relaying of the packet if the slave's distance would be a perfect match and result in an instant transmission. If a contention timer is active, the `uid_` is removed from the timer, remembered in the processing cache and the packet dropped.
ms_ == 2	In the last case the node is master of the junction. If the received packet is destinated for a different junction, indicated by the `mirph->pmdst_` entry not matching the MID of the node, the packet is discarded. If the `mirph->pmsrc_` field is matched by the MID value, then the packet is originated from this junction. If it is not from the master then there is another master in the junction and a `MIRP_PMRP` type packet is sent to the originator in unicast fashion while the received packet is dropped.

void recv(Packet *p, Handler*); *continued*	A variable delay to the sending of the `PMRP` has been added to avoid increasing traffic. The receive function now switches according to the packet type:
LSREQ	A link state request renews or adds an entry for the originating junction (`mirph->pmscr_`) to the junction database the master is fostering. Additionally the receiving master generates a link state reply containing all junction database entries of valid length[2] which do not contain a path entry equal to the connecting street (link).
LSREP	A link state reply results in an update of the `jdbv` for the one hop link, too. Furthermore all entries from the received reply are evaluated and the junction database is updated accordingly.
DATA	The final possibility is the reception of a data packet. After verifying if the packet is from a surrounding junction (master) a one hop link state update can be made if true, otherwise the packet is from a street node or a slave. After the last hop and successor information have been updated in the header fields the packet is handed over to the findJdst() function for further handling.

Chapter 6

Results

6.1 Node placement

In the execution of the simulation only a certain part[1] of the scenario layout has been used. The FleetNet simulation file number 61, which has been used by Christian Lochert previously contains only node movement data for this small part of the map layout. It has been recently discovered that two different simulation scales from Daimler Chrysler existed and unfortunately the results could therefore not qualitatively be compared. My scenario data consisted of 348 different vehicular nodes. To visualize the movement a plot has been assembled picturing all nodes during every second of the simulation. The small green line in the centre of the plot indicates the existence of an unused street. Either no traffic in the specific time period from 16:00:00 - 16:00:59 was measured on this street or it has been later removed from the files. In any case it has no influence on the simulation due to the length of the street. If a master node in junction number 39, which is the left end of the street, would send a link state packet the master in junction 40, being the right end of the green line, would receive the transmission in one direct hop, and the answer would be transmit in the same way. The plot suggests a completely covered simulation layout, but taking a closer look a different situation is shown:

[1]as denoted in figure 6.1 by the slashed line

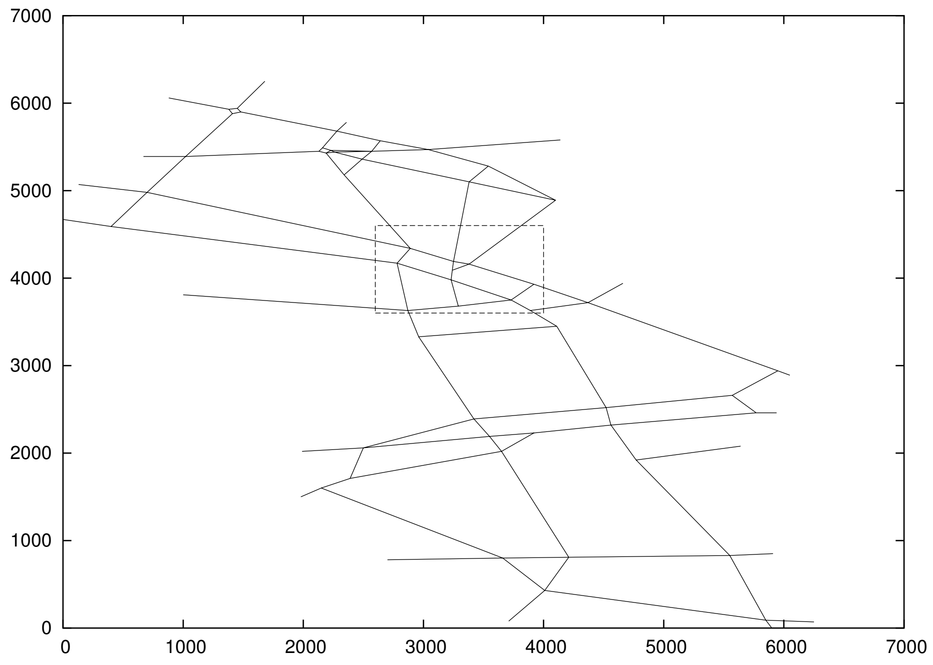

Figure 6.1: Visualization of the simulation area

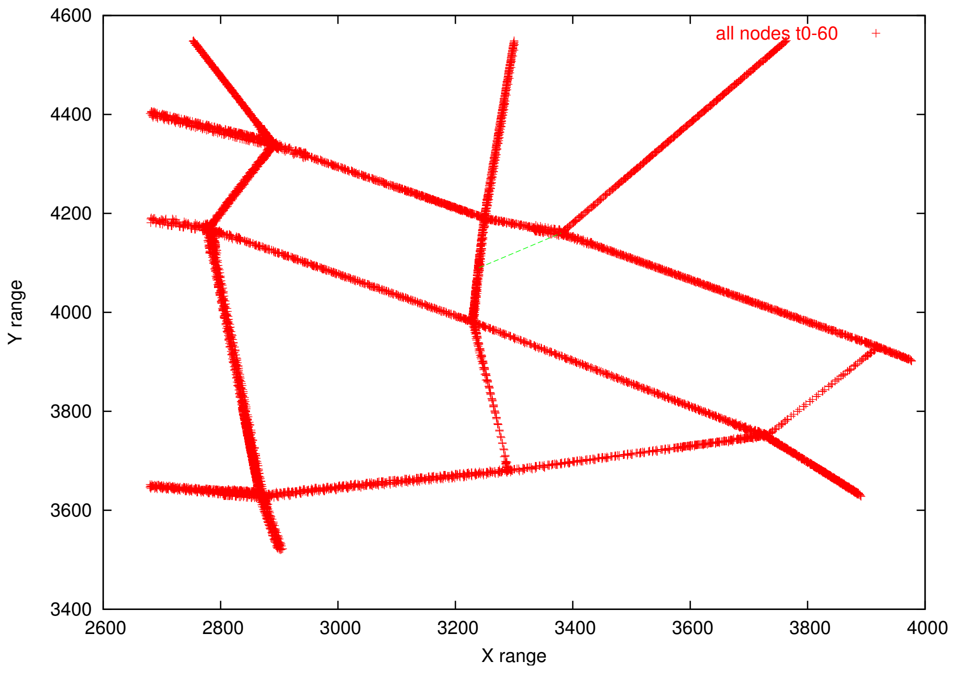

Figure 6.2: Location of all nodes from file 61 at each of the 60 seconds

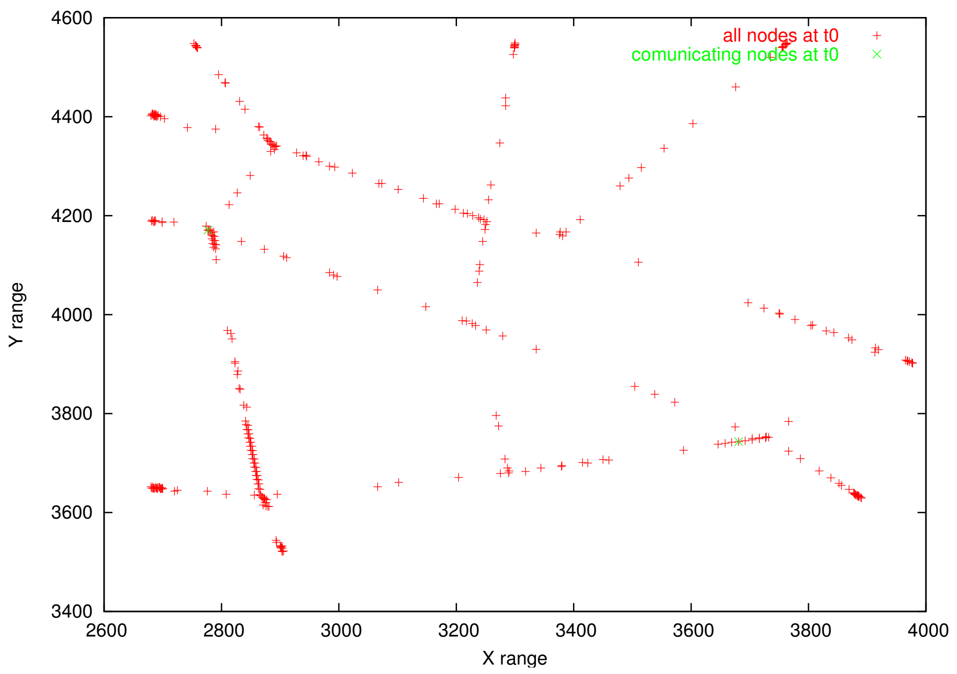

Figure 6.3: Location of all nodes from file 61 at `t=0`

In this figure only the active nodes at the very beginning of the simulation are plotted. The nodes are partly scattered and especially on the bottom roads the distance between some of them is larger than a normal radio range in the ns-2 simulation, which is 250 metres. All communicating nodes have a distance of 1000 metres ($\pm$*20 metres*). The packets are generated at an interval of 0.25, which is equal to 4 packets in a second, resulting in 100 packets send per communicating pair of nodes. As the first communication pattern is set to have a communication period from second 2 till the 27th second and a second communication time from the 30th second onward until the 55th second of the simulation the node layout at the starting of the second data transfer is shown, too.

The node layout here is not showing a better distribution, it is in some parts even worse. As my predecessor has decided to increase the radio range to 500 metres most of my simulations ran with this range too. The measurements of wireless transmissions in the real world even transported packets over a distance of 800 metres with success.

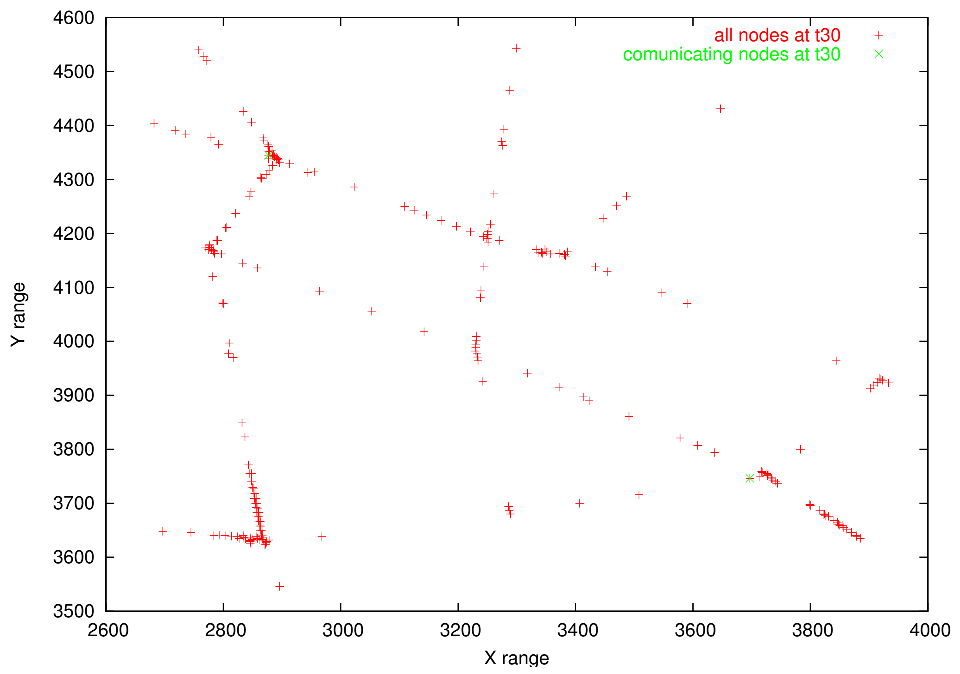

Figure 6.4: Location of all nodes from file 61 at `t=30`

6.2 Simulation results

6.2.1 Packet delivery ratio

The simulation results were rather varying. The following plot shows the latest results with a radio range of 500 metres, except for the first plot, which is simulated for 250 metres. The communicating nodes have been marked in the plot to the reader's notice. It can be seen that the packet delivery is between 35 and 60 % depending on the communicating nodes. On the other hand a distinct difference in the two transmissions can be observed, especially in the first, the 250 metres radio range transmission. This is due to the street links the packets use for the different destinations as well as another problem which will be discussed shortly. To analyse these results further a comunication between nodes across the bottom area has been set up and is shown in figure 6.6. The results for the communication of node 225 with 136 are shown in figure 6.7. To be able to better evaluate these results other simulations using the NULL MAC have been made.

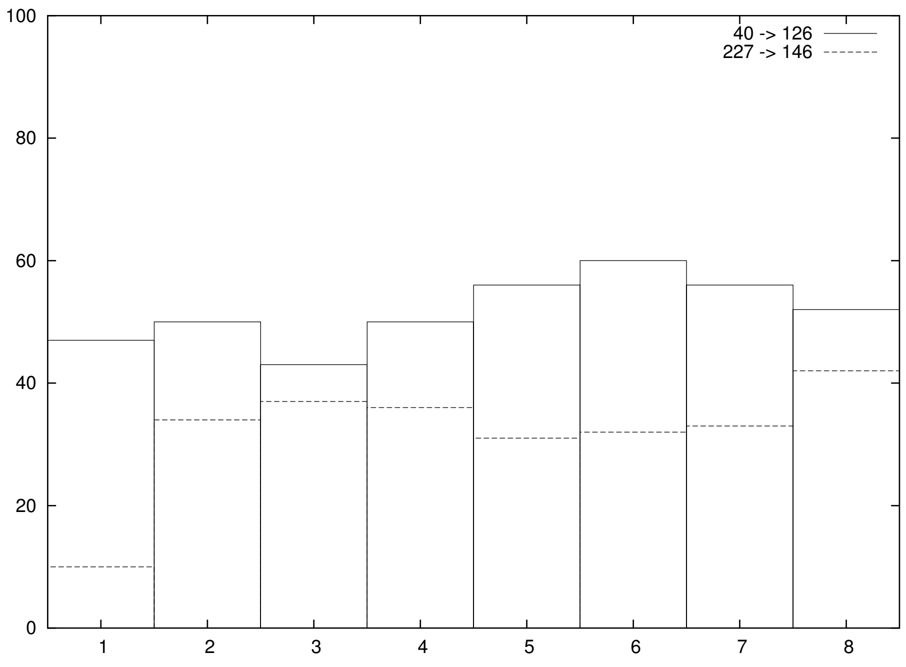

Figure 6.5: Packet delivery ratios

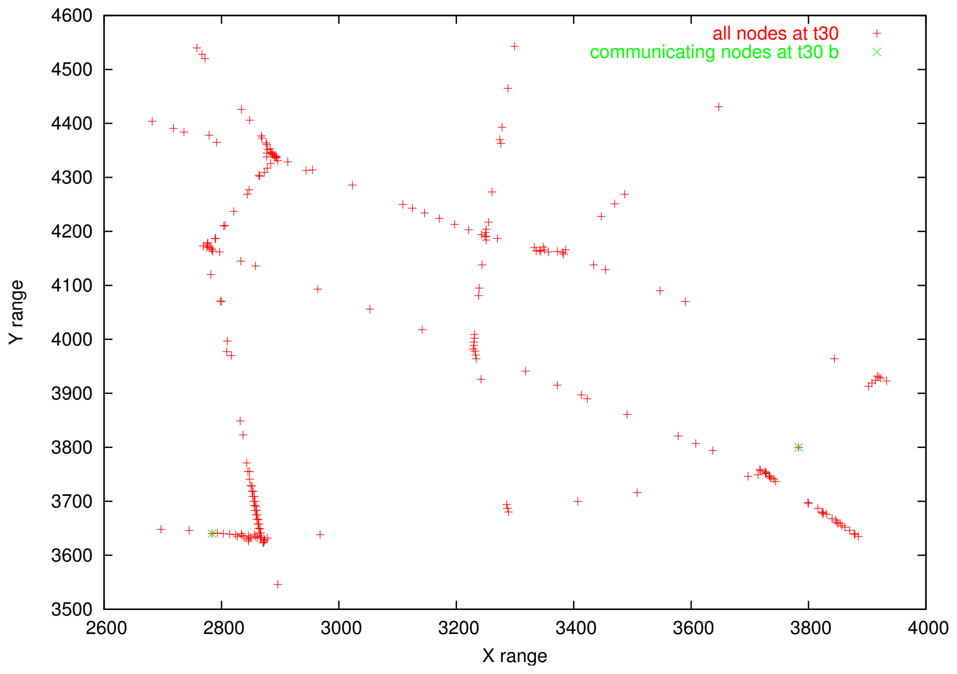

Figure 6.6: Sparsely populated link

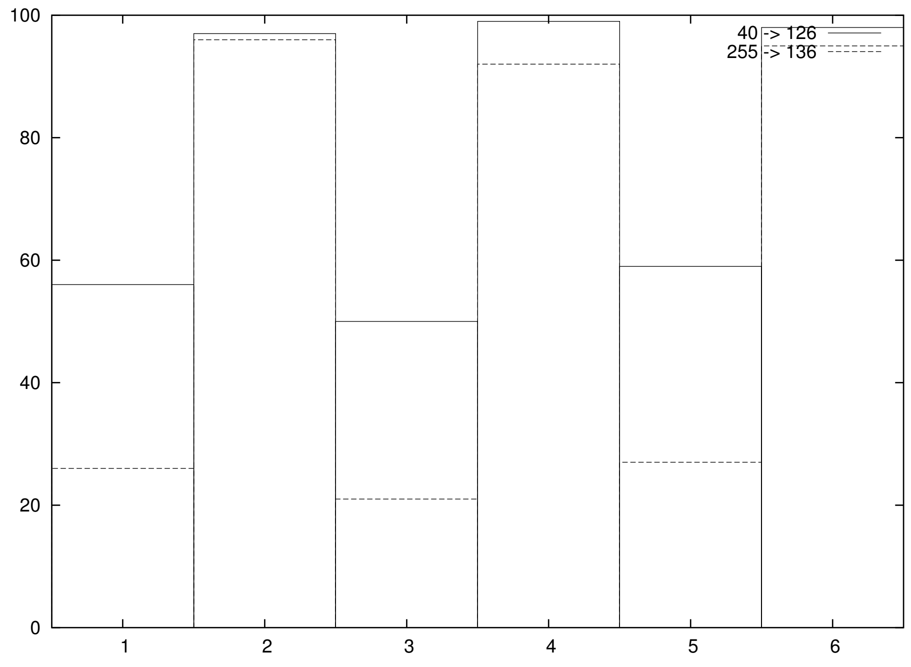

Figure 6.7: Packet delivery for node 225 to 136

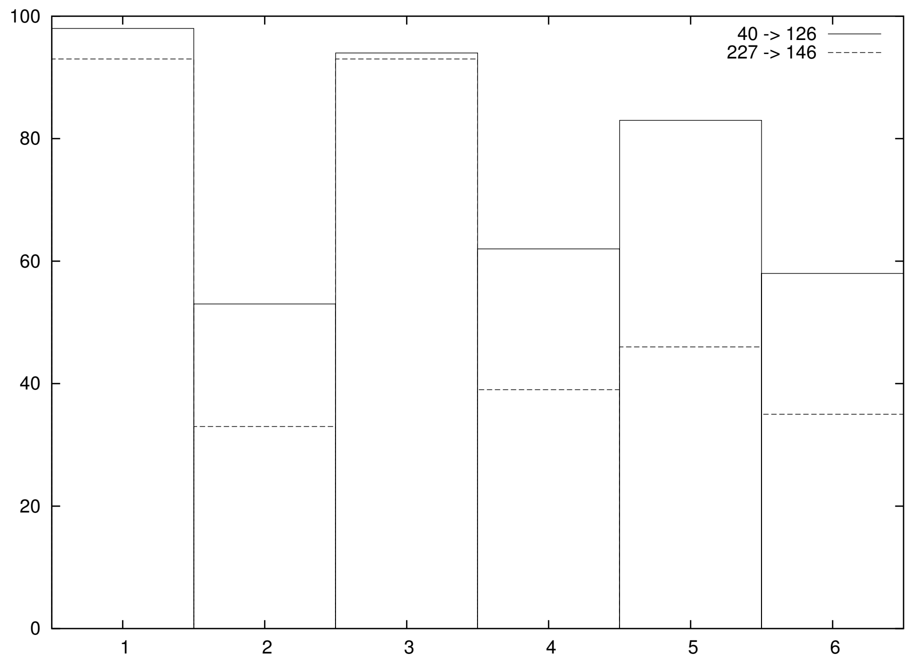

Figure 6.8: PDR data with/without NULL MAC usage

The last pair of plots (5+6) represents a radio range of 250 metres again. From these plots can be deduced that the main problem is a drop of packets due to occurring collisions. The range difference does not seem to have any effects on the delivery ratio. The NULL MAC simulations indicate good working of the routing decisions. It should be stated that no packet retry is activated which significantly lessens the probability of a successfull transmission. Unfortunately a stable version of one hop retry transmission could not be finished. A junction to junction retry approach could be possible too, relaying successfully transmitted packet IDs as well as neighbourhood link state information in the header of data packets.

To verify the link state functionality and the exchange of the master junction databases a small simulation layout had been created. Figure 6.9 shows the grid in the ad-hockey tool. The created communication pattern and member file assigned each node at an intersection junction status, the other nodes were set to street node status. The circle in the top left corner displays the used radio range of 250 metres. After verifying the functionality specific nodes were turned off to observe the reaction time of the master databases. Promising results could be achieved, which leaves the question why the results for the street scenario are so poor. Apart from the lack of packet retry the struggling masters can have a major part in this.

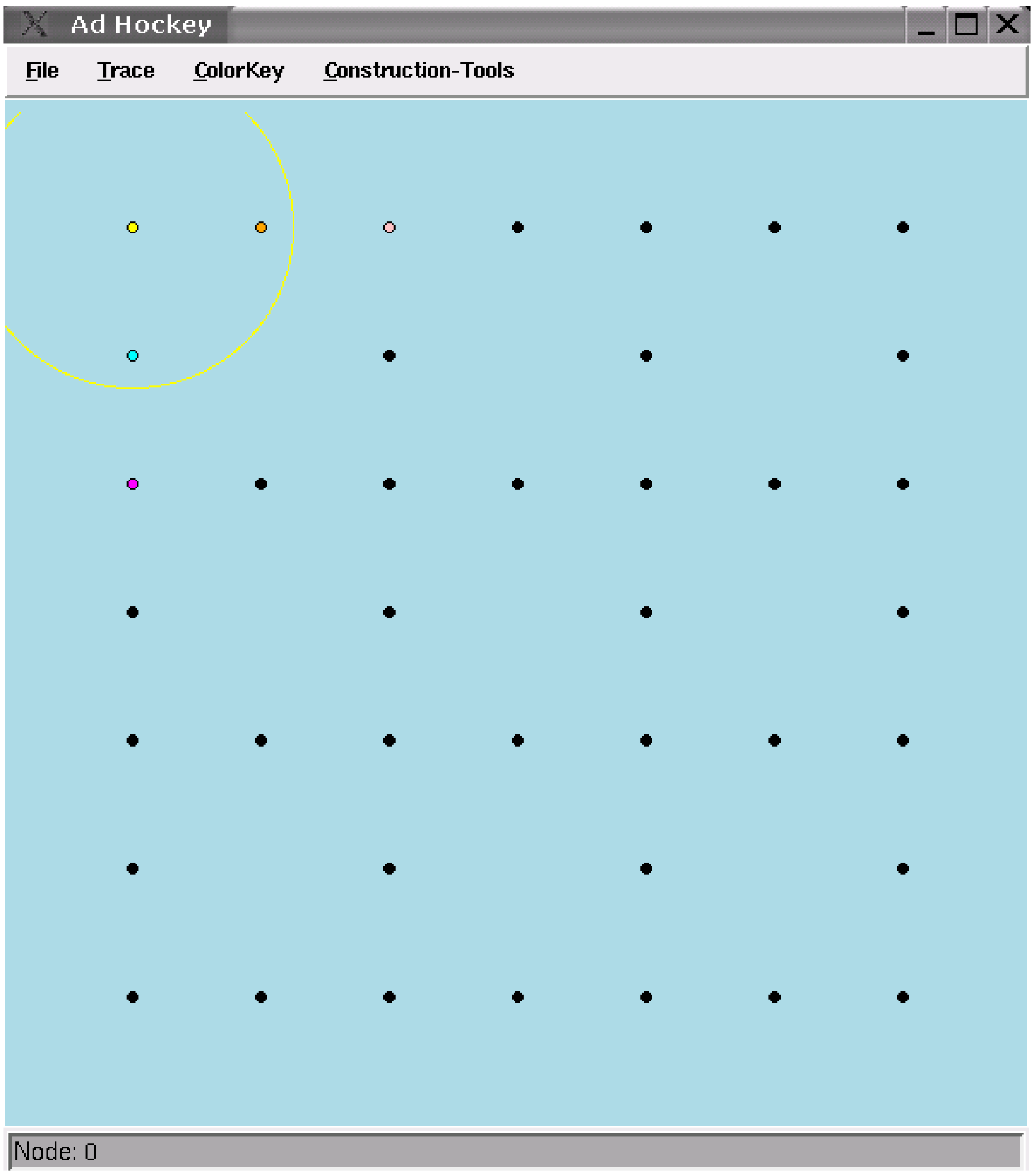

Figure 6.9: ad-hockey screenshot of grid.tcl layout

Chapter 7

Conclusion

7.1 Summary

This thesis presented the Map Information Routing Protocol, a new position-based routing agent for city scenarios. The general description and setup of the MIRP algorithm have been given, and some possible extensions introduced. The Map Information Routing Protocol divides the partitioning nodes in the simulation into destinct groups: Street nodes and junction nodes, the latter consisting of slave and master nodes. This state is set depending on the current position of the node in the map. Nodes inside a specific range around the junction centre gain junction node status. The aim is to have a single master residing in each junction channelling the traffic through 'his' junction making the routing decisions on the basis of a table of link states. This link state information is acquired by targeting special link state packets to all surrounding junctions, their masters generating specific reply answer packets. The street nodes perform only a basic contention based forwarding of packets from one junction to the next but link specific. All packets received for a different street are ignored. This way not only precise link states can be discovered but also the running into a local minimum [1] can be avoided as the responsible master should not route packets over a broken link[2]. The current

[1]as we are forwarding in greedy mode

[2]a dead end

implementation of the MIRP algorithm suffers from packet collisions.

The main idea of MIRP, bringing the routing decisions into the junctions, as superior knowledge of the topology could be accumulated there, also means more transmissions around the junctions[3]. As the one hop range of the junctions has increased traffic, part of it resulting from the control packets, these areas can be seen as areas of increased packet collision probability. If furthermore the radio range of the network devices is increased to 500[4] metres and the different junctions in the city scenario are now in direct radio range of each other, this probability is further increased. To conclude, the results of the simulation MIRP has potential if the collision problem can be eliminated. The main idea of gathering link state information in order to avoid dead end situations is worth further research.

7.2 Future Research

If the problem of several masters in a junction is to be solved, a possible approach could be an indirect switch in authority. Instead of broadcasting a `MIRP_MatJ` packet to claim the junction, a slave with an expired timer for junction privileges could only initiate a master contention request for the junction, evaluated by the `mts_`. All residing masters would then enter a forced contention period on the basis of their individual master timestamp, the longest residing master would win the contest and broadcast it's `MIRP_MatJ` packet. If such a packet is not broadcasted after a certain period, the slave itself becomes master and broadcasts the advertisement. The problem will lie in finding a suitable timespan especially with respect to the scale of the possible master timestamps.

As the control attempts in master and slave handling were found to be part of the bottleneck at the junctions exploring a different approach could be worthwhile. Early simulations with no observation about the number of masters actually residing in the junctions had not shown worse results and could lead a way to a possible tradeoff: *If every junction node collected*

[3]caused in part by the different close range packets

[4]a doubled range and an even greater area growth

link state information passing by in an autarchic way, sending this link state information by the collective in discrete time intervals means that far less bandwidth would be needed. A possible transmission of a packet by a new node in the junction, which has not yet collected much information would be a negligible cost, a junction node without any knowledge would simply not take part in the forwarding effort. The junction nodes could be handled similar to the standard contention. The backoff timer should be a combination of progress and the time at which the junction was entered by the node.

Internet sources

http://www.fleetnet.de
FleetNet Project

http://www.isi.edu/nsnam/ns/tutorial/
Marc Greis' Tutorial

http://nile.wpi.edu/NS/
Tutorial by Jae Chung and Mark Claypool

http://www.isi.edu/nsnam/ns/ns-documentation.html
ns-2 Documentation

http://inventors.about.com/library/inventors/bltelephone.htm
Alexander Graham Bell

Bibliography

[1] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad hoc networks. *Cluster Computing Journal*, 5(2), 2002.

[2] C. L. Fullmer and J. J. Garcia-Luna-Aceves. Floor Acquisition Multiple Access (FAMA) for Packet-Radio Networks. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '95)*, pages 262–273, Cambridge, MA, August 1995.

[3] H. Füßler, H. Hartenstein, J. Widmer, M. Mauve, and W. Effelsberg. Contention-Based Forwarding for Street Scenarios. In *1st International Workshop on Intelligent Transportation*, pages 155–159, Hamburg, Germany, March 2004.

[4] H. Füßler, M. Mauve, H. Hartenstein, M. Käsemann, and D. Vollmer. A Comparison of Routing Strategies for Vehicular Ad Hoc Networks. Technical Report TR-02-003, Department of Computer Science, University of Mannheim, July 2002.

[5] H. Füßler, M. Mauve, H. Hartenstein, M. Käsemann, and D. Vollmer. Poster: Location-Based Routing for Vehicular Ad-Hoc Networks. In *Proceedings of the eigth annual ACM/IEEE International Conference on Mobile computing and networking (MobiCom '02)*, Atlanta, Georgia, September 2002.

[6] H. Füßler, J. Widmer, M. Käsemann, M. Mauve, and H. Hartenstein. Contention-Based Forwarding for Mobile Ad-Hoc Networks. *Elsevier's Ad Hoc Networks*, 1(4):351–369, 2003.

[7] B. N. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the sixth annual ACM/IEEE International Conference on Mobile computing and networking (MobiCom '00)*, pages 243–254, Boston, Massachusetts, August 2000.

[8] M. Käsemann. Beaconless Position-Based Routing for Mobile Ad-Hoc Networks. Master's thesis, Department of Mathematics and Computer Science, University of Mannheim, February 2003.

[9] W. Kieß. Hierarchical Location Service for Mobile Ad-hoc Networks. Master's thesis, Department of Computer Science, University of Mannheim, 2003.

[10] C. Lochert, H. Hartenstein, J. Tian, H. Füßler, D. Herrmann, and M. Mauve. A Routing Strategy for Vehicular Ad Hoc Networks in City Environments. In *Proc. of IEEE Intelligent Vehicles Symposium (IV2003)*, pages 156–161, Columbus, OH, June 2003.

[11] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV). In *Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM '94)*, London, United Kingdom, August 1994.

[12] C. E. Perkins and E. M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, New Orleans, LA, February 1999.

[13] J. Raju and J. J. Garcia-Luna-Aceves. Scenario-based Comparison of Source-Tracing and Dynamic Source Routing Protocols for Ad Hoc Networks. *Computer Communication Review*, 31(5):70–81, October 2001.

[14] C. Tschudin, H. Lundgren, and E. Nordström. Embedding MANETs in the Real World. In *Proceedings of the IFIP-TC6 8th International Conference on Personal Wireless Communications (PWC '03)*, pages 578–589, Venice, Italy, September 2003.