BEACON-LESS ROUTING IN MOBILE AD HOC NETWORKS

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Thomas Bernoulli
November 2004

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Contents

Acknowledgement

First of all I would like to thank every body supporting me and contributing to the success of my diploma. I owe many thanks Professor Dr. Torsten Braun who gave me the opportunity to carry out my diploma in his research group *Computer Networks and Distributed Systems*. My special thanks go to Marc Heissenbüttel for supervising this work and contributing many valuable proposals. Thanks go to all the people in the research group who supported me in different topics. Last but not least I am greatly thankful to my family and my girlfriend who made this work possible, encouraged and supported me during the time of my studies.

Chapter 1

Introduction

In the last few years *mobile computing* proved to be not only a buzzword but a field of interest in computer industry. *Mobile computing* consists of many different aspects. Mobile devices are as well part of the *mobile computing* world as mobile applications, wireless networks such as cellular networks and wireless local area networks (WLANs), access technologies for wireless networks (for example 802.11, GPRS, and UMTS), and last but not least the relatively new field of mobile ad hoc networking.

Conventionally WLANs are used to ease access for mobile devices to wired networks. Two wireless nodes communicating with each other used the infrastructure as intermediate station. The idea of communicating without any fixed network infrastructure brought the field of mobile ad hoc networks (MANETs) into existence.

Mobile ad hoc networks consist of mobile devices with wireless communication facility. When it comes to exchanging data between nodes which are not in each other’s transmission range, some routing mechanism is needed. One of the intentions of research in the field of MANET is the development of adequate routing algorithms.

The first approaches tried to adapt existing routing algorithms to the specific characteristics of MANETs, look at OLSR in section 3.3 for an example. Traditional routing algorithms build their routing tables mainly based on topology information. This works fine as long as the topology remains relatively stable. But MANETs consist of mobile nodes which in some cases move quite a lot. In these cases topology based routing protocols come into trouble due to the frequent topology changes. This either results in outdated routing tables or in triggering routing table updates and spreading information about the topology changes throughout the network. Newer topology based routing protocols such as AODV (see section 3.1) try to avoid the network traffic of topology change information by being reactive. This strongly decreases the traffic overhead but does not solve the problem of outdated routing table entries.

As the name already states, one of the mobile nodes’ main property is their mobility. Mobility implies the changing of position. The fact that topology and position changes are related lead to the idea that position information could improve routing in MANETs: The promising approach of position based routing was born.

First approaches made only slight usage of location information, for example the Location-Aided Routing protocol (LAR). Research on the field continued and protocols making extensive use of location information emerged. An example for such a protocol is GPSR (see section

4.1 for details). These protocols do not maintain a routing table based on topology but based on location information. The information the routing tables are based on is disseminated using periodical message sending, thus generating bandwidth consuming control traffic.

In the technical report *BLR: A Beacon-Less Routing Algorithm for Mobile Ad Hoc Networks* [1], a new approach dramatically reducing control traffic has been proposed. The intention of this thesis was to implement the proposed routing algorithm in a network simulator and to evaluate several of its parameters. Furthermore improvements to the algorithm itself have been added.

The thesis first introduces mobile ad hoc networks and its characteristics. Chapter 3 and 4 give an overview of existing MANET routing protocols. In chapter 5 the Beacon-Less Routing Algorithm is explained in detail. When evaluating MANET routing protocols using simulators the used mobility model greatly influences the results. Therefore chapter 6 introduces and discusses the most common ones. Qualnet v3.6 is the simulator used to evaluate BLR. It is introduced in chapter 7, which is followed by a description of the BLR implementation in Qualnet v3.6 in chapter 8. The simulation scenarios and results are presented in section 9 which directly leads to chapter 10 where the conclusion and future work are presented.

Chapter 2

Mobile Ad Hoc Networks

In contrast to traditional wireless networks a mobile ad hoc network (MANET) does not require any fixed infrastructure or administration authority. A MANET consists of a collection of devices equipped with wireless communication facility, called nodes. Nodes which want to communicate and do not stay inside each others transmission range need intermediate nodes relaying their traffic. That is where the MANET routing algorithms come into play. Before having a look at existing routing algorithms some characteristics of MANETs and the most common wireless communication facility are presented.

2.1 Mobile Ad Hoc Network Characteristics

MANETs are not set up by any authority but are rather self-organizing, self-configuring, and self-administrating. These properties are really essential as it seems to be the only possibility to handle the special circumstances of a MANET. As the network consists of mobile node which occasionally move around, the topology is not stable. Moving nodes are not the only cause of changing topology. Devices can be switched on or off resulting in new nodes appearing or breakage of recently working links. Changing transmission conditions such as interferences can have the same effect. All those attributes of a MANET result in a rich topology changing rapidly and in unpredictable manner.

Other characteristics do not have such a direct impact on the network itself, but they should be considered in development of new routing algorithms or services based on MANETs. The property with the biggest impact is closely related to the mobility property: Devices have limited resources, such as energy and memory. Thus resource intensiveness is incommoding and should be avoided whenever possible.

2.2 Wireless Local Area Networks

In principle mobilie ad hoc networks can consist of nodes using any kind of wireless communication facility. But the well established wireless local area network (WLAN, 802.11 [2]) standard seem to be the most common used one. The implementation of BLR following later in this thesis uses WLAN as its underlying protocol too. WLAN has been defined to be the wireless counterpart of traditional ethernet protocol for local area networks (802.3 [3]). After the first definitions development moved on in the domain of wireless data transmission. Thus succeedingly faster standards for the physical layer of 802.11 have been published (802.11a [4], 802.11b [5], and 802.11g [6] are the most important ones).

Traditionally WLANs are used to provide access to wired networks through so called access points. In those scenarios the access point controls and coordinates the ongoing activities in the WLAN. In the alternative scenario two nodes communicate directly without any coordinating authority. This usage of WLAN is the starting point for MANET activities. The following part of this section gives an introduction on 802.11 features.

The 802.11 protocol defines two modes: The first one contains the *Basic Service Set* (BSS) and the *Extended Service Set* (ESS) and is used in traditional WLAN scenarios with access point infrastructure. The second one, called *Independent Basic Service Set* (IBSS) is used in ad hoc scenarios. Obviously, the IBSS matches the MANET requirements. The following description will not explain the IBBS but some 'low-level' mechanisms such as message sending. The understanding of these mechanisms will be needed to recognize their interaction with BLR.

Using the 802.11 protocol, a message can be sent using MAC level acknowledgement or not. MAC level acknowledgement can only be used for unicast transmissions because the target node has to be known by MAC layer address. Sending without any MAC level acknowledgement is called broadcasting. Broadcasted packets are received by any node within the sending one's transmission range, since *broadcasting* is used consistent to other fields of networking. Quite usual for broadcasted messages is the fact that successful delivery is neither guaranteed nor acknowledged. Unfortunately there exists no possibility to check if the transmission has interfered with another one. In wired networks this is not really a problem as transmission errors occur rarely and interferences with parallel transmissions on the same medium can be detected. In contrast thereto, wireless networks can suffer a wide range of transmission interferences, making the incorrect transmission of packets more likely. This has to be kept in mind if broadcasting messages.

When a node wants to transmit a message using unicast, it first listens for a certain amount of time whether any other node is sending or not. If the medium is not used the handshake as illustrated in figure 2.1 takes place: The node which wants to transmit data sends a *Request To Send* (RTS) message to the destination node. The destination answers sending a *Clear To Send* (CTS) message. After receiving the CTS the actual data packet is sent and if the transmission was successful, the destination sends a *Acknowledgement* (ACK) message. If no ACK is received, retransmission is performed (up to seven times). This mechanism not only checks that the receiving node is within transmission range but rather is some kind of medium reservation: The nodes within transmission range of a node sending a RTS or CTS message receive it too and therefore queue their own transmissions till the receiving node has sent the ACK. That solves the *hidden node problem*: If a sender A wants to communicate with B and a third node C exists which is only able to overhear B's transmissions, it would consider the medium to be free while A is sending data (which it cannot detect) but would disturb B's reception if it starts sending anything (figure 2.2 illustrates such a situation). Using the above described handshake avoids that: Node C is able to receive B's CTS and therefore queues its transmission.

If sending a message using broadcast, the node just invokes carrier-sensing (as in the unicast

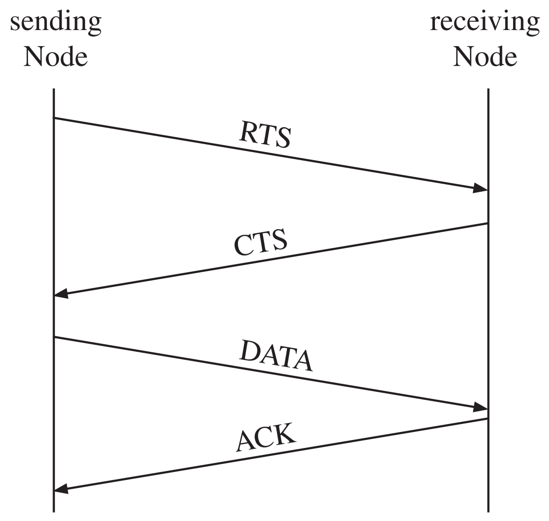


Figure 2.1: 802.11 unicast data sending: The node which wants to send a data packet initiates the handshake sending a RTS message. The addressed node answers with a CTS message, if it could receive the RTS. Once the first node received the CTS, it starts transmitting the DATA packet which is acknowledged by the receiver sending an ACK message as soon as the data arrives.

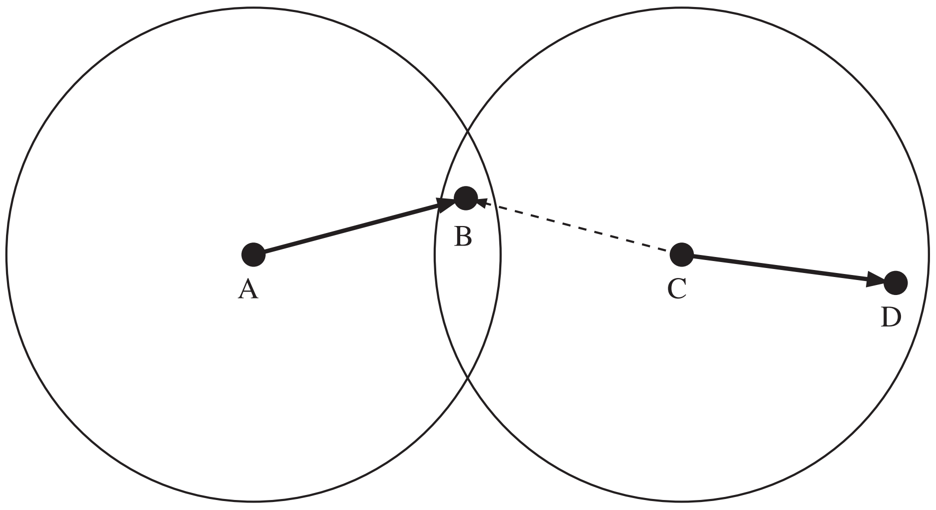


Figure 2.2: Hidden node problem: Node B is within the transmission range of node A and C, but A and C are not in each others transmission range. Consider the situation, where A is sending data destined for B. C, not able to detect the ongoing transmission, sends data to D on the same channel. At B's position both signals interfere and B can not receive A's transmission without errors. Because C cannot receive A's signals, C is not aware of A: From C's point of view A is *hidden*.

case too) and then starts sending the data packet if the medium is free. Because the sending node has no facility to check (at the MAC layer) if the packet reached the destination, retransmissions cannot occur.

An important and interesting element has not yet been described: 802.11 uses CSMA/CA which stands for *Carrier Sense Multiple Access with Collision Avoidance*. CSMA signifies that a node makes sure that no other traffic is transmitted on the shared physical medium (*carrier sensing*) before transmitting anything as the medium can be accessed by multiple nodes at the same time (*multiple access*). This mechanism has already been mentioned above when describing unicast and broadcast data sending. CA has the purpose to avoid data loss due to collisions. It can be implemented in different ways. The following paragraph describes the mechanism used in 802.11.

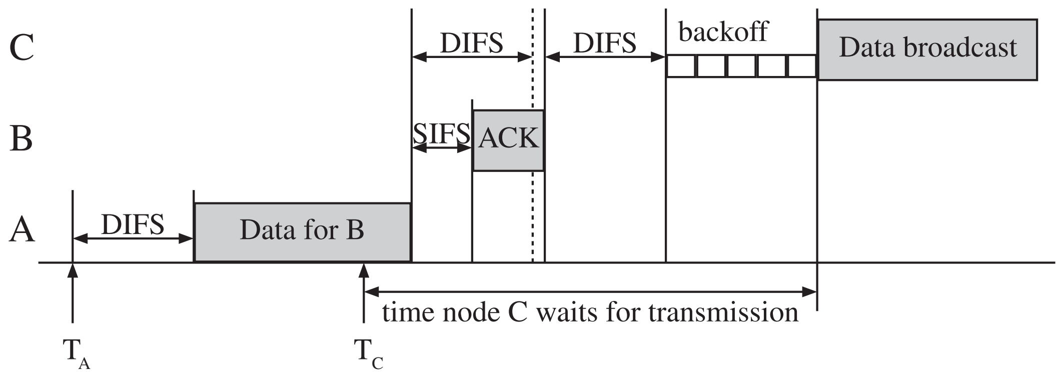


Figure 2.3: 802.11 CSMA/CA mechanism: Consider the situation where Node A has completed the RTS-CTS handshake with node B and wants to transmit the data packet. A first waits for DIFS and because the medium was free during this time, it starts the transmission. While this transmission takes place, at time T_C node C wants to broadcast a packet. It waits until the data transmission ends and then computes the *backoff* (which ended up by 5 slots) and starts waiting for DIFS + *backoff*. As soon as B decoded the data packet from A and the medium is free, it starts waiting for SIFS and then send the ACK. This ACK is sent during C is waiting, which causes to interrupt the timer. After the ACK is transmitted C waits again for DIFS + 5 slots (*backoff* already calculated and not yet decreased as the interruption occurred during the DIFS waiting time). After that, the medium is still free and was not busy in the meanwhile, so C start broadcasting its packet.

The first rule of the CA algorithm states, that a node must wait for a *Inter Frame Space)* (IFS) for the medium to be free, before it is allowed to transmit anything. There exist three different IFSs, which results in some kind of prioritization of the different sorts of packets: CTS and ACK packets only have to be retarded for a short IFS (SIFS), the shortest of the IFSs, so they have highest priority. The second IFS (PIFS) is not used in IBSS mode. The longest IFS is called DCF IFS (DIFS). DCF stands for *Distributed Coordination Function* which means that the DIFS can be prolonged in order to reflect a distributed decision to avoid collisions. This DCF works as follows: If the medium has already been free when the node wanted to start its transmission, the DIFS has a fixed length (one slot longer than the PIFS). If the medium was busy, the DIFS is the fixed length plus some random time called *backoff* which is calculated by the formula:

$$\textit{backoff} = \lfloor CW * random \rfloor * \textit{slot time}$$

CW	*Contention Window*: A value depending on occurrence of contention. It starts at 31 (CW_{min}) and is doubled up to a maximal value of 1023 (CW_{max}) every time the *backoff* timer times out and the medium is still busy. It is reset to CW_{min} after successful transmission.
random	A random value in the interval $[0, 1]$
slot time	The resulting time adding up the delay to turn on the sender, the signal propagation delay, and the time needed to detect a busy medium. For example $9\mu s$ for 802.11a and $20\mu s$ for 802.11b.

Once the *backoff* has been calculated, a node applying DCF waits for DIFS + *backoff*. If the medium becomes busy during this time, the countdown of the timer is paused and the timer value used as *backoff* if the medium becomes free again instead of calculation the *backoff* again (see figure 2.3 for an example). If the *backoff* times out without being able to transmit the packet, the procedure restarts but with a doubled contention window value. That happens in order to prevent a high collision probability if more nodes set a *backoff* timer (which is equivalent to more nodes waiting for a free medium to transmit packets). Therefore the available amount of *backoff* slots is increased by doubling the contention window (see formula and explanation for details) in the present of high network load.

The upcoming 802.11e standard improves the DCF to achieve a fine grained prioritization of the packets. This is done by assigning different minimal contention window values for different priority classes. More detailed description is way beyond the scope of this introduction to 802.11 but its mention worth as the *backoff* will come into focus again in section 5.5.

Chapter 3

Topology Based Routing Protocols

Topology based routing protocols build the routing tables based on the knowledge of the network's topology (or part of it) and therefore comprise some kind of topology discovery. In the majority of cases the nodes know only a small part of the whole network topology. The introduced protocols are tailored for routing unicast IP addresses but do not impede development of extensions such as for routing multicast IP addresses.

One distinctive feature of topology based routing protocols is the way they maintain their routing tables. It is either *reactive* or *proactive*. Reactive routing protocols establish routes on demand and only maintain them during their use for transmission. Therefore some kind of route detection is needed which is mostly accomplished by some flooding mechanism. In contrast to the reactive behavior, proactive routing protocols exchange routing information, generate and maintain routing tables independently of the current need of routes for sending packets. Generally those routing tables contain routes to every node in the network.

The following sections present AODV and DSR as representatives of the reactive protocol family and OLSR for the proactive protocol family.

3.1 Ad Hoc On-demand Distance Vector Routing Protocol (AODV)

The AODV [7] routing protocol uses separate routing control messages for exchanging topology information. Those messages are exchanged via UDP and normal IP header processing.

If the situation occurs that no valid route between two endpoints which would like to communicate exists, the node needing a route emits a Route Request (RREQ) message using limited broadcast mechanism: The TTL field is set to a value quite lower than the maximum in order that the RREQ is not broadcasted throughout the whole network but only within a given radius measured in hops. If a RREQ fails, the TTL is increased up to the maximum value.

As the RREQ reaches the destination or a node which has a valid route entry for this destination it is assumed that a valid route has been found and a Route Reply (RREP) message is generated to establish the route. This RREP is unicasted back the way the RREQ has reached the replying node. This is possible as every node which forwarded the RREQ cached a route back to the originator of the request. If the node originating the RREQ 'knows' that the destination will need a route back it can set a flag which instructs an intermediate node generating the

RREP to send some kind of RREP (called gratuitous RREP) to the destination as well. Using this mechanism the originating node can be quite sure that the established route is bidirectional. The distance vector aspect is done by clever handling of node specific sequence numbers which causes selective forwarding for RREQ and RREP. Not only the distance vector aspect but also the avoidance of loops is done by this sequence numbers.

As established routes fail or timeout, which can be prevented by using the optional mechanism of sending hello messages for active routes, a Route Error (RERR) message is generated and sent to affected nodes. As for every route entry the previous hop nodes are stored, its easy to select the affected nodes and send them the RERR. Receivers of a RERR update their routing table accordingly and then forward the RERR to other affected nodes.

3.2 Dynamic Source Routing Protocol (DSR)

A speciality of DSR [8] is its possibility of allowing *multiple* routes to any destination. A sender can select and control the routes used in routing its packets due to the protocol's explicit source routing property. Another characteristic of this routing protocol is its operability over unidirectional links.

The DSR routing protocol consists of the two main mechanisms *route discovery* and *route maintenance*. Whereas route discovery is the mechanism of finding a source route for a given destination to which no valid route exists, route maintenance is the mechanism by which the source node is able to detect that a link along a route in use no longer works.

The initiator of a route discovery sends a Route Request (RREQ) containing the address of the initiator and the target as well as a unique request identification. Each RREQ lists the addresses of every intermediate node which forwarded this packet. Every node receiving a RREQ forwards it if it has not already seen this RREQ, is already listed in the list of intermediate nodes or is the target of the RREQ. After appending its address to the list of intermediate nodes the node forwards the RREQ by transmitting it as a local broadcast packet. A node receiving a RREQ of which it is the target generates a Route Reply (RREP) which it sends back to the initiator of the route discovery. If the target node doesn't have a source route to the initiator, it has to initiate a route discovery itself. But to avoid possible infinite recursion of route discoveries, it piggybacks the route reply on the RREQ packet.

When originating or forwarding a packet using a source route, each node transmitting the packet is responsible for confirming that the packet has been successfully delivered to the next hop (by any feasible acknowledgement). If a delivery to the next hop fails, the link is treated as broken and a Route Error (RERR) message is generated and sent to any affected nodes.

Several additional features for route discovery and maintenance, such as caching overheard routing information, replying to RREQ using cached routes, preventing route reply storms, packet salvaging, automatic route shortening, increased spreading of RERR messages and optimized handling of queued packets for broken links, have the same goal as the optional DSR flow state extension: Namely minimizing routing overhead and end-to-end delay.

3.3 Optimized Link State Routing Protocol (OLSR)

As the name states, OLSR [9] modifies the classical link state routing protocol where the modifications are made with the goal of optimizing the protocol for its use in MANETs.

One of the main optimizations of OLSR affects the disseminating of topology information in the network, using the concept of multipoint relays (MPRs). A MPR is a node authorized to forward the messages in the flooding process of OLSR. The establishment of the flooding structure is based on periodic Hello message (HELLO) sending. HELLOs contain a list of the sender's neighbors and details about the link to them. Based on the information of all HELLOs received, a node selects its MPRs in such a way that it can reach every 2-hop neighbor (a neighbor's neighbor) which is not directly reachable, through its MPRs. Each HELLO also contains a detailed list of the sender's MPRs. Therefore a node listed as MPR receiving the HELLO hence will be aware of its function which consists in forwarding messages that are flooded through the network (nodes which are not MPRs do not forward those messages).

Based on the information about the neighborhood, nodes which have been selected as MPR periodically disseminates topology information through the whole network by sending topology control messages (TCs). In the TCs a node advertises the links to its neighbors (at least to the ones which have selected it as MPR). The TCs are spread using the flooding mechanism of OLSR.

By means of the information gathered by processing TCs and HELLOs, each node can calculate its routing table using a shortest path algorithm on the graph generated from the gathered information. Normal data packets are routed hop-by-hop, according to those routing tables.

OLSR does not only work on single interface devices but specifies functioning using multi-interface devices. Extensions of OLSR define the functioning of OLSR on devices with OLSR enabled and disabled interfaces, as well as MPR redundancy, taking into account link layer notifications and redundant topology information.

Chapter 4

Position Based Routing Protocols

Position based routing protocols use position information from the nodes to improve the routing. The currently available position based routing protocols assume the presence of some kind of location service to look up the position of the destination and a facility which enables each node to be aware of its position. Synonyms for position based routing are *geometric*, *geographic*, and *location based* routing.

4.1 GPSR / GFG

GPSR [10] tries to minimize the information a node has to collect, to make a router virtually stateless. The only knowledge a node has to collect is the position of its neighbors. Based on this knowledge, GPSR routes packets hop-by-hop.

In GPSR every node periodically sends Hello messages (called beaconing) to advertise its presence and position to potential neighbors.

If it comes to routing a packet, GPSR has two different modes: The *greedy mode* and the *perimeter mode*. GPSR always tries to forward packets in greedy mode because it is the more efficient one. When forwarding a packet in greedy mode, the current node searches its list of neighbors for the node closest to the destination which is at least closer than itself. If the search is successful, the node forwards the packet to this node. If the search ends without any resulting node, GPSR marks the packet being in perimeter mode and selects the next hop using the perimeter mode strategy.

The idea of the perimeter mode is as follows: A network can be modelled as a unit disk graph. If the graph is planarized it is possible to apply the right-hand rule to guarantee that a packet reaches the destination if a path exists [11]. Therefore the current node has to generate a subset of the neighbors list applying a distributed algorithm to create a planar graph. The two algorithms mostly used are the *relative neighborhood graph* or the *gabriel graph* algorithm. After accomplishing this task, the node applies the right hand rule to route the packet around the face which crosses the virtual line between itself and the destination. A packet stays in perimeter mode and is forwarded the same way (using the right hand rule) by the following hops, as long as the processing node is not closer to the destination than the node setting the packet into perimeter mode (which can be determined since the position of this node is stored in the packet header). If a packet in perimeter mode traverses the same link the second time, it is dropped immediately, which is a mechanism to avoid endless looping of packets if no path to the destination exists. In figure 4.1 a packet's path in GPSR is depicted. It illustrates greedy as well as perimeter mode.

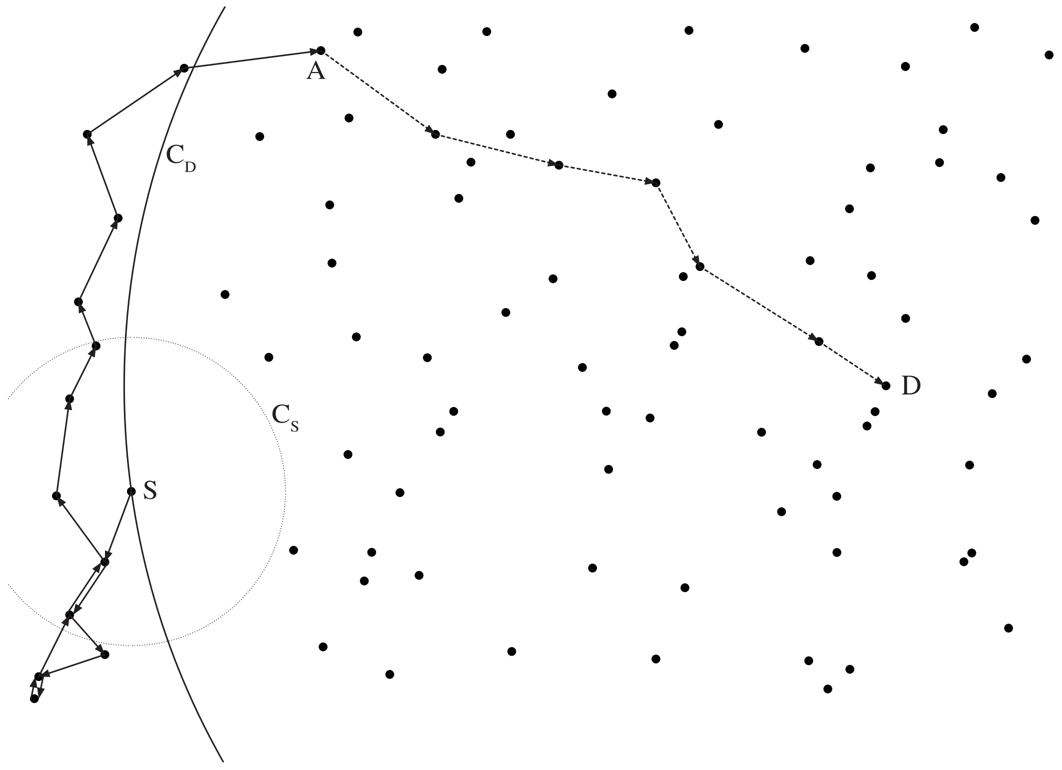


Figure 4.1: GPSR: Node S wants to send a packet to node D. Within S' transmission range (depicted by circle C_S) there exists no node with forward progress. Therefore S has to transmit the packet in perimeter mode. It follows the path in perimeter mode (indicated by drawn through arrows) till node A where it is closer to the destination than S which started perimeter mode (the circle C_D depicts this imaginary border). From node A until the destination it can be routed using greedy mode as in every hop's transmission range stays at least one node.

The basic idea of the protocol was first stated in [11] where the protocol is called GFG and defined at a higher level whereas the paper on GPSR [10] makes the purpose of the different parts more obvious and specifies some details, such as interaction with the MAC layer.

4.2 Terminode Routing

The objective of terminode routing [14] is to achieve scalability in large MANETs and being robust against inaccurate destination position information. The approach of terminode routing combines two routing protocols: *Terminode Local Routing* (TLR) and *Terminode Remote Rout-*

ing (TRR). While TLR is used if the destination is 'near' the processing node's location, TRR determines the path for long distance transmissions.

4.2.1 Terminode Local Routing (TLR)

TLR is a proactive topology based routing protocol. The big difference to other routing protocols of the same category is its restricted coverage. Each node only maintains routes for nodes at most *local radius* away. The bigger the *local radius*, the more nodes are covered and the more control message overhead occurs. A *local radius* of two hops turned out to be a good tradeoff. The routing tables are maintained be periodically sending hello messages. To be able to maintain a two hop neighborhood, hello messages announce information about the sending node and its immediate neighbors. When it comes to routing a packet, TLR uses a simple two hop link state routing protocol. Obviously only nodes in two hop neighborhood can been reached using TLR.

4.2.2 Terminode Remote Routing (TRR)

TRR consists of two parts: A hop by hop position based routing protocol called Geodesic Packet Forwarding (GPF) and the overlay protocol Anchored Geodesic Packet Forwarding (AGPF) to set intermediate locations. AGPF itself uses path discovery methods to obtain the paths the anchors are set for.

Geodesic Packet Forwarding (GPF)

GPF is similar to GPSR (see section 4.1). *Greedy mode* and *perimeter mode* are equal to the corresponding parts in GPSR. The difference consists in the additional check at the processing node: It checks whether the destination can be reached using TLR (TLR reachable). As soon as this condition is fulfilled, the packet is forwarded using TLR. Once a packet has been routed using TLR it cannot revert to TRR. This is only a restriction impeding the packet delivery if the destination position information in the packet is outdated. If the destination node is not TLR reachable for nodes at the given destination location, the packet would be forwarded around the destination location until TTL expires and the packet is dropped, thus wasting network resources.

To prevent this situation and handle position inaccuracy, nodes consider a potential looping scenario detected if the destination location is within their transmission range but the destination node is not TLR reachable. Two strategies are proposed to solve the problem. The first uses the TTL mechanism to restrict the forwarding to a few hops: The TTL field is set to a small number. The current implementation sets it to $min(3, current\ TTL)$. The second strategy is called *Restricted Local Flooding* (RLF). RLF is a neat algorithm for restricted flooding: A limited number of copies of the packet is sent towards different directions. The number of packets and their lifetime in hops (limited using a corresponding TTL value) are coordinated to cover a circle around the flooding node. The flooding uses GPF for routing the packets.

Anchored Geodesic Packet Forwarding (AGPF)

The purpose of AGPF is to circumvent large regions where GPF has to switch to *perimeter mode*. The main idea is that the source node sets a geographic path for the packet, thus AGPF is a loose source routing mechanism. This is achieved by adding geographic anchor points to the packet's header which have to be visited. An anchor point is considered reached as soon as the processing node stays within transmission range distance of the anchor point. The node at which an anchor point is reached deletes it and forwards the packet to the next one. After the last anchor point has been deleted, the packet is directly routed to the destination. Routing from anchor point to anchor point is carried out using GPF routing. Because the path given by the anchors avoids regions with low node density, GPF probably can forward the packet to the next anchor point using only *greedy mode*.

Two methods are proposed for path discovery: Friend Assisted Path Discovery (FAPD) and Geographic Maps-based Path Discovery (GMPD). FAPD has its name because every node keeps a list of other nodes, called friends. The name is quite appropriate as the nodes in this list have to cooperate. A node maintains routes to its friend for contacting them in order to find an anchored path to a given destination.

The maintenance of a summarized geographical view of the network by each node is the assumption taken by GMPD. This map contains areas where the density of nodes is higher than average. Based on that knowledge a node selects the anchors for routing a packet to its destination.

4.3 GOAFR$^+$

GOAFR$^+$ [12] divides the routing into the same two phases as GPSR (see section 4.1) does: A *greedy mode* for forwarding the packets if neighbors closer to the destination exist and a second mode called *face routing* to recover in situations where *greedy mode* fails. The goal was to find a routing algorithm which combines theory and practice. This means that the algorithm should be asymptotically worst-case optimal and provide average-case efficiency.

The *greedy mode* is identical to the *greedy mode* in GPSR. The beaconing is not discussed in detail as the knowledge of the neighbors' positions is an assumption.

What makes GOAFR$^+$ different from GPSR is the fall back mechanism which is an adapted *face routing*, originally introduced in [13]. The *face routing's* decisions are based on a planar graph generated using the gabriel graph algorithm. To find a next hop the right hand rule is applied. The distinguishing feature to GPSR is the condition for falling back into *greedy mode*. While GPSR uses a simple *closer to destination* rule, GOAFR$^+$ introduces sophisticated mechanisms to reach the goal set. For that purpose a virtual circle C centered at the destination D is maintained while forwarding the packet. The circle is initiated at the original source S of a packet with a radius $r_C = \rho_0|\bar{SD}|$ $(1 < \rho_0)$. While forwarding in *greedy mode* each node reduces r_C to $r_C = r_C/\rho$ as long as the current node stays within C $(\rho < \rho_0)$.

The node A first forwarding a packet using *face routing* initializes two counters. These counters are updated at each node while the packet is forwarded using *face routing*. While the first counter p counts the nodes visited which are closer to the destination than A the second counter q counts those farther away. The packet is forwarded using the right hand rule until a specific situation triggers a different action:

- The edge of the graph to the selected next hop crosses the circle C
  - for the first time: Switch from right hand rule to left hand rule. This results in routing the packet back to A and along the other border of the face.
  - for the second time: If no visited node is closer to D than A, enlarge C setting $r_C = \rho r_C$ and continue right hand rule forwarding. In the case at least one visited node is closer to D than A, forward to the closest one and when arriving switch back to *greedy mode*.
- $p > \sigma q$ (for a constant factor σ): The packet is forwarded back to the node B visited while *face routing* that is closest to D. B starts routing the packet in *greedy mode* again.

Packets may return back to A before any of the above conditions is fulfilled. If such a packet have not visited any node closer to D than A, graph disconnection is reported to the original source S. Otherwise the packet is forwarded to the visited node closest to D where it enters *greedy mode* again.

Chapter 5

Beacon-Less Routing (BLR)

Most of the position based routing protocols (GPSR, GOAFR) just exchanged the topology based routing protocols' weak point of outdated topology information: They have to bother with outdated position information of neighboring nodes.

This chapter introduces a Beacon-Less Routing algorithm (BLR) based on the technical report 'A Beacon-Less Routing Algorithm for Mobile Ad Hoc Networks' [1]. BLR is a position based routing protocol trying to avoid the well known problem of outdated position information of neighboring nodes.

5.1 Overview

The BLR approach tries to avoid the tradeoff to be made if using periodically broadcasted position announcement messages: Keeping position information of neighboring nodes up to date is a tradeoff between position inaccuracy and bandwidth consuming periodical broadcasting of position announcement messages. Because the low available bandwidth is one of the main concerns in mobile computing, BLR tries to save bandwidth by avoiding periodical sending of any kind of control messages. In addition, there is the benefit not having to address the problem of outdated information.

BLR is not an implementation of a whole network infrastructure but of a routing protocol addressing some known problems. Therefore some preconditions have to be fulfilled before BLR can be deployed. The first one is that the used links have to be bidirectional and the used antennas omnidirectional. The bidirectional property of links is also a prerequisite for our assumption that the MAC layer provides acknowledgement for sent unicast messages. Another assumption is that two system wide parameters are known by every participating node: *maximal additional delay* and *maximal transmission range*, which are explained in detail in section 5.2. In the current implementation the internet protocol is assumed network protocol and 802.11 as MAC layer protocol, but this is not a strict assumption, replacing those protocols is possible. The other assumptions are common ones for position based routing algorithms: Each node is aware of its position and nodes which inject packets in the network have a mechanism for looking up the destination's position at their disposal.

The following sections give a detailed description of the BLR protocol. Basically the protocol consists of two mechanisms, called modes:

- *Greedy mode* is used to forward packets with reasonable per-hop progress
- *Backup mode* is necessary to rout packets in regions where the greedy mode does not find a path

This classification is not brand new and already made in GPSR [10], GFG [11], and probably other routing protocols.

5.2 Greedy Mode

The originating node (the one that injects the packet in the network) first adds the BLR data header to the packet. The header is placed between the original IP header and the payload of the packet to route (see figure 5.1 for an illustration). Details about this design decision can be found in section 8.1. The originating node sets the *seqNr* field to a sequence number which together

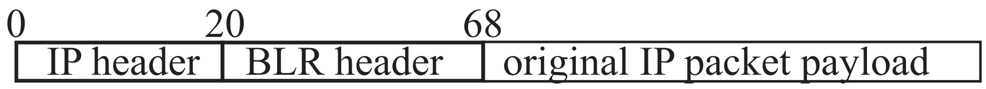


Figure 5.1: A packet after the BLR header has been inserted: The given numbers indicate the position in the packet in bytes.

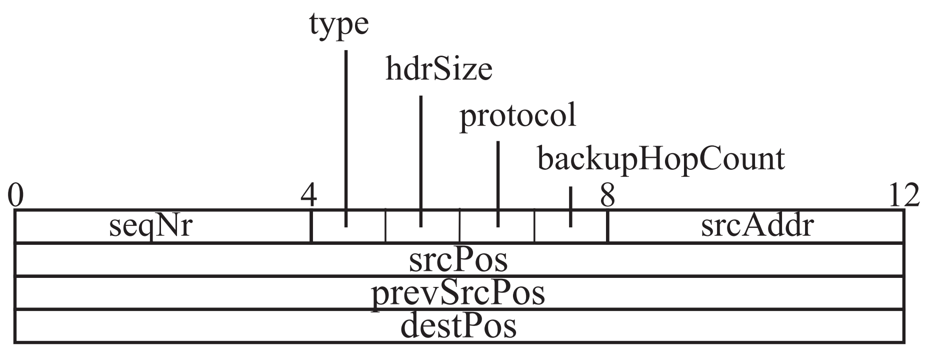


Figure 5.2: BLR data header: The fields, their position within the header and their size. Each line contains 12 bytes. The given numbers mark the position of the n^{th} byte in the header. The total header size is 48 bytes.

with the node's IP address makes this data packet identifiable throughout the whole network. For that purpose each node has a counter to derive the sequence number to guarantee that it does not set the same sequence number in different packets (the wrap around can be disregarded as the information does not persist long enough in the network that the wrapping can lead to

Table 5.1: BLR data header fields: Denotation of the fields in figure 5.2

seqNr	a sequence number which together with the originating node's IP address makes this data packet identifiable
type	specifies the type of this BLR header. Is set to DATA_PKT
hdrSize	the size of this header
protocol	the saved IP protocol field
backupHopCount	number of hops completed in backup mode
srcAddr	the IP address of the transmitting (current) node
srcPos	the position of the transmitting (current) node
prevSrcPos	the position of the previous transmitting node
destPos	the position of the destination

problems). The *type* field is set to DATA_PKT, which indicates that this packet is in greedy mode. The *protocol* field is set to the value from the *protocol* field in the IP header and that one is set to the value indicating that this is now a BLR protocol packet. As the header is inserted between the IP header and the data of the original IP packet, the node updates the *total length* field of the IP header to the new total length of the packet and recalculates the checksum. The *prevSrcPos* field is set to the same position as the *srcPos* field since there exists no previous hop. The other fields are set according to their designation.

After initializing the header a copy of the message is put in the *broadcasts sent buffer* and the node transmits the message using the MAC layers broadcast mechanism.

Any node receiving the packet will do the following processing: First it checks if its not farther away from the transmitting node than *maximal transmission range*. If this test fails, the packet is silently dropped.

The next test that is performed checks if the packet is *some kind of acknowledgement*. The packet fulfills the condition if it acknowledges a sent packet which is the case if it is an explicit acknowledgement (the type field in the header is set to ACK_PKT) or if it is a normal data packet but the transmitting node is nearer to the destination than the current node (more precisely: added more progress to the packet's way to the destination). If it comes out that the packet is *some kind of acknowledgement* the matching packet is removed from the *broadcasts sent buffer* and packet processed is silently dropped.

The last check which is made to decide whether the packet is dropped or forwarded, tests if the node is inside the *forwarding area*, which can be an area of any shape as long as it fulfills the following requirement: Every node inside the area can overhear the transmissions of the sender of the packet and of any other node inside the area (the nodes cannot be farther away from each other than maximal transmission range). If the processing node is not inside the forwarding area drops the packet silently. Every node which has not yet dropped the packet will now calculate a *additional delay*. The delay is in relation to the forward progress the node provides to the packet. The more progress the shorter the calculated delay is. Each node waits for the time of additional delay before broadcasting the packet. While the node is waiting it

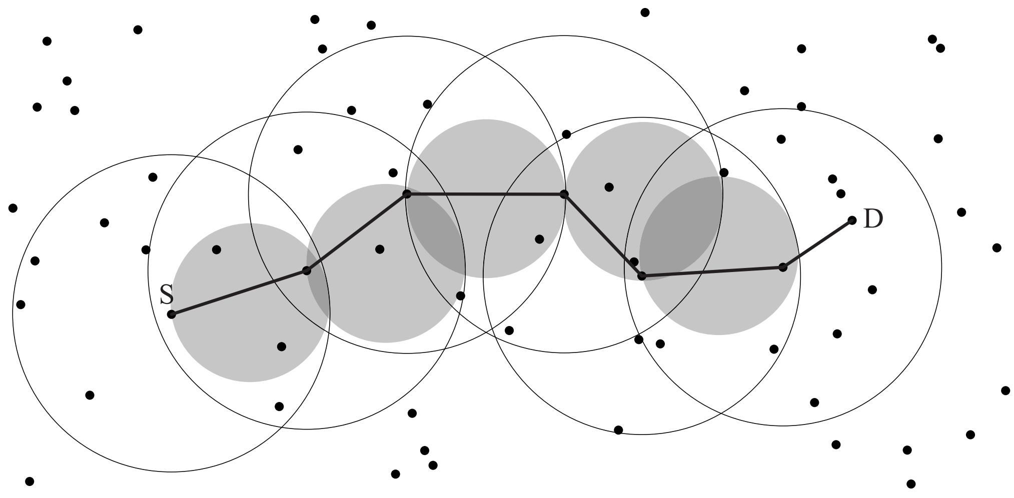


Figure 5.3: BLR greedy mode: The figure depicts the situation where node S sends a message to node D using greedy mode. The forwarding area is set to circle. The actual forwarding area at each hop is drawn by a grey filled circle. The empty circles printed with drawn through lines mark the bounds of the area which is within transmission range distance of the in the center of the circle. The path the packet is forwarded through follows the black printed lines between the forwarding nodes.

is listening if any other node is transmitting the packet. If that happens, the node deletes the packet from its waiting queue and drops it, hence only the node with the most forward progress finally transmits the packet (every node can overhear the broadcasting of the same packet of any other node as only nodes in the forwarding area potentially transmit it and the possibility to overhear transmissions of other nodes in the forwarding area is its the main property). For the case that the node does not overhear a transmission of the same packet, it stores a copy of the packet in its *broadcasts sent buffer* and before broadcasting the packet itself. The previous sender of the packet overhears any transmission of the packet as well and knows that an other node has forwarded a packet. Therefore, the further routing is not its responsibility and it deletes the packet from its *broadcasts sent buffer*. An example of such a greedy mode path is depicted in figure 5.3.

Once the packet reaches the destination, the BLR header is removed and the packet passed on to the next layer. Subsequently the destination sends a short acknowledgement packet (*type* = ACK_PKT) to inform the last hop that the packet reached its destination.

5.2.1 Forwarding Areas

The choice of the forwarding area is vital, as it determines the average number of nodes that potentially forward the packet. The bigger the forwarding area the higher the possibility that it contains at least one node which prevents the greedy mode from failing (not being able to route a packet to the destination). Not only the size but also the shape of the area has its effect: The farther away from the transmitting node the center of gravity of the shape is, the bigger is the

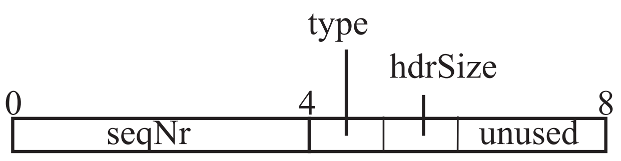


Figure 5.4: BLR acknowledgement header: The fields, their position within the header and their size. The given numbers mark the position of the n^{th} byte in the header. The total header size is 8 bytes.

Table 5.2: BLR acknowledgement header fields: Denotation of the fields in figure 5.4

seqNr	the sequence number of the packet that whose reception is acknowledged
type	specifies the type of this BLR header. Is set to DATA_PKT
hdrSize	the size of this header
unused	not yet used

average progress per hop [15].

In this work three different forwarding areas have been evaluated: *sector*, *releaux triangle*, and *circle* (see figure 5.5 for their shapes). The different areas cover diverse subareas of the circle C defined by the sending node S as center and its transmission range as radius r. These subareas are not the same size. Their size in percentage of the whole circle C:

- sector: $\frac{1}{6} \approx 17\%$
- reuleaux triangle: $\frac{1}{2} - \frac{\sqrt{3}}{2\pi} \approx 22\%$
- circle: $\frac{1}{4} = 25\%$

The circle covers the maximum possible area with the required property but its center of gravity is not as near to the destination as the one of the sector or the reuleaux triangle. The nearer the center of gravity is to the destination, the bigger the average progress a hop in the area provides to the packet. There exists obviously a tradeoff between the size of the area and the average progress per hop. Because the circle covers a bigger area than for example the sector, there exists a chance that there are more nodes inside the forwarding area than there would be if it had the shape of a sector. This additional node potentially has more progress than the others and therefore increases the possibility that the circle contains a node with good progress too. The interference of these properties shows that the circle is slightly better than the other areas [15]. If we take into account that the average number of successful hops in greedy mode is significantly higher if using a circle instead of other shapes [15], the circle suits best our needs.

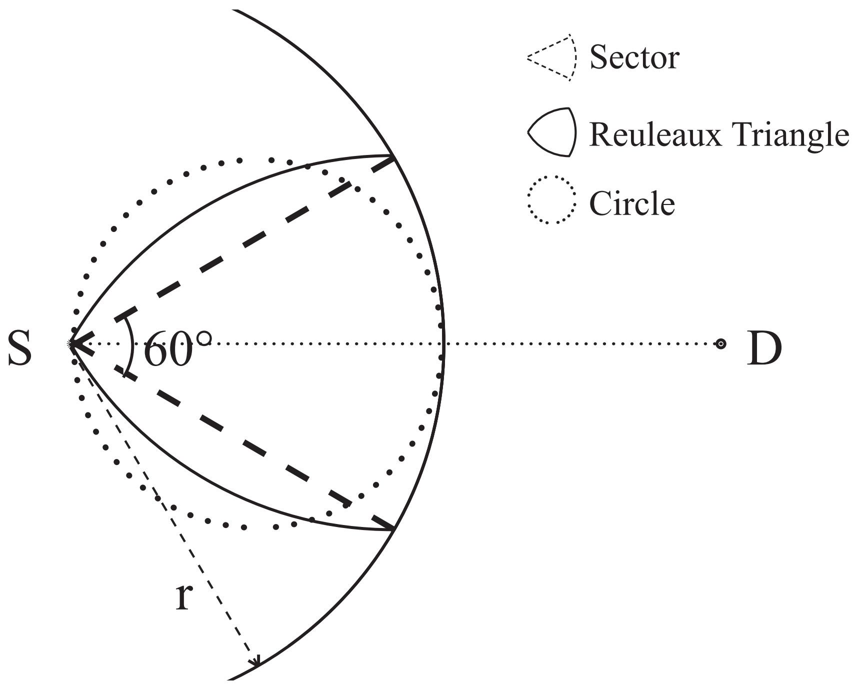


Figure 5.5: Forwarding areas. Each forwarding area lays within S's transmission range r and is centered in reference to the virtual line between S and the destination D.

5.2.2 Additional Delay Calculation

Any additional delay calculation results in a delay between 0 and the *maximal additional delay*. The delay determines the forwarding strategy such as *most forward progress, closest to destination*, *least deviation angle* (called 'Compass Routing' in [13]), and *nearest neighbor*. The *most forward progress* strategy for example can be implemented if the additional delay is smaller the more progress a node adds to a packet's path:

$$additional\ delay = \left(\frac{transmission\ range - progress}{transmission\ range}\right) maximal\ delay$$

Where *progress* is calculated as shown in figure 5.6; *maximal delay* is a shortcut for *maximal additinal delay* and *transmission range* for the *maximal transmission range*. The figure reveals an interesting detail of the most forward progress strategy: The node with the most forward progress, N_2 is not the closest to the destination. Node N_3 is inside the dashed circle C_D centered at the destination and with radius $\bar{N_2 D}$ which means that N_3 is closer to D than the node with the most forward progress. This situation occurs the more frequently the closer the packet gets to the destination, but still does not have really an impact on the overall performance.

Reflecting the forward progress is the main requirement for the additional delay, but not the only one. A problem arises if the network becomes denser and the additional delay is proportional to the forward progress: The forwarding area's distribution of the delays is not uniform (unless it is a rectangle). This means that the probabilities that a node's progress (assumed the

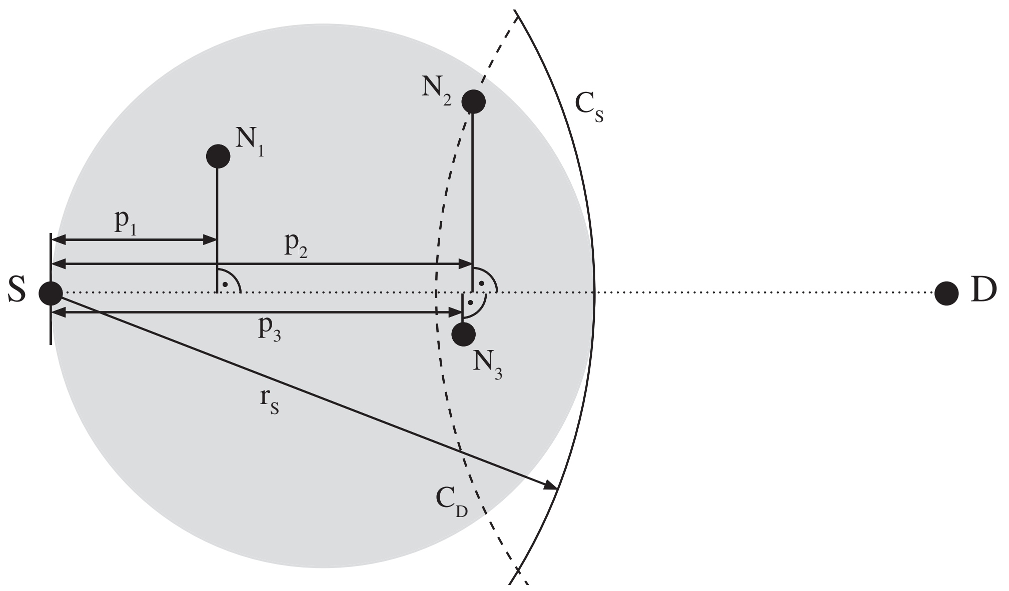


Figure 5.6: Progress measuring. If node S broadcasts a message, every node in the forwarding area (the circle in this case, depicted as grey filled area) calculates its additional delay. To be able to do that, the progress first has to be computed. The progress for a node N_i is the length of the projection of $\bar{SN_i}$ on the straight line between S and the destination D of the packet. Obviously, node N_1's progress p_1 is the smallest, followed by p_3 and p_2.

forwarding area is a sector) lies in the interval $[0, 0.5[$ and that it is in the interval $[0.5, 1[$ are not equal because about 28% of the sector's area is mapped to 50% of the progress range and the other 72% of the area are mapped on only 50% of the progress' range. This increases the probability that two nodes with good progress start transmitting the packet at the same time and therefore interfere each other's transmission. There are different possible consequences:

- The two sending nodes were the only one in the forwarding area: No other node will forward the packet and greedy mode fails.
- There was an other node in the forwarding area which received the signal of one of the nodes clear enough and therefore cancelled its own transmission: Greedy mode fails.
- There was an other node in the forwarding area but it received the interfered signal and could not decode it and hence did not cancelled its own transmission: The other node will forward the packet an the only loss is some progress.
- Each of the sending nodes reach a node that is able to decode the signal as it is far enough away from the second transmitting node but both receiving nodes are out or each other's transmission range: Greedy mode does not fail but the packet is duplicated.
- A combination of the mentioned options.

The given numbers apply to the sector but similar problematic distributions of the additional delay if calculated proportional to the progress, can be identified for every forwarding area shape, except the rectangle. Therefore the additional delay calculation should take the forwarding area's shape into account.

In a more general analysis [16] it has already been shown that exponentially distributed random timers, for example following the formula

$$additional\ delay = \left(\frac{e - e^d}{e - 1}\right) maximal\ additional\ delay$$

for the additional delay can improve performance if the number of responses is used as measuring criteria (where $d = \frac{source\ to\ current}{maximal\ transmission\ range}$, *source to current* is the distance from the last hop to the processing node).

5.3 Backup Mode

If a node does not receive any kind of acknowledgement for a packet sent in greedy mode during BROADCAST_TIMEOUT, it takes the packet from its broadcasts sent buffer to forward it in backup mode. Switches the packet to backup mode consists in changing the header from BLR

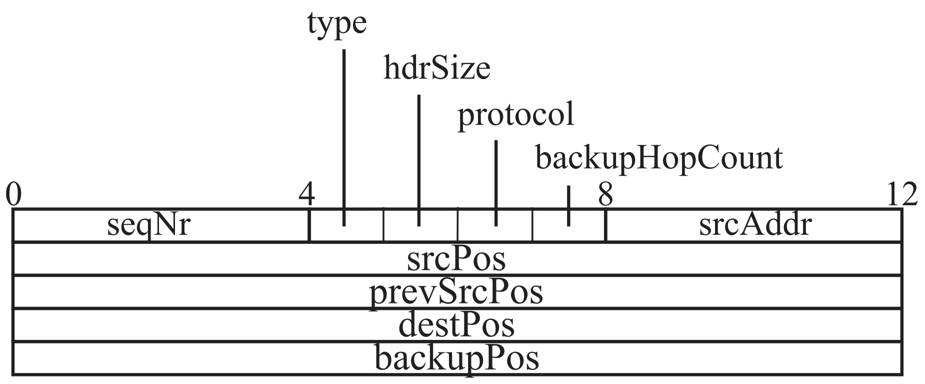


Figure 5.7: BLR backup header: The fields, their position within the header and their size. Each line contains 12 bytes. The given numbers mark the position of the n^{th} byte in the header. The total header size is 60 bytes.

data header to BLR backup header (thus adding the *backupPos* field), setting the backupPos field to its current position, and increasing the *backupHopCount* field by one. After the packet has been altered in such a manner, it is forwarded according to the applied backup mode. The technical report [1] describes two backup modes which make use of the whole transmission range of the processing node:

- *Clockwise backup mode* uses additional delays, similar to the greedy mode, but without forwarding areas. Section 5.3.1 describes the exact mode of operation.

- *Request response backup mode* let the processing node make the decision, to which node the packet is forwarded, which needs some mechanism to become aware of neighboring nodes. In section 5.3.2 the details of the *Request response backup mode* will be explained.

Table 5.3: BLR backup header fields: Denotation of the fields in figure 5.7

seqNr	a sequence number which together with the originating node's IP address makes this data packet identifiable
type	specifies the type of this BLR header. Is set to DATA_PKT
hdrSize	the size of this header
protocol	the saved IP protocol field
backupHopCount	number of hops completed in backup mode
srcAddr	the IP address of the transmitting (current) node
srcPos	the position of the transmitting (current) node
prevSrcPos	the position of the previous transmitting node
destPos	the position of the destination
backupPos	the position at which this packet entered the backup mode

A packet is routed in backup mode as long as the forwarding node's position is farther away from the destination than the node at which the packet has entered backup mode. The first node closer to the destination than the location stored in *backupPos* changes the BLR backup header back to BLR data header and tries to forward the packet in greedy mode.

5.3.1 Clockwise Backup Mode

The clockwise backup mode does not work totally different from the greedy mode as the 'selection' of the next hop is implicitly done by timers too. But in the clockwise backup mode every node which could receive the packet potentially forwards it, since there exists no limiting forwarding area. Once the angle α has been computed according to figure 5.8, the additional delay is calculated proportional to α:

$$additional\ delay = \frac{\alpha}{2\pi} maximal\ additional\ delay$$

Calculating the additional delay like that has the effect that the nodes with forward progress add shorter delays than the ones which do not have forward progress. If no node with forward progress exist, loosing progress is just inescapable.

When the node with the shortest additional delay broadcasts the packet, not every node that has set a delay timer is able to receive it (in contrary to the situation when using a forwarding are). Therefore the node which previously transmitted the packet broadcasts a Successful Transmission Notification message (STN). Every node receiving this STN checks if it has a corresponding timer set and if so, it cancels it. A final remark, concerning the maximum length of additional delay: It has to be longer than for the greedy mode, as there are potentially more nodes setting a timer and some nodes can only be notified about the successful forwarding with the STN.

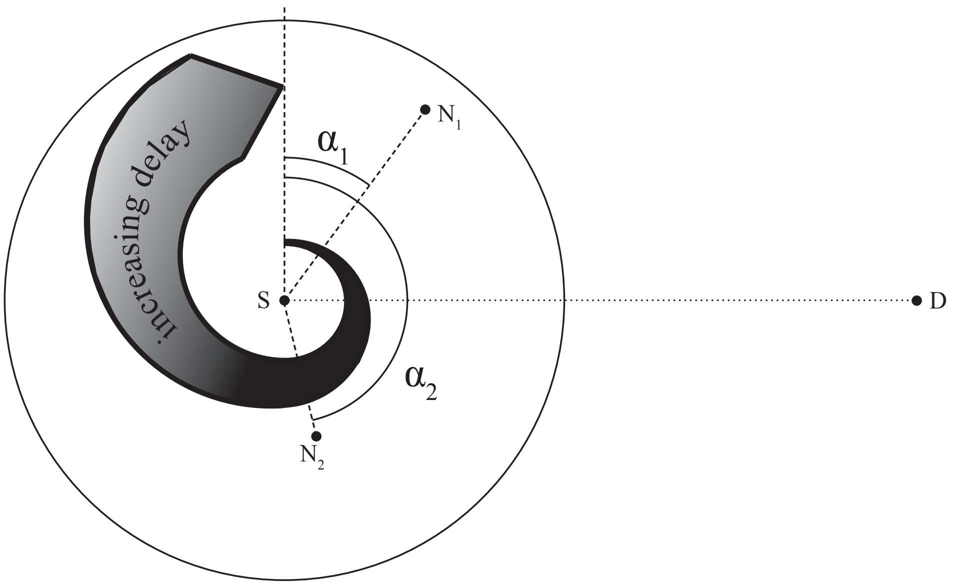


Figure 5.8: Clockwise delay: The line rectangular to the straight line connecting the transmitting node S and the destination D is the starting point for measuring α. α is clockwise measured from this line to the line between S and the processing node N_i. Thus α_1 is the angle for node N_1 and α_2 is the angle for node N_2 respectively. The additional delay is just proportional to α.

Unfortunately, it is quite easy to depict as scenario where clockwise backup mode fails, although there exists a path from the source to the destination. Figure 5.9 illustrates such a scenario. There exists no easy way to remove this weakness from this backup mode. Therefore it will not be taken into account in the further development of BLR.

5.3.2 Request Response Backup Mode

The request response backup mode is similar to the *perimeter mode* in section GPSR 4.1. Since BLR does not use beaconing, a node needs to send a *hello request* to ask the neighboring nodes to emit a position announcement message. After collecting these messages, the processing node extracts the planar graph out of the position information and forwards the packet according to the right hand rule using MAC layer unicast.

Table 5.4: BLR hello request header fields: Denotation of the fields in figure 5.10

seqNr	the sequence number of the packet that will be forwarded in backup mode
type	specifies the type of this BLR header. Is set to HELLO_REQ_PKT
hdrSize	the size of this header
unused	not yet used

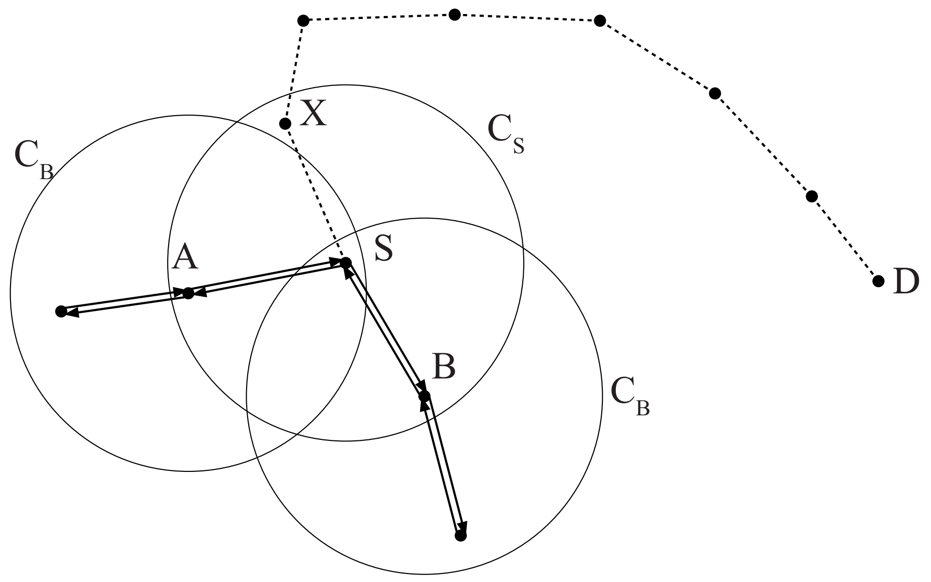


Figure 5.9: Clockwise backup mode failure: Clockwise backup mode will either loop the packet through the path indicated by drawn through arrows until the TTL exceeds, or a node (in the supposed implementation node S) drops the packet. But this backup mode will never find the path from S to D (indicated by dotted lines). There exists no possibility that node S can become aware of node X if clockwise mode is used as described. The circles C_i depict the transmission ranges of the nodes i.

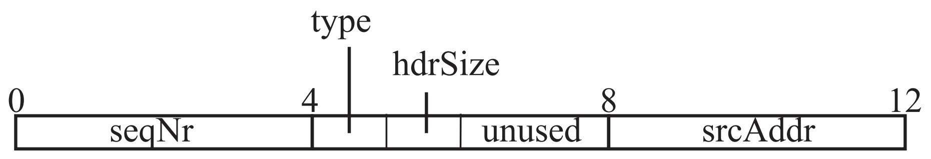


Figure 5.10: BLR hello request header: The fields, their position within the header and their size. The given numbers mark the position of the n^{th} byte in the header. The total header size is 12 bytes.

5.4 BLR Performance Improvements

In ideal networks, the described algorithm finds a path if there exists one and the packet is delivered. But there is still space for improvements. Two kinds of improvements exist: One addresses the imperfection of the algorithm itself (adding probably more delay), the other category of improvements tackle the problems coming up as a result of the conditions in a mobile ad hoc network.

The most serious problem is packet loss in greedy mode. This can occur if two nodes introduce similar additional delays and the backoff time introduced by the 802.11 MAC layer

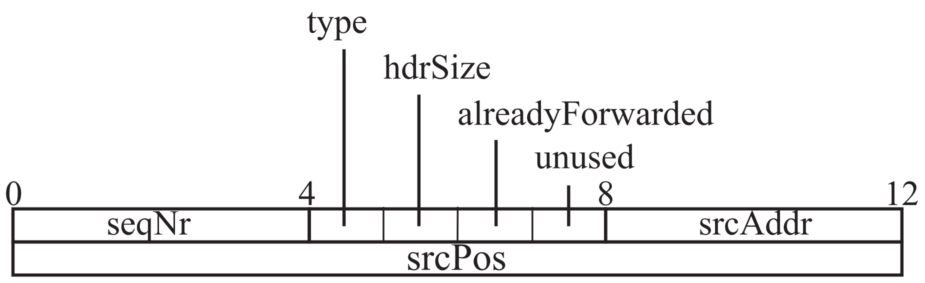


Figure 5.11: BLR hello header fields: The fields, their position within the header and their size. Each line contains 12 bytes. The given numbers mark the position of the n^{th} byte in the header. The total header size is 24 bytes.

Table 5.5: BLR hello header fields: Denotation of the fields in figure 5.11

seqNr	the sequence number of the packet that will be forwarded in backup mode
type	specifies the type of this BLR header. Is set to HELLO_PKT
hdrSize	the size of this header
alreadyForwarded	indicates that the sending node already received the corresponding data packet and forwarded it
unused	not yet used

protocol results in simultaneous broadcasting. The problem of packet loss in greedy mode leads to the problem of packet duplication. If a packet is lost the broadcasting node will not receive any kind of acknowledgement message and therefore try to send the packet in backup mode. Because both nodes will act like that, the packet is duplicated. Packet duplication can also occur if a node does not notice an other node forwarding the packet and therefore forwards the packet too. This can happen due to different transmission ranges or due to mobility of the nodes. Packet duplication leads to redundant network traffic which increases the overall network load and therefore the probability that contention occurs.

5.4.1 Advanced Acknowledgement Semantics

One of the first improvements is changing the acknowledgement handling. Up to this point, packets contained a *prevSrcAddr* field containing the address of the forwarding node one hop ago. With this address in the packet header it is easy to determine if a packet acknowledges a sent one. The drawback is that a node receiving a packet (in greedy mode) from a node nearer to the destination but with a different *prevSrcAddr* value, does not consider the packet as acknowledgement. This is no problem, as long every packet is always forwarded and every forwarding transmission is received by the previous hop node. As soon as this cannot be assumed anymore, it is superior to look at the sending node's position and consider a packet to be an acknowledgement if the sending node provides more progress to the packet's path than the node doing the acknowledgement check.

Using this mechanism for acknowledgement, duplicated packets and switching to backup mode can be prevented in the case that a node could not receive the signal of the next forwarding transmission (or the packet has not not been forwarded) but could overhear another transmission of the same packet. This situation for example occurs if a packet has already been duplicated and the next node uses a traffic history (see section 5.4.2 for details) and therefore recognizes the duplicated packet and does not forward the same packet twice, even if the previous hop was a different node.

5.4.2 Traffic History

A traffic history is useful to eliminate duplicated packets. The basic idea is that each node maintains a short traffic history. The history does not need to contain the whole traffic but some characteristics of the packets forwarded or being *some kind of acknowledgement* (according to section 5.2). If it comes to routing a packet, this history is used as additional criteria for the forwarding decision. Packets in greedy or clockwise backup mode are only forwarded if there exists no corresponding entry (same *type*, *sequence number*, and *IP address of the originating node*). Request response backup mode packets are forwarded even if a corresponding entry exists, as long as the *backupPos* field of the packet contains the coordinates of a location closer to the destination than the position of the processing node. The second option which leads to forwarding such a packet is the condition that the *backupPos* fields are equal and the entry's *backupHopCount* field contains a value lower than the one of the packet.

5.4.3 Tight MAC Layer Interaction

In a scenario where two nodes add about the same additional delay various problems arise. If they start transmitting at the same time, the packet cannot be received by all potential next hop nodes without an error, which may result in falling back to backup mode, although there probably are nodes within the forwarding area. That problem is addressed in section 5.4.4. This section provides a mechanism to avoid that two nodes which add about the same additional delay both transmit their packet although they could overhear each others transmission.

The depicted problem occurs due to processing delay and additional delay inserted by the *backoff* on the MAC layer (see section 2.2 for details about backoff delay): Once the packet is queued for transmission, it is not within the routing layer's control anymore. One consequence is that it normally cannot be cancelled if the transmission is delayed due to an other node forwarding the same packet. The solution presented is called *tight MAC layer interaction* and prevents multiple transmissions of the same packet which can lead to duplicated packets: Whenever a packet transmission has to be retained due to busy physical medium, the packet is passed back to the routing layer to affirm or cancel the transmission.

5.4.4 Cancelling Hello Messages

Packet duplication for example occurs if nodes do not receive *some kind of acknowledgement* for a packet sent in greedy mode although it has been received and forwarded. There exists a possibility to avoid this if using request response backup mode. Of course the node which forwarded the packet has still to be in transmission range. If this is the case and it receives the hello request, it sends a tagged hello message (*alreadyForwarded* field is set to 1) and the node requesting it will abort its backup mode transmission and consider the packet to be delivered successfully.

5.4.5 Request Response Backup Mode Improvements

To prevent unnecessary backup mode fall back, the node switching from greedy to backup mode does not strictly apply the right hand rule: It first searches for *any* node located closer to the destination than the current one. If there exist more than one node fulfilling this requirement, the one with the most forward progress is select as next hop. This mechanism helps to minimize the occurrences of packets routed over many hops in backup mode, although a path with each hop providing forward progress exists. A further improvement can be achieved if the processing node first searches for nodes in the forwarding area before for any node with forward progress. This addresses the problem of two nodes transmitting the packet at the same time and therefore interfering each others transmission.

The second improvement tackles a problem that occurs due to mobility and changing transmission conditions. A node forwarding a packet in request response backup mode sends it to one of the nodes which have sent a hello message. It selects the next hop by applying the right hand rule on the planarized graph (see section 5.3.2). In theory forwarding the packet to this node should be assured as the two nodes already exchanged packets. Due to node mobility or changed transmission conditions forwarding though may fail. In this case the forwarding node just sends the packet to the next node in the graph using the right hand rule. This procedure can be applied until the node tried to send unsuccessfully to every node in the graph.

5.4.6 Using Unicast Transmissions

The problems mentioned in section 5.2.2 would not occur if the packets were sent using MAC layer unicast transmission. Because BLR does not use beacons no neighbors are known which is a must to transmit packets using unicast. The solution is quite simple: If a node receives *some kind of acknowledgement* for a sent packet, it extracts and stores the position and the address of the sending node. This leads to some kind of routing table, called *route cache*. If another packet for the same destination has to be forwarded, the node sends it to the known next hop using unicast, but only if the following condition is fulfilled: The potential next hop's location stored in the route cache is within $transmissionrange - safetymargin$ from the current position of the processing node. This safety margin reduces the probability that a cached route no longer exists due to mobility of the nodes. If nevertheless the selected next hop node is not reachable, the packet is normally broadcasted and the invalid entry removed from the route cache.

Obviously the collected information is not valid for ever but expires after a *route timeout*.

Collecting routing information and choosing a next hop can be enhanced. Every node can overhear the traffic and extract the needed information out of the collected packet. Selecting the next hop can be performed using a node which already forwarded packets to the same destination, this is the easiest selection criteria. If nodes collect many route information it probably would be better to choose a next hop according to its position.

Using unicast transmissions can lower the impact of the known weakness of the BLR protocol described in section 5.5 and probably provide better end-to-end delay.

5.5 Known Weaknesses

Using 802.11 as MAC layer protocol has a drawback: The randomly chosen backoff time can interfere with the additional delay as it is a delay too. If no contention occurs, the backoff is in the interval $[0\mu s, 620\mu s]$. Because the backoff is chosen from a uniform distribution, the average backoff is $310\mu s$ which is about $\frac{3}{20}$ of a *maximum additional delay* of $2ms$ (which is one of the tested delays in section 9). This additional delay introduced by the MAC is not negligible. It is not that serious if the chosen maximum additional delay is higher. But higher maximum additional delay results in undesired higher end-to-end delay (see section 9.2.2). If contention occurs in the network, the backoff interval is increased up to the maximum of $[0ms, 20.46ms]$. It is obvious that this severely interferes with the additional delay if the maximum additional delay is not impractically big.

At the moment BLR comes into trouble if position inaccuracy occurs. The first point position inaccuracy comes into play is when a node checks if it stays within the forwarding area. In these situations position inaccuracy can lead to duplicated packet. The duplications occurs because a packet transmitted by a node outside the forwarding area cannot be received by every node inside the forwarding area. The position inaccuracy is not that severe when the additional delay is computed. It can result in packet forwarding by a node not providing the maximal possible progress to the packets, potentially resulting in additional hops. Position inaccuracy of the location looked up for the destination is the worst case: It can lead to packet loss as the packet cannot be delivered if the destination node is not situated at the indicated position. A possible solution is described in section 10.2.

Chapter 6

Mobility models

Mobility models are used in simulators to generate the nodes' movement patterns. The choice of the mobility model to evaluate a routing protocol for MANETs and the knowledge about the models properties are crucial to avoid misinterpretation of simulation results. There exist several models with varying main focuses and complexity. For some of them exist modifications, trying to reduce known weaknesses. The description focuses on the basic idea of the models and mentions modifications if they are noteworthy.

6.1 Entity mobility models

Entity mobility models generate the movement pattern of each entity, nodes in the case of MANETs, without consideration of other entities in the simulation. Thus no dependency between the movement patterns of different nodes exists.

6.1.1 Random Walk

A node randomly chooses a direction and a speed for its next move. The direction is chosen from $[0, 2\pi]$ and the speed from an interval given by *minimal speed* and *maximal speed*. The node will move using this parameters for either a constant time or a constant distance before restarting this procedure. If a node reaches the simulation area boundary during its movement, it 'bounces' off the border of the simulated area with an angle determined by the angle of incidence and continues its movement along this new path.

The main property of the generated movement patterns is that the nodes move around their initial position, which is the reason the Random Walk mobility model is sometimes referred to as Brownian Motion. Obviously the extent of movement is determined by the possible speeds and the decision if the nodes move for a constant time or a constant distance. Travelling for a constant distance intensifies the property of moving around the origin.

6.1.2 Random Waypoint

A randomly chosen destination in the simulation area and a randomly chosen speed in the interval [*minimal speed*, *maximal speed*] determines the next move of a node as it moves to the chosen destination by a constant speed. Upon arrival it pauses for *pause time* before making for a next place. There exist two implementation specific characteristics. The first concerns the initial setup: One implementation begins with a pause at the initial location, the other begins with all nodes moving. The second difference is in regard to the *pause time*: It can be derived from a uniform distribution from *minimal pause time* to *maximal pause time* or it can always be the same which makes it a tweaking parameter for the dynamics of the model.

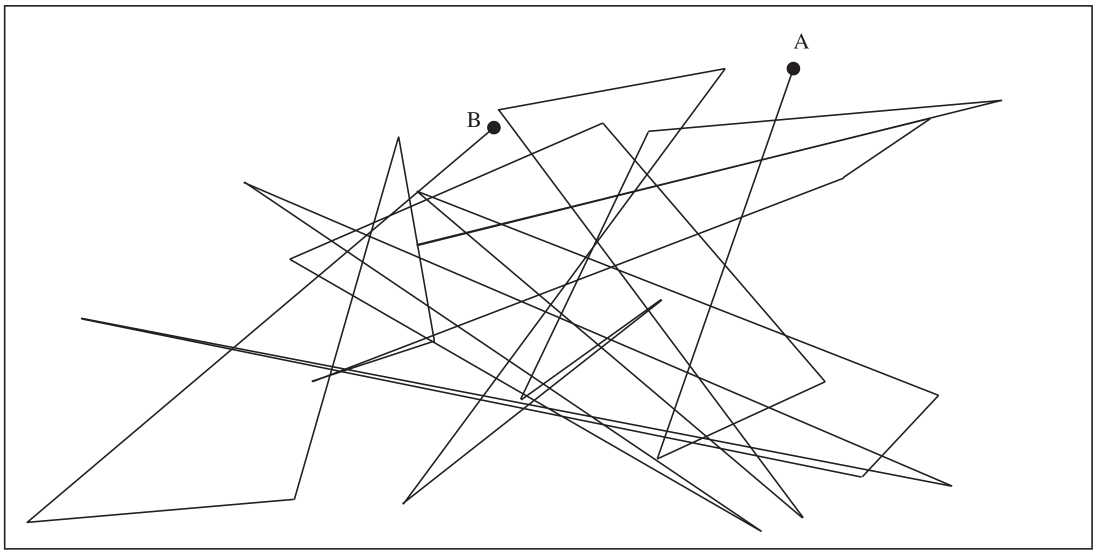


Figure 6.1: Random waypoint mobility model movement pattern: The picture shows a path of a node when using random waypoint mobility model, starting at point A and ending at point B. There is no concrete scale printed in this picture, as the pattern is not related to the size but only to the ratio of the width and length of the area. The characteristic of this mobility model, that nodes are more often near the center than at the border of the simulation area is already anticipatable.

The resulting movement pattern may vary quite a lot: Pause times over 20 seconds result in stable networks with few link changes per node even at high speeds [17]. An other property of the Random Waypoint model is the fact, that if the nodes initially are distributed according to one of the commonly used distributions (random, uniform, grid), the distribution changes over time to a distribution that can be approximated by

$$f(x,y) \approx f(x)f(y) = \frac{9}{16x_m^3 y_m^3}\left(x^2 - x_m^2\right)\left(y^2 - y_m^2\right)$$

where the nodes are more likely near the center than the edge of the simulation area (x_m and y_m being the x and y dimension respectively)[18]. This centering property can already be anticipated in figure 6.1, depicting a sample movement pattern. This changing in distribution can lead to unexpected effects and misinterpretation of results [19]. The easiest way to avoid this pitfall is to take into account only the simulation results over the period of time the nodes are distributed with the steady-state distribution. Because it's hard to guess the point in time the distribution will be close enough to the steady-state distribution for accurate simulation results,

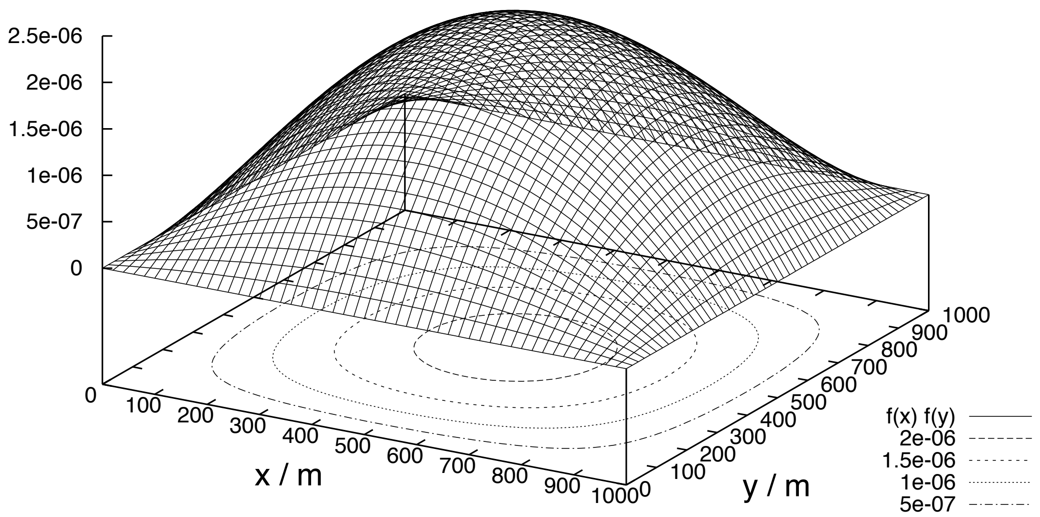


Figure 6.2: The spatial distribution of the random waypoint mobility model: The figure shows the two dimensional plot of $f(x)f(y)$ approximating the node's final distribution in the random waypoint model. Obviously, it is more probable that a nodes stays nearer to the center than to the border. (fig. copied from [18])

the superior way is to sample the start parameters for the nodes from the computed steady-state distribution [20].

6.1.3 Random Direction

A node chooses a random direction which it follows at a randomly chosen speed until the border of the simulation area is reached. At the border the node pauses for a specified time before choosing a new speed and direction, an angle in the interval [0, π], and restarting the movement sequence [21].

The main goal of this mobility model is to remove the changes in node distribution from the Random Waypoint mobility model [21]. This results in nodes well distributed, but in unrealistic behavior of pausing only at the edge of a given area.

There exists a slightly modified model called Modified Random Direction mobility model [21]. In this modified version nodes do not follow the randomly chosen direction up to the next border but move only a random length on the direction before pausing an choosing a new one. The behavior of this modified version is identical to the Random Walk mobility model altered by pause times.

6.1.4 Boundless Simulation Area (BSA)

In the BSA a relationship between previous and current movement direction and speed exists in order to limit the change in direction and speed per time unit to generate more realistic movement patterns. An other speciality of this mobility model has given it its name: the rectangular simulation area is folded to form a torus with the effect that nodes moving out of the simulation area enter it again at the opposite side thus creating a boundless simulation area (e.g. a node leaving the simulation area at the top enters at the bottom again) [22].

Since the changes in direction and speed are limited, the resulting moves lack of abrupt changes in direction and speed, which makes the movement patterns more realistic. This realism only persists, if the components working with the resulting movements are aware of the speciality of the boundless simulation area. If they are not, the boundless property results in ‘teleported’ nodes [19].

6.1.5 Gauss-Markov

The Gauss-Markov mobility model was designed to adapt to different levels of randomness via one tuning parameter [19]. This is achieved by taking the previous speed and movement direction into account when calculating the new direction an speed. This means that s_n and d_n, the speed and direction calculated in the n^{th} iteration of calculating new movement parameters, are calculated based on the speed and direction of the $(n-1)^{st}$ movement values:

$$s_n = \alpha s_{n-1} + (1-\alpha)\bar{s} + \sqrt{(1-\alpha^2)s_{x_{n-1}}}$$
$$d_n = \alpha d_{n-1} + (1-\alpha)\bar{d} + \sqrt{(1-\alpha^2)d_{x_{n-1}}}$$

where $\bar{s}$ and $\bar{d}$ are constants representing the mean values of speed and direction. $s_{x_{n-1}}$ and $d_{x_{n-1}}$ are random variables from a Gaussian distribution. α, where $0 \leq \alpha \leq 1$, is the parameter to specify randomness. $\alpha = 1$ leads to linear and $\alpha = 0$ results in completely random, Brownian motion. If a node gets close to the edges of the simulation area, the possible directions are restricted to those leading him off the edge. This procedure prevents undesirable effect at the edge of the simulation area.

6.1.6 City Section

In the City Section mobility model the nodes are restricted to move only on given streets, which can be the ones of a real city section. The streets may have different properties such as speed limits, maximum number of nodes per kilometer, and other rules which exist in the real world. Nodes start at a given location and move along the streets to a randomly chosen new location. Different path-finding algorithms can be applied to search for a path for a node from its current to its new location [23].

The nodes may have quite realistic movement patterns, according to the path-finding algorithm, the rules and the city ‘map’.

6.2 Group mobility model

Group mobility models try to model the movement patterns of groups of entities which are linked to each other in some way, thus some kind of dependency between the moves of the entities must be included in the model.

As explained in detail later, several known group mobility models can be modelled using the generic Reference Point Group Mobility model. That is the first reason only one group mobility model is presented. The second reason is the fact that group communication patterns are not a specific target of BLR routing deployment, hence group mobility models not in the focus.

Reference Point Group Mobility (RPGM)

The RPGM model represents the random motion of a group of nodes as well as the random motion of each individual node within a group [24]. A logical center of the group characterizes the group movement pattern. Each node has a individual reference point which is linked to the logical center as well as a movement pattern around its reference point. Thus two entity mobility models are involved in the RPGM: one for determining the moves of the logical center and one for the motion of the nodes around their reference points.

The RPGM is a generic group mobility model and the movement patterns it generates depend on the mobility model applied to the logical center and the individual reference points and further restrictions on the reference points or the nodes.

Applying particular restrictions to the reference points result in mobility models known as the *Column*, the *Nomadic Community* and the *Pursue* mobility model [19].

6.3 Discussion

In general there exists a tradeoff between the complexity of the model and the degree of accuracy to reality. Another fact influencing the choice of a mobility model is its pervasiveness in the research community, since the results should be comparable to other findings.

Group mobility models fit some particular scenarios really well, though are not commonly used. The City Section and the Gauss-Markov mobility model seem adequate for a lot of scenarios where MANETs are supposed to be useful, yet not commonly used neither. The Random Direction mobility model results in unrealistic node distributions whereas the BSA has the drawback of only being useful in environments where the propagation model is aware of the boundless area property. Random Walk can produce movement patterns which yield to semi-static networks (networks where link changes occur quite rarely) if the nodes are only allowed to travel in the same direction during a small amount of time or for a short distance [19]. The same applies to the random waypoint mobility model if the pause times are getting longer. A big advantage of the random waypoint mobility model is the fact that it is widespread and easy to apply, however the changing distribution problem should be solved (as proposed in section 6.1.2) and the resulting distribution kept in mind when interpreting simulation results.

Chapter 7

The Network Simulator Qualnet v3.6

BLR has been evaluated using the network simulator Qualnet v3.6. Qualnet v3.6 is a commercial network simulator from Scalable Network Technologies. This chapter will introduce Qualnet v3.6, explain its structure and give an overview of its most important specialities.

7.1 Introduction to Qualnet v3.6

Qualnet is a discrete event simulator (DES) implementing various signal propagation models, protocols on each layer of the network stack, and mobility models. Discrete event simulators model the continuous time by breaking it down into adequate time slices (Qualnet: $1ns$), resulting in a discrete environment. Events can only occur at those discrete moments in time. The core of a discrete event simulator is a scheduling mechanism providing the facility to schedule events. Hence everything modelled within such a simulator is modelled using events.

Qualnet v3.6 is implemented in C and uses various configuration files through which parameters at the different layers can be configured. An API on each layer of the network stack is provided. If the provided API does not fit one's needs it is possible to change the code of existing protocols: The source code for the protocols is provided with Qualnet. Not provided in source code is the core. It consists mainly of the scheduling mechanism and the signal propagation on the medium (wireless or wired).

To get something out of the simulations, facilities to analyze what is happening during the simulation are needed. Qualnet v3.6 offers two mechanisms to analyze the simulations:

- Trace files: The trace mechanism is still under development. The format of the generated trace file is quite cumbersome to handle. Tools for analyzing trace files are not provided, nor a specification of the trace file format. Trace output generation possibilities are inflexible.

- Statistic files: The statistic file provides extensive per node statistic for each layer. A well designed API makes it easy to print output to the statistic file.

7.2 General structure of Qualnet v3.6

Qualnet v3.6 is implemented in pure C, but the structure of Qualnet v3.6 emulates C++: The first evidence is the fact that there does not exist any global variable (at least in the delivered source code). The second and more important evidence is the presence of a central entity, the *Node* structure. It models a node in the network. Such a node contains its own network protocol stack, each layer's state, physical communication facility, and position information. The settings of these elements determine if the node is a router, a switch, a mobile node or anything else. The presence of this entity itself does not constitute the C++ emulation, but its use reveal it all the more: A pointer to a *Node* structure is the first parameter of *every* method of the API of Qualnet v3.6. Thus instead of a method call to a node object, a pointer to a *Node* structure is passed as argument to each method. The effect is the same because the called method will work only on the data of the object and the passed *Node* structure respectively.

Concerning the network stack, the most important part in a network simulator, Qualnet v3.6 is well structured. Every logical part is modelled as independently from other parts as possible. If some interaction can only be implemented by changing source code in implementations on other layers, this is done minimizing these changes. This modularity makes it quite easy to implement new protocols on the different layers.

7.3 The *Node* structure

As already mentioned the *Node* structure is the central entity in Qualnet v3.6. It contains two sorts of information: Data closely related to the simulation core like the node ID, host name, seed values for random number generation for this node, communication facilities and their properties, tracing options, partition information, and data internally used for scheduling events for this node. The other kind of data in the *Node* structure is assigned to the different parts that can be implemented when using the simulator. This includes the mobility data and most important, data storing the different network stack layers' states. These states are, from a developers perspective, the most important elements of the *Node* structure.

In the case the information of the *Node* structure passed as argument does not suffice, it is possible to access data from nodes other than the passed one: The nodes constitute a double linked list as every *Node* structure element contains a pointer *prevNodeData* and *nextNodeData* pointing to the previous and next node respectively. Hence, every node's data can be accessed if necessary. This can be useful if for example the current position of a specific node is needed.

A node's data can be accessed through an API. This is useful as for example the mobility data can not be read directly to obtain the current position of a node, but the current position has to be interpolated between stored locations.

7.4 Packets and Events in Qualnet v3.6

When talking about packets, messages, and events in a DES, attention has to be paid not mixing up the different meanings of these terms. Even when in case they are sometimes used synonymously. It has already been mentioned that in a DES every action taking place at a specific moment in simulation time is modelled as event. Qualnet models these events using a structure called *Message*. The factor making understanding of *Messages* difficult is, that *Messages* serve two purposes:

- A *Message* models an event according to the DES terminology: *Messages* can be passed to the scheduling mechanism which results in method invocation at a specific moment in simulation time.
- *Messages* are used to model packets in the network and therefore can be passed as arguments between network layer methods.

These two aspects need a detailed explanation as their understanding is crucial to implement anything in Qualnet v3.6. The following text explains the different usages of *Messages*. Every message handling method mentioned in this section is explained in detail in section 7.4.

Table 7.1: *Message* elements used for event modelling.

layerType	The network layer which will receive the message
protocolType	The protocol which will receive the message in the layer
eventType	Specifies the event this message is modelling
cancelled	Indicates that an event has been cancelled
info	Contains data that will be needed when the event takes place
infoSize	Indicates the size of the info field

The *Messages* structure contains various fields which are used to model an event (see table 7.1 for details). At the time a scheduled event takes place, the *Message* modelling the event is passed to the event processing method at the layer indicated by the *layerType* field. This method forwards the *Message* according to the *protocolType* to the corresponding protocol where specific actions will take place, according to the event the *Message* is modelling, indicated by the *eventType* field. But such a message does not only trigger an action but can also contain data needed to accomplish the activity. That is the moment when the *info* field comes into play. This field is the only possibility to preserve information from the current state for later use through the means of a *Message* (apart from misusing the packet field, introduced in table 7.2). Thus, after a message has been created, any information that will be needed later has to be copied to the *info* field (after allocating enough space by calling *MESSAGE_InfoAlloc*). Scheduling an event is done by calling the *MESSAGE_Send* method. Scheduled events can be cancelled (removed from scheduling) by the means of the *MESSAGE_CancelSelfMsg* method. When *Messages* are used to simulate packets, the fields listed in table 7.2 of the *Message* structure are used. Space for the payload of a packet can be allocated with a call to the *MESSAGE_PacketAlloc* method. When *MESSAGE_PacketAlloc* has been called, the packet can be resized using *MESSAGE_ExpandPacket* and *MESSAGE_ShrinkPacket* to enlarge and shorten it respectively. In the majority of cases a packet is resized due to header adding or removal at the different layers. To do this, the *MESSAGE_AddHeader* and *MESSAGE_RemoveHeader* methods should be called

Table 7.2: *Message* elements used for packet modelling.

packet	The packet as seen by a particular layer. This field an the packetSize field are updated if the packet's size is changed using one of the various methods to resize packets
packetSize	Indicates the size of the simulated packet.
virtualPayLoadSize	The size of virtual payload. Virtual payload is taken into account if calculating transmission delay. It is used for payload that does not really need to be transmitted byte by byte but should affect the transmission. Therefor only its size has to be stored and not the payload itself.

instead of *MESSAGE_ExpandPacket* and *MESSAGE_ShrinkPacket*, or packet tracing at the protocol level which performs the resizing is not possible.

Once such a packet *Message* is assembled it can be easily promoted through the network stack calling methods of adjacent layers and passing the message as argument. A good example is the *NetworkIpReceivePacketFromTransportLayer* method of the IP implementation. It has a parameter of type *Message* through which the transport protocol calling the method can pass the packet to forward. This is the first way to hand over packets between layers. Using this technique every additional information needed on the next layer can be passed as function argument.

Processing layer	**Receiving layer**	**scheduling mechanism**	**function calling**
Application	Transport	yes	no
Transport	Network	yes	yes
Network	Link	yes	yes
Link	Physical	no	yes
Physical	Link	no	yes
Link	Network	no	yes
Network	Transport	no	yes
Transport	Application	yes	no

Table 7.3: The different options to pass packets between layers: Each layer is listed with its adjacent layers. The table indicates if a packet can be relayed between them, scheduling the packet as event and by calling a function and pass the packet as argument respectively.

Passing messages as arguments is one way to propagate packets through the network stack. The second technique makes use of the dual meaning of a *Message*. If the next layer supports events modelling the receipt of a packet from the processing layer, the *layerType*, *protocolType*, and *eventType* fields of the message can be set accordingly (see table 7.1 for details). After setting these fields the message can be passed to the scheduling mechanism as a normal event calling *MESSAGE_Send*. The further processing of the event can occur immediately (if a delay of 0 is passed to the *MESSAGE_Send* method) or after a specified delay (for example to simulate processing delay). To achieve the forwarding of a packet on the network layer from the transport layer, the *NetworkIpReceivePacketFromTransportLayer* has been mentioned above. Scheduling a event results in the same processing, as this event implements the corresponding functionality.

Although these two techniques exists in Qualnet v3.6 to propagate packet, most layers offer only one option. The different layers and their possibilities to receive packets from other layers are listed in table 7.3.

Message API

The following list contains a selection of the most important functions of the API to handle *Messages*. There exist some macros not listed, to access data fields of the *Message* structure. It is advised to use them for consistent code, but it is not necessary.

- *Message *MESSAGE_Alloc(Node *node, int layerType, int protocol, int eventType)*
  Creates a new *Message* for the given layer and protocol. The *eventType* sets the corresponding field in the *Message* (see table 7.1 for details).

- *Message *MESSAGE_Duplicate (Node *node, const Message *msg)*
  Duplicates a message duplicating *every* element of the message.

- *void MESSAGE_Free(Node *node, Message *msg)*
  Because *Messages* are not allocated as normal structures but using the *MESSAGE_Alloc* method, the cannot be normally freed as well. This method has to be used to free *Messages*.

- *void MESSAGE_InfoAlloc(Node *node, Message *msg, int infoSize)*
  Allocates *infoSize* bytes of memory for the info field of the passed *Message*. It is important to use this method to allocate memory for the info field because it changes the infoSize field of the *Message* accordingly and internal optimizations can take place.

- *void MESSAGE_PacketAlloc(Node *node, Message *msg, int packetSize, TraceProtocolType originalProtocol)*
  Allocates a packet of the given size. This method can only be called once per message because it is assumed that if the packet is changed in following processing steps it is because headers are added or removed. Adding and removing headers should be done using the MESSAGE_AddHeader and MESSAGE_RemoveHeader functions.

- *void MESSAGE_AddVirtualPayload(node, msg, payLoadSize)*
  A macro adding the specified amount of virtual payload to the current virtual payload.

- *void MESSAGE_Send(Node *node, Message *msg, clocktype delay)*
  The name of this method is quite misleading: It does not send the *Message* through the simulated network but schedules it as event to take place after *delay* units of simulation time slices ($1ns$ in Qualnet v3.6).

- *void MESSAGE_CancelSelfMsg(Node *node, Message *msgToCancelPtr)*
  Cancels a scheduled event. This means that the event modelled by the message *msgToCancelPtr* is pointing to will never take place.

- *void MESSAGE_AddHeader(Node *node, Message *msg, int hdrSize, TraceProtocolType traceProtocol)*
  To add a header to the packet, this method has to be called as the internal elements are adjusted according to the given *hdrSize*. After calling this method, the *packet* field points to the start of the enlarged packet, thus *hdrSize* bytes of data can be copied to the packet without overriding any previously existing packet data. The *traceProtocol* parameter is not important for internal use but for tracing and internal assertion checking.

- *void MESSAGE_RemoveHeader(Node *node, Message *msg, int hdrSize, TraceProtocolType traceProtocol)*
  This method is the counterpart to the *MESSAGE_AddHeader* method. Added headers have to be removed using *MESSAGE_RemoveHeader*.

- *void MESSAGE_ExpandPacket(Node *node, Message *msg, int size)*
  This method can be used to enlarge the packet. As already mentioned, in most cases the resizing of a packet is due to header adding and removing as the packet propagates through the network stack. For these cases, *MESSAGE_AddHeader* should be used. For other purposes, *MESSAGE_ExpandPacket* serves the intended purpose (although the current implementation has some restrictions on how much a packet can be enlarged).

- *void MESSAGE_ShrinkPacket(Node *node, Message *msg, int size)*
  To short a packet, this function can be called. For *MESSAGE_ShrinkPacket* applies the same as for the *MESSAGE_ExpandPacket* method: It should not be used to manipulate the packet's size for header adding and removing.

7.5 Statistics in Qualnet v3.6

Most of the analysis when using Qualnet v3.6 is done analyzing the statistic file. Each run generates a statistic file listing detailed statistics for each layer on all nodes. The first part of a line in the statistic file identifies the entry:

<node id>, <interface address>, <instance id>, <layer>, <protocol>,

Except the *node id* which is the one from the node passed when calling *IO_PrintStat*, the other elements can be set to any value, even nonsensical ones. There exists no validity check. In many cases the *interface address* and the *instance id* are left blank because the statistics are not for one interface or instance but for all of them on a specific node. Following this beginning part of the line, the protocol specific output is printed (on the same line). Normally it uses the standard format (every statistics Qualnet v3.6 generates use it):

<statistic variable> = <value>

If this format is used, analyzing the statistics file is quite easy using a script. The succeeding cut-out contains the statistics of node nr. 29 on the physical layer, the link layer, and the first line of the network layer's statistics.

```
29,  , [0],  Physical,    802.11,Signals transmitted = 176
29,  , [0],  Physical,    802.11,Signals received and forwarded to MAC = 1795
29,  , [0],  Physical,    802.11,Signals locked on by PHY = 1832
29,  , [0],  Physical,    802.11,Signals received but with errors = 37
29,  , [0],  Physical,    802.11,Energy consumption (in mWhr) = 225.003
29,  , [0],       MAC, 802.11MAC,Packets from network = 241
29,  , [0],       MAC, 802.11MAC,UNICAST packets sent to channel = 2
29,  , [0],       MAC, 802.11MAC,BROADCAST packets sent to channel = 164
29,  , [0],       MAC, 802.11MAC,UNICAST packets received clearly = 4
29,  , [0],       MAC, 802.11MAC,BROADCAST packets received clearly = 1503
29,  , [0],       MAC,   802.11DCF,Unicasts sent = 2
29,  , [0],       MAC,   802.11DCF,Broadcasts sent = 164
29,  , [0],       MAC,   802.11DCF,Unicasts received = 4
29,  , [0],       MAC,   802.11DCF,Broadcasts received = 1503
29,  , [0],       MAC,   802.11DCF,CTS packets sent = 4
29,  , [0],       MAC,   802.11DCF,RTS packets sent = 2
29,  , [0],       MAC,   802.11DCF,ACK packets sent = 4
29,  , [0],       MAC,   802.11DCF,RTS retransmissions due to timeout = 0
29,  , [0],       MAC,   802.11DCF,Packet retransmissions due to ACK timeout = 0
29,  , [0],       MAC,   802.11DCF,Packet drops due to retransmission limit = 0
29,  ,    ,   Network,         BLR,packets broadcasted = 179
```

Physical and link layer statistic both output the *instance id* while the network layer leaves it blank. That is not astonishing as there exists mostly only one network layer instance for all interfaces.

Chapter 8

BLR Implementation in Qualnet v3.6

This chapter will present the BLR implementation in Qualnet v3.6 (see section 7 for an introduction). When it comes to implementation some decisions have to be made. This section itemizes the most important ones and discusses their tradeoffs.

8.1 BLR Integration in Network Stack

BLR is a routing protocol and as such conceptually part of the network layer. There are different possibilities to integrate BLR into the network stack:

- Every packet leaving the node is a IP packet containing a UDP packet whose payload is a BLR packet. Thus the original packet is encapsulated within a BLR packet which is then transmitted for example using UDP. This BLR packet would contain the BLR header followed by the original IP packet.
- BLR adds its header behind the IP header, saves the original IP header protocol field value and sets it to *BLR*.
- BLR adds its header in front of the IP header, before the packet is passed to the MAC layer. Using this mechanism, BLR conceptually lays between the network and the MAC layer.
- BLR creates a new IP packet, containing a new IP header, the BLR header and the original IP packet. Implemented like that, BLR is used as if it were not only routing but a transport protocol too.

All these options have their advantages and disadvantages. The first one is easy to implement but has the drawback that a packet's size is increased by 28 bytes due to the additional IP and UDP header. An other negative aspect is the fact that such a packet has to travel twice through the IP stack, consuming resources and increasing processing delay. Additionally if BLR is implemented that 'high' in the network stack, if using unicast, it sends the packets to a unicast IP address which results in an ARP request, thus generating additional network traffic.

If the BLR header is inserted behind the IP header, as in the second option, the IP header's protocol have to be save and reset. That could be prevented if the BLR header is inserted as an

IP option field as it is proposed for DSR [8]. There is no difference on the overall packet size but only in the processing if inserting the BLR header as IP option or just the way described above. Implementing BLR solely on the network layer (in contrast to the first option), it is aware of MAC addresses. Like that, no ARP requests occur because BLR can directly deliver the packets to the MAC layer, either by passing a broadcast MAC address or a unicast one, depending on the mode (broadcast greedy, unicast greedy, or backup) BLR wants to transmit the packet in. Thus signaling network traffic can be avoided. There exist a slight drawback of inserting the BLR header like that: Unless it is inserted as IP option, it is necessary to change the fragmentation mechanism on the IP layer. The fragmentation mechanism has to copy the BLR header and insert it in each fragment after the IP header or BLR will fail.

Adding the BLR header in front of the IP header, as in the third option, would be the best place to add the BLR header from a conceptual point of view. Especially with regard to the proposed improvement implementing BLR on the MAC layer (see section 10.2 for details). In contrast to the second option, this would avoid any changing of the fragmentation mechanism on the IP layer, but has all of the second option's advantages.

If BLR is implemented like some sort of transport protocol, as in the forth option, again an additional IP header wastes bandwidth. When inserting the BLR header like that, it is not yet decided if the packet should be sent using normal IP processing like in option one, or directly passed to the MAC layer as in option two and three; Both methods can be applied.

Given that bandwidth consumption is an important issue in wireless data transmission, only the second and the third option come into question. As already mentioned, from a conceptual point of view the third option is the most logical one. But there exists another factor: The complexity of the implementation. Because the IP layer in Qualnet v3.6 does not implement fragmentation, the problem of the second option does not exist. However, in a real world implementation this problem has to be addressed even if it is getting smaller because modern systems mostly use path maximum transmission unit (PMTU) discovery [25] to adjust the packet size to the transmission circumstances. If PMTU is used, IP fragmentation should not be needed. If that is kept in mind, the two options are equivalent with regard to the produced simulation results in Qualnet v3.6. Because the goal of this thesis was to evaluate the concept of BLR and not to verify different implementations in respect of their feasibility, the second option was the option of choice. The resulting IP packet and BLR processing works as illustrated in figure 5.1 and described in section 5.2.

8.2 Coordinate Format

One of the decisions to be made concerns the coordinate format. The coordinate format has direct consequences on the performance of the protocol due to the fact that stored coordinates make up the bigger part of the BLR data and backup header. The tradeoff is between the position accuracy and the size of the header. Using 64bit numbers for the x, y, and z coordinate doubles the needed space compared to using 32bit values. Assumed the used coordinate system covers the whole world (for example GPS), using 32bit values results in maximal position inaccuracy of $2.28m$. Loosing about two meters of accuracy is negligible if it is possible to halve the number of bytes needed to store the position. A little drawback concerns the simulation performance:

Qualnet v3.6 stores location information as three double values, having the effect that conversion has to be carried out which decreases the simulation performance.

To further reduce the coordinate's size the format can be tailored if used only in plains: The z coordinate can be omitted which saves $\frac{1}{3}$ of the space used.

8.3 Promiscuous Mode

A *BLR unicast* implementation according to section 5.4.6 should use promiscuous mode. Consider the following situation: Node A broadcasts a packet and the next hop B forwards the packet using unicast transmission. If node A does not listen to the medium using promiscuous mode it will not notice that the packet has successfully been forwarded by node B. Hence node A will retry forwarding the packet in backup mode and in doing so a duplication is carried out.

A second option to solve the problem: A node which received a broadcast packet and forwards the packet using unicast has to generate some kind of acknowledgement message and send it to the last hop.

8.4 Random Waypoint Mobility Model Implementation

Qualnet implements the random waypoint mobility model with all nodes moving at the beginning. The nodes can be distributed uniformly, randomly or following a grid. Because this implementation has the drawback of a changing distribution of the nodes and their parameters 6.1.2, a *stationary random waypoint distribution* has been implemented to distribute the nodes and to sample the nodes' speeds and pauses. This is the solution proposed in [20].

8.5 Resetting Traffic History Timeout Timers

Every entry in the traffic history times out after a given time δ. The question of when to time out an entry arises, if a packet is not forwarded because there exists a cancelling entry in the traffic history: Should the timer be reset to time out in $currentTime + \delta$ or just expire at its original timeout? If the timer is not reset and the δ low, the packet would be forwarded instead of cancelled if it arrives again (for example after a loop) and the traffic history entry meanwhile timed out. Therefore the second option seems more appropriate because the traffic history should contain traffic that was forwarded or was *some kind of acknowledgement* during the last δ time units (see section 5.2 for details about the *some kind of acknowledgement* definition). The argument against it is the fact, that the packet would have been forwarded if the traffic history would not exist. For the actual simulation results there exists virtually no difference between the two options, because the size of the traffic history (strongly related to the timeout value δ) was none of the parameters to optimize. Therefore the δ was set to $10s$ which is large enough to cover every packet that passes a node more than once because the maximal delay that has been found was far below. To make future optimization of these parameter possible, the first alternative has been implemented.

Chapter 9

Evaluation of BLR

9.1 Simulation Scenario and Fixed Parameters

Several parameters are all the same for all simulations. Especially all the physical and MAC layer parameters. They are set to values to simulate a 802.11b wireless network with a transmission range of 250m. IP layer properties such as queue size, queue type and so on, were left unmodified to their default values. Another constant over all simulations is the size of the simulation area: A plane of 1200x6000m. All simulations lasted 900s. The initial node positioning and the mobility model applied was the modified random waypoint mobility model, described in section 8.4. The pause time was always 0s because higher pause time can lead to pseudo static networks (see section 6.1.2 for details), which was not the intend scenario. The simulated traffic started at the 1^{st} second. The sending node (node 1) transmitted two packets per second until the number of sent packets has reached 1700. The packet's payload is 64bytes.

BLR has some parameters for buffer sizes and timeouts which have not been evaluated to a large extent (see table 9.1 and 9.2). Most of them have only an impact on the BLR performance up to a certain value. For example the time after which a entry in the traffic history 5.4.2 is deleted. Of course these parameter's values are not negligible because in a real world implementation resource consumption (energy, memory) is an issue to take care of. But the tuning of these values is beyond the scope of this thesis.

Later simulations will show that 2ms is a good *maximal delay*. Compared to that, *hello jitter* is relatively high (4ms). Using more jitter than necessary leads to needless delay. That is why it is desired to use a jitter as low as possible. One could argue that if the backup mode is needed, the average density is low, hence the danger of simultaneously transmitted hello messages is low too and the jitter can be small. This is true for the node starting the backup mode but the packet can reach regions where the node density is higher. If the *hello jitter* is too small, backup mode could fail in such situations. There are two reasons which can cause it to fail due to a small *hello jitter*:

- All hello messages transmitted have interfered with other hello messages and therefore have never correctly arrived at the node which has sent the hello request.

- Some of the messages are delayed due to congestion and the resulting 802.11 backoff mechanism (see section 2.2 for details) and arrive after the *wait for hello timeout* period.

Table 9.1: Greedy mode and general parameters

traffic history timeout: $10s$	The time after which an entry in the traffic history is deleted
transmission range: $250m$	Besides the settings for physical layer which have the effect that the transmission range is about 250m, the BLR protocol needs the *transmission range* as system wide parameter to calculate the *additional delays* (see section 5.1 and 5.2.2).
broadcast timeout: 5 *maximal delay*	The time a node that has broadcasted a packet waits for *some kind of acknowledgement* before forwarding it in backup mode
unicast timeout: $20ms$	The time a node waits for the MAC acknowledgement after transmitting a unicast message.
route timeout: $2s$	After *route timeout* an entry in the route cache is deleted
unicast security margin: $\frac{3}{10}$ *transmission range*	A node which is farther than (*transmission range - unicast security margin*) from the processing node will not be considered as next hop unicast. This helps to prevent trying to send to nodes already out of transmission range.

If the lost or delayed hello would have changed the forwarding decision, this can lead to loops and therefore message dropping (for example by the means of the traffic history, section 5.4.2). Figure 9.1 depict such a situation. The second problem can be addressed with two obvious approaches: The first one is to increase the *wait for hello timeout* for example to 5 *hello jitter*. The second solution is to increase the *hello jitter* itself. Obviously both options introduce additional delay, but the second beats the first one: It not only addresses the second cause of backup mode failing due to small *hello jitter* but also the first one: A larger *hello jitter* will lower the probability that hello message collisions occur.

9.2 Simulations

The simulations had three main purposes: Optimize parameters, verify presumptions, and find weaknesses. As already mentioned, a lot of parameters remained the same for all simulations. The parameters that changed are: maximal additional delay, forwarding area, backup mode, using unicast or not, the function used to compute the additional delay, and minimal and maximal speed in the random waypoint model. To set the minimal and maximal speed the desired average speed s_{avg} has been fixed. The speed interval has then been set to $s_{avg} \pm 10\%$. If nothing else is

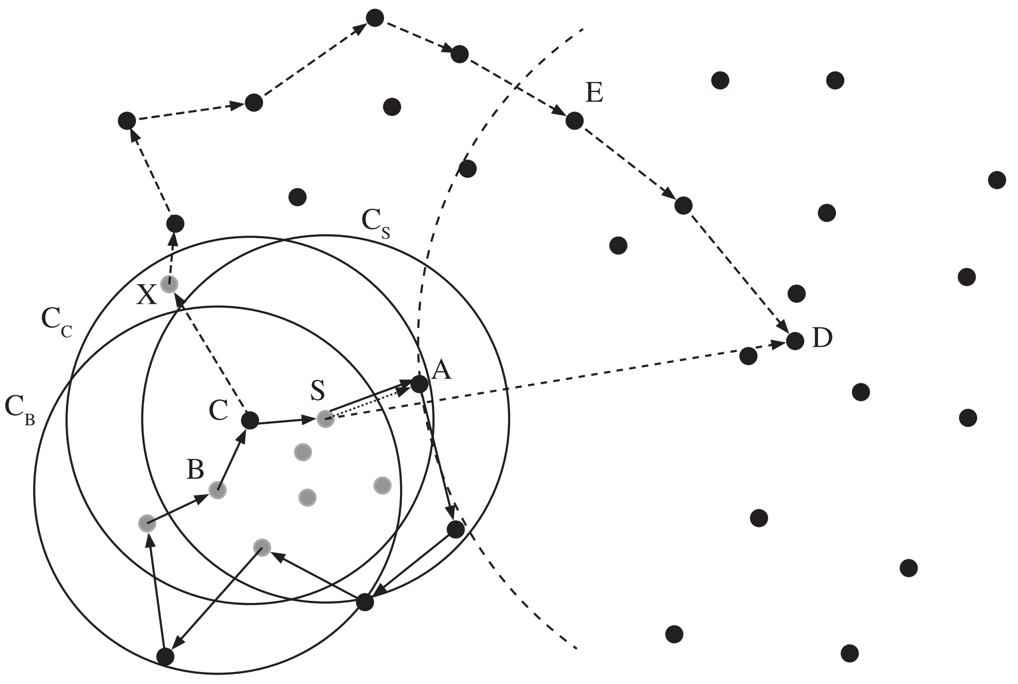


Figure 9.1: Lost hello message problem: Consider the scenario where node S wants to send a packet to destination D. The first hop to A can be done in greedy mode (indicated by the dotted arrow). Node A has not any neighbor in its forwarding area and therefore starts forwarding the packet in backup mode. It is assumed that no problems occur until the packet reaches node C along the path depicted by drawn through arrows. After receiving the packet, C sends a hello request. Supposed that the hello message from node X is delayed longer than for *wait for hello timeout* or lost due to interference with another hello message, which both is quite probable as in C's transmission range (indicated by a circle C_C) eight nodes are located (which are printed grey) which will all emit a hello message. If the hello message from X is not available at the forwarding decision moment, the packet will be forwarded to node S. From there it is forwarded to A which will drop it. In that way the packet is lost although a path exists: From C through X in backup mode till node E, where it would switch back to greedy mode. Two hops later it would have reached the destination. Obviously this packet loss occurs because X is not within the transmission range of the node B or S, which both would have forwarded the packet to X. (the two additional circles C_B and C_S depict B's and S' transmission range respectively).

stated, simulations have been performed either in *slightly dynamic* networks with $s_{avg} = 5m/s$ or in *highly dynamic* networks, whose average speed is $20m/s$.

The diagrams with the simulation results display average values and 10% two sided confidence intervals.

9.2.1 Forwarding Area Comparison

The first set of simulations was performed to figure out the best forwarding area. To achieve that goal quite a lot of simulations must be performed and interpreted because the performance of

Table 9.2: Backup mode parameters

maximum clockwise delay: 5 *maximal delay*	The maximal delay for packets forwarded in clockwise backup mode
clockwise timeout: 2 *maximum clockwise delay*	The time after which clockwise backup mode is considered failed if no acknowledgement has been received
hello jitter: $4ms$	hello messages answering a hello request have to be generated and sent during the interval $[0, hellojitter]$
wait for hello timeout: 1.5 *hello jitter*	After sending a hello request the node waits for *wait for hello timeout* before continuing the processing
request response timeout: $20ms$	The time the node waits for an acknowledgement for a transmitted request response backup mode packet before trying to send it to the next hop as described in 5.3.2
hello timeout: $2s$	The time a hello packet and its information remain valid

a forwarding area is related to other parameters: The maximal additional delay, the number of nodes and probably to the mobility of the nodes.

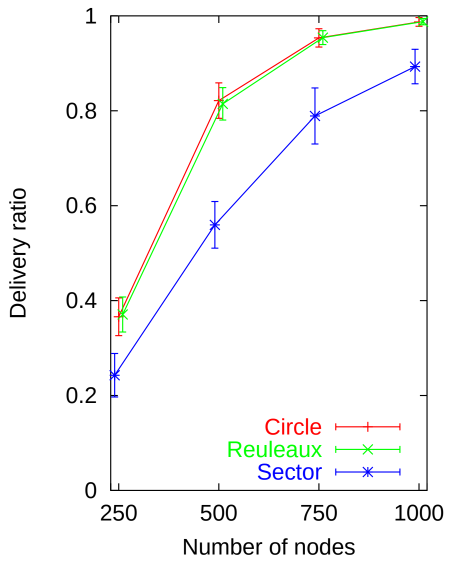
The relation between the forwarding area and the maximal additional delay is as follows: The bigger the forwarding area, the more nodes are potentially located inside the area. The more nodes set a timer to forward the packet, the higher is the probability (for a fixed maximal additional delay) that collisions occur, resulting in the problems mentioned in section 5.2.2. Obviously the number of nodes in the simulation area has a direct impact on that probability too: The higher the overall node density, the more nodes are potentially in a forwarding area.

Thus a hypothesis for the simulations can be formulated: For low node density, the biggest forwarding area should perform best because it has the highest probability to contain at least one node. If the node density is getting higher, two opposite effects take place. If more nodes stay within the forwarding area, there is a higher chance that one with good forward progress exists, which improves the performance. The opposite effect concern the higher collision probability if more nodes stay within the forwarding area. These effects only packet forwarded in greedy mode, therefore backup mode is inactivated for these simulations.

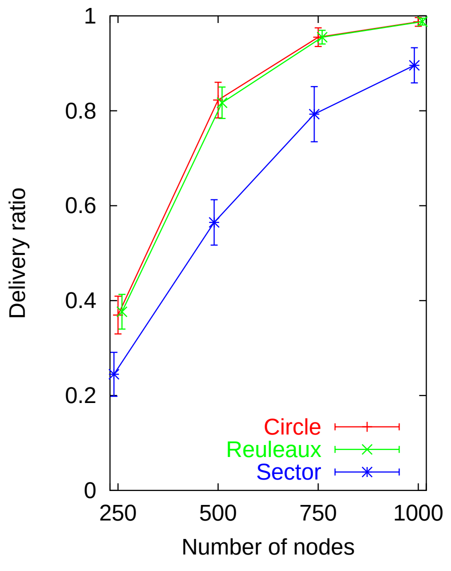
This resulted in simulations with varying maximal additional delay, forwarding areas and number of nodes for scenarios with *slightly dynamic* and *highly dynamic* networks.

As the figures 9.2 and 9.3 show, the circle and the reuleaux triangle are equal in respect of delivery ratio. The supposed effect that a smaller forwarding area (the sector) would outperform the bigger ones in scenarios with high node density cannot be confirmed. A positive fact is that BLR does not come into trouble if the network's topology is highly dynamic. That is not surprising, because BLR does not stores any information about its neighbors.

Because the reuleaux triangle and the circle show equal results concerning the most impor-

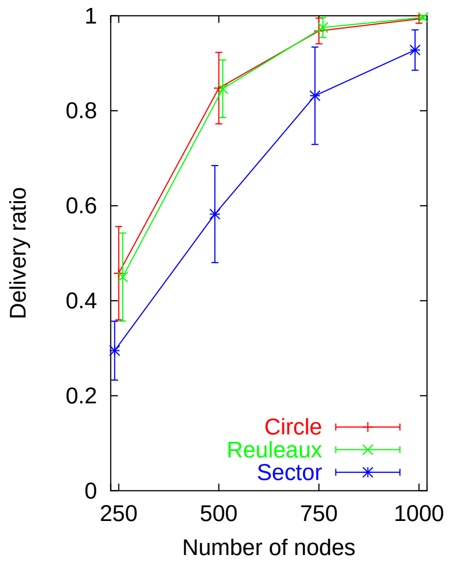


(a) 2ms maximal additional delay

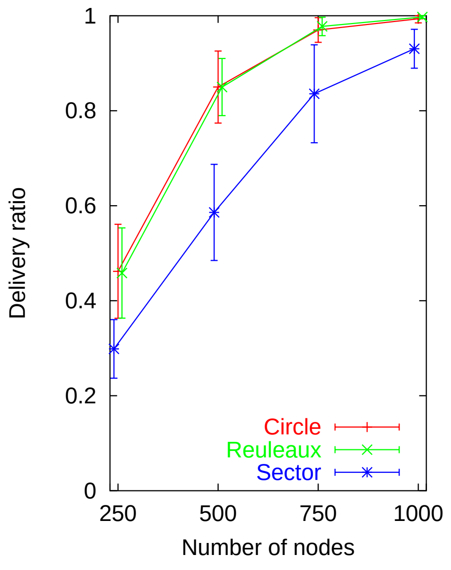


(b) 10ms maximal additional delay

Figure 9.2: Forwarding area comparison in highly dynamic networks.



(c) 2ms maximal additional delay



(d) 10ms maximal additional delay

Figure 9.3: Forwarding area comparison in slightly dynamic networks.

tant measure (delivery ratio) it does not really matter which one is chosen for further simulations. As the circle in theory has a slightly better performance and in the simulations the reuleaux triangle was not better, the circle has been chosen for further simulations.

9.2.2 Maximal Additional Delay Comparison

As mentioned in section 9.2.1, the maximal additional delay probably has its impact on the delivery ratio. A relation between the end-to-end delay and the maximal additional delay can be taken for sure because the maximal additional delay destines the delay added at each hop. As already mentioned, the shorter maximal additional delay, the higher the probability of packet collisions and packet loss. Thus a tradeoff has to be found between end-to-end delay and delivery ratio.

For these simulations, backup mode is turned off too. They are performed only in *highly dynamic* networks because the results from section 9.2.1 showed that the network dynamics have negligible impact on the greedy mode performance. As already stated in section 9.2.1, the circle is used as forwarding area.

As obvious from the diagram in figure 9.4 the additional maximal delay has no impact on the delivery ratio. Thus it has the expected impact on the end-to-end delay which is obvious too: Higher maximal additional delays produce higher end-to-end delays. Another effect is not really clear at the first glance: The end-to-end delay has a peak at 500 nodes. With 250 nodes the end-to-end delay is fairly low, rises to its peak at 500 nodes and then drops slowly the denser the network gets. The end-to-end delay is measured only for packets reaching the destination. In low density networks, a packet needs more progress per hop to reach the destination, as in average less hops are possible before greedy mode fails [15]. Figure 9.4(c) affirms this supposition: The hop count for 250 nodes is really low compared to the other node densities. Hops with more progress add a lower delay, thus the resulting end-to-end delay is quite low. If the network becomes denser, packet delivery ratio is higher but at the price of using hops with less progress, higher additional delay, and therefore higher end-to-end delay. If the network gets really dense, the probability that a node with good progress exists within the forwarding area increases again, resulting in a trend of lower end-to-end delays for denser networks.

The fact that the end-to-end delay lowers more for higher maximal additional delay can be explained with the fact, that in denser networks more collisions of forwarding nodes occur with the effects described in section 5.2.2, resulting in the effect illustrated by the figure 9.4.

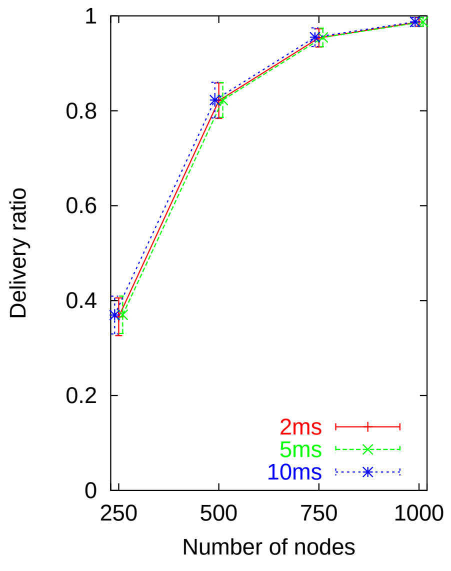
Because this results are very clear, further simulations will be performed using maximal additional delay of 2ms.

9.2.3 Evaluating the Effects of Unicast

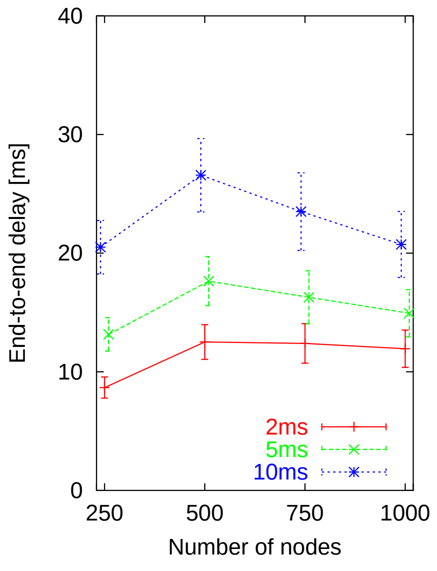
As mentioned in section 5.4.6, unicast transmission of packets has some advantages. It is supposed that the average end-to-end delay will be lower than with normal greedy forwarding.

For these simulations, again the backup mode is turned off. As unicast transmissions would reveal its weaknesses especially in *highly dynamic* networks, the simulations concentrate on these scenarios. As in other simulations, the forwarding area is set to *circle*. The safety margin is set to $\frac{3}{10}$ of the transmission range (see section 5.4.6 for an explanation of the safety margin).

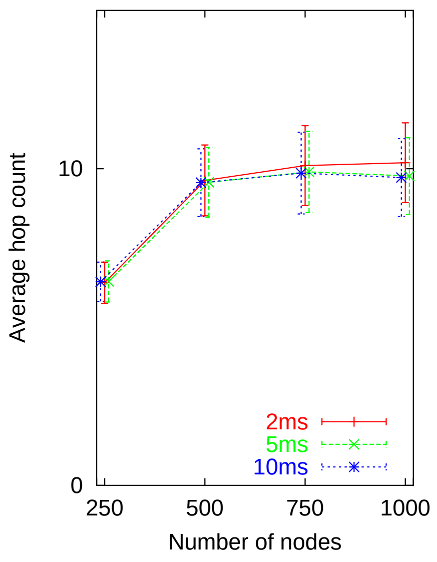
Delivery ratio and hop count nearly stay the same using unicast or just plain greedy mode (see figure 9.5). What is slightly different, yet not really different as the confidence intervals



(a) Delivery ratio comparison



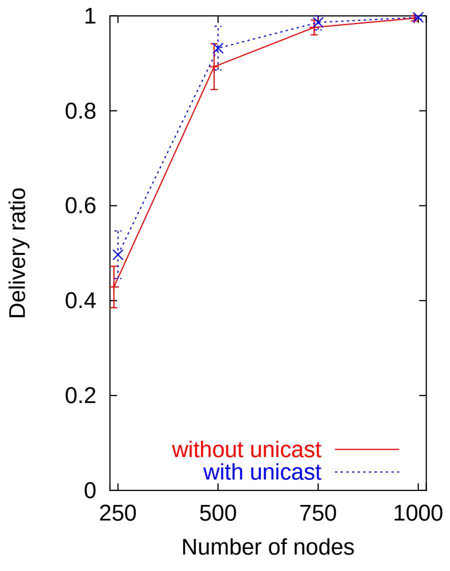
(b) End-to-end delay comparison



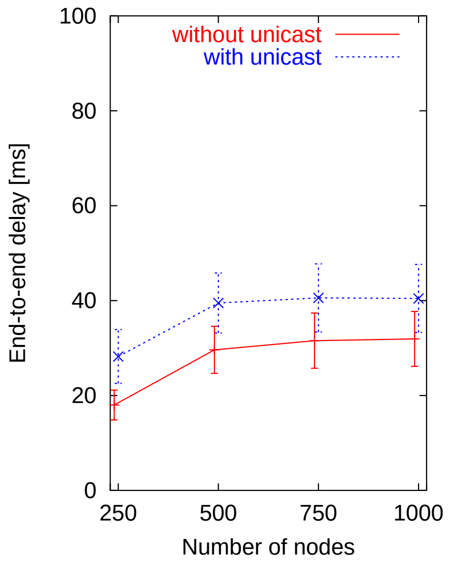
(c) Hop count comparison

Figure 9.4: Maximal additional delay comparison in slightly dynamic networks.

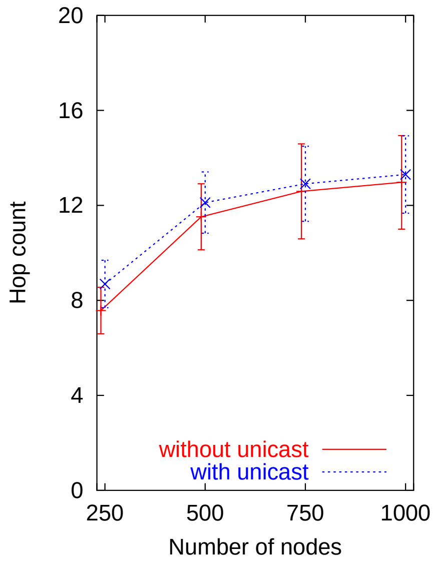
still overlap, is the end-to-end delay. But not as expected! The end-to-end delay is higher if using unicast than in plain greedy mode. An explanation is not that hard to find: In such highly dynamic networks as the simulated one, it is quite probable that nodes move out of each others transmission range. In this case, unicast retries seven times to carry out the RTS/CTS handshake without success before rebroadcasting it, which noticeably increases the end-to-end delay.



(a) Delivery ratio comparison



(b) End-to-end delay comparison



(c) Average hop count comparison

Figure 9.5: Evaluation on the effects of unicast in highly dynamic networks.

Looking at the diagram comparing the average number of hops a slight trend can be found: Unicast transmissions seem to have a slightly higher hop count. This is quite logical: A node forwards packets with a specific destination to the node whose address is stored in the route cache. If an other node providing more progress to the packet's path moves inside the forward area, it is not 'detected' until rebroadcasting occurs due to the route becoming invalid or expiring.

Normal greedy mode immediately uses the better positioned node as next hop. But as already mentioned, this is only a slight trend and should be evaluated further.

9.2.4 Request Response Backup Mode vs. Greedy Mode Comparison

Up to this point only greedy mode options have been evaluated. This section tests the request response backup mode of section 5.3.2 which elevates BLR to a fully functional protocol: Without the backup mode BLR is not able to deliver packets in certain networks, even if a path to the packet's destination exists.

The backup mode has been tested in highly dynamic networks. The crucial backup mode parameters *hello jitter* and *wait for hello timeout* are set to 4ms and 6ms respectively. These are the values proposed in section 9.1. The forwarding area is set to circle.

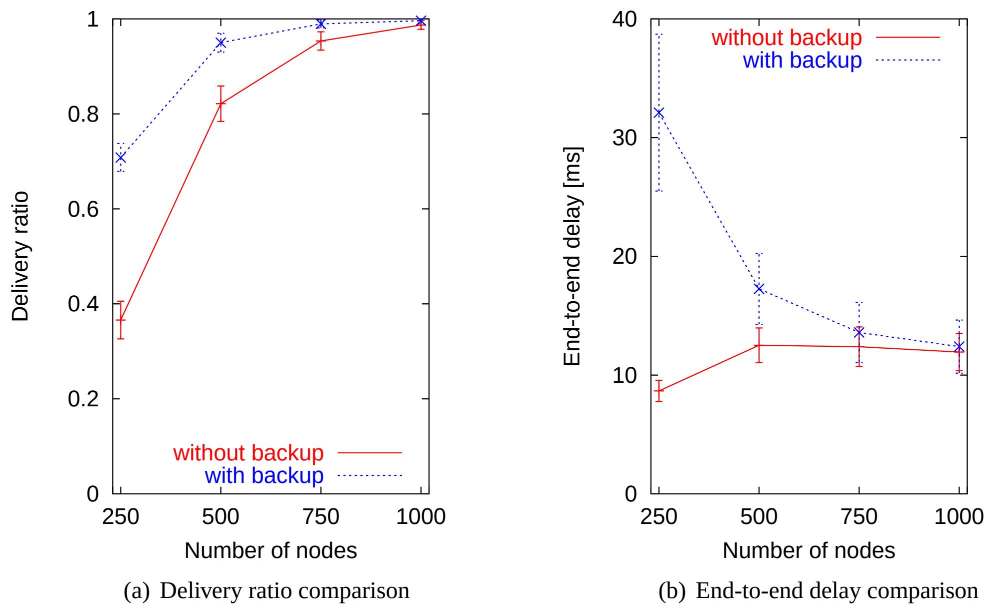


Figure 9.6: Evaluation of the response backup mode in highly dynamic networks.

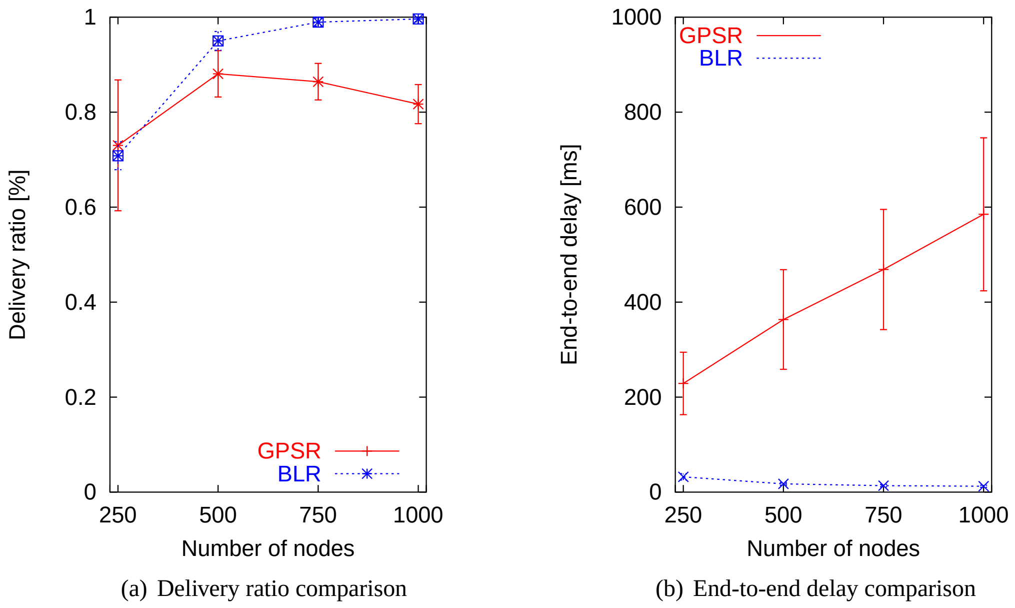
Looking at figure 9.6(a), the request response backup mode improves BLR's performance independently from the node density. What changes is the relative performance gain compared to plain greedy mode. There exists an explanation for this phenomenon: In networks with low node density greedy mode fails quite often where backup mode still can find a path. As the networks get denser, greedy mode does not fail that often, so that switched on backup mode does change the routing of the packet.

Packets forwarded in backup mode experience more delay as greedy mode packets. This is a consequence of the fact, that a node forwarding a packet in backup mode first sends a hello request message and then waits for a certain amount of time for hello messages before forwarding the packet to the next hop. Figure 9.6(b) shows that the delay of BLR with backup mode decreases in denser networks, which can be explained with the fact, that in denser networks fallback to backup mode is not needed that often and therefore the higher delay occurs less frequently.

9.2.5 BLR vs. GPSR

BLR is not the only routing protocol for mobile ad hoc networks. Therefore it has to prove its right to exist. One of the protocols with good performance is GPSR (see section 4.1 for details). In this section BLR and GPSR are compared respecting their delivery ratio and end-to-end delay.

Of course BLR is tested using its backup mode. The forwarding area is set the already established circle. The GPSR parameters are set to their defaults. The scenarios were highly dynamic network.



(a) Delivery ratio comparison (b) End-to-end delay comparison

Figure 9.7: Comparison of BLR and GPRS in highly dynamic networks.

In networks with low node density GPSR and BLR provide about the same delivery ratio. Figure 9.7(a) also reveals that GPSR is outperformed if the networks get more dense: With 500 nodes in the simulation area (1200x6000m, which results in about 70 $nodes/km^2$) GPSR produces its best simulation results. After that point GPSR's delivery ratio decreases while BLR still achieves an improvement. An explanation can be found in the high mobility of the nodes and the periodic hello message sending of GPSR (each node broadcasts one hello message every 1.5 seconds). Probably the network gets overloaded by the hello messages, resulting in packet loss that does not occur due to missing knowledge of a path but due to contention. The second cause of the bad GPSR performance probably is the high link breakage rate due to the high mobility of node. Both explanations are affirmed by the increasing end-to-end delay if the network becomes denser, see figure 9.7(b). In the first case, the delay increases because the packets have to be queued. In the second case due to failure of the RTS/CTS handshake and the following retrying to send it to another neighbor, or, even worse, to switch to perimeter mode.

9.3 Discussion of the Simulation Results

The simulations showed that the maximal additional delay does not have to be impractically big but only $2ms$ to achieve good results. This saves BLR from producing long end-to-end delays.

The circle as forwarding area affirmed its superiority above the sector, which has been supposed due to analytical results. With the given number of simulations, it is not possible to find out if the reuleaux triangle or the circle performs better.

The request response backup mode showed its usefulness. But the results have not been compared to a measure such as ’how many packets can be delivered due to the network graph’. This question is equal to the question about the number of packets which are not delivered although a path to the destination exists. Such measures should be applied in further simulations.

The unicast improvement does not revealed the supposed effect. It probably has to be improved or it is only superior to normal greedy mode in scenarios with high network load.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

The results presented show that BLR is robust against topology changes. It performs nearly equally good in highly dynamic networks and networks with less topology changes. This result is not surprising as BLR does not collect any information that can become outdated if the topology changes. The high scalability can be anticipated looking at the good results even in dense networks.

The combination of the request response backup mode with the greedy mode outperforms GPSR which combines greedy and a fall back mode (perimeter mode) too.

The unicast mode did not show the excepted results (lower end-to-end delay). Thus the RTS/CTS handshake takes its time and invalid route cache entries which results in rebroadcasts impede better results. The approach mentioned in section 10.2 of moving BLR into MAC layer probably could combine the two results.

10.2 Future Work

BLR is not yet perfect and can be further improved. Some aspect to evaluate were beyond the scope of this thesis but could reveal interesting knowledge about BLR characteristics. This section lists some aspects that are worth to be evaluated in the future.

- BLR calculates the *additional delay* according to the *most forward progress* strategy (see section 5.2.2 for details). It would be interesting to check the impact of the choice of the forward strategy on the overall performance. Especially the difference between the *most forward progress* and *closest to destination* strategy would be interesting as the *most forward progress* sometimes prefers a node farther from the destination to a node closer to the destination (see figure 5.6 for details and an example configuration).

  An other category of *additional delay* calculating function improvements have been mentioned in section 5.2.1. These functions tailored for the used forwarding area could decreases the possibility of packet collision. Of course these ideas can be combined with different forward strategies mentioned above.

- The current implementation of BLR using unicast (see section 5.4.6 for details) uses a cached route until the node does not fulfill the condition of being within the circle with radius $transmissionrange - safetymargin$, the route expires, or a transmission fails. As seen in the simulation results in section 9.2.3, there exists a trend that using unicast results in more hops. This is easily explained as nodes which move into the forwarding area and would provide more progress are not taken into account.

  A possible solution to eliminate this drawback of using unicast is the periodic retransmission in normal greedy mode. That means that even in cases where the route is still valid, nodes providing more progress can be detected. One could argue that this is in fact equal to a shorter route time out. For a static rebroadcasting time, this is true. But the advantage of this approach is, that the decision to rebroadcast can be made depending on the progress the current route provides. This would result in faster rebroadcasting if the current next hop provides only a small progress. Next hop's which provide really good progress can be used till the route timeout is reached. That which will not happen if regularly traffic uses this route, because the route cache entry is updated every time on overhearing a forwarding of a packet.

- Up to now BLR has not been evaluated in the situation where the network is highly loaded and suffers contention. Future work should evaluate those situations because especially greedy mode could run into trouble as mentioned in section 5.5. Another hypothesis could be verified in these simulations: Up to now, using unicast does not really yield any gain compared to simple broadcasting. That would probably change under high network load.

- At the moment a node forwarding a packet in the request response backup mode does not cache hello messages. To increase the end-to-end delay performance a node could cache hello messages for a specific time to use them for relaying following packets to the same destination. Because cached hellos can become invalid, an in depth survey should be performed on the drawback and a tradeoff has to be found.

- At the moment, BLR is implemented on the network layer. The tighter interaction with the MAC layer (see section 5.4.3 for details) already offers some advantages. Implementing BLR directly on MAC layer opens various optimizations. The RTS/CTS mechanism of 802.11 could be modified that the CTS is returned using an *additional delay*. Problems that could emerge especially in situation with high network load can probably be handled like that. Obviously thorough analysis and testing will be needed to prevent drawbacks that could occur because the RTS/CTS handshake has its role in providing a method to handle high network load (see section 2.2 for details).

- BLR assumes that a mechanism to look up the destination's position exists. Because such a mechanism probably consumes bandwidth too, it would be interesting to do simulations using one of the know mechanisms, for example VHR [26].

- At the moment BLR does not implement any mechanism to deal with position inaccuracy which can lead to problems (see section 5.5 for details). Two different problems arise due to position inaccuracy. The first one affects the position inaccuracy of the nodes currently involved in the forwarding process (additional delay computing nodes and the last hop). Position inaccuracy of potential forwarding nodes can lead to nodes forwarding the packet although they are not located within the forwarding area but consider to be due position inaccuracy of their own positions. That can cause packet duplication as not all positions in the forwarding area can receive the transmission. The same problem arises if the last hop is not located at the position it has stored in the *prevSrcPos*. It is quite difficult to solve. A possible solution is to add a *safety margin* to the forwarding area, thus the forwarding area gets smaller. The positive effect is, that but even transmissions of nodes which consider to be inside but are not, can be overheard by all nodes within the forwarding area.

The second problem affects the position inaccuracy of the destination's looked up location. In the current implementation, a node which forwards the packet and considers itself to be within the destination's transmission range, does not switch to backup mode but just drops the packet if it does not receive any acknowledgement. Terminode's *Geodesic Packet Forwarding* suggests *Restricted Local Flooding* to solve this problem (described in section 4.2.2). Probably this approach would be adequate in BLR too and should be evaluated in further simulations.

Bibliography

[1] M. Heissenbüttel and T. Braun, "BLR: A beacon-less routing algorithm for mobile ad-hoc networks," Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, Tech. Rep. IAM-03-001, March 2003.

[2] IEEE 802.11 working group, "Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," IEEE standard, March 1999. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11-1999.pdf

[3] IEEE 802.3 working group, "Part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications," IEEE standard, December 2001. [Online]. Available: http://standards.ieee.org/getieee802/download/802.3-2002.pdf

[4] IEEE 802.11 working group, "High-speed physical layer in the 5 ghz band," IEEE standard, September 1999. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11a-1999.pdf

[5] ——, "Higher-speed physical layer extension in the 2.4 ghz band," IEEE standard, September 1999. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11b-1999.pdf

[6] ——, "Amendment 4: Further higher data rate extension in the 2.4 ghz band," IEEE standard, June 2003. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11g-1999.pdf

[7] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," IETF RFC 3561, July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561

[8] D. B. Johnson, D. A. Maltz, and Y.-C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (dsr)," IETF INTERNET-DRAFT, April 2003. [Online]. Available: http://www.ietf.org/proceedings/03jul/I-D/draft-ietf-manet-dsr-09.txt

[9] T. Clausen and P. Jacquet, "Optimized link state routing protocol," IETF RFC 3626, October 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626

[10] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *MOBICOM'00*, Boston, USA, August 2000, pp. 243–254.

[11] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, “Routing with guaranteed delivery in ad hoc wireless networks,” in *Proceedings of 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Seattle, August 1999, pp. 48–55.

[12] Y. Z. A. Z. Fabian Kuhn, Roger Wattenhofer, “Geometric ad-hoc routing: Of theory and practice,” Boston, USA, July, pp. 63–72.

[13] H. S. E. Kranakis and J. Urrutia, “Compass routing on geometric networks,” in *11th Canadian Conference on Computational Geometry*, Vancouver, August 1999, pp. 51–54.

[14] L. Blazevic, J. Y. Le Boudec, and S. Giordano, “A scalable routing scheme for self-organized terminode network,” in *Proceedings of Communication Networks and Distributed systems modeling and Simulation conference (CNDS)*, San Antonio, Texas,USA, January 2002.

[15] M. Heissenbüttel, T. Braun, T. Bernoulli, and M. Wälchli, “Blr: Beacon-less routing algorithm for mobile ad-hoc networks,” *Elsevier’s Computer Communications Journal (ECC)*, vol. 27, no. 11, pp. 1076–1086, 2003, submitted.

[16] J. Nonnenmacher and E. W. Biersack, “Scalable feedback for large groups,” *IEEE/ACM Trans. Networking*, vol. 7, no. 3, pp. 375–386, June 1999.

[17] J. Boleng, “Normalizing mobility characteristics and enabling adaptive protocols for ad hoc networks,” in *Proceedings of LANMAN 2001: 11th IEEE Workshop on Local and Metropolitan Area Networks*, 2001, pp. 9–12.

[18] C. Bettstetter and C. Wagner, “The spatial node distribution of the random waypoint mobility model,” in *Proceedings of the First German Workshop on Mobile Ad-Hoc Networks (WMAN)*, 2002, pp. 41–58, gI Lecture Notes in Informatics.

[19] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” *Wireless Communications and Mobile Computing (WCMC)*, vol. 2, no. 5, pp. 483–502, 2002, special issue on Mobile Ad Hoc Networking.

[20] W. Navidi and T. Camp, “Stationary distributions for the random waypoint mobility model,” *IEEE Transactions on Mobile Computing*, vol. 3, no. 1, pp. 99–108, 2004.

[21] P. M.-S. E. Royer and L. Moser, “An analysis of the optimum node density for ad hoc mobile networks,” in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2001, pp. 857–861.

[22] Z. Haas, “A new routing protocol for reconfigurable wireless networks,” in *IEEE International Conference on Universal Personal Communications (ICUPC)*, San Diego, USA, October 1997, pp. 562–565.

[23] V. A. Davies, “Evaluating mobility models within an ad hoc network,” Colorado School of Mines, 2000, master’s thesis.

[24] G. P. C. C. X. Hong, M. Gerla, “A group mobility model for ad hoc wireless networks,” in *Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 1999, pp. 53–60.

[25] J. Mogul and S. Deering, “Path mtu discovery,” IETF RFC 1191, November 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1191

[26] S. Giordano and M. Hamdi, “Mobility management: The virtual home region,” EPFL, Lausanne, Switzerland, Tech. Rep. SSC/1999/037, October 1999.