BEACON-LESS ROUTING: AN IMPLEMENTATION FOR GNU/LINUX

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Tobias Roth
2005

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

Contents

Acknowledgement

Thanks go to Professor Dr. Torsten Braun as the head of the *Computer Networks and Distributed Systems* research group, who allowed me to carry out my diploma thesis in the interesting field of MANET computing. Marc Heissenbüttel supervised my work and walked me through the whole year, providing valuable hints whenever I needed them. Matthias Scheidegger often helped me with various problems in the realm of system programming and the C language in general. My girlfriend Liliane Mérinat supported me with a lot of love and understanding during my writing of this thesis. And last but not least, I would like to thank everybody else from the research group who helped my with scientific and administrative issues.

Chapter 1

Introduction

Little more than a decade ago, mobility and communication were two completely separate aspects of life. With the emergence of cellular phones and wireless computer networks, these two aspects began to converge. Nowadays, both mobility and communication are an integral part of daily life for millions of people. The people want to communicate whenever they feel like it, and wherever they are. This revolution of mobile communication started with the widespread adoption of cellular phones. In 2002, mobile phones outnumbered fixed-line phones for the first time, counted worldwide. These days, the percentage of the worlds population that communicates with a mobile phone is approaching 50%. It is likely to assume that the demand for wireless Internet will grow in similar dimensions, although probably at a slower rate. With the introduction of new communication technologies such as WLAN, UMTS or EDGE, high-speed internet access for mobile devices becomes a reality.

The mobile phone and the mobile internet device are merging into one unit, which becomes smaller and more portable with every year. The ever increasing number of mobile communication devices in operation offers an opportunity to deploy new methods of communication. Since its beginning, wireless communication has been handled by means of a fixed infrastructure. In the case of wireless networking, stationary units called base-stations are used as relays that allow mobile devices to communicate with each other. The limitations of this mechanism are obvious. It depends on a fixed infrastructure that possibly is expensive to set up, and the area that can be covered is limited. To allow a larger coverage, multiple base stations need to be connected via a wired network. It is apparent that with the increasing number of mobile units, these devices should communicate directly with each other, and be used to relay traffic between two distant devices. This eliminates the necessity of base stations unless connectivity to another network is required. The field of Mobile Ad-Hoc Networks (MANETs) embraces this concept.

The primary challenge in MANETs is how to establish a route between two distant devices. Early approaches were modelled after existing routing algorithms, where designated devices, the routers, build routing tables based on the topology of the network. All non-routing devices send their traffic via the routers. Since the topology remains mostly stable in fixed networks, the routing tables need to be updated only infrequently. This is different for mobile networks, where devices are moving around and thus often enter or leave the communication range. If traditional routing algorithms are used in mobile networks, frequent updates of routing information to account for topology changes are the result, together with often outdated routing tables. To make

routing more efficient in mobile ad-hoc networks, position information can be used instead of topology data. Routing algorithms that make use of position data are called position-based or geographic. The Beacon-Less Routing algorithm (BLR) falls into this category.

The Beacon-Less Routing Algorithm[1, 2] uses a stateless approach, where no knowledge about the position of nearby nodes is needed to make routing decisions. BLR showed very promising results in network simulations[3]. The intention of this thesis is to produce a prototype implementation of the Beacon-Less Routing algorithm, and to try to verify the results achieved in the network simulator as much as possible. It was chosen to do the implementation in the C programming language, as this promises a highly portable solution.

Chapter 2 discusses other implementations of position-based routing protocols suitable for real-world measurements. Chapter 3 describes the BLR algorithm briefly, and chapter 4 discusses the details of the implementation. This is followed by the measurements presented in chapter 5, which make up the largest part of this work. Finally, a few suggestions for improvements are made in chapter 6, where the work is also concluded.

Chapter 2

Related Work

Many position-based mobile ad-hoc routing protocols have been proposed over the past few years. An overview over them is given in [4]. Among the more prominent ones are GPSR[5], Terminode Routing[6] and GOAFR+[7]. The most commonly used approach to evaluate and study proposed algorithms is to carry out computer simulations. For BLR, this step has been taken in [3]. However, as often stated[8, 9], it is extremely difficult to model the physical layer of wireless networks realistically during computer simulations. As shown in [10], the radio model, which is responsible for simulating signal propagation and interference during a computer simulation, highly affects the significance of results obtained from network simulations. Thus, the logical next step after computer simulations is to create an implementation that can be used to conduct real-world tests, both indoor and outdoor, and to conduct measurements under realistic conditions. In [11], first results of real-world tests with BLR have been published, and this thesis describes the work done in more detail.

Implementations suited for carrying out real-world tests exist for most protocols, for example one for GPSR can be found in [12]. In [13], indoor tests with the AODV[14] protocol have been conducted, although the results were rather disappointing. The poor performance obtained in the experiments have been attributed to problems with the wireless network cards. In [15], a small test network of six statically positioned nodes has been used to evaluate an implementation of ODMRP[16].

The objective of other related projects was to create testbeds for evaluating wireless ad-hoc networks, and to compare different protocols to each other. The evaluated protocols are either existing implementations, slight adaptations thereof, or entirely new implementations. In [17], a general overview over the requirements of testbeds for mobile ad-hoc networks is given and the difficulties encountered during deployment of one are outlined.

In [18], experiences in building a testbed are described. The problems with evaluating wireless ad-hoc protocols denoted in this work are as follows: The wireless signal propagation is highly unpredictable, as factors such as weather, terrain or random interfering signals make it very difficult to achieve reproducible measurements when doing real-world, outdoor experiments. Obtaining packet routes over multiple, different hops requires a deployment of dozens of nodes over a large area. This inevitably results in difficulties in managing the coordination of the individual nodes during the experiment, as well as in controlling their movement. Transmission ranges are greatly influenced by external factors and bidirectional links cannot be taken for

granted. Furthermore, non-scientific problems are also discussed, such as the difficulty to find the large number of people required to conduct an experiment as well as explaining to casual bystanders (and possibly the authorities) what is going on.

In [9], a testbed was developed with the purpose of providing the possibility to directly compare wireless ad-hoc networks with a minimum of administrative overhead. The testbed has been set up indoors inside a single room. Virtual separation of the nodes, in order to achieve that some nodes are not in the transmission range of others, is achieved with RF attenuators. A facility to easily control the attenuation and to switch between different virtual topologies has been built. Mobility is simulated in software, and only allows for discrete movement of the nodes, i.e. for jumps between different positions. The advantage of this approach is that it greatly simplifies protocol development and experiments, as only one person is needed to carry out an experiment. The obvious disadvantage is that the previously mentioned influences from an outdoor environment are not accounted for during measurements.

In [19], a complex wireless network testbed for outdoor tests has been constructed. It is built in an area of 7 km^2 and consists of static, vehicular and airborne nodes, as well as management stations with internet connectivity. Preliminary tests with the DSR[20] protocol have been conducted in this testbed.

Yet another, very promising approach to solve the problem of precise modelling of real-world conditions with the purpose of evaluating wireless ad-hoc network protocols is taken in [10]. Instead of first implementing a protocol in a general purpose network simulator, followed by an implementation for real-world devices, an existing real-world implementation is run via so-called *Direct Execution* inside a newly developed simulator. The advantage is that the same codebase can be used for both real-world tests and simulations, which increases the significance of comparisons between results from the simulation and results from outdoor experiments. When porting a protocol to be directly executable, only a small number of system calls, those responsible for receiving and sending packets and for accessing the real-time clock, have to be adapted for the implementation to work in the simulator. The measured overhead in CPU performance was minimal, while the overhead in memory usage was larger, although not so high as to render the approach unfeasible. After a real-world experiment has been performed, the simulator can be fed with mobility and connectivity traces, to exactly recreate the outdoor environment. Another advantage of the direct execution approach is that the CPU work-model is implicitly given, and does not have to be emulated by the simulator. Existing simulators usually do not consider delays caused by the scheduling mechanism of the operating system, on which the routing algorithm is executed. The main conclusion drawn in [10] is that a simple stochastic radio propagation model with parameters typical to an outdoor environment can produce acceptable results in a standard simulator. The results can be greatly improved with the help of connectivity traces, however. The free-space and the two-ray radio propagation models largely exaggerate the radio transmission range, and thus misinterpret network conditions, which can have false conclusions as a result.

Another work[21], which is very similar to the presented one, deserves to be mentioned. It consists of a protocol proposition for a position-based routing algorithm, which is to be used in vehicular networks. However, its focus lies on protocol design and implementation, and measurements are only used to demonstrate the correctness of the implementation. Two experiments

with four nodes and fixed positions have been carried out.

Chapter 3

The BLR Protocol

This section describes the BLR[1, 2, 3, 11, 22] algorithm in a formal way, as well as delivering the necessary background information needed to understand the specification.

3.1 Assumptions

Each node is aware of its own geographical position. This can be achieved through the Global Positioning System (GPS), Galileo, or other positioning services[23]. Levels of altitude, as delivered by GPS or Galileo, are not considered in this specification for reasons of simplicity. As with all position-based routing protocols, the source of a transmission must have a possibility to detect the geographical position of the destination. Several schemes for this so-called localisation have already been proposed, [24] gives an overview.

There are two system-wide constants, called `MAX_DELAY` and `TRANSMISSION_RANGE`, which are known by all nodes. The former indicates the maximum time that a packet is intentionally delayed per hop, as described in the next section. The latter denotes the theoretical maximum transmission radius of a node. The algorithm assumes the unit disk graph model, where two nodes can communicate with each other if their intermediate distance is no more than `TRANSMISSION_RANGE`. As a consequence, all links are assumed to be bidirectional and antennas are omnidirectional.

3.2 Greedy Mode

When BLR operates in dense networks, it forwards packets in greedy mode. It allows to forward packets in a completely stateless and distributed manner. The advantage of BLR over other position-based routing protocols is that no information about neighbouring nodes is needed to make routing decisions, which eliminates the need for a periodical transmission of beacons. This has advantages such as preservation of battery power and avoiding both interferences with normal data transmissions and problems caused by outdated routing tables.

If a source node has a packet to send, it first determines the geographic position of the destination node. It stores this information in the packet header, along with its own position. The packet is then broadcast and received by all nodes in transmission range. With the position

of the source node, the previous node and the destination node, nodes receiving a packet have all the information needed to calculate whether they are inside a specific area, called the *Forwarding Area*. Nodes outside this area drop the packet, while those inside prepare for relaying it. The forwarding area is located between the sending node and the destination node and its diameter is restricted by the `TRANSMISSION_RANGE` constant. It is chosen so that all nodes located inside this area are in each others transmission range. An infinite number of areas with these properties exists, and [22] shows that a forwarding area in the shape of a circle is suited best for BLR. Figure 3.1 illustrates the forwarding area.

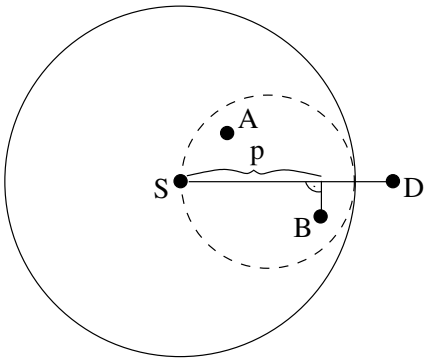


Figure 3.1: The transmission area is denoted by the solid circle, the forwarding area by the dotted circle.

The source node S is always located on the border of the circle that surrounds the forwarding area. The forwarding area is divided in half by the straight connection between the source node and the destination node. Nodes A and B are both located inside the forwarding area in the above example.

Nodes that receive a packet and are located inside the forwarding area use the concept of the *Dynamic Forwarding Delay* (DFD) to assure that only one node forwards the packet. This means that before actually relaying the packet, each node waits for a specified amount of time, called *Add_delay*. This amount is dependent on the position of the node in respect to the previous hop and the constant `MAX_DELAY`. It is calculated by the following formula, where r stands for the `TRANSMISSION_RANGE` and p stands for the progress:

$$Add\_delay = MAX\_DELAY \cdot \left(\frac{r-p}{r}\right)$$

The progress p of node A is defined as the projection of the distance between node S and node A on the line from S to the final destination D. The DFD function shown above ensures that the node that introduces the most progress forwards the packet first, which is node B in figure 3.1. All other nodes attempting to forward the same packet cancel their scheduled transmission, as soon as they overhear the relaying by another node. The source node also overhears the transmission of the packet, and thus receives an acknowledgement that the packet has been successfully relayed, which is called *passive acknowledgement*. The only change that a forwarding node performs on a packet it relays is substituting the position of the previous hop with its own position. The packet is forwarded in this manner until the final destination is reached. The

destination is the only node that has to actively send an acknowledgement packet, since it not relays the received packet any further.

3.3 Backup mode

When a sending node does not receive a passive acknowledgement in a certain amount of time after sending a packet, it assumes that no other node is located inside its forwarding area. It then enters backup mode and tries to discover an alternate path to the destination. It does this by broadcasting a short request for beacons, and all nodes within transmission range reply with a packet indicating their position. In this scenario, nodes that are located closer to the destination than the sending node are said to introduce forward progress. If such a node exists, it is chosen as the next hop. If not, the sending node extracts a planar sub-graph, e.g. a Gabriel Graph, for its neighbourhood and forwards the packet via unicast according to the right-hand rule. A Gabriel graph contains the edge between vertices v_i and v_j, if no other vertex v_k is within the circle with diameter $[v_i v_j]$. The extraction of the planar sub-graph is necessary in order to prevent packets to enter a loop. The distance between the node where greedy mode failed and the destination is stored in the packet header. As soon as the packet arrives at a node with a smaller distance to the destination, the packet is forwarded in greedy mode again.

3.4 Unicast Optimisation

After a node has detected the relaying of one of its packets through a passive acknowledgement, it knows which node is optimally placed on the route towards that specific destination. It now can send subsequent packets to the same destination directly to that optimally placed hop via unicast. Forwarding nodes that know the optimal next hop can relay by unicast and without applying DFD, which can reduce end-to-end delay significantly. After a certain amount of time, a packet is broadcast again, so that it is possible for the algorithm to react to topology changes that may have happened in the mean time.

Chapter 4

Protocol Implementation

4.1 Development Environment

The target platform of the implementation is GNU/Linux. The Gentoo Linux[25] distribution was used for development, although any modern GNU/Linux distribution that is based on the Linux Kernel 2.6 will work with the implementation. The Linux kernel needs to be compiled with the following configuration options enabled: `CONFIG_TUN`, `CONFIG_PACKET` and `CONFIG_IP_NF_IPTABLES`. The option `CONFIG_TUN` is needed to create the virtual tunnel interface, `CONFIG_PACKET` is needed to create the raw socket used for sending and receiving packets and `CONFIG_IP_NF_IPTABLES` is needed to block incoming unicast packets from not triggering error messages. This will be explained later in more detail. Additionally, the appropriate drivers for wireless cards and serial drivers for the GPS devices are needed. During the implementation, the following tools were used: The GNU C compiler version 3.3.3 [26], the GNU debugger [27] and the Subversion revision control system [28].

4.2 Portability

Keeping the code portable to other unix systems was not a requirement. However, the code is written in ANSI C and only two GNU/Linux-specific constructs have been used. These are the usage of the *select()* systemcall and the *pf_packet* facility. The System V variant of *select()* typically sets a timeout value before exit, but the BSD variant does not. Since this timeout value is used in the BLR application, the behaviour of the GNU/Linux variant of *select()* has to be simulated when considering a port to non-Linux systems. This can be easily done. *Pf_packet* does only exist on GNU/Linux, and for a port to a non-Linux operating system, the packet reception and sending code will have to be rewritten. Both the tuntap driver and the MAC-layer filter do not pose portability problems, as the BPF filter used for MAC layer filtering originated in BSD, and the tuntap device has already been ported to most unix variants.

4.3 Design Decisions

Even though an implementation in the simulator existed already[3], it was necessary to make certain design decisions for cases where the implementation in the simulator could not be followed exactly. Aside from the overall architecture of the application, the main questions were what packet format to use and how to integrate the application into the network stack.

4.3.1 Tunnelling or Stack Integration

When implementing a routing protocol prototype, two options are available: the use of tunnelling or an integration of the protocol directly into the network stack. With tunnelling, a BLR-specific packet header is wrapped around the payload, and the resulting BLR packet is encapsulated inside a UDP packet, which is then transmitted via IP. Applications that want to send BLR packets typically do this with the help of an API that the BLR implementation has to provide. The GPSR implementation[29] is an example of an implementation that takes this approach. With a direct integration into the network stack, an outgoing packet is intercepted at some point in the network stack, and the BLR header is inserted between the existing packet headers. This involves adapting the existing headers (Ethernet or IP header, depending on where the BLR header is inserted), to accommodate for the change of the packet. When such a manipulated packet is received by a destination node, the BLR application removes the BLR header and reverses the changes done to the other headers, before delivering the original packet. Intermediate hops simply update all the necessary headers and forward the packet.

The advantage of tunnelling is that it is simpler to implement and manage, since a standalone application can be developed, which is only loosely coupled with the network stack. On the other hand, direct integration into the network stack offers several advantages over tunnelling:

- Application compatibility: With tunnelling, each application has to use an API to send network traffic, whereas with a direct integration, network access is transparent for all applications. This means that existing applications such as browsers or email clients can use a BLR network without having to be changed.

- Accuracy: The behaviour of the algorithm can be matched closely by the implementation without having to take the underlying protocol of the tunnel into account. This makes modelling the algorithm easier and the resulting application is a more accurate implementation of the algorithm.

- Performance: The overhead introduced with tunnelling is larger than with a direct integration, both in terms of processing time and packet size. With a direct integration, the size overhead per packet is only the size of the BLR header that is added. With tunnelling, a UDP header is needed in addition to the BLR header. Furthermore, when tunnelling inside IP/UDP, ARP messages are needed for relaying packets by unicast. The ARP mechanism is not needed with a direct integration, since packets can be sent directly to the next hop, because the Ethernet destination address can be set by the BLR application. When a tunnelled packet is received, the surrounding headers are usually stripped completely before

the packet is read, and then built again when the packet is forwarded. With a direct integration, the existing IP header is reused, as it does not need to be changed when a packet is forwarded.

One of the requirements for this thesis was to stay as consistent as possible with the implementation done in the simulator. This was also an important reason to favour a direct integration over the tunnelling approach.

4.3.2 Placement in the Network Stack

Due to time constraints and the prototype nature of the implementation, only an implementation in user space was feasible. This means that instead of modifying packets directly in the kernel, the BLR application has to somehow intercept packets sent by user space applications such as browsers or email clients. It can then add the BLR header to the intercepted packet, decide whether to use greedy mode, unicast optimisation or backup mode, and then pass the packet to the network adaptor. To remain consistent with the simulator implementation, the BLR header will be placed between the IP header and the IP payload. Details about how the BLR application is hooked into the network stack are given in the next sections of this chapter.

4.4 Architecture Overview

Figure 4.1 shows how the BLR application is hooked into the network stack and gives an overview over the operation of the application. The dashed arrows signify incoming traffic, the continuous arrows outgoing traffic. The diagram is divided into a user space section and a kernel section.

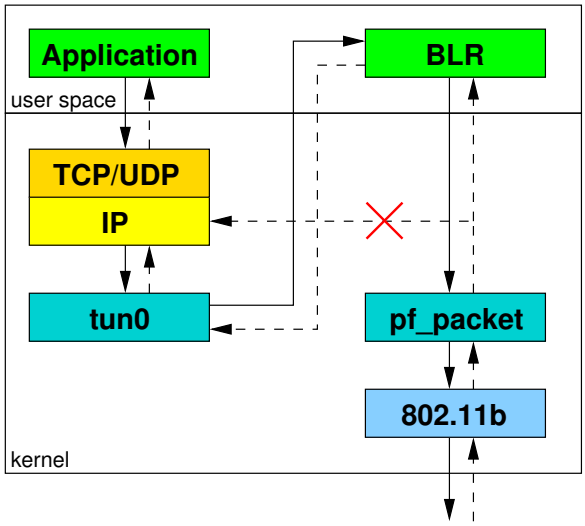


Figure 4.1: Architecture overview

The BLR application interacts with the network stack at two points: the *tun0* interface[30] and the *pf_packet* socket[31]. Communication between a local application and the BLR application happens via *tun0*, and communication with the wireless network adaptor happens via *pf_packet*. Outgoing packets are handled as follows. A routing entry is set in the system routing table that sends all traffic with an endpoint in the BLR test network to the *tun0* interface. The *tun0* interface provides the BLR application with raw IP packets coming from local user space applications. All traffic with an endpoint not in the BLR test network is subject to normal routing decisions made by the kernel, and is thus not sent to the BLR application. Once the BLR application has read a packet from *tun0*, it inserts the BLR header and updates the IP header to accommodate for the changes caused by the insertion of the BLR header. It then sends the manipulated packet to the wireless network adaptor via the *pf_packet* socket, which allows to send and receive raw Ethernet frames or IP packets directly to and from the network adaptor.

Incoming packets, which the BLR application reads from the *pf_packet* socket, are either forwarded to the next hop or passed to localhost, depending on the destination address in the IP header. When a packet is to be forwarded, the BLR application updates the BLR header of the packet and additionally delays it by the newly calculated *Add_Delay*, before it is passed via *pf_packet* to the wireless network adaptor. When this host is the final destination for the packet, the BLR header is stripped off, and the modifications made to the IP header by the BLR application that is running on the original sender node are reversed. The resulting IP packet is then sent to the *tun0* interface, from where it goes through the system routing table before it reaches the application layer.

A problem occurs because *pf_packet* actually creates a copy of all incoming packets. The copy is passed to the BLR application, the original packet takes the normal way through the kernel, as shown by the crossed out line in figure 4.1. Why this is a problem and how it has been solved is described in section 4.8.

4.5 The BLR Application in Detail

The architectural overview given in the previous section showed how the BLR application is integrated into the network stack. This section describes the parts that the BLR application consists of. It is split into three separate processes which communicate with each other via pipes.

- The GPS process is connected to an external GPS device which it polls for changes in geographic location. It parses the GPS data and passes position updates to the main process.
- The sendqueue process receives outgoing packets together with the packet-specific value *Add_Delay* from the main process. It sends packets after they were delayed for the specified amount of time.
- The main process does everything else. Specifically, this is packet reception from both the localhost and the wireless network adaptor, packet header transformation and updating, *Add_Delay* calculation, and managing packet timeouts, unicast routes and a list of duplicate packet IDs.

Figure 4.2 gives an overview over the different processes. The blue boxes symbolise separate processes. Everything inside the dashed line is part of the BLR application. The arrows going in and coming out of the dashed box symbolise connections between the BLR application and the rest of the system, these connections are provided by the kernel.

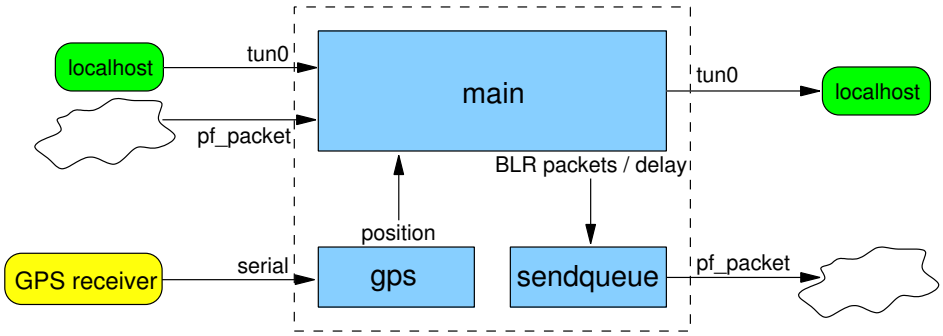


Figure 4.2: Architecture details.

4.5.1 GPS Handling

To find out its current position, each node can be connected to a GPS [32] device, or, alternatively, read fixed coordinates from a file. The connection to a GPS device is made through an RS232 interface over which the NMEA protocol[33] is used for communication. Most commercial GPS devices offer support for this protocol. However, in order to also receive a precise time synchronisation signal, the device must provide a PPS[34] signal, which is not provided by all devices. The precision of the position data is greatly influenced by the GPS signal quality, which depends mostly on the number of satellites that are in sight.

4.5.2 Sendqueue

The sendqueue process is responsible for queueing packets according to their individual *Add_delay*. It receives packet/delay tuples from the main process and maintains an ordered queue of all packets that are waiting to be sent. Whenever the *Add_delay* for a packet has expired, the packet will be sent. The queue is implemented in a separate process in order to separate packet delay handling from packet handling and processing done in the main process. Another task handled by the sendqueue process is the deletion of packets from the queue, whenever the main process detects that another node has already forwarded a pending packet.

4.5.3 The Main Process

The main process does most of the packet handling. It consists of various subsystems:

- The main routine handles process initialisation and is responsible for receiving packets from *tun0* and *pf_packet*.

- It maintains the current position which it receives from the GPS process.
- The host table stores information about peers, namely their most recent position and the next hop to reach them while using the unicast optimisation.
- The packet list maintains packets that were sent and not yet acknowledged, together with a timeout value for each packet. Retransmissions in case of a timeout are also handled by the packet list.
- The recently received table (RRT) contains all packet IDs that have been processed recently and should be ignored when encountered again.
- The backup queue contains outgoing packets that wait to be sent until the backup mode setup is completed and a decision about the next hop has been reached.
- Localisation, i.e. the acquisition of the geographical position of the destination, is done by the main process. It is described in more detail in the next chapter.

4.5.4 The BLR Header

A diagram of the BLR header is shown in figure 4.3. It is inserted between IP header and IP payload and contains information necessary to route BLR packets. The fields are: packet type (1 byte), original protocol (1 byte), sequence number (2 bytes), backup distance (4 bytes), source position (8 bytes), destination position (8 bytes), previous position (8 bytes). The overall size is 32 bytes.

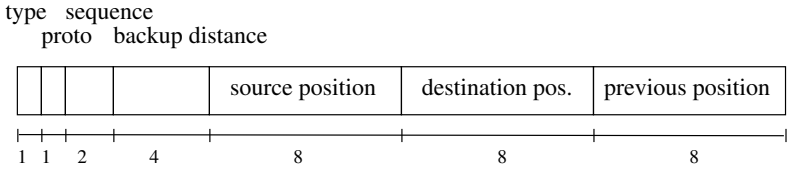


Figure 4.3: The BLR header.

In the simulator implementation different headers are used for different packet types. For instance the headers added to packets sent in greedy mode are different than the headers of data packets used in backup mode, since not all information needed during backup mode is also needed during greedy mode. For simplification, this subdivision is not made in the real-world implementation. However, the header size in the real-world implementation is already 2 bytes smaller than the header size used for greedy mode packets in the simulator implementation, so for the majority of packets, the real-world implementation uses smaller headers. The fields are used as follows: The type field indicates the type of the BLR packet, for example greedy mode packet, location request and so on. The protocol field stores the protocol number of the encapsulated protocol (TCP or UDP in most cases). This field is necessary since its counterpart in the IP header is overwritten with the protocol number for BLR, when a BLR packet is created. The

sequence field is used together with the source IP address of the IP header to uniquely identify each packet. The backup distance stores the distance between the point where greedy mode failed and the destination node. It is only used during backup mode. The position fields in the header consist of two 32 bit integers that store latitude and longitude. These are computed into Cartesian coordinates where necessary.

4.5.5 Internal Structure

The following section gives an overview of the composition of the BLR application. It lists configuration values, packet types, operation modes, source files and the main functions. In the next section, the various modes BLR can send packets in are explained with the help of examples.

The following configuration parameters are tunable at compile time:

- `TRANSMISSION_RANGE`: the theoretical transmission range
- `MAX_DELAY`: the upper boundary for *Add_Delay*
- `FWAREA`: what forwarding area to use
- `TIMEOUT`: the time to wait before a packet is retransmitted
- `DISCOVERY_TIMEOUT`: the time until discovery replies are processed
- `REFRESH_INTERVAL`: the invalidation interval

The invalidation interval defines an interval after which the application switches back to greedy mode, in case it previously was in backup or unicast mode. Additionally, the cache of the geographic positions of other nodes can be purged, but this functionality has not been used during the measurements.

The following packet types exist:

- `DATA`: All packets that contain only payload and are not used to exchange protocol control messages have this type.
- `LOCATION_REQUEST`: This type of packet is sent when a node has a packet to send but does not know the geographic position of the destination. It contains piggybacked payload.
- `LOCATION_REPLY`: A node sends a packet of this type after receiving a location request. If two-way traffic is assumed, location replies are not sent.
- `ACK`: An explicit acknowledgement packet is sent either when a node is the final destination and receives a packet it has to acknowledge, or when a node receives a duplicate, in order to prevent further retransmissions of the same packet.
- `DISCOVERY_REQUEST`: This type of packet is used during the setup phase of backup mode, to discover what nodes are located in transmission range.

- `DISCOVERY_REPLY`: A discovery reply is sent in response to a discovery request, it can be seen as a sort of position beacon, which is only sent on request.

`LOCATION_REPLY`, although implemented, was not used during measurements for various reasons explained in the next chapter.

A packet can be sent in one of three modes, in accordance with the specification of the BLR algorithm:

- `GREEDY` (also sometimes called `BROADCAST`)
- `UNICAST`
- `BACKUP`

Unicast packet transmission will be treated as an independent mode and not as a simple optimisation, thus the terms *unicast optimisation* and *unicast mode* are used interchangeably. From an architectural point of view it makes sense to separate unicast sending from the greedy mode.

During packet creation, several fields in the IP header need to be adjusted, to accommodate for the insertion of the BLR header into the packet. All these changes will be reversed before the final destination passes a packet to *tun0*.

- Source address: Packets the BLR application receives from the *tun0* interface have their source IP address in the IP header set to the address of the *tun0* interface. This needs to be changed to the IP address of the wireless network interface, since this is the interface where the packets leave the host.
- Packet length: The length needs to be increased by the size of the BLR header that has been added.
- Protocol field: This field needs to be changed to BLR (the protocol number 254 is used, which is reserved for testing purposes). The original protocol number is saved in the BLR header.
- Header checksum: After changing any of the header fields, the checksum of the IP header needs to be recalculated.

The BLR application is split into several source and header files. The various files and their tasks are the following:

- maindriver.c: Initialisation and packet reception
- config.h: Contains protocol and application tunables
- filter.h: Defines MAC filters used during indoor tests
- blr.c, blr.h: BLR protocol handling

- gps.c, gps.h: The GPS process
- sendqueue.c, sendqueue.h: The sendqueue process
- hosttable.c, hosttable.h: The host table
- packet list.c, packet list.h: The packet list
- backup queue.c, backup queue.h: The backup queue
- rrt.c, rrt.h: The Recently Received Table
- tunnel.c, tunnel.h: Creates the virtual interface *tun0* and sets an entry in the system routing table that directs all traffic destined to the BLR test network to this interface.

The following core functions are referenced in the coming sections:

- *blr_create()*: creates a new BLR packet by adding the BLR header and adjusting the IP header. Created packets are either passed to the sendqueue or stored in the backup queue.
- *blr_forward()*: Handles incoming packets with an IP address that does not match the local address. Depending on the packet type and the current mode, *Add_Delay* is calculated and the packet/delay tuple is passed to the sendqueue for forwarding.
- *blr_assemble()*: Handles packets destined for localhost. The BLR header is stripped and the IP header updated, then the resulting IP packet is written to the *tun0* interface. When a `LOCATION_REQUEST` packet is received, the position of the requesting node is stored and the piggybacked `DATA` packet is extracted and sent to localhost via *tun0*. This is explained in more detail in the next chapter.

Both *blr_forward()* and *blr_assemble()* also handle `ACK` packets, `DISCOVERY_REQUESTS` and `DISCOVERY_REPLIES`.

4.6 BLR Modes

This section illustrates the route a packet takes through the system and the BLR application, as it is sent from the source node over the wireless network, forwarded by intermediate hops and finally received by the destination node. To simplify the explanation, the three modes are treated separately.

4.6.1 Greedy Mode

Figure 4.4 shows an example scenario with the nodes involved in a transmission. S and D denote the endpoints of a transmission, A and B denote possible intermediate hops. The dotted cycle represents the forwarding area and the continuous circle represents the transmission range of node A.

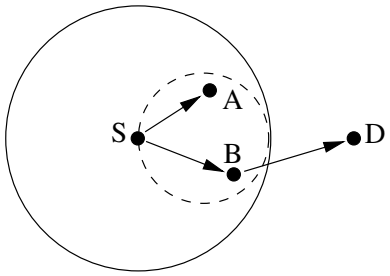


Figure 4.4: Greedy packet sending.

A local application at node S is sending a TCP packet to node D, which is an endpoint in the BLR test network. Because a route in the system routing table sends all traffic with such endpoints to interface *tun0*, the main process of the BLR application on node S can read the packet from there. The packet already comes encapsulated in IP/TCP. The packet is then passed to the function *blr_create()*. First, this function checks the host table whether the position of node D is known. This example assumes that the position of node D is already known to node S. Next, the BLR header is added between IP header and IP payload. The fields in the BLR header are set to contain the following values: The type is `DATA`, the protocol is TCP, the sequence is set to the current value of a sequence counter internal to the BLR application and the backup distance is set to zero, since the packet is being sent in greedy mode. The source position field is set to contain the current position of node S, the destination position will contain the position of node D, and the field containing the previous position is set to zero, because the packet is newly created and not received from another node. Then, the IP header is updated to accommodate for the insertion of the BLR header, as already described in previous sections. The packet is then passed to the sendqueue process, along with values for *Add_delay* and with the Ethernet address of where to send the packet. For newly created packets, *Add_delay* is always zero, and in greedy mode, the target Ethernet address is always the broadcast address. The sendqueue process then immediately sends the packet via *pf_packet*, and stores a copy of the packet in the packet list, in case it needs to be retransmitted.

Both node A and B receive and process the packet, since it is broadcast on the MAC layer. The main processes of both nodes pass the packet to their *blr_forward* functions, since the destination address in the IP header is set to that of node D, and thus neither node A nor B is the endpoint for the packet. Both nodes first check if they are in the forwarding area. If so, they calculate a new value for *Add_delay*, which is slightly higher for node A than it is for node B. They pass the packet together with the calculated delay and the target Ethernet address (always the broadcast address during greedy mode) to the sendqueue process, which waits for the amount of time specified by *Add_delay*. Node B sends the packet first. Node A overhears this transmission, and the sendqueue process deletes the pending packet instead of sending it. By overhearing the forwarding of the packet from node B, node S knows that node B will handle the packet from now on, and it deletes the copy of the packet it has previously stored in the packet list. This mechanism is called *passive acknowledgement*, or in short: *passive ACK*.

Once node D receives the packet, it is passed to the *blr_assemble()* function, since the des-

tination address in the IP header matches the IP address of node D. A packet with type ACK, called a *final ACK*, is broadcast to indicate that the packet has reached its final destination and will not be forwarded anymore. Then, the position of node S is extracted from the BLR header and saved in the host table, before the BLR header is stripped off. The IP header of the resulting packet is updated, in order to reverse the changes done by the *blr_create()* function of node S. Finally, the packet is written to *tun0*, where it is received by the kernel and passed to the local application running on node D.

If node S does not know the position of node D when attempting to send the packet, transmission works as follows. The host table check in the *blr_create()* function fails, and the packet type field in the BLR header is set to LOCATION_REQUEST instead of DATA. The packet is forwarded by both node A and node B, since its type is set to LOCATION_REQUEST, and packets of that type are sent through a simple flooding mechanism instead of via BLR. See section 4.7.2 for more details. When node D receives the packet, it is passed to the *blr_assemble()* function, where the position of node S is extracted from the BLR header and stored in the host table. Then, the packet type field in the BLR header is changed from LOCATION_REQUEST to DATA and the packet is run through the *blr_assemble()* function again, where it is then processed like a normal DATA packet. Please note that the localisation mechanism, although integrated into the BLR application, is not part of the BLR algorithm as such, and is only used to provide a working real-world implementation.

4.6.2 Unicast Optimisation

The differences between unicast optimisation and greedy mode will be explained with the scenario from figure 4.4. Once node S receives the *passive ACK*, it knows that for packets destined to node D, the next hop will be node B. The MAC address of node B is thus stored in the host table. The same happens when node B receives the *final ACK* from node D. The next packet is sent from node A to node B and from node B to node D directly via unicast. After a specified amount of time, called the invalidation interval, each host purges its host table entries, and the application goes from unicast mode back to greedy mode when the next packet arrives.

4.6.3 Backup Mode

Backup mode is entered when a packet sent in greedy mode has not been acknowledged after the time defined by the TIMEOUT constant. See figure 4.5 for an example scenario. In this scenario, node S first tries to send a packet in greedy mode, with the destination being node D. Upon sending, node S stores a copy of the packet in the packet list. After the packet timed out due to no other node forwarding it, the packet is removed from the packet list and put into the backup queue, and the algorithm switches to backup mode. The backup mode setup, during which the next hop on the route to node D will be searched, begins. Until the search is completed, packets from *tun0* with destination D are not sent but stored in the backup queue. At the beginning of the backup mode setup, node S prepares a DISCOVERY_REQUEST packet: It is a BLR packet without payload, encapsulated in an IP packet. The position of the destination is put into the BLR header and the backup distance field in the header is set to zero. This indicates that the

DISCOVERY_REQUEST emerged from the node where greedy mode failed and backup mode started. The destination address in the IP header is set to the IP address of node D.

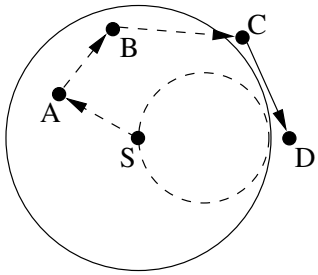


Figure 4.5: Packet sending in backup mode. Dashed lines indicate unicast traffic, continuous lines indicate broadcast traffic. Only the path of the DATA packets is depicted, the packets sent during backup mode setup are not shown.

Node A and B both receive the DISCOVERY_REQUEST packet and reply to node S by sending a DISCOVERY_REPLY packet. They store the distance between their current position and the position of node D in the BLR header. In order to make it possible for node S to distinguish between discovery replies meant for other destinations than D, the IP address of the destination, D in this case, is stored as packet payload. After sending the discovery request, node S waits for the amount of time specified in the constant DISCOVERY_TIMEOUT, and stores all replies it receives. After the timeout, the replies are processed. Since node A and node B both have a negative forward progress, node S applies the right-hand rule, which yields node A as the next hop. Node S can then send all the packets it has stored in the backup queue, after setting the distance between itself and node D in the backup distance field of the BLR header of every packet. When node A receives the packet it first checks whether the backup distance in the BLR header of the received packet equals zero. Since it does not, the packet must be forwarded by applying the right-hand rule. The packet is stored in backup queue and then node A conducts the backup setup procedure. It receives discovery replies from nodes S and B. After applying the right-hand-rule, it takes the packet from the backup queue and sends it to node B. The backup distance in the BLR header of the transmitted packet is still set to the distance between node S and node D. Node B repeats the same procedure. However, the DISCOVERY_REPLY packet it receives from node C indicates that node C has forward progress (i.e. is closer to node D than node S is). Node B sets the backup distance to zero and sends the packet to node C. Once node C receives the packet in backup mode, but with the BLR header field backup distance set to zero, it changes from backup mode to greedy mode and forwards the packet by broadcast.

4.7 Differences to the Implementation in the Simulator

Because of the inherently different nature of the simulator and a GNU/Linux system, differences between the implementation in the simulator and on the GNU/Linux system are manifold. This section lists the most important differences.

4.7.1 Header structure

Only one header is used instead of different headers for different packet types. The rationale is less complexity. The header used in the real-world implementation is already 2 bytes smaller than the header for `DATA` packets used in the simulation implementation. And since it is assumed that BLR operates in greedy mode most of the time, there is not much benefit in introducing different headers for different packet types.

4.7.2 Localisation

In the simulator, localisation information is provided by the simulator. Thus, when making the step from simulations to a real-world implementation, some sort of localisation service is needed. While localisation is a research aspect of its own and several solutions have already been proposed [24], it was not something central for this thesis, and so a simple solution for finding the geographical position of the destination node had to be found. Considering the small number of nodes in the test network, it is feasible to use a flooding mechanism, where the source node sends out a LOCATION_REQUEST packet which is then forwarded exactly once by each node that receives it. When the destination node receives the packet, it replies with a LOCATION_REPLY packet that contains its current position. This method is very simple, yet when using it, there are a few points that need to be considered.

How long should position data stay valid? This is of course largely dependent on the mobility behaviour of the nodes. While [2] suggests methods to judge position data by additional information about node movement, these techniques have not been used in the current implementation, for reasons of simplicity. Instead, position data is invalidated periodically, and the source is forced to send out a new location query. This method can be further simplified when two-way communication is assumed: In case of TCP for example, invalidation can be skipped, as a changing position of the destination can simply be extracted from packets returning from the destination. This method was eventually deployed, since during all measurements, two-way traffic was used. A further advantage of this method is that location requests do not interfere with measurement results, since it is not necessary anymore to periodically transmit location request packets.

A further optimisation of this method is possible: The first DATA packet sent by the source node can be piggybacked onto the LOCATION_REQUEST packet, thus avoiding the need for an extra packet. Since the destination node can extract the position of the source node from the LOCATION_REQUEST packet, it can immediately send the next data packet, and LOCATION_REPLY packets are not needed anymore (still assuming two-way traffic). So the only additional traffic that is generated by localisation are the additionally forwarded packets by nodes not in the direct path between source and the destination.

It is questionable whether the applied method of an integrated, optimised localisation service is the right way to go. With a separate localisation service, with which the BLR application only interacts by asking it for position data, the BLR application would be much more flexible. In the current implementation, flexibility is traded for performance. With an external localisation service, it would also be possible to separate the communication medium used by BLR from the medium used by the localisation service. This allows to separate the measurement results

from the localisation overhead and decreases the complexity of the BLR application. To conduct outdoor tests however, a simple, working localisation service is needed, which can use the same communication medium as BLR, since this is the only medium available.

A problem that has not been specifically addressed arises when the source node sends packets out faster than the packets carrying location data from the destination can arrive. The possibility of this happening increases with higher sending rates as well as increasing number of nodes (which increases the average distance between source and destination). Since in the current case the test network is small and data is sent at relatively low rates, this did not affect measurements.

The conclusion is that the described method for localisation is sufficient for the scope of this thesis, but would have to be refined for larger or more complex scenarios.

4.7.3 MAC-layer Control

When broadcasting, a packet is acknowledged through a passive ACK, i.e. by the detection of the relaying of the packet through another node. During unicast, this is not possible, as demonstrated by figure 4.6.

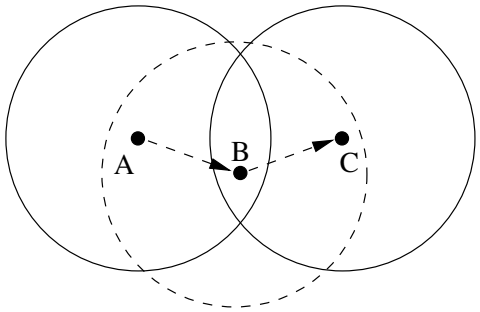


Figure 4.6: Unicast relaying. Transmission ranges of A and C are denoted by the continuous circles, transmission range of B is denoted by the dashed circle.

Node A sends a packet and waits for the relaying of this packet by another node. If node B relays the packet as broadcast, node A can overhear this transmission and will interpret it as a successful relaying. But if node B forwards the packet to C by unicast, node A will node hear this transmission and assume that the packet timed out. However, in 802.11b unicast packets are explicitly acknowledged on the MAC-layer. If the MAC-layer does not receive such an acknowledge in time, it may retransmit the packet up to 7 times before giving up. In the simulator, the MAC-layer can signal successful relaying or a timeout to the upper layer, and the simulator implementation makes use of this: the BLR part of the simulator implementation is noticed when a packet has been successfully transmitted or when it timed out. In the real-world BLR implementation however, this is not easily possible. The MAC-protocol is largely implemented in the firmware of the 802.11b card, which makes accessing the said functions virtually impossible. Acknowledging every unicast packet explicitly with a short ACK packet would nearly double the number of packets sent during unicast transmissions, and thus render the unicast optimisation pointless. Another method to detect the relaying of unicast packets had to be found. The solution is to put the outbound interface into promiscuous mode, so that unicast packets to other

nodes can be received and processed by the BLR application. The drawback of promiscuous mode is obvious: since much more traffic has to be processed, power consumption increases. This is discussed in more detail in section 6.3.3.

Care has to be taken not to make packet timeouts too small: If the BLR timeout is chosen too close to the overall MAC-layer timeout (including all possible retransmission attempts), it may be possible that the BLR implementation detects a timeout and starts a retransmission, while the MAC-layer is still trying to resend the same packet. The same packet will then be resent twice, once by unicast through the MAC layer, and once as broadcast by the BLR application, (since the BLR applications falls back to greedy mode after an unsuccessful unicast transmission).

4.8 Challenges during Implementation

This section describes different challenges encountered during the implementation of the BLR algorithm. Some are not specific to a real-world implementation and might also happen in the simulator, depending on the amount of details that the simulator considers. However, the likelihood of their appearance under real-world conditions is probably higher than in simulated environments. Others challenges encountered are specific to the presented real-world implementation on the GNU/Linux platform, and can be avoided entirely in a simulated environment.

Problems that can affect the outdoor operation of an ad-hoc networking protocol are usually caused by four factors:

- Positional inaccuracies due to imprecise GPS data
- Terrain characteristics such as obstacles that block or reflect signals
- Mobility-induced changes of topology
- Timing problems caused by clock skew, inconsistent hardware quality or inaccurate clock synchronisation methods

As these factors are omnipresent in the field of MANETs, it is interesting to see how the BLR implementation is affected by them and what solutions exist for minimising their impact. Except for mobility-induced changes, these factors are usually not modelled in simulations.

4.8.1 Collisions

The lab testbed consists of five nodes, lined up on a desk. Since they are all located in the same room, they are inside each others transmission range. For measurements that require some nodes to be outside of the transmission range of other nodes, a virtual topology was simulated. This may lead to an artificial increase of collisions, since packets that virtually cannot collide may do so in reality because all nodes are inside the same collision domain. For the same reason however, the hidden terminal problem[35] is avoided during lab tests, while it may cause collisions during the outdoor tests. It is thus not clear whether the chance of collisions is higher in the lab environment with a simulated topology than it is with an outdoor setup, where the nodes are spaced several hundred meters apart.

Collisions are especially disruptive during backup mode, since backup mode relies on the exchange of control packets. Collisions may happen during the setup phase of backup mode as well, not only during the actual packet transmission. Instead of simple packet loss, which can be handled by upper network layers, collisions during backup mode setup can cause existing paths to fail or to be altered. For example, if two nodes send discovery replies which collide, routing may fail, even though a path exists. See section 6.2.1 for a possible solution to these problems.

4.8.2 Unicast Anomalies

Since it sometimes happens that multiple nodes in the forwarding area forward the same packet (as described in the next section), the sender may receive two ACKs for the same packet. For reasons of simplicity, in the current implementation only the first ACK is taken into account. This may lead to the situation where a suboptimal node is chosen as the next hop for unicast forwarding. While this causes some loss of progress, its impact is limited temporally to the length of the invalidation interval.

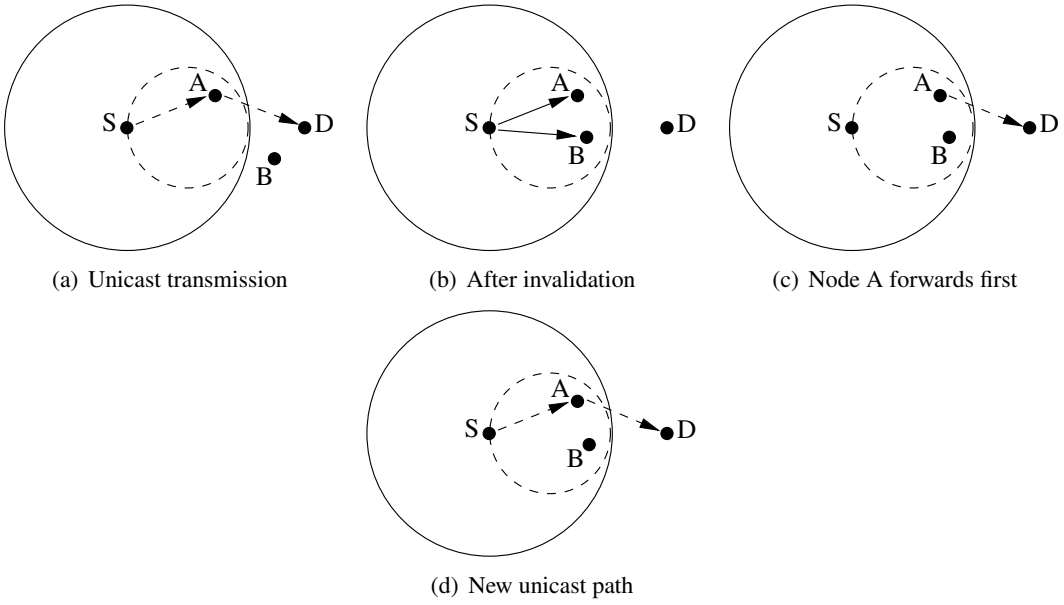


Figure 4.7: Unicast forwarding.

The scenario described in figure 4.7 however results in a permanent selection of a suboptimal node, at least until topology changes require a reselection. Figure (a) shows the initial situation where unicast traffic is transmitted from node S to D via A. This continues until invalidation happens at node S. The next packet is then broadcast as seen in figure (b). Since node A did not yet invalidate, it can forward the packet immediately by unicast, while node B applies *Add_Delay* before attempting to broadcast the packet. As seen in figure (d), S chooses node A as the next unicast hop towards D, since node A forwarded the packet faster than node B.

The solution to this is to allow unicast forwarding of a packet only when the packet was already received by unicast. Node A in the above scenario would then not be allowed to forward the packet faster than node B since it has to broadcast and thus is forced to take *Add_Delay* into account.

4.8.3 Duplicate Packets

Under ideal conditions, duplicate packets do not occur. Because of the four reasons mentioned at the beginning of this XXX chapter however, they do happen in practise. This section describes a few example scenarios where duplicate packets occur. Figure 4.8 shows the most common cause for duplicates experienced during the measurements. The relaying of a packet is not detected fast enough. Node S sends a packet towards node D. Both node A and B attempt to forward it. Node B will introduce the smaller *Add_Delay* and send first. Because of an anomaly node A does not overhear the relaying and sends the same packet again. Node D receives this as a duplicate.

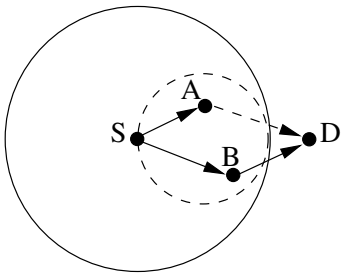


Figure 4.8: Relaying of a packet is not detected fast enough.

One reason for this is outlined in section 4.8.7, another reason could be an obstacle between A and B, that makes it impossible for A to overheard the relaying by B. In figure 4.9, the formation of a duplicate caused by mobility is shown. Source S sends a packet, node A which is currently moving receives it. At the time of reception, node A is located at position A1, close to the border of the transmission range of node S. At the time node A actually forwards the packet, it has moved to position A2 which is outside the transmission range of node S. Since S now cannot detect the relaying of its packet, it will resend it, causing a duplicate.

To reduce the impact of duplicates, a filtering mechanism was implemented. At various points during packet procession, the packet is matched against a table containing the recently received packets, called the RRT. However, packet IDs may only be added to the RRT if they were received while the receiving node was inside of the forwarding area of the sending node. Otherwise, the receiving node may add a packet that he should relay at a later point in time, for instance when greedy mode fails and the sending node resends the packet in backup mode.

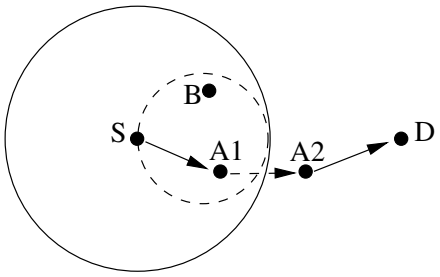


Figure 4.9: A duplicate caused by mobility.

4.8.4 IP Fragmentation

IP fragmentation occurs when a network interface receives a packet that is too large for it to handle. Depending on the value of the DF (Do not Fragment) bit in the IP header of the packet, the IP stack either fragments the packet into smaller fragments and forward them, or it sends back an ICMP error packet, indicating that it would like to fragment but cannot due to the DF bit being set. Problems with packet fragmentation could have been avoided entirely by placing the BLR header between the Ethernet header and the IP header. This way, each IP fragment would have had the complete BLR header prepended and would contain all data necessary to find its destination. But with the current implementation, where the BLR header is part of the IP payload, only the first IP fragment contains the BLR header and thus the information needed to find a path to the destination. Subsequent fragments will not contain any BLR information. Therefore, IP fragmentation has to be avoided in the current implementation. To achieve this, the MTU of the virtual tunnel interface is decreased by the size of the BLR header, which makes sure that the maximum packet size is not larger than the standard MTU of 1500 bytes. Otherwise, packets with the size of 1500 bytes would exceed the maximum packet size, due to the addition of the BLR header, and would have to be fragmented. Furthermore, the DF bit is set for all outgoing packets and PMTU[36] discovery is used to handle links for which the packets are too large. This offloads the responsibility of fragmentation to the TCP stack.

4.8.5 Cloned Packets

Pf_packet works by creating a copy of an incoming packet and delivering this copy to the BLR application. The original packet is passed to the kernel for normal processing. Since only the packet passed to the BLR application is desired, the original packet has to be blocked somehow. This is achieved by deploying an IPtables [37] packet filter right after the *pf_packet* facility. This filter blocks all incoming traffic that has the protocol number of BLR set in the IP header. For broadcast traffic alone, this filter would not be necessary, since the kernel simply drops broadcast traffic with a protocol number for which there is no open socket. It does not reply with an error message, since with broadcast traffic, this would not make sense. However, when the kernel receives unicast traffic with an unhandled protocol number, it does send an ICMP error message

to the sender. Without the packet filter, every packet sent by unicast would trigger such an error message. For outgoing packets, no filter is necessary, since all packets destined to the BLR network take the defined route through the virtual interface *tun0*.

4.8.6 Short-range Testbed in Indoor Scenarios

To allow testing of topologies where certain nodes are outside the transmission range of others, connectivity between nodes has to be restricted somehow. Since during indoor tests and implementation, all nodes were located in the same room and therefore inside each others transmission range, the problem was to artificially remove connectivity between some nodes. The most elegant solution was to implement a MAC-layer filter. This filter operates directly on the *pf_packet* socket, which results in the BLR application never seeing packets that it should not see due to the simulated topology. This approach saves processing work on the side of the BLR application, since the kernel does all the necessary filtering. The implementation of the MAC-filter is done by means of the Berkeley Packet Filter [38] language. The GNU/Linux implementation is called LSF, short for *Linux Socket Filter*, and is compatible with the BPF language.

4.8.7 Interrupt Granularity

In the network simulator, `MAX_DELAY` values as low as 2 milliseconds could be achieved. The meaning of `MAX_DELAY` is briefly repeated here to make the following explanation more understandable: When in broadcast mode, a forwarding node introduces *Add_Delay* in the range $[0, MAX\_DELAY]$ milliseconds before it forwards a packet. The actual value used for *Add_Delay* depends on the position of the node: the better it is located, the lower *Add_Delay* will be. When it was tried to put this into practise in the real-world implementation, a system-wide limit was encountered: The granularity of the timer interrupt. This granularity is defined by a compile-time kernel constant called HZ. On Linux kernels older than 2.5, this constant is set to 100, which means that timer events hit at a frequency of 100 Hertz, or once every 10 milliseconds. On newer kernels, the value has been increased tenfold, resulting in timer events every 1 millisecond.

In practise, this means that the *select()* system call, which is used extensively in network programming, will return at 1 millisecond intervals only. This largely impacts scenarios where multiple nodes are located relatively close together and all attempt to forward the same packet. Values for *Add_Delay* in the range $[0, 1000]$ microseconds all result in *select()* sleeping around 1000 microseconds, values for *Add_Delay* in the range $[1001, 2000]$ microseconds result in an actual delay of about 2000 microseconds and so on. Therefore, nodes with similar distances from the sender will forward a packet at the same time, rendering the method for selecting the optimal forwarder by position useless.

To limit the impact on the measurements, affected scenarios have been tested with a larger `MAX_DELAY` and nodes were spaced out more widely, so that each node gets an effectively unique *Add_Delay*.

While theoretically possible, a further increase of the HZ value is not yet completely supported by the Linux kernel. The drawbacks of raising the HZ value, however, is also raising overall timer overhead. With switching from a value of 100 for HZ to a value of 1000, ten times

more timer interrupts are generated, so the overall overhead for timer interrupts increases accordingly. For a personal computer, these overheads are negligible, for a small mobile device, they might not be.

4.9 Lessons Learnt

If the step from a prototype to a commercial application will be taken, it would make sense to implement a few things differently. The insights that have been gained during the course of this work will be discussed in this section. On the implementation side, the following improvements would be interesting to investigate: The use of separate processes that communicate via pipes can be improved upon in two levels. First, instead of using pipes for communication, shared memory could be used. While more complicate to implement, using shared memory offers a performance benefit over using pipes. Even more performance improvements could be achieved with a multi-threaded design. This would be a necessary step if high traffic networks with multiple sources and destinations should be handled. To improve the time-critical sections that handle packets that require an *Add_Delay*, these parts need to be moved from user space into the kernel.

On the architectural side, the integration of the BLR header between IP header and IP payload is disputable. On the one hand, the encapsulation inside IP worked well, and was efficiently implemented. On the other hand, BLR has characteristics of a routing protocol as well as characteristics of a link layer protocol. For instance, it is responsible for finding a route to the target, but also for packet retransmissions. In the simulator implementation, BLR is tightly coupled with the MAC layer, by hooking into the timeout and retransmission mechanism of 802.11b. This tight coupling is not possible in a real-world implementation, since access to the MAC layer is very limited for consumer-grade 802.11b equipment. An implementation of BLR as MAC protocol would thus make sense from an architectural point of view, as well as from an implementation point of view.

Chapter 5

Measurements

5.1 Test Scenario and Parameters

Measurements have been conducted to verify the correctness and robustness of the implementation, as well as to get insight into how BLR performs under real-world circumstances, as opposed to simulated environments. The first part of the measurements were done indoors. Indoor measurements allow for a wider range of parameters and topologies to be tested, since testing can be conducted by a single person, and is generally less time-consuming than outdoor testing. Also it is possible to minimise the influence of external factors such as weather, interference through obstacles or position inaccuracies caused by variations in GPS signal reception. Outdoor measurements, while restricted in their significance because of these external factors, are still very valuable, as they test the robustness and correctness of the algorithm even better than indoor measurements. Another advantage of outdoor measurements is that they can be conducted under the same conditions that would apply when the algorithm would be used in a real device, something which is not possible with indoor tests.

The measurements are carried out with a set of five laptops, of different brands and models. All laptops are equipped with an 802.11b network card set to transmit with 2Mbit/s in ad-hoc mode. RTS/CTS has been disabled on all cards. The power-save mode of the cards is turned off and no encryption is used. The tests run on Gentoo Linux [25] with the Linux kernel 2.6.8.

One big problem was to synchronise the time between the nodes. While GPS would allow for a very precise setting of the time[34], the use of GPS device on every node as a source for time is impossible indoors, since GPS receivers need a clear view to the sky in order to function. Trials to synchronise the nodes via the NTP protocol[39] failed, due to the reason that the laptops were of different brands and the drift of their internal clocks varied greatly. Even after hours of synchronisation with NTP via Ethernet, time drifted quickly enough to not allow for precise measurements. It was thus decided to send two-way traffic and measuring time only on one node and thereby eliminating the necessity for synchronisation.

Traffic sending is always initiated from one node only. The traffic is sent by the unix ping utility, which produces ICMP traffic. As packet size, the default of 56 bytes was chosen, together with the ICMP header this adds up to 64 bytes of payload. Unless stated otherwise, 2000 ICMP echo requests were sent, together with the echo replies, this results in a total of 4000 data packets per measurement. Echo requests are sent out with a rate of 10 packets per second unless stated

otherwise.

The parameters that can be varied between measurements are limited. The maximum number of nodes is fixed and mobility is very limited, or even nonexistent in the case of indoor tests. As [3] has shown, the forwarding areas circle and ruleaux perform identically well in simulations with low node density. Therefore, the circle has been chosen as forwarding area for all measurements. The lower boundary of `MAX_DELAY` has been evaluated during the implementation already: Below 5 ms, the implementation will not perform correctly due to the problems mentioned in 4.8.7. This leaves the topology, the upper boundary of `MAX_DELAY` and the timeout parameters *TIMEOUT* and *DISCOVERY_TIMEOUT* for variation between measurements. In addition, the different BLR modes (greedy, unicast optimisation and backup mode) can be compared. Of course, general aspects such as delivery time and packet loss have been examined as well.

5.2 Indoor Tests

The indoor tests were all conducted inside an office, with the five laptops lined up on a desk. Since GPS devices are not usable inside buildings, the positions of the nodes have been evaluated before each measurement and were then been hard-coded. The sending node is always set up with latitude and longitude both zero. This simplifies the determination of the positions of the other nodes. The Chain topology, for example, can then be created by spacing out all nodes east of the sending node on the equator, all with a latitude of zero and with a constant increase of longitude. To have more complex topologies implemented, latitude and longitude were determined by trial and error, and then fine-tuned with a separately implemented utility. This utility periodically sends out position beacons and displays all nodes in range and their distance to each other. The maximum transmission range, with is a fixed constant in the BLR protocol, has been set to 250 meters during all tests.

Once all nodes were positioned, a virtual topology was ready. It is called virtual, because the actual topology still had to be enforced through the MAC filters. If this would not have been done, the destination node would be able to directly receive packets from the source node, since those nodes are both located in the same room. This MAC filter is also hard-coded, as it is only used for indoor experiments, and not really part of the BLR algorithm. During outdoor tests, no MAC filter is necessary, since the nodes are spaced out so far that that only neighbouring nodes can overhear each other.

After the positions have been determined and measurement parameters were configured, the BLR application was launched and the ping utility was used to send ICMP traffic. Finally, another custom script was used to gather output from all nodes, parse the output and create statistical data of the measurement.

5.2.1 Topologies

For indoor tests, six different topologies have been chosen. A variety of different test cases is possible with these topologies. For easy referral, they were given short names that should reflect their structure. The following topologies have been used:

- Chain: All five nodes are lined up on a straight line, with equal distances between all neighbours. The distance is the maximally possible distance so that neighbouring nodes can still overhear each other. This means that source and destination node each have one direct neighbour, the other nodes each have two. Figure 5.1 illustrates the Chain topology.
- Pairs: The nodes also lie on a straight line, but they build pairs. The distance between paired nodes is 20 meters, the distance between nodes that are not pairs is the maximum distance of 249 meters. The number of neighbours for each node is the same as with the Chain topology. See figure 5.2 for a visualisation.
- Contention1: Source node and destination node are spaced out twice the maximum distance, and the forwarding area of the source contains the other three nodes, as indicated in figure 5.3. Only one node from within the forwarding area is in range of the destination node.
- Contention2: The difference to Contention1 is that the destination node is in range of all other nodes except the source node, as shown in figure 5.4.
- Uninvolved: This is the same as Contention1 except that only one node is located in the forwarding area of the source. The other two nodes are outside of the forwarding area and thus uninvolved in the direct transmission. See figure 5.5.
- Backup: With only five nodes, not many interesting topologies for backup-mode evaluation are possible. The one tested looks as show in figure 5.6. The forwarding area of the source node is empty, and the first and second node after the source have no forward progress. The first node to introduce forward progress is the third hop.

The following two pages show all topologies. Unless noted otherwise, dots signify nodes, the source is the leftmost node, the destination is the rightmost node, circles signify the transmission range and line style or fill patterns are only used to enhance visibility.

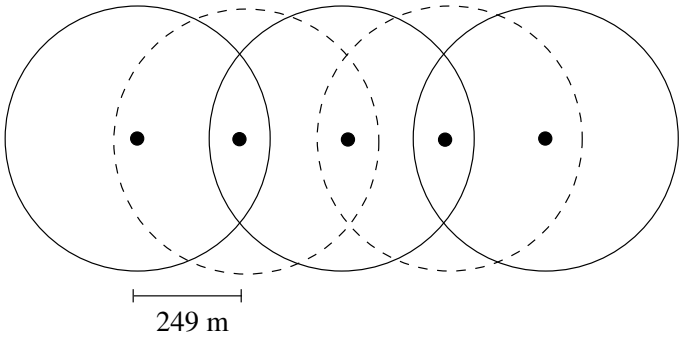


Figure 5.1: The *Chain* topology

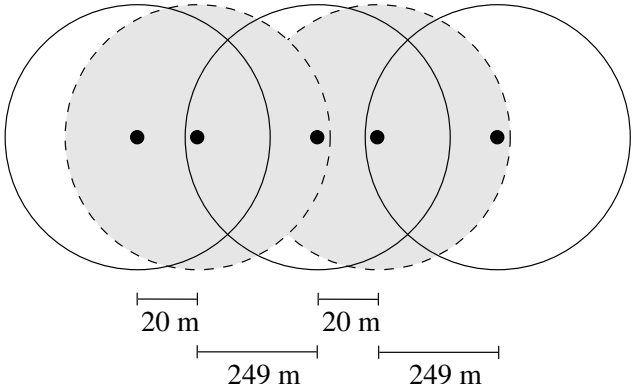


Figure 5.2: The *Pairs* topology

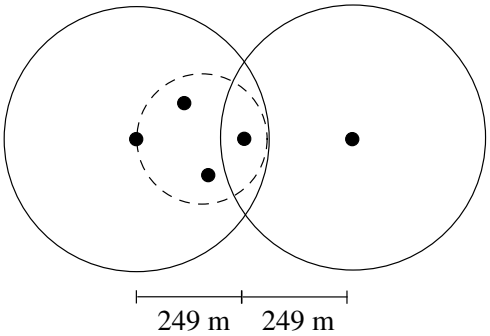


Figure 5.3: The *Contention1* topology. The dashed circle denotes the forwarding area of the source node.

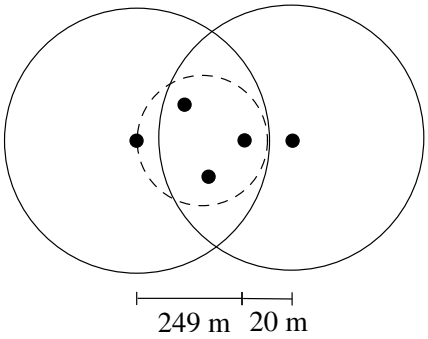


Figure 5.4: The *Contention2* topology

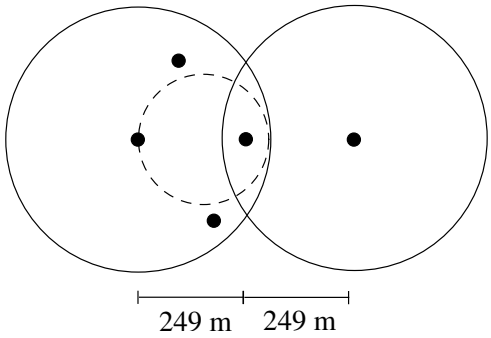


Figure 5.5: The *Uninvolved* topology

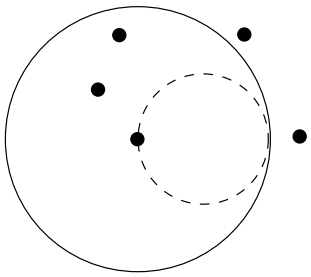


Figure 5.6: The *Backup* topology. The dashed circle denotes the forwarding area of the source node.

5.2.2 Performance Overview

The first few measurements were conducted to get an impression of how well the BLR implementation can perform in an optimal scenario. The goal was to find out how much overhead the BLR protocol and implementation adds, and how much additional delay per hop can be expected.

All tests have been carried out in the Chain topology and with a `MAX_DELAY` of 5 ms. In this section, all results denote the end-to-end delay between source node and destination node, whereas in all following sections, the results represent the round-trip-time, i.e. the time it takes from the moment the source node sends an ICMP echo request until it receives the corresponding ICMP echo reply. The reason that this section uses a different metric than all following sections is that network performance measurements are usually expressed in end-to-end delays. For symmetrical topologies such as the Chain, it is possible to approximate the end-to-end delay by halving the round-trip-time. However for non-symmetric topologies, this approximation does not work, since the path from the source node to the destination node might be more expensive in terms of delay as the path from destination to source. Since many comparisons between symmetric and non-symmetric topologies will be made in the next sections, it is more appropriate to use the round-trip-time in results for the other sections of this chapter.

Fig 5.7 shows the end-to-end delay in the Chain topology with five nodes. The average end-to-end delay was 8.74 ms for four links. The delay that one BLR hop induces is therefore about 2.19 ms.

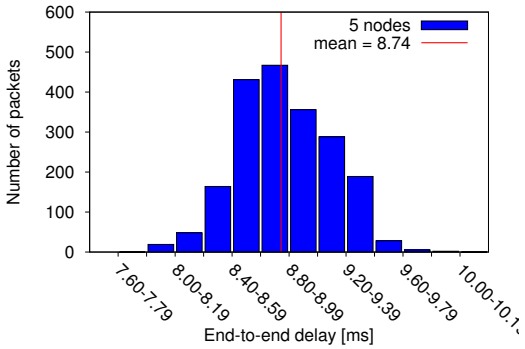


Figure 5.7: Chain topology, five nodes

To illustrate the amount of overhead the BLR protocol and implementation causes, the next measurements were made with only two nodes. The test data has been transmitted once without BLR (i.e. a normal 802.11b transmission between two nodes in ad-hoc mode) and once with BLR in greedy mode. Figure 5.8 shows that the two end nodes in a BLR transmission induce an overhead of about 0.25 ms each. Please note that this value cannot be directly compared to the 2.19 ms achieved in the previous measurement. The 2.19 ms include the time required for the complete data transmission, whereas the 0.25 ms achieved in this experiment only denote the overhead that the BLR protocol induces.

It can also be seen that the values in the BLR scenario are much more scattered. The probable

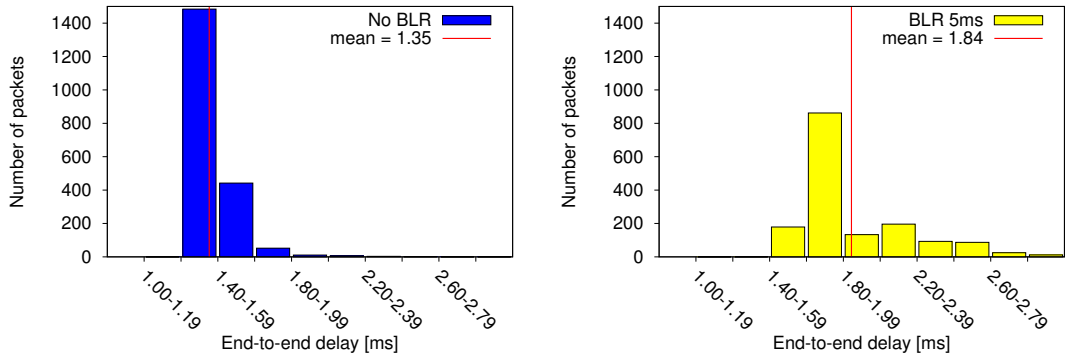


Figure 5.8: Two nodes, without BLR (left) and with BLR in greedy mode (right)

reason for this is that BLR is implemented in user space, which may cause unpredictable delays because of the context switches between kernel and user space. This high deviation has been observed in all measurements with BLR, although in is sometimes not clearly visible in the graphs due to the class distribution of the histograms.

Since in the above scenario no actual forwarding takes place, the next interesting question is how much overhead BLR adds when forwarding. To approximate this, a comparison is made between the Chain topology with five nodes (as seen in figure 5.7) and a Chain topology with only four nodes (as seen in figure 5.9). The difference in end-to-end delay between these measurements approximately indicates the additional delay that one more BLR node induces in the Chain topology. Thus, one additional BLR node that is optimally positioned (as in the Chain topology) will add a delay of about 1.5 ms.

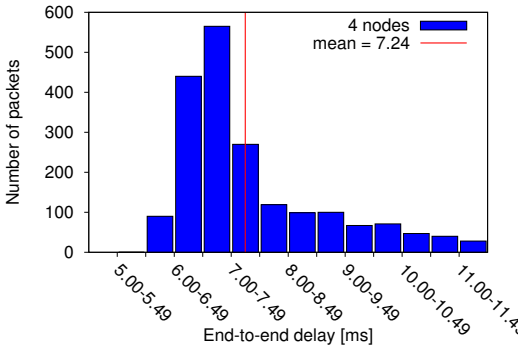


Figure 5.9: Chain topology, four nodes

When this value is compared to the previously calculated delay of 2.19 ms per hop, the conclusion can be drawn that the endpoints of a transmission induce a higher delay than the forwarding nodes. This is not unexpected, as endpoints have to add (or remove) the BLR headers, recalculate the IP checksums and read (or pass) the packet from (or to) the *tun0* interface. The time a packet takes between the *tun0* interface and the ping utility, where time is measured, also needs

to be taken into account when comparing values.

In all the previous measurements, data was sent at a rate of 10 packets per second. What happens if the rate is changed? As more packets means more possible collisions, it is to be expected that the end-to-end delay increases with an increase of the sending rate, because more retransmissions are necessary. Since the application is not multi-threaded, an increase of the sending rate can result in a slowdown as well, in case incoming packets can not be processed immediately. This is very well illustrated in figure 5.10. The deviation of the end-to-end delays in the scenario with the higher sending rate is much higher than the deviation in the scenario with the lower rate. This indicates that more packets need to be retransmitted when the sending rate is increased. The average end-to-end delay also increases with higher sending rates.

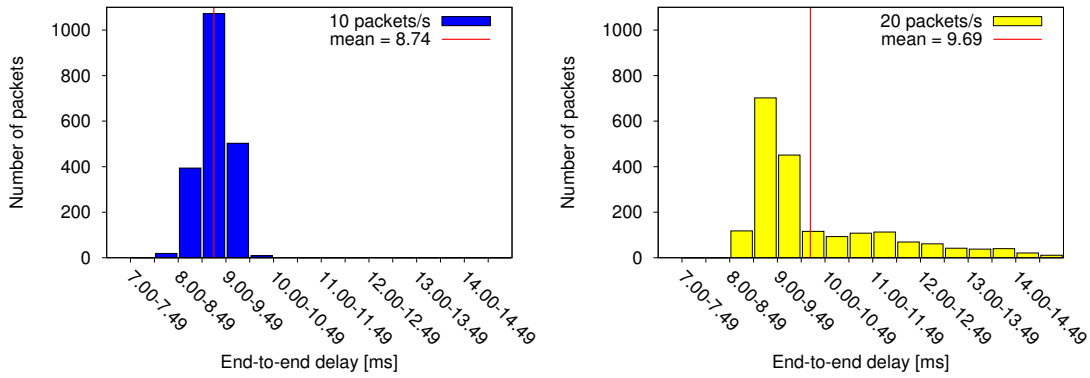


Figure 5.10: Chain topology, different packet sending rate

5.2.3 Profiling

A simple attempt at profiling the BLR implementation has been made, in order to find out where the in code the implementation spends the most time. Since a number of context switches between user space and kernel space as well as context switches between BLR processes are involved, it is difficult to calculate a meaningful number for BLR-induced overhead. At four positions in the application code, timestamps were taken. This was at the reception of a packet through *tun0* or *pf_packet*, before the main process writes a packet to the pipe connected to the sendqueue process, when the sendqueue process reads from the pipe, and when the sendqueue sends the packet.

The time between packet reception and writing to the queue, where no process switch is involved, typically takes between 100 microseconds and 400 microseconds. The time that elapses between writing a packet to the pipe in the main process and read it in the sendqueue process lies in the same range. The time between reading and sending a packet inside the sendqueue process takes anywhere between 100 and 1000 microseconds. Thus, as mentioned in the previous chapter, the *select()* system call inside the sendqueue process, where the application waits for the time specified in *Add_Delay* before a packet is sent, is where the implementation spends most of the time. The time not accounted for when comparing these values against the 2.19 ms achieved

in the previous section is the time a packet spends outside of the BLR application. Since the actual time spend in this section varies greatly, the implementation would benefit from moving that part into kernel space to reduce this variance.

5.2.4 Influence of MAX_DELAY

In this section, the influence of the BLR parameter `MAX_DELAY` is examined, and the measurement results are used to verify the correctness of the implementation to some extent. From now on, all measurement results are specified as the round-trip-time of the ICMP packets being sent. Figure 5.11 combines the results of two measurements in the Chain topology with five nodes. The first measurement was conducted with a `MAX_DELAY` of 5 ms, the second with a `MAX_DELAY` of 25 ms. In the Chain topology, all forwarding nodes are located close to the border of the forwarding area of their predecessor, therefore they provide maximal possible progress. This means that they introduce a minimal delay, as can be seen in the formula used to calculate *Add_Delay*:

$$Add\_delay = MAX\_DELAY \cdot \left(\frac{r-p}{r}\right)$$

Since in the Chain topology the progress p is almost equal to the transmission range r, it is obvious that the value of `MAX_DELAY` has little influence over the *Add_Delay* in this topology. So the expected result is that the value of `MAX_DELAY` does not influence the round-trip-time, and this can be confirmed when looking at figure 5.11. The average round-trip-times are 17.49 ms for a `MAX_DELAY` of 5 ms and 17.46 ms for a `MAX_DELAY` of 25 ms. Both averages are inside each others confidence interval which confirms that the measured values are statistically equal.

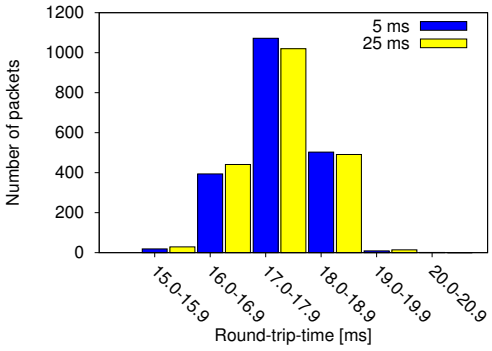


Figure 5.11: Chain topology, different values for `MAX_DELAY`.

On different topologies however, the value of `MAX_DELAY` should have an effect on the round-trip-time. To illustrate this, a measurement with the Pairs topology has been made. The results are shown in figure 5.12. As expected, the increase of `MAX_DELAY` has a significant effect in the Pairs topology, since on the way between source node and destination node, two of the three forwarding nodes are located close to their predecessor and therefore introduce a high *Add_Delay*. For the returning packets (ICMP echo replies), one of the three forwarding nodes introduces a high *Add_Delay*.

These measurements show that the implementation works as expected with changes of the value of MAX_DELAY.

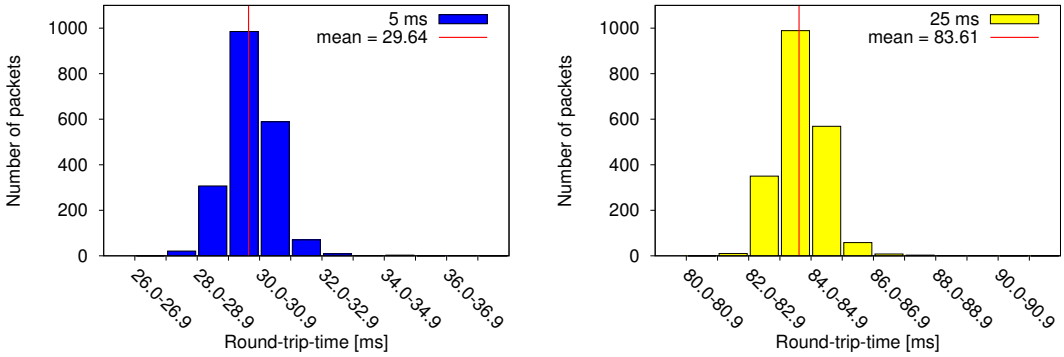


Figure 5.12: Round-trip-times in the Pairs topology, with different values for MAX_DELAY. Please note the different scale on the x-axis.

5.2.5 Influence of Additional Nodes

As already mentioned in section 4.8.7, when the forwarding area contains multiple nodes, the implementation does not behave as intended in the specification of the algorithm. Nodes cannot detect fast enough when another node already forwarded a packet that they are also to forward. This results in packets forwarded by multiple nodes whenever more than one node is located in the forwarding area. In the topologies Contention1 and Contention2, this was immediately noticeable in tests made with a MAX_DELAY of 5 ms. To increase the chance of the algorithm performing as expected with those topologies, the measurements in this section have all been conducted with a value of 25 ms for the parameter MAX_DELAY. Figure 5.13 shows the results from measurements with topology Contention1, and 5.14 shows Contention2.

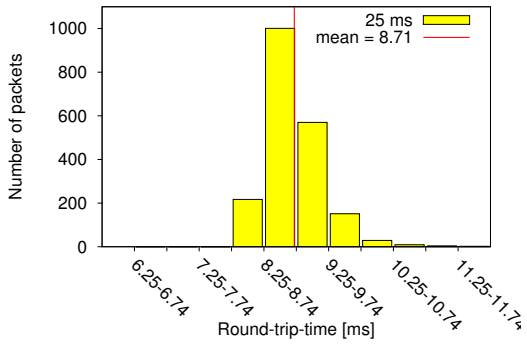


Figure 5.13: Contention1 topology, sent with a MAX_DELAY of 25 ms

Both topologies show very similar results. The algorithm performs slightly better for Con-

tention1 than for Contention2. This is not unexpected as the path for the ICMP echo reply packets is not the same in both topologies. The forwarding node for returning packets in Contention2 introduces a slightly higher *Add_Delay* than the forwarding node in Contention1, since it is located a bit closer to its predecessor. This is not clearly visible in figures 5.3 and 5.4.

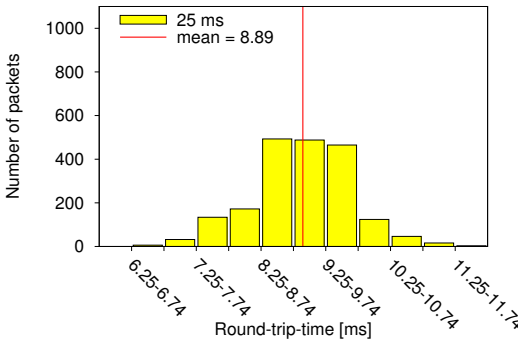


Figure 5.14: Contention2 topology, sent with a `MAX_DELAY` of 25 ms

The algorithm went to backup-mode a bit more often in the Contention1 topology than it did in the Contention2 topology (still only about 10 times per 2000 packets, this is discussed in more detail in section 5.2.10). This is also not surprising. In the Contention1 topology, only one node can directly relay packets between the source node and the destination node. In the Contention2 topology, all forwarding nodes are reachable by both source and destination. This increases the chances for successful delivery in the Contention2 topology. And since backup mode packets are filtered in the displayed results of the measurements, this observation does not contradict the better performance of the Contention1 topology mentioned in the previous paragraph.

In both contention topologies, the amount of packets forwarded by sub-optimally positioned nodes was only around 0.1%. This shows that with a high enough `MAX_DELAY`, the interrupt granularity problems described in section 4.8.7 are less severe. The topology called Uninvolved performed as expected with zero packets forwarded by nodes positioned outside the forwarding area.

5.2.6 The Unicast Optimisation

In this section, the behaviour of the unicast optimisation and the performance improvements that result from it are examined. In theory, the various topologies benefit differently from enabling the unicast optimisation. For example only a minimal *Add_Delay* per hop is introduced in the Chain topology, because all forwarding nodes are already optimally placed at the border of their predecessors forwarding area. Surprisingly, enabling the unicast optimisation in the Chain topology results in a notable performance improvement, as figure 5.15 shows. Apart from the difference in the round-trip-time, the two measurements showed similar characteristics.

The performance difference between the two measurements can be explained as follows. As described in 4.8.7, even a minimal *Add_Delay* may result in an actual delay of up to 1 millisecond. When sending with the unicast optimisation, packets are sent directly without an

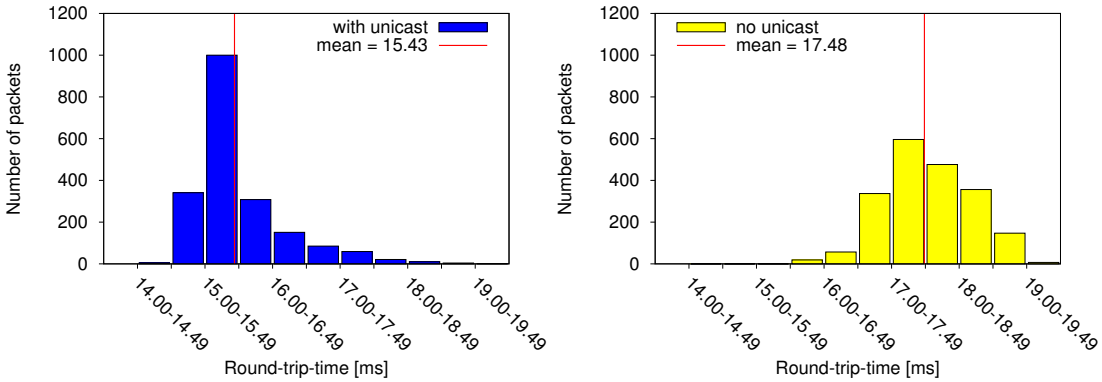


Figure 5.15: Chain topology, unicast compared to broadcast. The unicast optimisation offers a benefit over greedy mode, even though the forwarding nodes are already optimally positioned

additional *Add_Delay*. In the Chain topology, packets are forwarded six times, which means that greedy mode suffers from an additional delay of six added random delays between zero and 1 millisecond, when compared to the unicast optimisation. The measured difference of about 2 milliseconds seems to fit into this hypothesis, although the vague assumptions concerning the actual delay do not allow for a more precise reasoning. Furthermore, since BLR in the current implementation operates in promiscuous mode, both broadcast and unicast packets have to be examined by the BLR code. If the implementation would operate in non-promiscuous mode, sending by unicast would have an additional advantage: Nodes that are not directly involved in a transmission could discard those packets that are not destined for them already at the MAC layer. The packets would not have to be inspected by the BLR code, which would save additional processing time.

In other topologies than the Chain, the performance gained through the unicast optimisation is even more noticeable. Figure 5.16 shows various histograms with round-trip-times for the Pairs topology.

Several conclusions can be drawn from these measurements. Besides the substantial improvement in performance that the unicast optimisation delivers, one can see that sending in unicast mode makes the value of `MAX_DELAY` virtually irrelevant: Both measurements that had unicast optimisation enabled resulted in a similar average round-trip-time. The conclusion that can be drawn is that the higher the value for `MAX_DELAY` is, the more improvement unicast optimisation provides. The average round-trip-time for the Chain topology is exactly the same as for the Pairs topology when the unicast optimisation is enabled, since except for the distance between nodes, both topologies are equal. Interestingly, when sending in unicast, the performance for a `MAX_DELAY` of 25 milliseconds (15.43 ms average) was a bit better than for 5 milliseconds (15.37 ms). Since the averages lie slightly outside the 95% confidence interval of each other, this is more than a statistical deviation. The values stayed the same with repeated tests, and no apparent reason for this phenomenon could be found.

In the contention topologies, the improvement in round-trip-time that is achieved by the unicast optimisation is of course smaller than with other topologies, since the contention topologies

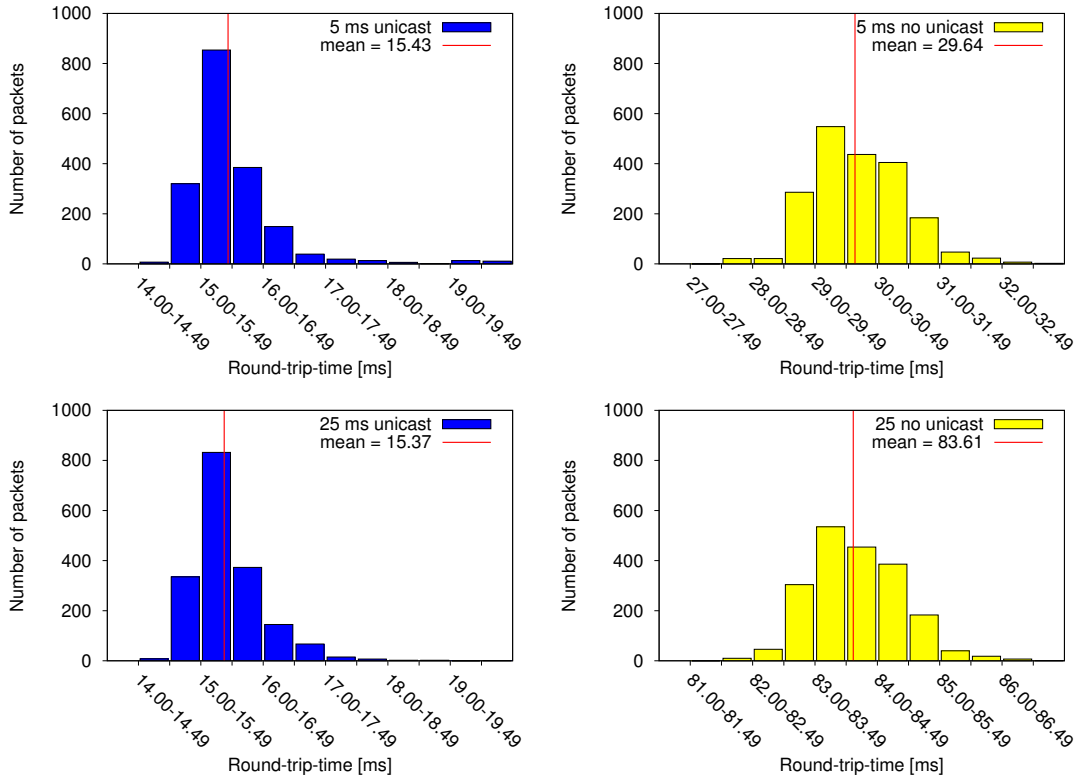
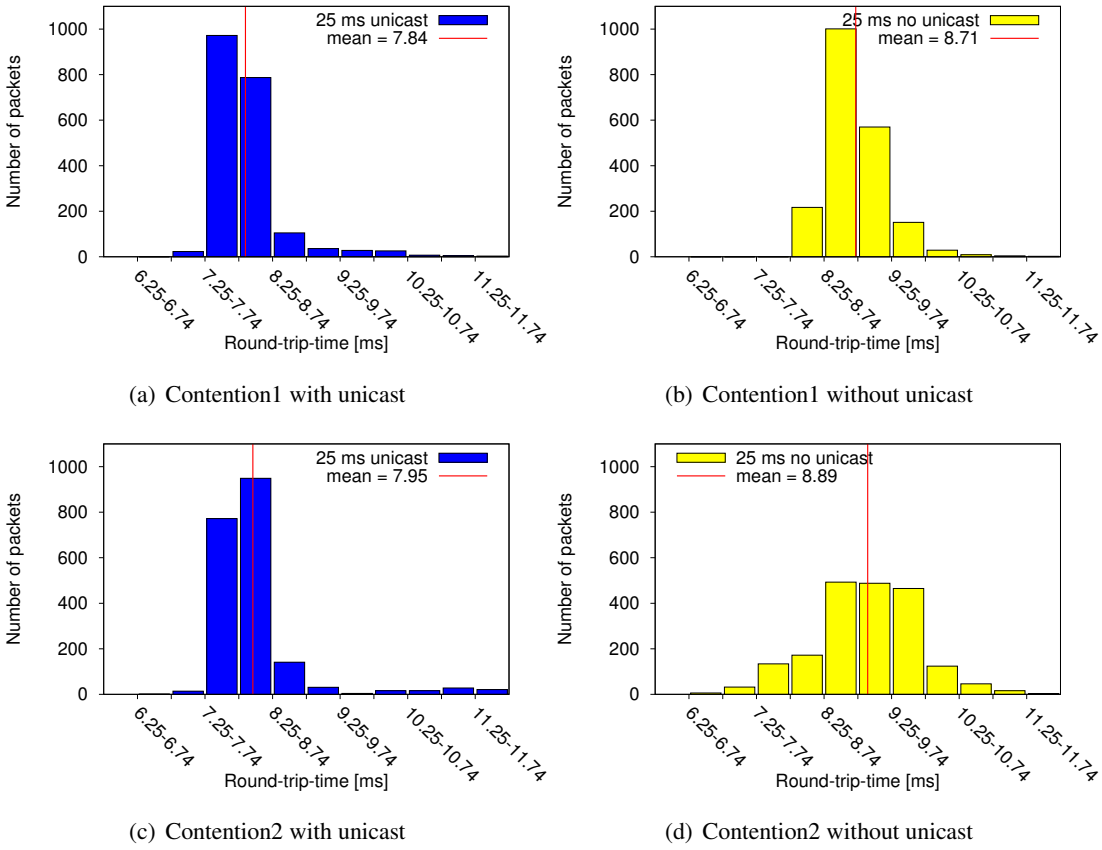


Figure 5.16: Pairs topology, unicast optimisation compared to greedy mode, with different values values for `MAX_DELAY`

have only one hop between source node and destination node. Figure 5.17 shows the results of the measurements in the contention topologies. In addition to these small performance improvements, there were no notable differences between greedy mode and unicast optimisation. There is also not much difference between the Contention1 and Contention2 topologies. Again the Uninvolved topology behaved virtually the same as its counterpart, Contention1.

One thing specific to all unicast measurements has been detected. About 2% of duplicated packets were sent. These were data packets and the additional ACKs that these duplicated data packets caused. All those packets were sent by node number two, the first hop after the source node in the Chain topology. This node number two did not receive 2% of all final ACKs sent by node number one. Node number two thus dropped to broadcast mode and resent the packets it did not get ACKs for. Since the source node already received the first packet sent by node number two, it sent an additional ACK to stop node number two from retransmitting the packets. Switching the position of node number two showed that the problem was specific for this node and did not have to do with the algorithm or the implementation. The hardware of node number two seems to be defective. In the measurements, this defect had no effect on the round-trip-times, because the source received and recorded the packets in question already the first time it received them, and detected the retransmissions as duplicates. This shows the importance of a

well-implemented detection of duplicates.



(a) Contention1 with unicast

(b) Contention1 without unicast

(c) Contention2 with unicast

(d) Contention2 without unicast

Figure 5.17: Contention topologies, unicast optimisation compared to greedy mode with `MAX_DELAY` set to 25 ms in all measurements

5.2.7 Influence of the Invalidation Interval

Most measurements with unicast optimisation have been made with different invalidation intervals (i.e. the interval after which the algorithm goes back to greedy mode to discover a change in topology). The values tries were one second, five seconds and ten seconds. However, no notable difference between measurements with different intervals could be detected. Since the optimal invalidation interval is strongly dependent on the mobility model and node density, this aspect has not been further evaluated.

5.2.8 Backup Mode

Backup mode is dependent on a few additional parameters. These had been evaluated in a few test runs, before the actual measurements were conducted. A value for `MAX_DELAY` of 25 ms does not work, because the location information does not arrive fast enough. The problem are

not the location requests, as those are not being transmitted via BLR, but flooded, as described in section 4.7.2 . The forwarding of packets during flooding is independent from MAX_DELAY, flooded packets are forwarded immediately. The location replies however are sent via BLR, and are thus subject to the delays induced by backup mode (which in turn depend on MAX_DELAY). The value of MAX_DELAY has therefore been set to 5 ms for all measurements with the Backup topology.

The other two parameters that influence the behaviour of the backup mode are TIMEOUT (the time after which a packet is resent) and DISCOVERY_TIMEOUT (the time to wait for discovery replies). Both timeouts have been set to fixed values. On the one hand, these timeouts may not be chosen too high, or the algorithm will not work. On the other hand, performance is increased if these timeouts are chosen as small as possible. For TIMEOUT, $3.5 \cdot$ MAX_DELAY seems to work well. While the theoretical minimum for this value is MAX_DELAY, this is to low in practise, since processing time at the receiving node must be taken into account. For DISCOVERY_TIMEOUT, a value of 7.5 milliseconds was chosen. Higher values resulted in too much delay, and lower values resulted in missed discovery replies.

In figure 5.6, the topology used for backup mode was shown. Packets will be sent in the following manner: After the source node has sent the initial location request, it sends the first packet in greedy mode. Since the forwarding area of the source node is empty, backup mode will be entered after the TIMEOUT. After the completion of the neighbour discovery, the packet will then be sent to the leftmost node (in the figure), obeying the right-hand rule. The leftmost node will then do neighbour discovery again and continue with the right-hand rule, since no node in range can introduce forward progress. It sends the packet to the next node (the node in the upper left corner), where neighbour discovery takes place again. Since now the next node introduces forward progress, the right-hand-rule does not apply anymore and the packet is sent directly to the node with forward progress (the upper right node). This node goes back to greedy mode and the destination receives the packet by broadcast. On the way back, only two nodes are forwarding, both of which can introduce forward progress. However, both of these forwarding nodes will first try to send the packet in greedy mode, before resolving to backup mode, because no nodes are located in their forwarding area. That means that on the way between destination and source, packets suffer from a TIMEOUT twice.

This description shows that it is a bit difficult to make statements about the performance penalties that incur when the algorithm switches to backup mode. The increase in round-trip-time that backup mode causes is highly dependent on the topology layout. But as figures 5.18 and 5.19 illustrate, this increase also depends on the rate at which packets are sent.

In the first measurement, shown by figure 5.18, data was sent at the usual rate of 10 packets per second. Unlike with all other measurements, the round-trip times are split into two distinct groups. The cause for this is the following. When packets are sent at a high rate, it may happen that an outgoing packet is received by the BLR application at the time when the discovery of the neighbour is still in progress. In such a case, the packet is stored in the backup queue until the next neighbour has been determined. Only then all packets in the backup queue are sent. Therefore, packets that go directly into the backup queue have a lower round-trip-time than those packets that are the initiators of the neighbour discovery procedure.

Figure 5.19 shows the results for the same setup, but with a packet sending rate of only one

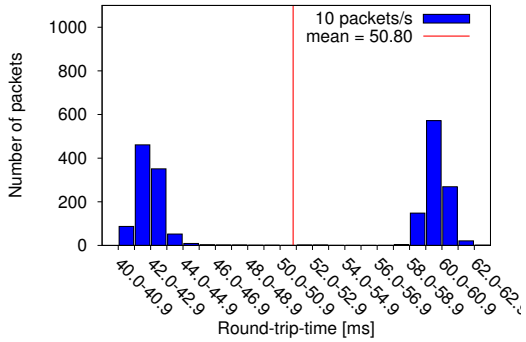


Figure 5.18: Backup topology, packets are sent at a rate of 10 packets per second and a `MAX_DELAY` of 5 ms

packet per second. There is now only one distinct group, because all packets suffer from the delay that neighbour discovery causes.

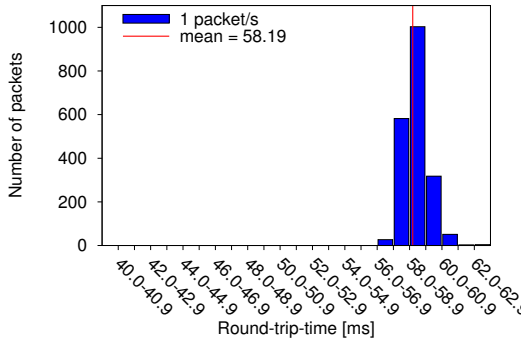


Figure 5.19: Backup topology, packets are sent at a rate of 1 packet per second and a `MAX_DELAY` of 5 ms

At a sending rate of 1 packet per second, the backup mode is about three times slower than the unicast optimisation for the Chain and Pairs topologies, and about 1.5 times slower than the Pairs topology in greedy mode. However, these values have a rather low significance for reasons previously discussed. Other than that, the backup mode behaved similar to the other modes. The amount of duplicates was the same for the Backup topology and packet loss was equally low than with other setups (see section 5.2.10 for more details).

5.2.9 Simulated Mobility

To test the behaviour of the implementation under mobility, two measurements with simulated mobility have been conducted. In these tests, the movement of one node was simulated by changing its coordinates at a specific moment after the start of the experiment. This resulting movement is discrete, that is the node is jumping to its next position rather than moving smoothly. Only nodes already at the border of the forwarding area are moved in this manner,

with the result that a small change in position already results in a topology change, which causes different routing decisions to be made by the nodes. Where necessary, the MAC filters have to be adjusted along with the node positions to reflect the change in topology.

The first measurement illustrates the switch from greedy mode to backup mode and back to greedy mode. The node positions are shown in figure 5.20.

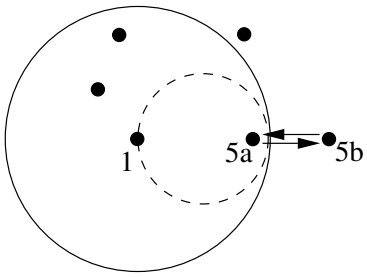
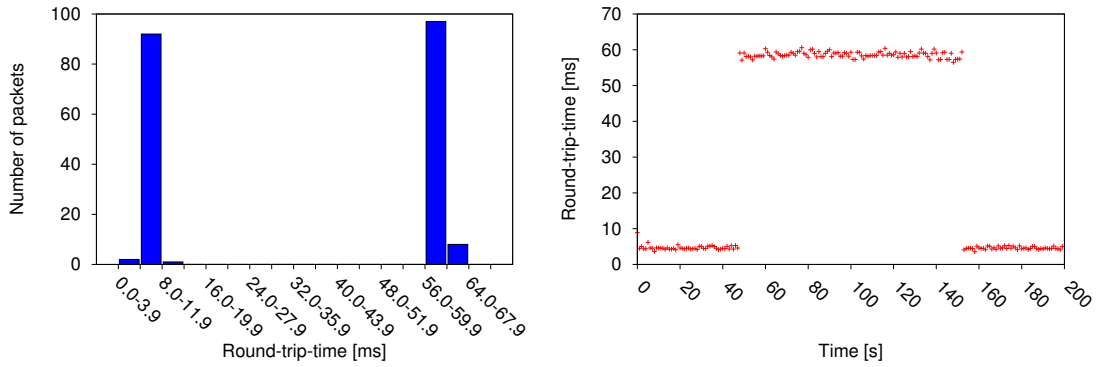


Figure 5.20: Mobility with the Backup topology

The topology starts out with a situation similar to the Backup topology. However, node number 5 will start at position 5a, which lies inside the forwarding area of node number 1. In this situation, packets can be sent directly from node number 1 to node number 5 in greedy mode. After the experiment has been started, the nodes hold their positions for about 50 seconds, then node number 5 moves out of the forwarding area of node number 1 to position 5b. In that situation, node number 5 can only be reached via backup mode. This topology is the same as the Backup topology already used in previous measurements. After another 100 seconds, node number 5 moves back to its initial position 5a, so that greedy mode is possible again.

The results do not show any unexpected behaviour, as seen in figure 5.21. The histogram shows that about half of the packets are sent in greedy mode, and half are sent in backup mode.



(a) Histogram

(b) Distribution over time

Figure 5.21: Backup topology with mobility, sent at a rate of 1 packet per second.

In the distribution of the round-trip-times over time, the two position changes of node number five, and thus the change from greedy to backup mode and vice versa, can be seen very well. As

packets are always tried to be sent in greedy mode,

The topology used in the second measurement is also based on the Backup topology, as shown in 5.22. At the start of the measurement, the topology is equal to the Backup topology

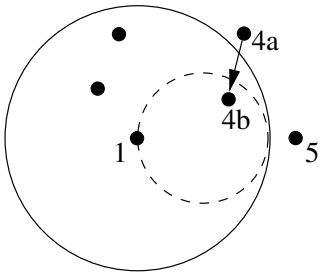
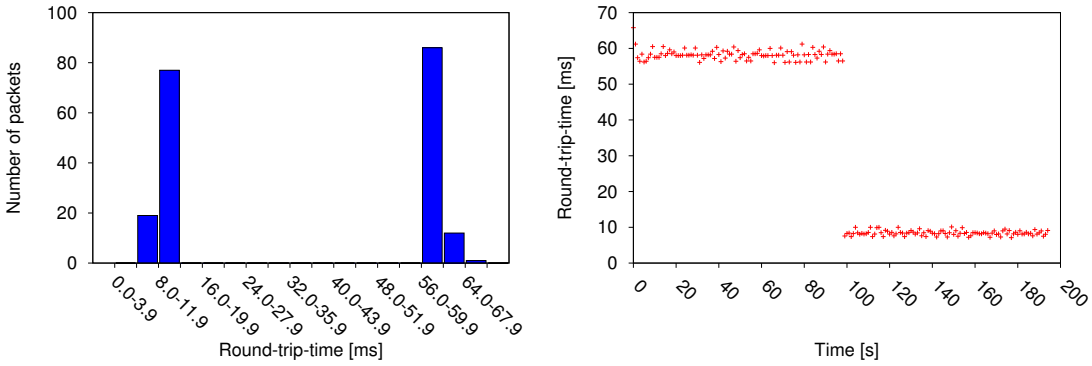


Figure 5.22: Mobility with the backup topology.

defined in section 5.2.1. Node number 4 is located at the position 4a. About 100 seconds after the experiment has started, node number 4 jumps into the forwarding area of node number 1, to position 4b. It stays at this position for the rest of the experiment. It can be seen that the round-trip-time during backup mode is about the same as in the previous measurement. The round-trip-time during greedy mode is higher in the second experiment, since the packets are forwarded via node number 4, and not sent directly from node number 1 to node number 5, as in the previous experiment.



(a) Histogram

(b) Distribution over time

Figure 5.23: Backup topology with mobility, sent at a rate of 1 packet per second.

5.2.10 Reliability

During all measurements with the Chain, Pairs and Contention topologies, delivery rate was 100%. During backup mode measurements, a packet loss of 0.5% has been experienced. About 1% of packets got duplicated during transmission, however they were always detected and destroyed at the next node by the BLR implementation so that the duplicates never reached the

application level. Sometimes, the algorithm switched to backup mode because of a packet timeout. This happened between once and twice per 2000 packets. A possible explanation for this is the increased chance of collisions as explained in 4.8.1. In the measurements, this switch to backup mode caused some outliers, which have been filtered from the data presented in the graphs.

5.3 Outdoor Tests

To get insight into how the BLR algorithm and implementation operates under realistic conditions, a range of outdoor measurements has been conducted. The idea behind these outdoor tests was to set up the nodes in the same topologies and with the same parameters that were used in the indoor tests, to see if the implementation works correctly and to compare the measurement results of indoor and outdoor tests. The outdoor measurements differ from the indoor measurements as follows: Firstly, the positions are gathered from GPS devices instead of being computed before the experiment. And secondly, the MAC filters were removed and the nodes set up with intermediate distances between them great enough to really separate them. The parameter `TRANSMISSION_RANGE` was adjusted from the 250 meters used in the lab to the real transmission range of the wireless transmitters.

5.3.1 Difficulties Encountered

The outdoor measurements proved to be rather difficult to conduct, for a variety of reasons, which will be explained in the following.

The GPS devices do only work reliably with a clear view of the sky. If buildings or trees narrow this view, accuracy drops. During the measurements, the accuracies received in an open field was about 8 meters, with buildings or trees blocking part of the sky view, this dropped to between 20 and 30 meters. Since signal reception depends on the number of satellites in view, this means that reception is not only influenced by obstacles such as trees or buildings, but also by the current time and the position of the satellites. While the DGPS[40] mechanism allows to compensate these inaccuracies, using DGPS was not possible in the scope of this work.

During test setup, another problem was encountered. The assumption of bidirectional links made in the specification of the BLR algorithm did not hold very well in the outdoors. Often, even in the open space, of two laptops, only one node was receiving signals transmitted by the other node. Since different models and brands of laptops and network cards were used during the course of this work, it is difficult to say whether the reason for these asymmetric links could have been avoided if homogeneous material would have been used. As stated in [18], wireless signal reception is not only highly dependent on the wireless card and antenna used, but also on many external factors such as weather, obstacles and signal-reflecting surfaces. This high dependency on external factors has the result that even a small shift in position or a rotation of the wireless transmitter can result in a drastic change of signal reception.

The maximum transmission range of the wireless cards used was measured at 330 meters, with one specific pair of laptops and cards. With a Chain topology containing five nodes, this would result in a distance of 1.32 kilometres between source and destination. As it is difficult to

find an area of that size, that has a clear view of the sky and allows for people conducting experiments, a compromise had to be made. Topologies were only approximated, taking the available terrain into account. The Chain topology for instance was set up along a longer, curved promenade. The curves, trees and other obstacles reduced the signal reception, which in turn reduced the distance between source node and destination node in a natural way without changing the underlying topology.

Furthermore, all measurements had to be conducted on days without rain, for obvious reasons, and during evening hours. One attempt to conduct the measurements in the afternoon had to be cancelled, because it was impossible to read the laptop screens in broad daylight.

5.3.2 Test Setup

To help laying out the nodes according to a topology, a utility was implemented that periodically announces the current position of a node, and displays the distance to other nodes in range. It can read the current position either from a GPS device or from a file containing coordinates. With the help of this tool, the individual positions of the nodes in a topology have been determined. To remove the effect of GPS inaccuracies, instead of live GPS data, these previously determined, fixed coordinates have been used during the measurements. It was noticed that organic material such as grass, ground, trees and human bodies strongly block wireless signals. This had the effect that a laptop standing on the ground had a highly reduced transmission range, since the signal only propagates in a hemisphere around the antenna. And since in the open space the signal is not reflected by anything, this has the effect that if the surface between two laptops standing on the ground is not perfectly flat or contains an obstacle, the two laptops can overhear each other only at very short distances. Figure 5.24 illustrates this.

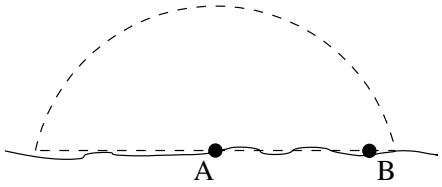


Figure 5.24: Two nodes A and B. While node B would be inside the transmission range of node A, signal reception is blocked by the ground line.

By putting all laptops onto chairs during the experiment it was possible to compensate the signal blocking effect of bumps in the ground. After the nodes have been positioned, their position was saved and the GPS device removed. To have the nodes operate with a fixed distance during the measurements helped to reduce the impact of GPS inaccuracies.

To minimise the impact of the difference in transmission strength of the various nodes, it was necessary to set the parameter `TRANSMISSION_RANGE` for each measurement separately. As it will be explained with the help of figure 5.25, the parameter `TRANSMISSION_RANGE` must be set equal to the lowest actual transmission range of all nodes. For example, if it would be set to r, packets sent from A to C would be received by C directly, which works. But sending packets from C to A does not work: When C sends a packet, it is received by both A and B. Since A will forward the packet faster than B, B will cancel its pending transmission of the packet, as it

overhears that A has already forwarded the packet. But C does not overhear the forwarding by A and will resend the packet. If TRANSMISSION_RANGE is set to s instead, the algorithm works: A will receive the packet sent by C, but will drop it since it is located outside of the forwarding area. B forwards the packet, A receives it again and can now forward it further, while C is seeing this relaying by B as passive acknowledgement.

By carefully laying out the node topology and setting the parameter TRANSMISSION_RANGE correctly, asymmetrical links did not cause problems. Laying out a single topology took between twenty minutes and an hour.

The tests for the Chain and Pairs topology were carried out alongside a promenade running through an accommodation located in Bolligen, a suburb of Bern. The promenade was curved and flanked by trees and buildings on both sides. The tests for the Contention topology were conducted on a grass field in Ostermundigen, Bern, called "Kleine Allmend".

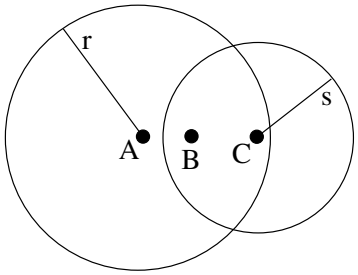


Figure 5.25: Node A with a large transmission range of r, and node C with a smaller transmission range of s.

5.3.3 Measurements Based on the Chain Topology

The Chain topology was set up as illustrated by the map shown in figure 5.26. Buildings and trees blocked the clear line of sight between the nodes, allowing to reduce the mean distance between two nodes to below 100 meters. The parameter TRANSMISSION_RANGE was set to 65 meters. Setting up the nodes with an intermediate distance close to the transmission range did not work out, the actual transmission ranges of the devices varied too greatly, partly due to the different hardware, but also due to the terrain and due to obstacles.

The results of the measurements can be seen in figure 5.27. The average round-trip-times achieved in the outdoor experiments were only marginally higher than those achieved in indoor measurements. In greedy mode, the average RTT outdoors was 17.76 milliseconds, whereas indoors, it was 17.48 milliseconds. See figure 5.15 for reference. With the unicast optimisation enabled, the average RTT was 15.75 milliseconds outdoors compared to 15.43 milliseconds indoors. A packet loss of about 1% was experienced in the outdoor measurement, while comparable lab experiments showed a 100% delivery rate. The performance benefit that unicast mode offers over greedy mode does not change when going outdoors.

Figure 5.26: Node placement for the Chain measurements

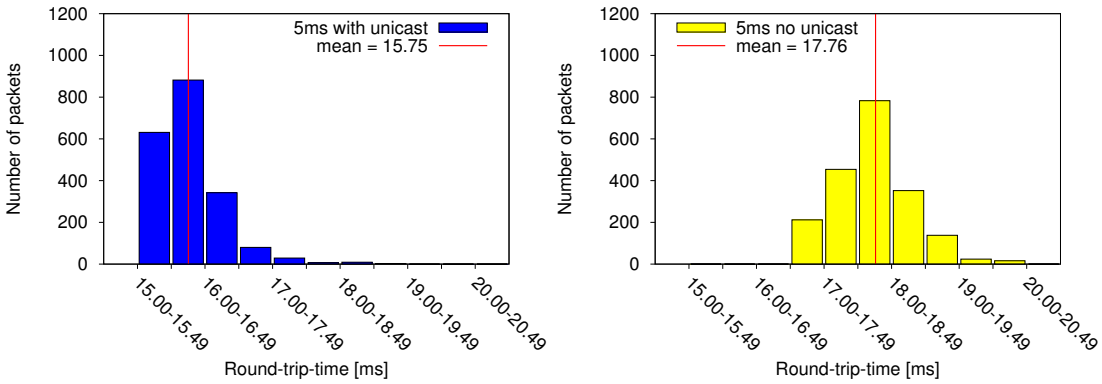


Figure 5.27: Chain topology, measured outdoors.

5.3.4 Measurements Based on the Pairs Topology

The Pairs topology was set up along the same path as the Chain topology, using the Chain topology as starting point. Nodes two to five were all moved approximately the same distance

closer to node number one. Then node number four and five were moved the same distance again closer to node number three. The node positions were then slightly corrected, to better match the actual Pairs topology. Both nodes two and four are located significantly closer to their predecessor than the other nodes, yet again, a perfect setup was not possible. Figure 5.28 shows the node positions indicated on a map.

Figure 5.28: Node placement for the Pairs measurements

The results of the measurements are shown in figure 5.29. The performance during measurements with the unicast optimisation turned on showed similarities to the measurement with the Chain topology. The parameter `MAX_DELAY` has virtually no influence during unicast mode. With a `MAX_DELAY` of 5 ms, an average round-trip time of 15.78 ms was measured outdoors, and 15.73 ms was measured with a `MAX_DELAY` of 25 ms. In comparison, the indoor measurements yielded 15.43 ms and 15.37 ms respectively, see figure 5.16 for a reference. Thus, the small decrease in performance for outdoor experiments that has been seen in the Chain topology can also be seen in the Pairs topology.

The outdoor measurements with greedy mode show a significantly better performance than those achieved the lab: Indoors, 29.64 ms was measured for a `MAX_DELAY` of 5 ms, and 83.61 ms for a `MAX_DELAY` of 25 ms. In the outdoor experiments, 24.61 ms for 5 ms and 70.06 ms for 25 ms was measured. A reason for these unexpected results could not be found, which empha-

sises again that it is not always meaningful to directly compare lab experiments with outdoor experiments, as too many external factors such as weather, obstacles and signal-reflecting surfaces can influence the outcome of outdoor experiments in an unpredictable manner.

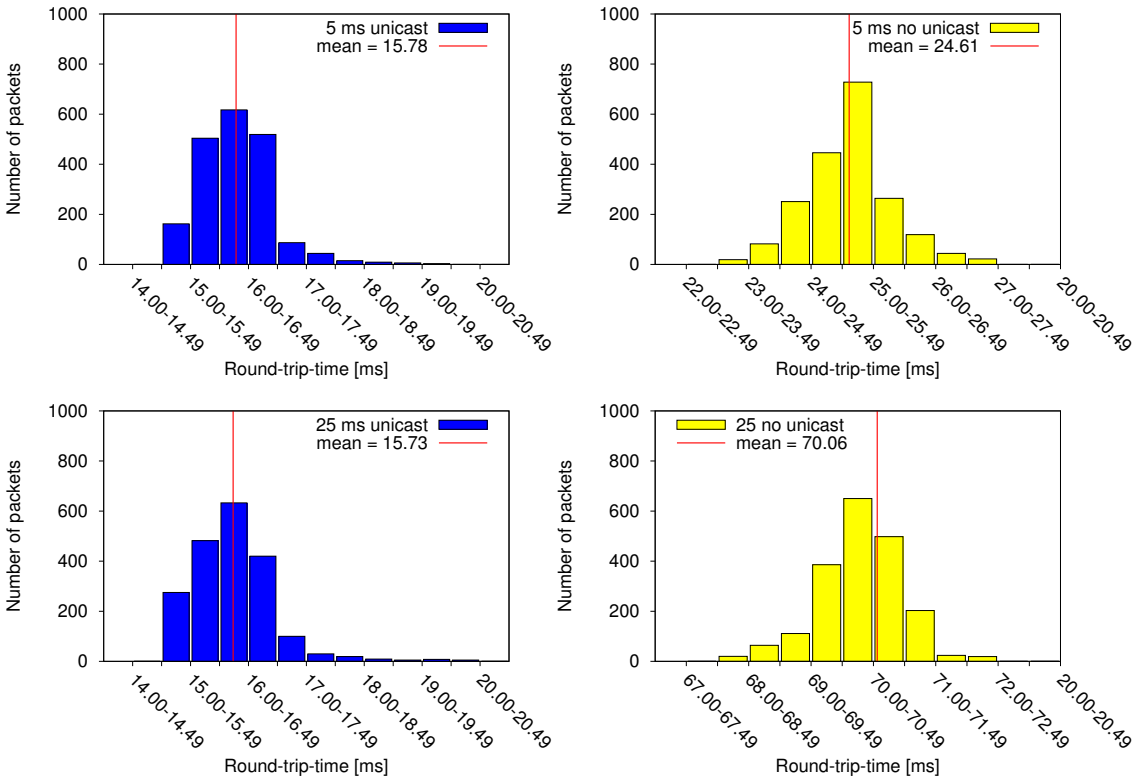


Figure 5.29: Pairs topology measured outdoors, unicast optimisation compared to greedy mode, with different values values for `MAX_DELAY`

5.3.5 Measurements Based on the Contention Topology

This measurement was conducted at a different place than the last two experiments. The setup was a bit simpler, since only three nodes had to be positioned at a maximal distance, and the remaining two nodes could be positioned almost arbitrarily somewhere inside the forwarding area of the source. The performance does not differ much from the lab measurement, the outdoor measurement shows only a marginally slower round-trip-time. The results can be seen in figure 5.31.

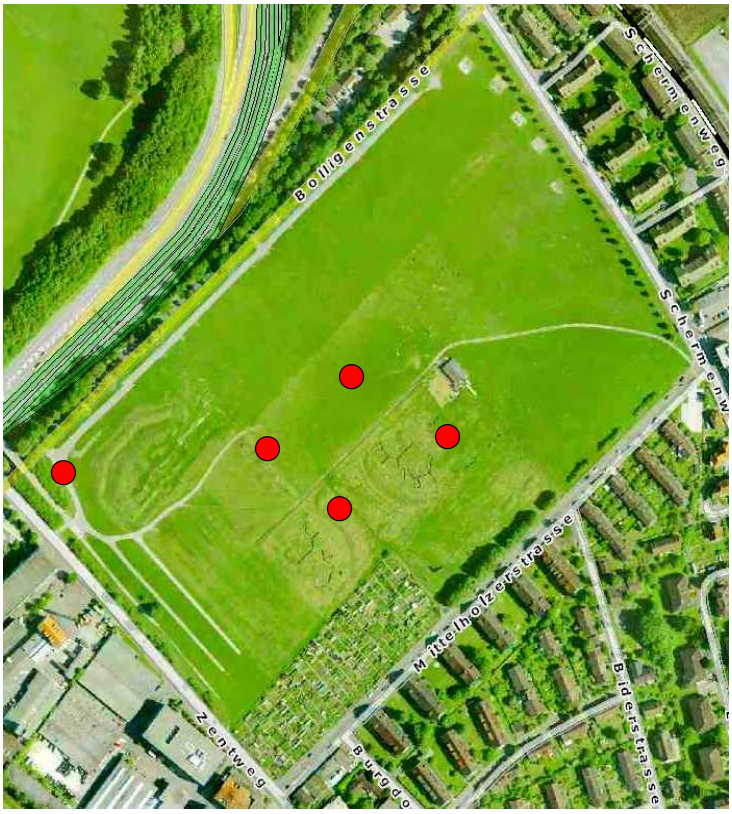


Figure 5.30: Node placement for the Contention measurements

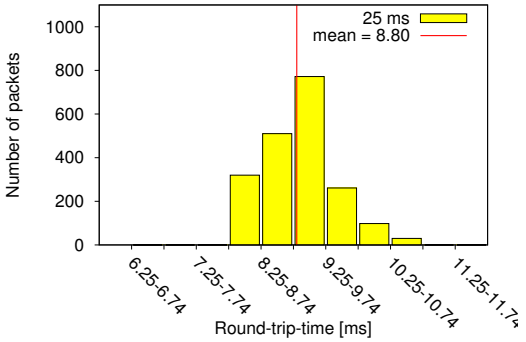


Figure 5.31: Outdoor measurement with the Contention1 topology

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The measurement results have shown that developing a real-world implementation of the Beaconless Routing Algorithm is possible and that the resulting application is performing reasonably well in the lab. The relevance of the results of the outdoor tests should not be overestimated, however. Carrying out real-world outdoor experiments is very complicated and due to many external factors, reproducibility of the results is difficult to achieve.

Both in indoor and outdoor measurements, the BLR implementation performs correctly, reliably and packet delivery is fast. The outdoor measurements were rather difficult to conduct, and showed a performance similar to that of the indoor measurements, with one exception, where the outdoor setup outperformed a similar indoor setup. A definitive reason for this phenomenon could not be found.

Depending on the field of application, many optimisations to the algorithm and the implementation are possible. Some are necessary to achieve optimal performance. BLR, like every other mobile ad-hoc routing algorithm, is not suited for every scenario equally well.

The main weakness of the algorithm are the strong assumptions made about the radio model. The outdoor experiments showed that it is unrealistic to assume a transmission range which is the same in every environment and for every node. Choosing a small value for the transmission range parameter prior to conducting an experiment worked when using only five nodes. Experiments with more nodes are needed to evaluate whether using a short transmission range works equally well in larger networks. The unit disk graph model also proved too strong an assumption to work well in real-world outdoor conditions. However, the Cross-Link Detection Protocol (CLDP)[29] could remedy this problem at least for backup mode. CLDP, given an arbitrary connected graph, produces a sub-graph on which face traversal cannot cause routing failures, regardless of radio irregularities and localisation errors.

6.2 Proposed Optimisations

During implementation and testing, various ideas have been developed about improving the algorithm and implementation, and about future measurements which could be of interest. Parts

of these ideas stem from literature, and parts are based on personal experiences gathered during the course of this work.

6.2.1 Backup Mode TTL

During backup mode, collisions are especially disruptive, since backup mode relies on exchange of control packets. Instead of simple packet loss, which can be handled by the upper network layers, collisions during backup mode setup can cause existing paths to fail or to be altered, as it has been described in section 4.8.1. To reduce the negative effect this can have, a sort of Backup Mode Time-To-Live field could be added to the packet header. If a packet sent during backup mode arrives back at the point where backup mode initially failed, it may be resent in backup mode again, if the TTL allows it. The topology may have changed since initial sending for two reasons: mobility of the nodes may have caused a real change in topology, or collisions of data packets may have happened during the first tour of the packet and may have caused the topology to look different than it actually is. Before reporting failure to the upper layers, backup mode could be tried again in that case. Future experiments will have to show whether collisions or mobility-induced changes in topology happen often enough to present a major problem and to warrant the introduction of a backup mode TTL.

6.2.2 Data Caching

At various points in the algorithm, it is possible to cache information to improve performance. For example, position data of next hops during unicast and of the destination node during regular transmissions is already cached for a certain amount of time. During backup mode, it would be possible to cache neighbour information as well. Since the current implementation operates in promiscuous mode, it would be possible for nodes that overhear a unicast transmission in which they are not directly involved, to cache next hop information if suitable for their current position. Another, more complex idea that involves caching is that nodes could extract positions from all packets they overhear. A sort of quality tag could be added to cached position information, which is based on speed and direction of nodes. Position data of slow-moving nodes or nodes moving towards the caching node would be tagged with higher quality, since it is less likely that this information gets outdated quickly.

Of course, caching is always a trade-off between complexity of the implementation and performance. Outdated information can quickly lead to degraded performance, or the processing overhead of acquiring information could be too large. Extensive evaluations have to show if and to what amount caching of data is of benefit. After all, the simplicity of the stateless manner of greedy mode is an important advantage of BLR over other position-based routing algorithms.

6.2.3 Adaptive Retransmission Timeouts

Timeouts should generally be as small as possible to reduce delays during mode transitions. In the current implementation, timeout values are constants, which have been chosen based on experiments. An adaptive algorithm could be developed, so that timeout values start with a

small value, and are gradually increased whenever timeouts occur. In [18], adaptive retransmission timers are deemed a necessity when the retransmission algorithm is situated above the link-layer. In BLR, this is inevitably the case for transmissions in greedy mode, since no retransmission scheme is available in the 802.11b MAC-layer for broadcast packets. For unicast and backup modes, retransmissions can theoretically be handled by the MAC layer, however this is not possible in the current implementation for reasons already stated. Therefore, adaptive retransmission timers might make sense if implemented in the context of the current implementation. Again, the benefit of such an enhancement would need to be evaluated with experiments.

6.2.4 Preemptive Unicast mode

Since it is possible to extract signal strength values from passive acknowledgements, the next hop during a unicast transmission could be chosen not only depending on its position, but also on signal strength. Nodes with low signal strength or nodes travelling towards the border of the transmission range could be ignored when making the decision for a next hop during unicast mode. Also, unicast mode could be exited prematurely if the hop used shows signs of decreasing signal strength.

6.3 Future Tasks

BLR showed to be a very promising routing protocol for mobile ad-hoc networks. Yet, some things need to be improved to reach a production-quality application.

6.3.1 MAC-layer Integration

For an ideal operation of BLR, the implementation would have to be moved into the MAC layer, since timeout and retransmission handling is an integral part of the BLR algorithm: Retransmission is handled by switching from unicast mode to greedy mode or from greedy mode to backup mode, and the switching is triggered by packet timeouts. The current implementation is suboptimal in that respect, since it has to do timeout management and retransmission initiation independent from the MAC-layer (as explained in section 4.7.3). Implementing BLR in the MAC layer and giving this implementation complete control over timeout and retransmission handling would thus be beneficial. Another advantage of this approach is that promiscuous mode could be avoided, since BLR can now send acknowledgements during unicast mode in the same way that the 801.11b MAC-layer does. The increased control over the medium that an implementation in the MAC-layer gives could be leveraged further, for example by use of power control and directional antennas, as described in [41] and [42].

6.3.2 Kernel Implementation of Critical Parts

As described in section 4.8.7, the current implementation is suboptimal when used in scenarios with high node density. In is thus imperative that the timing behaviour of the implementation is improved. This could be achieved by moving the time-critical parts of the program from user

space to the kernel. The use of a kernel with real-time properties could also be examined, as this would possibly benefit the overall behaviour of the algorithm.

6.3.3 Power Measurements

Examination of power consumption has been left out completely in this thesis. For environments where low power operation is required, many optimisations of the current algorithm are possible. The factors power consumption, performance and code complexity have to be evaluated against each other. In static sensor networks for example, backup mode could be eliminated, if the sensors are positioned intelligently. Also, removing the unicast optimisation would reduce complexity and have a positive effect on power consumption, since promiscuous mode could be avoided. On the other hand, with adaptive sending power and a delay function that favours closely located nodes over distant ones, sending in unicast could mean an improvement in power consumption. Thus, measurements of the power consumption behaviour in various situations should be performed, to gain an understanding on where in the algorithm the most power is used.

6.3.4 Extensive Measurements and Simulations

To gain more insight into how the protocol behaves under real-world conditions, additional experiments with more nodes are necessary. As reproducibility of the experiments is desired, the approach of direct execution, as described in [10], might be worth to be considered for future experiments. This has the additional benefit that it makes comparing real-world measurements with results obtained through simulation easier.

Bibliography

[1] M. Heissenbüttel and T. Braun. BLR: A Beacon-Less Routing Algorithm for Mobile Ad Hoc Networks. Tech. Rep. IAM-03-001, Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, March 2003.

[2] 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(11):1076–1086, 2003. Submitted.

[3] T. Bernoulli. Beacon-Less Routing in Mobile Ad Hoc Networks. Institute of Computer Science and Applied Mathematics, University of Bern. Master's thesis.

[4] S. Giordano, I. Stojmenovic, and L. Blazevic. Position based routing algorithms for Ad Hoc networks: A taxonomy. In *Ad Hoc Wireless Networking*. Kluwer, 2003. To appear.

[5] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MOBICOM'00*, pp. 243–254. Boston, New York, August 2000.

[6] 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, January 2002.

[7] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proceedings of the 22nd ACM Symposium on the Principles of Distributed Computing (PODC)*, pp. 63–72. Boston, New York, July 2003.

[8] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Tech. Rep. CSD-TR 02-0013, University of California, Los Angeles, February 2002.

[9] S. Sanghani, T. X. Brown, S. Bhandare, and S. Doshi. EWANT: The Emulated Wireless Ad Hoc Network Testbed. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*. Boulder, Colorado, March 2003.

[10] J. Liu, Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. Kotz, and L. F. Perrone. Simulation Validation using Direct Execution of Wireless Ad-Hoc Routing Protocols. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS)*. Kufstein, Austria, May 2004.

[11] M. Heissenbüettel, T. Braun, T. Roth, and T. Bernoulli. GNU/Linux Implementation of a Position-based Routing Protocol. In *Proceedings of the IEEE ICPS Workshop on Multi-Hop Ad Hoc Networks: from theory to reality*, pp. 27–35. Santorini, Greece, July 2005.

[12] Y.-J. Kim and R. Govindan. A GPSR implementation. [Online]. Available: *http://www.cs.cmu.edu/ bkarp/gpsr/gpsr.html*, accessed March 21, 2005.

[13] S. Desilva and S. Das. Experimental Evaluation of a Wireless Ad Hoc Network. In *Proceedings of the 9th Int. Conf. on Computer Communications and Networks (IC3N)*. Las Vegas, Nevada, October 2000.

[14] C. E. Perkins and E. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of MobiCom '99*, pp. 90–100. New Orleans, Lousiana, February 1999.

[15] S. H. Bae, S.-J. Lee, and M. Gerla. Multicast Protocol Implementation and Validation in an Ad hoc Network Testbed. In *Proceedings of IEEE ICC 2001, Helsinki, Finland*, pp. 3196–3200. June 2001.

[16] S. H. Bae, S.-J. Lee, W. Su, and M. Gerla. The Design, Implementation, and Performance Evaluation of the On-Demand Multicast Routing Protocol in Multihop Wireless Networks. *IEEE Network, special issue on Multicasting Empowering the Next Generation Internet*, vol. 14(1):70–77, 2000.

[17] W. Kiess, S. Zalewski, A. Tarp, and M. Mauve. Thoughts on Mobile Ad Hoc Network Testbeds. In *Proceedings of the 1st Workshop on Wireless Network Measurements (WiNMee 2005)*. Trentino, Italy, April 2005.

[18] D. A. Maltz, J. Broch, and D. B. Johnson. Experiences designing and building a multihop wireless ad hoc network testbed. Tech. Rep. CMU-CS-99-116, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1999.

[19] S. Jadhav, T. X. Brown, S. Doshi, D. Henkel, and R. G. Thekkekunnel. Lessons learned constructing a Wireless Ad Hoc Network Test Bed. In *Proceedings of the 1st Workshop on Wireless Network Measurements (WiNMee 2005)*. Trentino, Italy, April 2005.

[20] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, chap. 5, pp. 139–172. Addison-Wesley, 2001.

[21] G. Wépiwé. Design and Implementation of a Routing Mechanism for a Wireless Ad Hoc Network. Department of Electrical and Computer Engineering, Institute for Open Communication Systems, Technical University of Berlin. Master's thesis.

[22] M. Heissenbüttel. *Networking in Ad-hoc Networks*. Ph.D. thesis, University of Bern, CH-3012 Bern, Switzerland, Jun 2005.

[23] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-Free Positioning in Mobile ad-hoc Networks. *Cluster Computing Journal*, vol. 5(2):118–124, 2002.

[24] T. Champ. Location Information Services in Mobile Adhoc Networks. Tech. Rep. MCS-03-15, Colorado School of Mines, Golden, Colorado, October 2003.

[25] Gentoo Foundation, Inc. Gentoo Linux. [Online]. Available: *http://www.gentoo.org*, accessed March 21, 2005.

[26] Free Software Foundation. The GNU C Compiler. [Online]. Available: *http://gcc.gnu.org*, accessed March 21, 2005.

[27] Free Software Foundation. The GNU Project Debugger. [Online]. Available: *http://www.gnu.org/software/gdb/gdb.html*, accessed March 21, 2005.

[28] CollabNet, Inc. The Subversion Revision Control System. [Online]. Available: *http://subversion.tigris.org*, accessed March 21, 2005.

[29] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation, NSDI*. Boston, New York, May 2005.

[30] M. Krasnyansky. Virtual Point-to-Point devices. [Online]. Available: *http://vtun.sourceforge.net/tun*, accessed March 21, 2005.

[31] A. Kleen and M. Wilcox. The Packet(7) manpage. [Online]. Available: *http://unixhelp.ed.ac.uk/CGI/man-cgi?packet+7*, accessed March 21, 2005.

[32] A. Gifford, S. Pace, J. McNeff, and V. Zhang. One-Way GPS Time Transfer. In *Proceedings of the 32nd Annual Precise Time and Time Interval Meeting (PTTI)*, pp. 137–146. November 2000.

[33] NMEA. The National Marine Electronics Association. [Online]. Available: *http://www.nmea.org*, accessed March 21, 2005.

[34] M. A. Lombardi, L. M. Nelson, A. N. Novick, and V. S. Zhang. Time and Frequency Measurements using the Global Positioning System. *Cal Lab Int. Jour. of Metrology*, vol. 3:16–33, 2001.

[35] C. L. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Networks. In *SIGCOMM*, pp. 39–49. 1997.

[36] J. Mogul and S. Deering. Path MTU Discovery. IETF RFC 1191.

[37] H. Welte. Netfilter: firewalling, NAT and packet mangling for Linux. [Online]. Available: *http://www.netfilter.org*, accessed March 21, 2005.

[38] V. J. Steven McCanne. The BSD Packet Filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference*. San Diego, California, December 1993.

[39] D. L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF RFC 1305.

[40] L. S. Monteiro, T. Moore, and C. Hill. What is the Accuracy of DGPS. In *Proceedings of the Annual Meeting of the Institute of Navigation*. Cambridge, New York, June 2005.

[41] T. Ueda, S. Tanaka, D. Saha, S. Roy, and S. Bandyopadhyay. A Rotational Sector-based, Receiver-Oriented mechanism for Location Tracking and medium Access Control in Ad Hoc Networks using Directional Antenna. In *Proceedings of the IFIP conference on Personal Wireless Communications PWC*. venice, Italy, September 2003.

[42] D. Saha, S. Roy, S. Bandyopadhyay, T. Ueda, and S. Tanaka. A Power-Efficient MAC Protocol with Two-Level Transmit Power Control in Ad Hoc Network Using Directional Antenna. In *Proceedings of the 5th International Workshop on Distributed Computing (IWDC)*. Calcutta, India, December 2003.