TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DISSERTATION

Cashflow: a virtual currency system for mobile ad hoc networks

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. Harmen R. van As
E388 – Institut für Breitbandkommunikation
Technische Universität Wien

und

Univ.Prof. Mag. Dr.techn. Schahram Dustdar
E184 – Institut für Informationssysteme
Technische Universität Wien

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. Mag. Lukas Wallentin
Matr.-Nr. 0104725

Wien, im November 2010

Abstract

The concept of mobile ad hoc networks allows connecting mobile nodes without the need of additional infrastructure. In such networks, every node acts as router, forwarding packets on behalf of other nodes. Therefore, mobile ad hoc networks possess no single point of failure, which makes them interesting for military usage, where mobile ad hoc networks have been an active research field for more than 30 years. Another traditional area for the usage of mobile ad hoc networks are emergency scenarios, where networks need to be deployed but no infrastructure is available.
Today, since mobile devices like smart phones and laptops are nearly ubiquitous, the concept of mobile ad hoc networks can be used to connect these mobile devices to extend the range of infrastructure and to provide the bases for new applications. However, new issues have to be solved if nodes belong to multiple authorities and not to a single one like in military or emergency scenarios. One of these issues is how to motivate nodes to participate in these networks and forward packets on behalf of other nodes. Generally, nodes belonging to different authorities have no motivation to participate and provide services to other nodes without getting rewards.
Focusing on this issue, we present and evaluate in this thesis Cashflow, a virtual currency system to motivate nodes to participate in mobile ad hoc networks. The basic idea is that nodes pay other nodes for provided services. This concept motivates nodes to participate and prevents free riders, which use services of the network without providing services for other nodes. In contrast to other virtual currency systems, Cashflow is a market-based and channel-oriented virtual currency system. We show in this thesis that the usage of supply and demand as basis for pricing in combination with virtual channels provides a number of benefits compared to other virtual currency systems, which in most cases use auction or fixed price schemes for pricing. These benefits include besides other the inclusion of the nodes context into pricing, implicit load balancing, reduction of payment overhead, the ability of the user to control his participation degree, and the ability to reschedule transmission if the current pricing level is high. Additionally it allows an easy integration of the mobile ad hoc network into the Internet, which is not considered by other virtual currency systems. Further, the market concept allows the development of route discovery algorithms using fee information for routing decisions. Such an algorithm has been developed and integrated into Cashflow, allowing nodes to use the cheapest routes to other nodes. Therefore, Cashflow solves in contrast to other virtual currency systems not

only the problems how to pay and how much to pay, but also how to find the most cost effective route. Since payment functionality in networks could be interesting for a number of business scenarios, Cashflow provides open interfaces to enable the usage of its payment and route search functionality for applications. Therefore, Cashflow can be used as platform to develop business applications in ad hoc networks.

Zusammenfassung

Das Konzept von mobilen ad hoc Netzen erlaubt es ohne Einsatz von Infrastruktur mobile Knoten zu verbinden. In einem solchen Netz agiert jeder Knoten als Router und leitet Pakete für andere Knoten weiter. Daher besitzen mobile ad hoc Netze keinen einzelnen Fehlerpunkt, was sie für Anwendungen im militärischen Bereich interessant macht, wo seit mehr als 30 Jahren aktiv an mobilen ad hoc Netz geforscht wird. Ein weiterer ursprünglicher Einsatzbereich von mobilen ad hoc Netzen sind Katastrophen-Szenarien wo Netze zur Kommunikation benötigt werden aber keine Infrastruktur vorhanden ist.
Heutzutage, da mobile Geräte wie Smartphones und Laptops beinahe allgegenwärtig sind, kann das Konzept der mobilen ad hoc Netze dafür verwendet werden, die vorhandenen mobilen Geräte zu vernetzen, um die Reichweite von der vorhandenen Infrastruktur zu erhöhen und um die Grundlage von neuen Anwendungen zu schaffen. Allerdings müssen neue Problemstellungen gelöst werden, wenn die Knoten verschiedenen Besitzern und nicht einem einzelnen, wie das bei militärischen oder notfalls Szenarien der Fall ist, gehören. Eine dieser neuen Problemstellungen ist die Frage, wie man Knoten dazu motiviert an einem solchen Netz teilzunehmen und Pakete für andere Knoten weiterzuleiten. Grundsätzlich haben Knoten von verschiedenen Besitzern keine Motivation an so einem Netz teilzunehmen und Services anderen Konten zur Verfügung zu stellen ohne dafür eine Entlohnung zu erhalten.
Im Hinblick auf diese Problemstellung präsentieren und evaluieren wir in dieser Dissertation Cashflow, ein virtuelles Währungssystem für mobile ad hoc Netze um Knoten zur Teilnahme an solchen Netzen zu motivieren. Die grundsätzliche Idee dieses Systems ist, dass Knoten andere Knoten für geleistete Services bezahlen. Das motiviert Knoten an dem Netz teilzunehmen und verhindert sogenannte free rider, also Knoten, die Services von anderen Knoten nutzen, ohne selbst Services anzubieten. Im Gegensatz zu anderen virtuellen Währungssystemen für mobile ad hoc Netze basiert Cashflow auf einem Marktsystem und ist kanalorientiert. In dieser Arbeit wird gezeigt, dass die Verwendung von Angebot und Nachfrage als Grundlage der Preisberechnung in Kombination mit einem virtuellen Kanalsystem eine Reihe von Vorteilen gegenüber anderen virtuellen Währungssystemen bringt, die meist Fixpreis oder Auktionsmechanismen zur Preisfestsetzung verwenden. So erlaubt das Preisfestsetzungssystem von Cashflow unter anderem den Kontext des Knotens mit einzubeziehen, verfügt über einen implizierten

Lastverteilungsmechanismus, reduziert den Overhead der durch das Bezahlen erzeugt wird, gibt dem Benutzer die Möglichkeit selbst zu bestimmen zu welchen Grad er an einem Netz teilnimmt und erlaubt es den Knoten, Übertragungen zu verzögern, falls das aktuelle Preisniveau hoch ist. Außerdem erlauben diese Konzepte eine einfache Intergration von mobilen ad hoc Netzen in das Internet, was von anderen virtuellen Währungssystemen nicht unterstützt wird. Weiters ermöglicht das Marktkonzept die Entwicklung von Routingalgorithmen, die den günstigsten Pfad zwischen Knoten finden. Ein solcher Routingalgorithmus wurde entwickelt und in Cashflow integriert. Daher löst Cashflow im Gegensatz zu anderen virtuellen Währungssystemen nicht nur das Problem wie und wie viel für die Services der anderen Knoten zu zahlen ist, sondern auch wie die Konten die kosteneffektivste Route finden. Da die Zahlungsfunktionalität auch für eine Reihe von Geschäftsszenarien interessant sein könnte, bietet Cashflow offene Schnittstellen um die Bezahl- und Routensuchfunktionalität Anwendungen zur Verfügung zu stellen. Daher kann Cashflow auch als Plattform für die Entwicklung von Bezahlservices in ad hoc Netzen verwendet werden.

Acknowledgements

I would like to express my deep gratitude to Professor Harmen R. van As who has given me the opportunity to join the Institute of Broadband Communications and supported me throughout my PhD studies. I also appreciatively acknowledge the second reviewer, Professor Schahram Dustdar, for agreeing to evaluate this thesis.
Further, I want to thank all people directly or indirectly involved in this thesis and to show my appreciation for all the assistance I have received throughout my work. Particularly I want to thank Dr. Joachim Fabini, DI Christoph Egger, DI Mag. Marco Happenhofer, DI Mag. Michael Hirschbichler, and Dr. Markus Sommereder for their support.
Finally, I want to thank my family and friends for their continual support and encouragement.

Vienna, November 2010 Lukas Wallentin

Contents

Chapter 1

Introduction

Mobile ad hoc networks have been an active research field for more than 30 years. Traditionally, mobile ad hoc networks were developed for military usage [Ephremides02] where communication between mobile units in the battlefield is crucial. The mobile ad hoc concept allows deploying networks without the need of additional infrastructure and, which is even more important from the military point of view, it possesses no single point of failure. A typical military scenario for the usage of ad hoc networks would be a tank division, where each tank is equipped with a radio communication system and a router to provide forwarding services to other tanks. In this scenario, the deployed mobile ad hoc network provides a robust communication system without the need of infrastructure, which additionally tolerates the loss of participants.

Besides military usage, emergency situations are also typical usage scenarios for mobile ad hoc networks. In emergencies like earthquakes, communication is of high importance for rescue forces. If existing infrastructure is not available, mobile ad hoc networks allow the fast deployment of alternative communication systems.

A third classical usage area of mobile ad hoc networks is the area of sensor networks. The idea is to equip hundreds of sensors with radio transmitters and drop them on the area, which should be observed. An example would be a chemical accident where an area has been contaminated. To get a detailed analysis about the contamination degree, sensors can be dropped from a low-flying plane on the contaminated area. The sensor nodes build an ad hoc network to forward the measurement results to the emergency

forces. This scenario differs from the previous two, since energy is in sensor networks of much greater importance. In the previous scenarios, the energy consumption of the radio and routing equipment is relatively small, compared to the energy needs of a tank or a car. However, in many cases sensor nodes rely on batteries as energy resource, making energy the limiting factor for node's lifetime. [Perkins08]

Advantages in wireless communication and computer technology have expanded possible applications and usage scenarios for mobile ad hoc networks. For instance, the mobile ad hoc network concept can be used to connect cars to build so called vehicular ad hoc networks. By using vehicular ad hoc networks, cars could exchange automatically information about traffic, local conditions like ice on the road or accidents.

Since mobile devises like laptop computers and smart-phones are already ubiquitous today, also these devises could be used to establish ad hoc networks as platform for new applications. However, these new applications result in a number of new issues, which have to be solved. One of these issues is the problem how to motivate nodes to participate in an ad hoc network, when nodes belong to distinctive authorities. Nodes belonging to distinctive authorities have no motivation to forward packets on behalf of other nodes. However, cooperation is essential for the establishing of mobile ad hoc networks.

Currently there are two concepts to prevent selfish behavior and stimulate cooperation in ad hoc networks: reputation systems and virtual currency systems [Nahrstedt09]. Reputation systems enforce fair behavior by excluding selfish nodes from the network. Additionally reputation systems stimulate cooperation since nodes can rely on other nodes service as compensation for own contributions. Virtual currency systems on the other hand adapt the "pay for service"-concept to ad hoc networks. Nodes compensate for other nodes services by paying a fee using credits. Nodes can either earn credits by providing services to other nodes, or buy them from outside of the system. It is worth noting that virtual currency systems not only stimulate cooperation between nodes, but also additionally could raise interest in ad hoc networks for a number of business applications.

Virtual currency systems proposed in literature mainly focus on how to pay and on how much to pay for data transmission along a given path. As pricing function, in most cases auctions or fixed price schemes, based on game theoretical considerations, are used. These schemes do not consider the context of nodes nor adapt to user's needs. However, we argue that the consideration of node's context and the user's ability to control the degree

of participation in networks is crucial for the acceptance of virtual currency systems as enabler for ubiquitous ad hoc networks.

Focusing on these issues, we present Cashflow, a virtual currency system to motivate nodes to participate in ad hoc networks and to prevent selfishness. This system distinguishes itself from other virtual currency systems by using a channel concept for data transmission as well as a market system for pricing. The combination of channel and market concept results in a number of positive system characteristics. It gives users control over their participation degree, allows Cashflow to consider the context of nodes, and provides implicit load balancing and access control functionality.

1.1 Motivation

Besides the fact, that virtual currency systems in mobile ad hoc networks are still a relatively new and active research field with a number of open issues to solve, there exist two causes, which make this research area especially interesting.

First, mobile ad hoc networks, when connected to other networks and deployed using equipment belonging to different authorities like smart phones or laptop computers, provide the basis for a number of potential applications. For instance, the combination of mobile ad hoc networks with cellular networks could be used to extend cell coverage. Additionally, if single cells are high loaded, traffic could be routed to other cells with lower traffic [Cavalcanti05]. Besides the usage of mobile ad hoc networks for normal data exchange, these networks could be used to distribute local information automatically. For example, cars connected over a mobile ad hoc network could warn other cars about accidents or ice on the road. Further emergency warnings could be distributed to users, which are not connected to other networks.

The combination of mobile ad hoc networks and virtual currency system provides the basis for commercial applications. To give an example, the integration of WLAN access points in mobile ad hoc networks using a virtual currency system would allow users to access the Internet over all participating access points. The owners of the access point would profit from this concept since they could earn money using already available equipment by selling unused bandwidth to other users. The users of this service might profit by lower prices compared to Internet access over today's cellular networks or faster internet connection.

The commercial factor leads to the second cause, which motivates research

in the area of virtual currency systems in mobile ad hoc networks. The deep integration of a payment system in mobile ad hoc networks provides the potential for a number of business scenarios not directly related to mobile ad hoc networks. For instance, it allows transferring credits using any equipment implementing the virtual currency system. With other words, each mobile node implements the functionality of a credit card. With respect to this capacity of virtual currency systems, Cashflow was designed so that it can be used as basis for commercial applications. It provides payment functionality, which can be used by commercial applications, and search functionality, which allows applications to find not network relevant services in the surrounding area.

1.2 Main contributions

The main contributions of this thesis are:

1. The architecture of the virtual currency system Cashflow. Using a modular approach, different functionalities are encapsulated in different modules. This concept allows adapting and extending parts of the system without interfering with other parts. For instance, one module categorizes links depending on their signal quality. If the radio interface provides special functionality for link assessment, a special statistic module could be written to use the additional functionality without altering other modules.

2. The development of a special routing algorithm, which fits the needs of virtual currency system. In contrast to other algorithms, which in most cases are optimized to find the shortest route between nodes, this algorithm allows to find the cheapest route between nodes under the consideration of quality requirements. The algorithm was optimized by using artificial delaying to change racing conditions, which leads to an decrease of routing overhead.

3. The development of a new kind of pricing system which uses demand and supply to determine the fee nodes charge for packet forwarding. Additionally, the pricing algorithm includes the node's context and the preferences of the user. This allows the make the forwarding services of nodes more or less attractive for other nodes. The result is that for instance nodes, running low of battery, only participate minimally to the network compared to nodes where energy is not an issue. By including the user's preferences into pricing, Cashflow allows users to

regulate their participation degree, which is important for the user's acceptance of the system.

4. The introduction of the channel concept into virtual currency systems. The channel concept allows informing nodes about the fee they have to pay for data transmission to another node before the actual transmission starts. This is again important for the acceptance of the virtual currency system. Additionally it allows nodes to reschedule data transmissions in situations where the price for data transmission is high. Further, the channel concept allows to reduce the payment overhead, since nodes could pay for packet forwarding in bulk and have not to pay for every singly packet separately.

5. The introduction of implicit load balancing functionality as part of the virtual currency system. The combination of the pricing function, which includes the nodes context into the fee calculation, the channel concept, and the routing algorithm, optimized for virtual currency systems, acts as load balancing system. With an increase of the load, the ratio between supply and demand changes resulting in a higher price. Since nodes could use the route discovery algorithm of Cashflow, they can avoid high price areas of the network and consequently avoid highly loaded network parts. This results in a load balancing effect, which will be analyzed in detail in the evaluation part of this thesis.

6. The integration of mobile ad hoc networks using Cashfow as virtual currency system within the Internet. To the best knowledge of the author, Cashflow is the first virtual currency system, which allows connecting mobile ad hoc networks to the Internet. Other virtual currency systems focus on payment within a mobile ad hoc network. Using a newly developed concept, Cashflow allows nodes within mobile ad hoc networks to search for nodes connected to the Internet and negotiate a fee for the usage of the Internet connection. Additionally Cashflow provides payment functionality to pay for the used Internet service. Further Cashflow adapts the mobile IP concept to allow nodes to be available over the Internet even if they are connected over a mobile ad hoc network. The possibility to connect to the Internet over a mobile ad hoc network is again an important factor for the user acceptance and for a number of applications.

7. The search for nodes providing internet access is realized in a generic way with the result that Cashflow can be used as platform to integrate

additional pay-services into the mobile ad hoc network. Therefore, Cashflow makes mobile ad hoc networks interesting for a number of business scenarios.

8. A prototypical implementation of Cashflow in IBKSim. This implementation allows developing and evaluating new modules and applications for Cashflow using thousands of virtual nodes.

9. The last contribution of this thesis is the evaluation of Cashflow including the proposed algorithms.

Parts of these concepts and results have been published in several scientific papers, including [Wallentin10a] and [Wallentin10b]

1.3 Structure of this thesis

The remainder of this theses is structured as follows: Chapter 2 presents a short introduction on mobile ad hoc networks and gives an overview over routing protocols in mobile ad hoc networks. Further, it provides a short introduction into the IEEE 802.11 standard, which is important for the understanding of the artificial delay's influence on the quality of route search. Additionally this chapter presents current solutions to prevent selfishness in mobile ad hoc networks and to motivate node's participation.

Chapter 3 presents Cashflow. It starts with a requirement analysis followed by a presentation of Cashflow's core concepts. The next part presents the architecture of Cashflow in detail and proposes algorithms for the different modules of Cashflow. After the presentation of the architecture, different attacks on Cashflow are discussed and how they are prevented. The chapter continues with a solution to integrate Cashflow into the Internet before it concludes with a comparison of Cashflow with other virtual currency systems.

Chapter 4 focuses on the evaluation of Cashflow. It presents a detailed analysis of the presented routing protocol including an evaluation of the artificial delays influence on several parameters. Additionally this chapter includes an analysis of the proposed pricing function. The last part of Chapter 4 focuses on the effect of Cashflow's implicit load balancing functionality.

Chapter 5 concludes this thesis with a summary and gives an outlook of potential further work, followed by Annex A, which presents additional diagrams belonging to the evaluation of the route discovery protocol.

Chapter 2

Related work

This chapter gives a short overview over related work. It starts with a short description of the IEEE 802.11 MAC protocol, since for the evaluation of the system this standard is used for communication. Therefore, basic knowledge about this protocol is needed to understand the interplay between the virtual currency system Cashflow and the MAC protocol. The next section of this chapter introduces the concept of mobile ad hoc networks. The last two sections focus on two specific mechanisms of mobile ad hoc networks: on routing mechanisms and on concepts stimulating cooperation. As part of this theses, a new routing algorithm especially designed for the requirements of virtual currency systems was developed which will be presented in Chapter 3. Therefore, the section on routing algorithms provides an overview over basic routing concepts in mobile ad hoc networks. The section about cooperation in mobile ad hoc networks gives an overview over other virtual currency systems, as well as over other mechanisms to prevent selfishness in mobile ad hoc networks and stimulate cooperation. A more detailed overview over mobile ad hoc networks including the topics presented in this chapter can be found in [Nahrstedt09]

2.1 Introduction to the IEEE 802.11 family

The IEEE 802.11 [IEEE99] [IEEE07] standard specifies layer 1 and 2 of the OSI model [OSI84] for wireless communication in local area networks. Besides the original standard, a number of extensions have been standardized. These extensions define for instance additional coding and transmis-

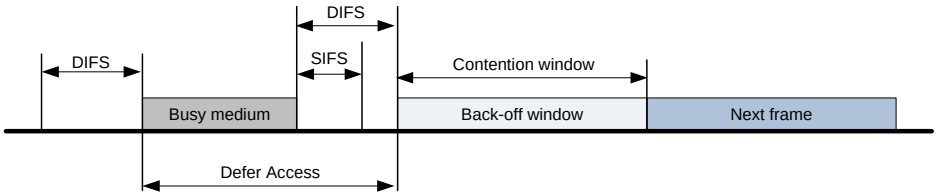


Figure 2.1: Basic shared medium access

sion schemes to gain higher throughput [IEEE00] [IEEE03] [IEEE09], new security schemes [IEEE04] or support for quality of service [IEEE05]. However, this section focuses only on the MAC protocol as specified in [IEEE99], since this protocol is relevant for this thesis to understand the interplay between Cashflow and the MAC protocol as it is presented in Chapter 4, where the simulation results are discussed.

The IEEE 802.11 standard specifies two methods for accessing the shared medium. The first method is the distributed coordinator function, which is mandatory for all devices using the IEEE 802.11 standard. This distributed access method can be used by nodes of a mobile ad hoc network to access the shared medium. Therefore, it will be discussed in this section in detail. The other method is the point coordinator function, which is an optional method for accessing the shared medium. This method is used by access points to coordinate the client nodes access to the shared medium. However, since this method is of minor importance for mobile ad hoc networks, it will not be discussed any further.

Figure 2.1 visualizes the basic access scheme of the distributed coordinator function using a time line. Before a node transmits data over the shared medium, it senses if the shared medium is free for a certain time span. This period of time is called DIFS, which stands for distributed coordinator function interframe space. If the medium is not free during the whole time, the transmission is deferred until the medium is free for the required time period. When the medium was free for the duration of the distributed coordinator function interframe space, the node sends data. The transmission time is marked with *Busy medium* in Figure 2.1. During this period all nodes, while sensing that the shared medium is busy, defer their access to avoid a disruption of the transmission. After the transmission, the node, which has used the shared medium, cannot directly transmit additional data over the shared medium. If the node would only wait again until the distributed coordinator function interframe space has passed before it transmits again data, other nodes, which have deferred their access because of

the node's transmission, would also start to transmit data at the same time. This is the case since they would also have sensed that the shared medium was free during the complete distributed coordinator function interframe space. Since this behavior would lead to collisions, the so-called back-off algorithm as to be performed by nodes after they have transmitted data over the shared medium. After the transmission, the node randomly chooses a back-off time. After the pass of distributed coordinator function interframe space, the node also has to wait additionally until the back-off time has passed, before it is allowed to transmit data again. If during the back off time the shared medium is used by another node, the node stops its back off timer and continues to run the timer if the medium was free again for the period of the distributed coordinator function interframe space. Since the value of the back-off timer is randomly chosen, the probability that two or more nodes access the medium at the same time decreases. However, it is not completely excluded that multiple nodes choose the same back-off time and therefore try to access the shared medium simultaneously, which leads to collisions as described before. If collisions occur, nodes learn about them because of an acknowledge scheme defined in the standard, which will be explain later in this section. When nodes learn that a frame was lost, they calculate a new back-off time before they retransmit the frame using a larger range of possible back-off timer values. This decreases the probability, that again two or more nodes choose the same back-off value. The time, when all nodes are waiting until their back-off time passes, is called contention window. As soon as the back-off time of a node has passed, the node is allowed to transmit over the shared medium before it calculates a new back-off time.

In Figure 2.1, besides the distributed coordinator function interframe space, also the short interframe space is plotted. This interframe space is used by the acknowledgment scheme of the IEEE 802.11 protocol to send acknowledge frames with high priority back to the transmission's source. Since nodes, wanting to transmit data, have to wait at least for the duration of the distributed coordinator function interframe space, the acknowledge frames, send after the short interframe space, are transmitted earlier.

Figure 2.2 visualizes the acknowledge scheme specified in the IEEE 802.11 standard. As described before, if a node wants to transmit data using the shared medium, it has to wait until the shared medium has been free for the duration of the distributed coordinator function interframe space, and optionally until the back-off time has passed. If the node has already waited until the back-off time has passed, but did not had any data to transmit, the node can immediately transmit data after the distributed coordinator

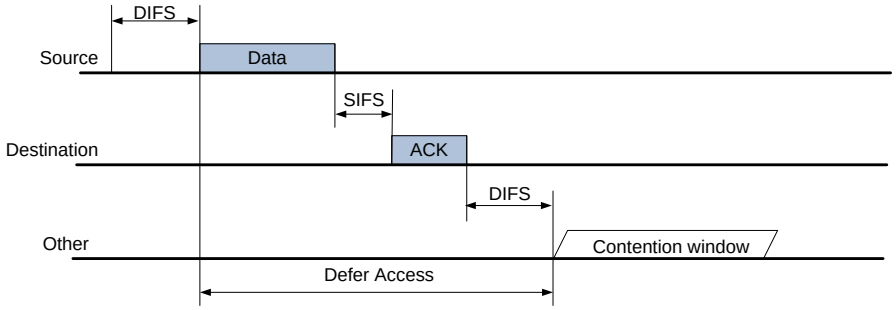


Figure 2.2: Acknowledge mechanism of IEEE 802.11

function interframe space has passed as shown in this figure. When the node starts its data transmission, other nodes defer their transmission until the medium is free again. After the transmission, the target node returns an acknowledge frame back to the source after the time of the short interframe space has passed. When the source receives the acknowledge frame, it has the verification that the destination node has received the frame correctly. Otherwise, if the source node receives no acknowledge frame after the sort interframe space, a frame loss has occurred. To be specific, this means that the destination node has not received the frame, the frame has not been transmitted correctly, or that the acknowledge frame was lost or invalid. As reaction on these events, the source node retransmits the frame. To do so, it chooses a new back-off time from an increased value range and tries to retransmit the frame again after the back-off time has passed. If also the retransmission fails multiple times, the frame gets discarded, since the source node assumes in this case that there is no connection to the destination node.

This scheme avoids collisions if all nodes of a network could sense the transmission of all other nodes. However, in many cases this is not given. It could happen that a node is only in the range of the source or the destination node and therefore could not detect all transmissions. For instance, a node, which is only in the range of the destination node and not of the source node and wants to send a frame itself to the destination node, will not sense the transmission from the source to the destination node. Therefore, it will start its own transmission to the destination node, which results in a collision of the two simultaneous transmissions. The result is that the destination node receives none of both transmissions and both frames have to be transmitted again. This problem is known as hidden station problem [Tobagi75].

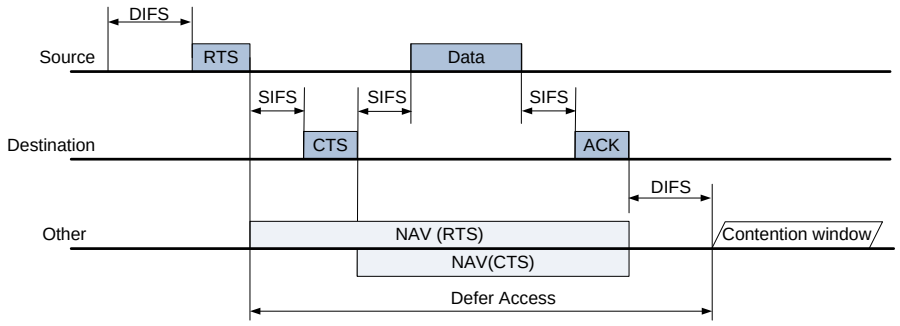


Figure 2.3: Request to send / clear to send mechanism of IEEE 802.11

Since this problem is especially serious if the frames are large, an additional virtual carrier sense mechanism was introduced by the IEEE 802.11 standard. The mechanism, known as request to send / clear to send (RTS/CTS) mechanism, is visualized in Figure 2.3. In contrast to the acknowledge scheme described before, two additional messages are exchanged before the actual data transmission is performed. The first frame is the request to send frame, transmitted by the source node to the destination node. This frame includes information how long the complete transmission will take. All node receiving this frame set their network allocation vector (NAV) accordingly, which can be seen a kind of counter or timer preventing nodes to access the shared medium if set. The destination node returns after the short interframe space a clear to send frame, including again information about the duration of the remainder transmission. Again, nodes receiving this frame set their network allocation vector accordingly. Using this mechanism, hidden stations learn about the transmission and consequently will not interrupt it. However, during the exchange of the request to send and clear to send frames, the hidden station problem is still an issue. Nevertheless, if the data frame is relatively large compared to the request to send and clear to send frames, the probability that the request to send or the clear to send frames get interrupted is smaller compared to the scenario, where the data frame is send directly. On the other hand, if the data frame is small, a transmission without the request to send / clear to send mechanism could be more efficient, especially if there is only low traffic in the network.

The acknowledgment scheme, as well as the request to send /clear to send mechanism is only used for direct communication. To broadcast frames, only the basic access scheme is used, meaning that a node broadcasts a frame as soon as is senses that the shared medium is free for the distributed coordinator function interframe space. Since no acknowledgment mecha-

nisms is used, the source node does not know which nodes have received a broadcast and which not. As it will be described later, many routing protocols of mobile ad hoc networks use broadcasts to flood networks with route requests. The flooding in combination with the hidden station problem could lead to quality problems of the route search, since potential routes might not get considered due to frame losses, as described in [Ni99]. Therefore, for the development of route discovery algorithms for mobile ad hoc networks, the interplay between MAC-protocol and route discovery algorithm has to be considered.

2.2 Mobile ad hoc networks

A mobile ad hoc network is a mesh network, which is spontaneous formed by so-called nodes. Nodes are platforms equipped with wireless communication technology, a router and one or more hosts. All these components might be integrated into one single device. For instance, a node could be a special equipped sensor platform, a smart-phone, laptop computer, a car, or an airplane. Nodes, which are free to move around, act as router and forward packets on behalf of other nodes. This concept allows to form wireless mesh networks without the need of additional infrastructure. [Macker98]. Additionally, such networks possess no single point of failure and can therefore tolerate the loss of nodes.

Due to the dynamic nature of mobile ad hoc networks, routing in these networks is challenging. As result, there exist a number of different routing strategies with different assets and drawbacks. External factors like the mobility degree influence the efficiency of the protocols. For instance, proactive routing protocols flood information about topology changes over the whole network, independent if this information is currently needed or not. Therefore, this category of routing protocols have a relative small route discovery overhead, compared to reactive routing protocols, in networks where nodes show sporadic movement. Reactive routing protocols only perform route discovery when needed by nodes. Therefore, if nodes show a sporadic mobility pattern, the route discovery overhead is greater, compared to proactive routing protocols, since each node has to perform route discovery for the first data transmission to another node. However, if nodes show a very active mobility pattern, proactive routing protocols, which try to keep topology information up to date, show ongoing route discovering activity. This is especially contra productive in scenarios, where nodes exchange data sporadically. In these scenarios, reactive routing protocols have a lower overhead, since they perform only for actual transmis-

sion route discovery. Another issue for mobile ad hoc networks, which is also related to the routing, is scalability. Targeting this issue, a number of routing protocols have been developed, which divides networks into clusters, introduce hierarchies, or use location information to optimize routing and increase scalability. Since also applications and services like Cashflow, which are incorporated in mobile ad hoc networks, have specific needs on routing, resulting in new issues and challenges, the next section discusses routing in mobile ad hoc networks in more detail.

Nevertheless, routing is not the only challenge for mobile ad hoc networks. Like in other networks, security is an important issue. The wireless nature of mobile ad hoc network and the fact, that nodes forward data of other nodes and could therefore intercept communication, aggravates the security issue. Another important issue is the energy consumption of nodes. Because of the mobile nature of nodes, in many scenarios, nodes are battery powered with the consequence that energy supply is limited. Therefore, especially in sensor networks, which, depending on the usage scenario, should work up to months or years, the energy consumption has to be considered.

As stated before, the usage of the mobile ad hoc network concept to connect mobile office equipment belonging to different users requires a rewarding scheme to motivate users to participate on the one hand, and, on the other hand, to prevent selfish behavior. Since this thesis focuses on this issue, similar mechanisms are also presented in this chapter. However, the usage of mobile office equipment leads additionally to other challenges. One of these challenges is the integration of mobile ad hoc networks, build out of mobile office equipment, into other networks like the Internet or cellular networks. In this scenario, the nature of mobile ad hoc networks changes from an isolated, self-sufficient network to an extension of other networks, leading to a number of issues. An overview over open issues in integrating mobile ad hoc networks into other networks is given in [Cavalcanti05].

Summarizing, the mobile ad hoc network concept has the potential to make the step from a niche technology to a technology, included in every mobile equipment, to extend the coverage of existing infrastructure and provide ad hoc connection between multiple participants. This also reflects in current initiatives to establish standards for mobile ad hoc networks allowing to connect equipment form different manufacturers [IEEE08] [Macker10].

2.3 Routing in mobile ad hoc networks

In mobile ad hoc networks, routing protocols are needed for forwarding packets between network nodes, which are outside of each others broadcast range. In consequence, intermediate nodes are required to act as routers to forward packets. Optimal routing is one of the key factors to provide high performing mobile ad hoc networks. Therefore routing is one of the major research fields since the beginning of the concept of mobile ad hoc networks. The huge number of routing protocols proposed in literature can be classified depending on their route determination strategy as shown in Figure 2.4. Reactive or on-demand routing protocols perform a route search only when required, whereas proactive protocols maintain routing information so that every node in the network always knows the complete topology and consequently can transmit packets immediately to any node of the network, without performing extra route search. Additionally, there exist a number of protocols, so called hybrid protocols, which combine these two classes.

In mobile ad hoc networks using proactive routing protocols, network nodes continually monitor their environment. When a node detects a change in his environment, it informs other nodes about this event. Typically, this is done using some kind of flooding, whereas flooding overhead is reduced using known information about the network. As mentioned before, the use of proactive protocols has the benefit that packet transmissions can start immediately, without performing extra route search. However, this benefit comes at the cost of scalability problems, caused by the size of routing tables respectively due to caching of topology data. Additionally, frequent topology changes can cause high costs to keep the topology information up to date, which is especially problematic, if network activity is low and consequently updated information is not always used.

The Dynamic Destination-Sequenced Distance-Vector Routing Protocol (DSDV) [Perkins94], which is based on Bellman-Ford routing [Cheng89], and the Global State Routing (GSR) protocol [Chen98] are examples for early generation proactive routing protocols, developed on basis of routing concepts, originally designed for wired networks. Both protocols are flat protocols, meaning that every node has exactly the same functionality. In Dynamic Destination-Sequenced Distance-Vector Routing, if a node detects a topology change it broadcasts an update message, which could include either incremental update information or a complete routing table. Nodes that receive such a message update their routing table and accordingly broadcast the changes. The disadvantage of this blind flooding strategy is

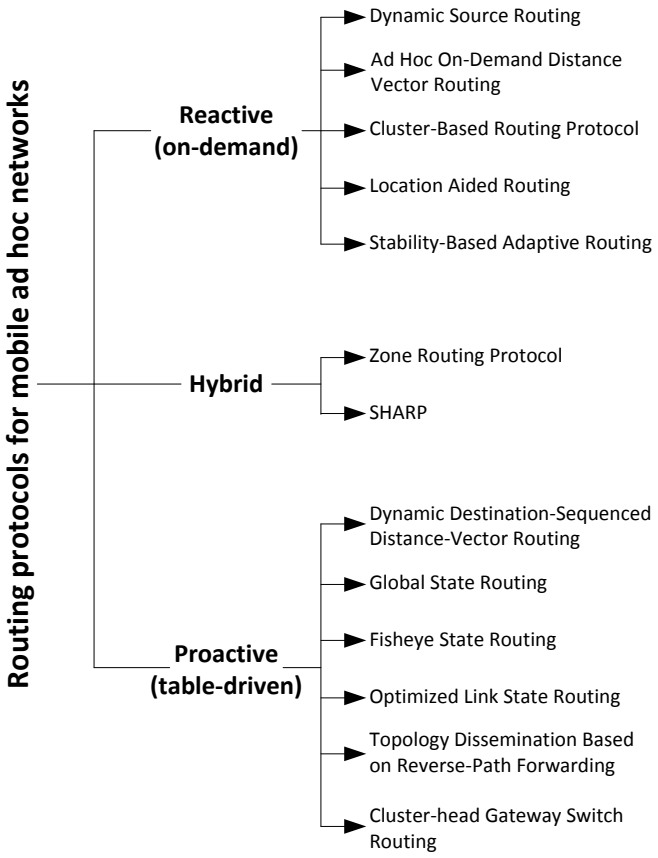


Figure 2.4: Overview over routing protocols in mobile ad hoc networks

that it can lead to broadcast storms [Ni99].

In Global State Routing, nodes broadcasts link state information periodically and not event triggered. This has the advantage that information about short disconnections between nodes is not propagated over the complete network. However, again every node broadcasts an update message if it has changed its routing table.

The Fisheye State Routing (FSR) [Pei00] protocol, which is based on Global State Routing, tries to reduce the flooding overhead by broadcasting update information from distant nodes more seldom than from nearer nodes. Hence, nodes have an exact knowledge about the topology of their neighborhood, but their knowledge about the complete network is not always complete. Other protocols like the Optimized Link State Routing Protocol

(OLSR) [Jacquet01] and Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) [Bellur99] tries to optimize flooding in dense networks by reducing the number of re-broadcasting nodes. In Optimized Link State Routing Protocol this is done by calculating for every node the minimum set of nodes needed to flood a message to all two-hop neighbors. Just the selected nodes, so called multipoint distribution relays, generate and re-broadcast topology control messages, whereby the flooding overhead is reduced. Hierarchic protocols like Cluster-head Gateway Switch Routing [Chiang97] builds clusters, where it is up to a specific node in the cluster, the cluster head, to exchange routing information with other clusters. This leads to a much better usage of the resources then blind flooding.

Reactive routing protocols try to reduce the routing overhead by searching for a route only when required. Especially in scenarios with high mobility and low communication activity networks can profit from this strategy. Dynamic Source Routing (DSR) [Johnson96] and Ad Hoc On-Demand Distance Vector Routing (AODV) [Perkins99] are typical examples for reactive routing protocol. Both protocols use blind flooding as route discovery strategy. When a route to a specific node is needed, and there is no cached routing information from a previous search to the target node, a route request packet is flooded though the network. When the packet eventually reaches the destination node, this node will send a route reply packet back to the source of the route request. In Ad Hoc On-Demand Distance Vector Routing the intermediate nodes learn during the route discovery to which neighbor node they have to forward a packet to reach a certain node. In Dynamic Source Routing, the intermediate nodes make their routing decisions based on the routing information included in every packet. To reduce the number of route requests, both protocols uses caching techniques.

Cluster-Based Routing Protocol (CBRP) [Jiang98] reduces the flooding overhead by introducing a hierarchy. It forms clusters where the cluster head is responsible for forwarding and exchange of route requests. An alternative procedure to reduce the routing overhead is by using additional information like the location of the nodes if available. Location Aided Routing (LAR) [Ko98] calculates an area using the positions of source and target nodes. Only nodes within this zone are allowed to forward route request packets, by what the flooding is reduced to the proximate direction of the target node. All of these protocols have in common that they tend to prefer the shortest path between two nodes. The Signal Stability-Based Adaptive Routing Protocol (SSA) [Dube97] follows another strategy: The aim of Signal Stability-BasedAdaptive Routing is to establish stable routes through a wireless network. Every node monitors the signal strength of the neighbor

nodes and classifies connections as weak and strong. Route requests are only forwarded along strong connections, which results in more stable routes than using just the shortest path. The drawback of this solution is that if there exist just paths with at least one weak connection between a source and a destination node, the protocol will not find any route to the target node.

Hybrid routing protocols combine reactive and proactive routing techniques. Generally, reactive techniques are used within a limited region around a node and proactive techniques for nodes farther away. This approach can reduce the routing overhead and can increase the performance, if nearby nodes are more likely to collaborate. Examples for hybrid routing protocols are the Zone Routing Protocol (ZRP) [Haas99] and SHARP [Ramasubramanian03].

2.4 Cooperation in mobile ad hoc networks

Participation of nodes in mobile ad hoc networks cannot be taken for granted if the nodes belong to different authorities like private users and companies. Generally, nodes have no interest to participate in network wide tasks like forwarding packets on behalf of other nodes voluntarily, without any reward. Hence, mechanisms are needed to stimulate cooperation between nodes. Current research focuses on two ways to achieve this task. Reputation systems try to stimulate fair behavior by excluding selfish nodes from the network. The motivation of nodes to participate in networks using reputation systems is that they can expect support of other nodes as compensation for provides services. Several systems have been proposed to perform this task, including OCEAN [Bansal03], CORE [Michiardi02] and CONFIDENT [Buchegger02a] [Buchegger02b].

Virtual currency systems use (sometimes virtual) money as reward for provided services to motivate nodes to cooperate. The basic idea is that the sender of a packet pays the relaying nodes a fee for forwarding packets. This is the case in the packet purse model in Nuglets [Buttyan01]. The source of a packet adds a number of nuglets, the virtual currency of the system, to the packet it wants to transmit to a target node. Each forwarding node removes some nuglets from the packet to compensate its forwarding costs. If there are not enough nuglets left in the packet, it gets discarded. The problem for the source of the packet is to estimate, how many nuglets should be included in a packet. It has to overestimate the needed amount to reach the target node, but the higher the overestimation, the higher the loss for a node, if the packet gets lost for instance due to transmission failures. The other model proposed in [Buttyan01] [Buttyan00] is the packet trade model. In contrast to the packet purse model, the receiver pays for packets. Each

intermediate node buys packets from the previous node in the path and tries to sell it for a higher price to the next node. Consequently, the target node pays effectively for the complete transmission. A combination of both modes is possible. As pricing strategy an auction scheme or alternatively a fixed price scheme is used.

To prevent fraud, Nuglets needs tamper-proof hardware. Otherwise, it would be possible for nodes to extract more nuglets from passing packets than entitled to. The need of special hardware makes the usage of Nuglets in scenarios, where node consists out of mobile office equipment, difficult. In Nuglets it is additionally possible to lose money by losing or discarding packets, which lead to a decrease of available nuglets in the system over time. Therefore a type of compensation system for lost nuglets is needed, which is however not indented by the system.

To overcome the issues of Nuglets, the virtual currency system Sprite, presented in [Zhong03], proposes a credit system for payment instead of a virtual hard currency. A credit clearance service is introduced to determine the charges. When a node forwards a packet, it keeps a receipt and later sends the collected receipts to the credit clearance service, which compensates the node for provided services. Using this concept, no tamper-proof hardware is required, which allows to deploy this system using no special hardware. Additionally the loss of a packet does not result in the loss of the virtual currency. In Sprite, the credit clearance service is also used to prevent fraud by using game theory. The reward for a forwarded packet depends if the packet was received by the target and in the case the packet was not received by the target, the reward additionally depends if the next hop along a path has received the packet or not. Using this conditional rewarding scheme, no motivation for malign behavior, like dropping a packet but claim to have forwarded it, is given.

Both virtual currency systems are not interconnected with the routing functionality of the network. They solve the problem how to pay for packet forwarding along a path, but they do not provide a general mechanism how to transmit a packet using the cheapest path. An exception is the fix price scheme of Nuglets, where the shortest path is compulsorily the cheapest path. Using the auction scheme in Nuglets, if there is more than one route to the target node, the intermediate node chooses the route where the next hop provides the cheapest service. Again, this does not have to be the optimal choice from a global view.

Commit [Eidenbenz05], which is based on Ad hoc-VCG [Anderegg03], bridges the gap between route discovery and auction for pricing. In Ad

hoc-VCG, when a source node needs a route to another node, it broadcasts a route request. Every time a node receives a route request, it checks whether the packet contains new topology information. If so, it adds information about the link over which it has received the request, for instance the transmission power, to the packet and rebroadcasts the request. Using this technique, the destination node finally learns the complete topology including the additional parameters and can use this information to calculate the best route. It is worth noting, that using this route discovery technique, every node has to forward up to n^2 route requests per route search in comparison to 1 like in other protocols. Additionally, the complete route selection is done by the target node, which could lead to trust and security issues. In COMMIT the overhead of the discovery phase for every node is reduced by forwarding just information about new links a node has learned. Additionally, an upper limit that the sender is willing to pay is included in the request. Therefore, the forwarding nodes can decide during the route discovery phase, whether the forwarding of information about a new received path is feasible or not, and react correspondingly.

Another virtual currency system is iPass [Chen04a], which differs from other virtual currency systems by the way it determines the fee for packet forwarding. In contrast to the other virtual currency systems presented in this section, it uses a flow-oriented scheme for pricing. In iPass, each node represents an auction market where flows passing through are bidding for the limited bandwidth. Each packet contains four fields relevant for pricing: the request rate, which determines the preferred size of the flow in bytes per second, the current rate, which corresponds to the actual size of the flow in bytes per second, the bid for bandwidth and the actual charge for the packet, which is initiated to zero and gets decreased whenever the packet is forwarded by a node. Since the bandwidth is limited, flows are assigned bandwidth depending on their bid. For instance, when each of three flows request 40% of the bandwidth of a node, only the two flows with the highest bid become 40% assigned. The third flow, which is the flow with the lowest bid, becomes only 20% assigned. Therefore, the current rate field of packets belonging to the third bidder is set correspondingly to 20%, so that over a feedback mechanisms the source node learns that it has to increase its bid to get the requested 40% of bandwidth. The actual fee nodes pay for packet forwarding depends on the bids of flows, which did not receive all the bandwidth they requested. In the given scenario, the price per byte corresponds to the bid of the third flow, which got only assigned 20% of the bandwidth. If there would exist an additional fourth flow which would also request 40% of the bandwidth, the price would be the sum of one third of

the bid from flow three and two thirds of the forth flow's bid. In contrast to other virtual currency systems, iPass prevents overload of single paths by coupling the pricing function with the availability of bandwidth.

Even if there exist solutions to integrate payment into mobile ad hoc networks to motivate nodes to participate, a number of issues remain unsolved. Among others, these issues include the user's possibility to control the participation degree of nodes, the efficient coupling of virtual currency systems with routing mechanisms to allow nodes to use the cheapest route to a target and the inclusion of node's context.

Chapter 3

Architecture of Cashflow

This chapter [1] presents Cashflow, a flow oriented virtual currency system designed to prevent selfish behavior in mobile ad hoc networks and to motivate nodes to participate. The first part of this chapter specifies usage scenario for Cashflow. These scenarios are used to formulate requirements and assumptions leading to the different concepts Cashflow is based on. After the description of the concepts, Section 3.3 presents the architecture of Cashflow in detail, including all its modules and algorithms. The following section discusses possible attacks on Cashflow and describes how these attacks are prevented. Section 3.5 extends the usage scenarios of Cashflow by describing how Cashflow can be integrated in hybrid and Internet connected networks. The chapter concludes with a comparison of Cashflow with other virtual currency systems.

3.1 Usage scenarios of Cashflow

Cashflow was designed to encourage nodes belonging to different authorities to participate in a common wireless network. For the development, the focus laid on scenarios like a shared office building, where the communication infrastructure is temporarily not available, or, to give another example, a hotel, where members of a traveler party, accommodated in different rooms, what to exchange pictures using their mobile equipment without leaving their room.

In both described scenarios, communication between different nodes is possible, provided that intermediate nodes, equipped with wireless communication technology, are willed to join a common wireless network and provide

[1] Parts of this chapter have been published in [Wallentin10b] and [Wallentin10a]

services like routing and forwarding for other network nodes. Additionally, it can be assumed that in most cases not all nodes are moving at the same time. Taking the office example, a user might use a laptop at his office for some time, than he brings his laptop to a meeting where the laptop stays again at nearly the same time during the whole meeting, and after the meeting he takes the laptop back to his office. So this laptop would show a nomadic mobility pattern during a normal work day.

Besides location, also the context of this laptop changes over time. During the usage in the user's office, the laptop might run on AC power, while during the meeting, the internal battery is used as power source. While running on battery, the energy consummation is a much more important issue, compared to the time while running on AC power. Therefore, during the time running on battery, the user might not be interested to provide the same services for the network as while running on AC. The energy source might not be the only parameter influencing the user's willingness to provide services. Since routing needs resources from the computer, like processing time and memory, users running resource-intensive applications like games, 3D rendering applications or simulations, and therefore needing their computer resources for themselves, might not be interested to provide larger parts of their resources for network services. Therefore it was assumed for the design of Cashflow that it is important to consider the context of the node for compensation.

The described scenarios allow making expectations concerning the traffic. Because of the size of images and other digital documents, it is assumed that in many cases a data transmission between nodes will include a number of packets and not just a single one. Additionally, protocols like TCP require a number of packet exchanges for the transmission of even a single data bit. Consequently, for the development of Cashflow, a specific traffic pattern was assumed which is different compared to typical traffic pattern of other wireless networks like sensor or vehicular networks.

Naturally, scenarios, where no infrastructure exists, are not very common today. Especially the scenario, describing a shared office with no telecommunication infrastructure, might be an extreme example. Never the less, for the development of the virtual currency system, scenarios with no infrastructures were assumed, since from the system's point of view, an infrastructure device, like a wireless access point, can be seen as a normal node with some special parameters. However, even if infrastructure devices are not explicitly considered as special nodes for the description of Cashflow, Section 3.5 of this chapter discusses the usage of Cashflow in hybrid and

Internet connected networks in detail.

Summarizing, Cashflow was developed focusing on scenarios where the following assumptions are fulfilled:

1. Network nodes are equipped with wireless communication technology.

2. There exists at least one direct or indirect connection between every node.

3. There exists no additional infrastructure or the existing infrastructure cannot be used.

4. Network nodes belong to different authorities.

5. Nodes can be motivated to provide services to other nodes by using some kind of compensation.

6. Network nodes consists of mobile office equipment like laptops, smartphones, and personal digital assistant.

7. Nodes are free to move, but most networks nodes show a nomadic behavior, meaning that not all nodes are moving all the time.

8. In most cases, nodes want to transmit a number of packets to other nodes, not just single packets.

9. Nodes are willing to pay a certain maximum fee for services of other nodes.

10. Nodes want to pay as little as possible for the service of other nodes.

11. All nodes have sporadic access to the internet, but it is not required that nodes have internet access during their participation in a wireless network.

Additionally it was assumed that some of the following characteristics of nodes might be true:

12. Nodes have different preferences regarding to their participation in networks, independent from the reward they receive for their services.

13. The node's context has an influence on its willingness to participate in wireless networks.

All these assumptions provided a framework for the development of Cashflow. Some assumptions, like 1 to 3, are essential in all ad hoc networks, since otherwise the deployment of a mobile ad hoc network is not possible or makes no sense. Other assumptions, like 4 and 5, are essential for integration of virtual currency systems, because if all nodes would belong to one single authority, cooperation would not be an issue and therefore a system like Cashflow would be obsolete in such kind of network. This is also true in the case that nods cannot be motivated to participate using rewarding schemes. All the rest of the assumptions define real life scenarios. Some of these assumptions result in restrictions in the system others provide optimization potential. For example, the usage of regular mobile office equipment like laptops and smart-phones in such a network results in the requirement that the system should not rely on special hardware like tamper-proof hardware, which would simplify the integration of a virtual currency system from the viewpoint of security. On the other hand, the assumption of a specific mobility and traffic pattern allows optimizing the routing as well as the data transfer for these specific scenarios. All these assumptions lead to a system, which is optimized for a special type of scenario. Naturally, Cashflow can also be used in scenarios, where some of the assumptions are not true. In this case, some functions of Cashflow might not be used or there is an additionally overhead. For example, in the case that Cashflow is used in scenarios where tamper-proof hardware is given, other payment strategies might perform better than the standard payment system integrated in Cashflow. However, because of the modular the design of Cashflow, which will be described in Section 3.3, by changing some modules of the system, it can be optimized to fit specific scenarios without changing the complete system.

3.2 Basic concepts of Cashflow

To consider all mentioned assumptions, Cashflow is based on three concepts: a channel concept for data transmission, a market concept as pricing strategy and a credit system for payment. The channel concept as well as the market concept is unique compared to other virtual currency systems for mobile ad hoc networks. The following paragraphs will describe the different concepts and their correlations. However, the actual realization of the proposed concepts will be described in Section 3.3, while discussing the nodes architecture.

3.2.1 Channel concept

The channel concept of Cashflow considers the assumption that in many cases nodes want to exchange a number of packets consecutively and not just sporadically single packets. The basic idea of the channel concept is that if a node wants to transmit a number of packets to another node, it opens a channel to the destination node. It is important to note that channels in Cashflow are bidirectional, meaning that the source node, the initiator of the channel, can transmit data to the destination node and vice versa. A channel is characterized by two parameters. The first parameter is the duration. A channel is open for a certain time during which the source and the destination node can exchange data. After the expiration of the duration time, the channel is closed and the source node has to open a new channel if it wants to communicate with the destination node again. However, the source node, and only the source node, can request an extension of the duration time, which might be granted by the intermediate nodes.

The second parameter is the throughput. It specifies how many packets will be transmitted per second over the channel. Since packets have a defined maximum size, intermediate node can estimate the maximum bandwidth a channel needs. Consequently, nodes can reject channel requests if the needed bandwidth exceeds the available one and therefore avoid overload situations. Besides the avoidance of overload situations, by using the channel concept, nodes can regulate their maximum throughput depending on their current situation. This regards the needs of assumptions 12 and 13. But not only intermediate nodes profit from the channel concept. As soon as a node has successfully opened a channel, it can be confident that the connection to the destination node will not get interrupted abruptly because of upcoming traffic of other nodes. In other virtual currency systems it might happen that an intermediate node stops to forward packets on behalf of a node, because another node requests its services and is willing to pay a higher fee than the other node is currently paying. Even if the node whose transmission has been interrupted continues its transmission by interrupting the interrupters traffic likewise, still some packets have been dropped as consequence of the interruption. Besides the additional delay, caused by the retransmission of the dropped packet by the source node, the source node has to pay the intermediate nodes along the path to the node, which has dropped the packet, even if the packet has never reached the destination node. The reason for this is that these intermediate nodes have forwarded the packet and therefore earned a reward, even if the packet did not reach the target. (Compare with [Zhong03] and [Buttyan01]) Therefore, it is in

the interest of the user, if such abrupt interruptions are prevented by the system.

3.2.2 Market concept

The channel concept has influenced the way nodes determine their service fees. Cashflow uses a market concept to calculate the fees charged by nodes for forwarding packets. The basic idea of this concept is that nodes continuously adapt the fee they charge for the so called standard channel depending on supply and demand. The standard channel is a channel with a fixed throughput and duration value known to every node. Based on the price of the standard channel a node can derive the price of any other channel with any parameterization. When a node wants to open a channel along a given path, it sends a channel request message, including parameter information about the channel, along the path to the destination. The destination node replies with a channel response message. The intermediate nodes include the value of the fee they would charge for the requested channel into the response message, so that the original source nodes finally gets the information how much the channel would cost. Based on the fee of the channel and the price the source node is willing to pay, it can either accept this offer and open the channel or reject it. By adjusting the price, nodes can make themselves, respectively their routing service, more or less attractive for other nodes. Another point of view would be that nodes can express by the fee they charge their willingness to provide additional channels. Given that each node has a certain preferred throughput value. If the current throughput is significantly higher than the preferred throughput, the node could increases the fee with the result that some of the open channels might not be extended by the channels source node after the duration time expiration. Additional the probability increases that nodes requesting a channel reject the channel after receiving the offer because of the price. Therefore, by increasing the fee for channels, nodes can reduce the throughput. The opposite is true when a node decreases its fee. In this case, channels running over this node become cheaper and therefore more attractive to other nodes.

Cashflow additionally uses the fee as instrument for load balancing. Since nodes increases their prices in high load situations, channels through high loaded parts of a network are more costly than through parts with low load. Assuming it is in many cases in the interest of nodes to get a service for the lowest possible fee, nodes are interested to open channels using the cheapest path to the destination and consequently avoid high loaded

parts of a network if possible. This functionality requires a route discovery protocol, able to find the cheapest route from a source to a destination node. Such a protocol has been developed as part of Cashflow and will be presented in Section 3.3.3. However, Cashflow can also be used in combination with other routing protocols, but such a combination might result in a performance penalty of the load balancing functionality.

Besides load balancing, the combination of the channel with the market concept can also be used to prevent local overload situations. When nodes receive a channel request, they estimate the additional usage of the shared medium based on the channels parameters. If the additional usage would cause an overload, they charge an infinite high price, which corresponds to a reject. Additionally, the node increases the fee it would charge for a standard channel, since such a reject indicates an imbalance of supply and demand. Similar to this overload detection, Cashflow allows each node to define a maximum throughput. If a node receives a channel request and its parameters indicate that the additional throughput caused by this channel would exceed the maximum throughput, again the node can reject the request by charging a infinite high fee. As it will be described in detail in Section 3.3.7, parameters, like the preferred and maximum throughput, the minimum price a node charges, and the maximum usage of the shared medium, are used by Cashflow to allow users to influence their participation degree. Additionally, these parameters can be used to react on the node's context.

3.2.3 Credit concept

For the actual payment, Cashflow uses a credit based system where nodes use a kind of virtual credit card to pay rewards for used services. This concept has two advantages compared to other systems: First it can be implemented completely in software and therefore the systems does not rely on tamper-proof hardware like used in Nuglets [Buttyan01]. As mentioned before, it is assumed that Cashflow is used in scenarios, where network nodes are mobile office devices like smart-phones or laptop computers, which are in most cases not tamper-proof. Therefore, the requirement of tamper-proof hardware would hamper the usage of Cashflow. The second benefit is the flexibility in the payment amount's size, compared to real world money or micropayment systems using hash chains [Tewari03b], [Tewari03a]. Using a credit system, it is possible for nodes charging fees worth a fraction of a credit unit, with the consequence that no node has to pay an overhead caused by rounding to the next amount of money supported by the system.

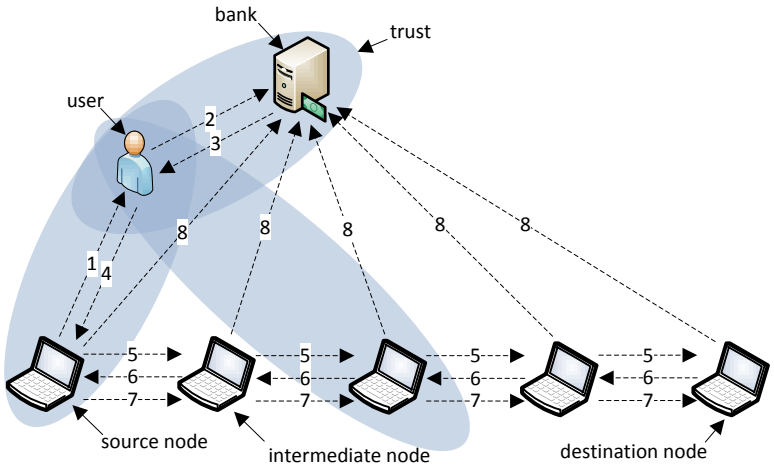


Figure 3.1: Credit system of Cashflow

It is worth noting that the usage of a credit system has also a downside concerning the users privacy. As it will be described later, credit systems uses banks or credit clearance services for accounting and payment. Using payment information, the bank or credit clearance service provider could reconstruct connections between nodes and might use this information to estimate correlations between nodes. However, the bank additionally gets detailed information about the intermediate nodes used for forwarding packets. Therefore, the bank or credit clearance service can be used for fraud detection and also for the exclusion of misbehaving nodes from the network by freezing their accounts.

Due to the benefits of credit systems, similar concepts are also used by other rewarding systems like Sprite [Zhong03] and iPass [Chen04b]. However, the credit system used by Cashflow is optimized for the usage in combination with channels, resulting in a reduction of the payment overhead compared to Sprite and iPass.

The credit system used by Cashflow is based on public key infrastructure [Katz08]. As pictured in Figure 3.1, the system distinguishes between three roles: bank, user, and node. A node is an actual physical device, which is equipped with wireless communication technology and can therefore participated in wireless as hoc networks. Each node possesses a unique key pair: a public and a private key. Every node belongs to at least one user. In this context a user could be a real person using the node, which could

be a for instance a laptop. A more abstract kind of user could be a company. From the system's point of view, a user is somebody who wants to use a node to use services of a network or to provide services to a network to earn a reward. Each user possesses a unique key pair and additionally at least one account at a credit clearance authority, which in this context is called bank. It is assumed that nodes and their users have exchanged their public keys in a secure way, so that user and node could rely on the keys for authentication. Therefore, mutual trust exists between nodes and their users, as pictured in Figure 3.1. Mutual trust also exists between users and banks. Banks also possess key pairs, which are not only used for authentication, but also for issuing credit certificates used by the nodes for payment. Besides the issuing of credit certificates, banks provide accounts for users. A user can transfer money to the account to use this money to pay for used network services. On the other hand, when a node belonging to a user has earned a reward for provided services, the bank transfers money from the account of other users to the user account. When a user provides more services than he uses, the bank can disburse the money to the user. In short, an account in this credit system is similar to a bank account.

When a node needs a credit certificate it sends a request to its current user as pictured a step 1 in Figure 3.1. Now the user can request a new certificate for the node at the bank (step 2). The request includes the following fields:

$$\{userAccount, nodePublicKey\}_{\text{sigUser}} \tag{3.1}$$

An overview over the meaning of the fields is given in Table 3.1. As mentioned before, it is assumed that the user and they node have already exchanged their public keys, so the node does not have to send its public key again. By allowing the user to include the account number in the request, it is possible that a user has multiple accounts at the same bank. For example a user could have a private account and additionally a business account.

Assuming that the user has no outstanding debts at the bank, the account number included in the request belongs to the user, and the signature indicates that the message was not altered, the bank issues a credit certificate, which includes the following fields:

$$\{accountNr, nodePublicKey, expirationDate\}_{\text{sigBank}} \tag{3.2}$$

This certificate states that the bank accepts invoices until the expiration date, signed by the node to which the public key in the certificate belongs and that the account included in the certificate will be debit. The bank

Field	**meaning**
$\{\cdots\}_{\text{sigUser}}$	user's signature
$\{\cdots\}_{\text{sigBank}}$	bank's signature
$\{\cdots\}_{\text{sigSN}}$	signature of a channel's source node
$\{\cdots\}_{\text{sigIntNodeX}}$	signature of intermediate node X of a channel
$nodePublicKey$	public key of a node
$bankPublicKey$	bank's public key
$SNPublicKey$	source node's public key
$accountNr$	account number
$accountNrSN$	account number of a channel's source node
$accountNrINX$	account number of intermediate node X of a channel
$expirationDate$	expiration date of a certificate
$channelParameters$	channel parameters, compare with section 3.3.5
RH	Hash value of a channel request
$channelId$	unique ID of a channel
$feeIntNodeX$	fee intermediate node X of a channel charges
$statisticChannelIdX$	statistic data of channel X
$nodeListChannelIdX$	nodelist of channel X

Table 3.1: Description of data fields

keeps a copy of the request and the issued certificate, and, as step 3, it transmits the credit certificate back to the user. The user forwards in step 4 the certificate together with the public key of the bank in a signed message to the node. The complete message includes the following fields:

$$\{\{accountNr, nodePublicKey, expirationDate\}_{\text{sigBank}}, \\ bankPublicKey\}_{\text{sigUser}} \tag{3.3}$$

By verifying the user's signature, the node can prove that the message was not altered. Additionally it proves that the node has the permission from the user to use the account quoted in the credit certificate for payments. Additionally the node is allowed to accept invoices from nodes having a credit certificate signed by the bank to which the public key in the message belongs. In the basic version it is assumed, that there is only one bank in

the system. However, if there are more banks in the system, the user needs a list of the public keys from the other banks, signed by the user's bank, and additionally needs to forward this list to the nodes belonging to the users.

After all nodes have performed this initial phase, every node possesses a valid credit certificate, the permission of the user to use this certificate, and the public key of the bank so that it can accept certificates issued from this bank. It is worth noting, that a node can accept certificates signed by the bank as long as the user has not explicitly forbidden to accept certificates issued by this bank. Therefore, if the credit certificate of the node has expired, the node can nevertheless provide services to the network. Otherwise, even if a certificate is not expired, nodes might reject it if it will expire within a short time period. That is because the expiry date in the certificate states the latest moment when the bank will accept invoices signed by the node holding the certificate. Therefore, if the certificate will expire within a short period, nodes might reject the certificate, because they might have no possibility to transfer the singed invoice in the period left to the bank. It is important to note that nodes does not need to have a connection to a bank all the time. For instance, if a node's credit certificate is valid three months and nodes only accept credit certificate which expire in more than a month in the future, node only have to communicate with the bank once a month. Assuming the bank is reachable over the Internet, this means that nodes only need sporadic Internet access, as stated in assumption 11.

When a node requests a channel to another node, it includes the certificate into the signed request as pictured in step 5. Therefore a request has the following structure:

$$\{\{accountNrSN, SNPublicKey, expirationDate\}_{\text{sigBank}}, channelParameters\}_{\text{sigSN}} \tag{3.4}$$

Each intermediate node can verify that the credit certificate is valid by the signature of the bank. Additionally each intermediate receives the public key of the requesting node and its account number. Using the public key in the certificate, intermediate nodes can also verify that the request belongs to the owner of the certificate. After verifying the request, each node calculates a hash value RH (request hash) out the request using a cryptographic hash function like SHA-512/384 [FIP02]:

$$\{\{accountNrSN, SNPublicKey, expirationDate\}_{\text{sigBank}}, channelParameters\}_{\text{sigSN}} \Rightarrow RH \tag{3.5}$$

When the request reaches the destination, the destination answers to the request by sending a response message (step 6). Each intermediate node inserts a signed field, including the hash value of the credit certificate, the account number of the source node, the unique channel ID, which is part of the channel's parameters, and the fee it charges for the requested channel. Consequently, the source node receives a message containing the following information:

$$\begin{aligned} &\{RH, accountNrSN, channelId, feeIntNode1\}_{\text{sigIntNode1}}, \\ &\{RH, accountNrSN, channelId, feeIntNode2\}_{\text{sigIntNode2}}, \\ &\{RH, accountNrSN, channelId, feeIntNode3\}_{\text{sigIntNode3}}, \\ &\qquad \ldots \end{aligned} \tag{3.6}$$

The source node can now compute the total cost of the channel by summing up all the fees included in the response. By using the hash value RH, the source node can verify that the fee corresponds to the original request. The replacement of the original request by the hash value is done to reduce the amount of data, which has to be transferred to open a channel. The source node cannot verify all the signatures of the intermediate nodes, but this is not required. The signatures of the nodes are required, when nodes transmit invoices to the bank, to verify that nodes only book invoices belonging to them. The response message can be seen as offer. The source node can now reject the offer by dropping it or accept it. To accept it, it has to sign the offer of every node, resulting in the following structure:

$$\begin{aligned} &\{\{RH, accountNrSN, channelId, feeIntNode1\}_{\text{sigIntNode1}}\}_{\text{sigSN}}, \\ &\{\{RH, accountNrSN, channelId, feeIntNode2\}_{\text{sigIntNode2}}\}_{\text{sigSN}}, \\ &\{\{RH, accountNrSN, channelId, feeIntNode3\}_{\text{sigIntNode3}}\}_{\text{sigSN}}, \\ &\qquad \ldots \end{aligned} \tag{3.7}$$

This information is send back to the destination node, as it is pictured as step 7 in Figure 3.1. Each intermediate node extracts its signed offer, which can now be seen as contract between source and intermediate node or as invoice. Intermediate nodes can verify that the invoice is valid by using the source nodes signature and the public key form the corresponding credit certificate. Additionally they can prove that the original offer has not been modified by checking their signature. After the verification of the invoice, it is stored until there is a connection to the bank. Additionally, after the channel has been closed, nodes store the number of forwarded packets along the channel as statistic.

As soon as there is a connection to the bank, nodes transmit in a signed message their invoices together with the corresponding statistical data and their account number (step 8). If a node has opened a channel, it additionally sends the list of involved intermediate nodes to the bank. The statistical data and the list of intermediate nodes are used for fraud detection. Since it is not directly required for payment, even if a node, which has opened channels, refuses to transmit the node list, the intermediate nodes receive their earned reward. However, it is in the interest of each innocent node to detect malicious nodes and therefore nodes are motivated to provide this data to the bank.

In short, the bank receives messages from nodes with the following fields, in this example from the intermediate node 1:

$$
\begin{gathered}
\{ \\
\{\{RH1, accountNrSN, channelId1, feeIntNode1\}_{\text{sigIntNode1}}\}_{\text{sigSN}}, \\
statisticChannelId1, \\
\{\{RH2, accountNrSN, channelId2, feeIntNode1\}_{\text{sigIntNode1}}\}_{\text{sigSN}}, \\
statisticChannelId2, \\
\dots \\
channelId9, nodeListChannelId9, \\
\dots \\
accountNrIN1 \\
\}_{\text{sigIntNode1}}
\end{gathered}
\tag{3.8}
$$

In this example, the message includes invoices and statistical data of two channels from the same source. Additionally it includes the node list from one channel opened by the node itself. When the bank receives a signed message containing an invoice it first verifies that the message has not been manipulated by checking the signature. This can be done since the bank has received the public keys of the nodes when it issued credit certificates. As mentioned before, the bank keeps copies of issued credit certificates and therefore can verify that it has issued a credit certificate to $IntNode1$ with the account number $accountNrIN1$. Since it is possible that a bank has issued a number of credit certificates to the same node with different account numbers, the number of the account, to which the bank should book the earned fee, has to be included in this message. Then the bank extracts the statistical data and uses the channel IDs to relate the statistical data to a channel until all nodes have transmitted their invoices related to the channel.

For the actual payment, the bank verifies that the invoice, the request hash value, the account number used by the source node, and the fee, has been signed by both, source and intermediate node. If this is the case and the invoice has not been handed in before, the bank transfers the fee form the source nodes account, $accountNrSN$, to the account of the intermediate node, $accountNrIN1$. Therewith the payment transaction is finished.

Summarizing the credit systems provides a flexible way to perform the payment. Users could have multiple accounts and could use multiple nodes. It is also possible that a number of different users use a node. In the described system, only one bank is assumed. However, the system can be extended to support multiple banks. This could be done by implementing a web of thrust between the banks and by providing the public keys of all banks to the nodes. Like in other payment systems, security is an important issue. Therefore, this topic is discussed together with other security considerations in Section 3.4 after the detailed description of Cashflow's node architecture.

3.2.4 Combining the concepts

After the description of the different concepts used by Cashflow, this section briefly introduces the interplay between all the components by providing an usage scenario. To enable nodes to participate in ad hoc networks using Cashflow, they first must acquire a certificate from the bank. As soon as the bank has issued the certificate, nodes can participate in ad hoc networks without having a connection to the bank, until their certificates expire. All nodes within an ad hoc network periodically update the basic fee they charge for forwarding, based on supply and demand. Additionally, they monitor the quality of wireless links to other nodes.

When a node needs to transmit data to another node, which is not a direct neighbor, it might have to perform a route search to find the cheapest route to the target node, whereby restrictions concerning link quality must be considered. For the detection of the cheapest route, the route discovery algorithm not only considers the basic fee nodes charge but also penalties caused by the context of intermediate nodes. Provided the target node is part of the network, the route search results in at least one path to the target node. Additionally, the node receives information about the price level of different nodes along discovered routes, which they can compare with prices received from previous route discoveries.

Using routing information, a node can request channels along discovered paths. The request results in an offer for the channel or a reject, if the

channel would overload nodes along the path. The requesting node can decide to either accept the channel or reject it, if the fee is too high. When the channel is open, the source node can exchange data with the target node until the channel expires. However, the source node can request an extension of the channel, if it wants to continue to exchange data. For charging, intermediate nodes keep receipts of the channels. As soon as a node can connect to the bank server, it transmits its receipts to transfer the earned fees to its account. Additionally it transmits statistical data, which is used for the detection of malign nodes. Depending on the balance of the bank account, the user might have to transfer money from an external source to the bank account, or the bank disburses the money to the user. This is important, since the bank will not issue new certificates to user's nodes after the expiration of old certificates, if the user's account has been overdrawn.

3.3 Node architecture

The following section describes the node architecture in detail, including the interfaces and functionality of the different modules, and provides a number of algorithms for the realization of the modules.

3.3.1 Overview

The core functionality of Cashflow is implemented into the nodes. Nodes implement Cashflow's route discovery algorithm, forwarding and pricing functionality, collect statistical information, assess links, handle payment, and provide interfaces for users and applications. To realize all the described functionality, Cashflow uses a modular architecture, where the different functions are realized in different modules. The benefit of this modular design lies in its flexibility, since single modules can be exchanged without altering others. Therefore, it is for instance possible that if a node's radio interface already provides link assessment functionality, Cashflow can use it by using a statistic module, optimized for the specific radio interface. Another example would be the pricing strategy. By exchanging the pricing module, a node can use different pricing strategies, depending on its situation or preferences.

Figure 3.2 gives an overview over the architecture and visualizes a number of other components Cashflow interacts with. From the viewpoint of external applications, Cashflow's functionality is wrapped in the shell module. The main purpose of the shell module is to provide interfaces between Cash-

flow's modules and external applications and functions. Therefore, it can be seen as a kind of container, containing all the other modules of Cashflow. The shell module includes six other modules: Pricing module, safe module, statistics module, controller module, routing module, and scheduling module.

The pricing module provides the pricing functionality. As stated before, by exchanging this module, the pricing strategy can be altered. This module calculates the fee a node charges for a specific channel request and keeps the basic fee up to date.

The safe module does the local accounting, including the collection of receipts, balancing, and communication with the bank. The safe module also stores the node's credit certificate, the node's keys, and foreign public keys. It is also used for signing messages. Therefore, the safe module implements all safety critical functionality needed for payment.

The statistics module assesses the link quality to other nodes. It categorizes links based on signal strength, error rate, and duration. This statistical information is used by the routing algorithm as well as by the pricing function.

The controller module provides functionality to open channels to other nodes and handles channel requests from other nodes. For each channel, the controller module starts a channel controller instance, which handles and maintains a single channel.

To provide the needed routing functionality, the routing module was introduced into the shell. It performs and handles route requests and stores routing information. Additionally it transmits control data along paths outside of channels. This functionality is needed for the establishing of channels.

The last module is the forwarder module. It is used for data forwarding along channels. Since data is not immediately forwarded but delayed in the case that a node tries to transmit more data along a channel than the channel is able to transport, this module includes also scheduling functionality.

As stated before, the shell module needs to communicate with a number of external modules and components, and therefore provides a number of interfaces for these modules. The first external component is the configuration application. The configuration application provides an interface for the user for configuring Cashflow. The next external component considered by Cashflow is the control application. The main functionality of the control application is to request channels with specific parameters from Cashflow to other nodes and forward data over these channels. Besides the size and

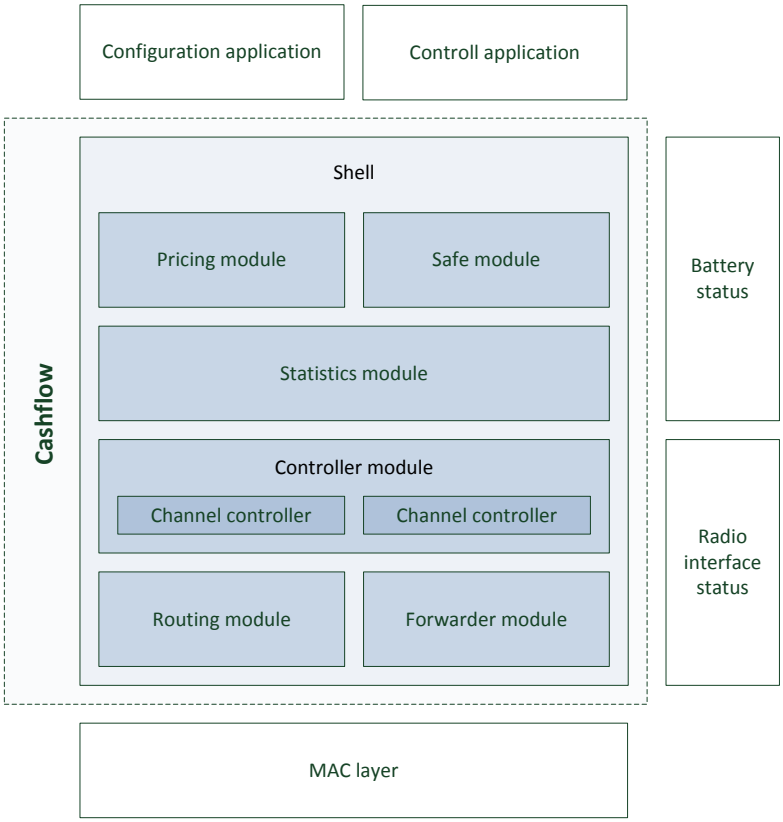


Figure 3.2: Overview of Cashflow's node architecture

the target node of a channel, the parameter includes also information about the maximum fee Cashflow is allowed to pay for the requested channel. The control application itself gets the data from higher layers. An example of such a control application is described in Subsection 4.3.1

The next external component is the node's energy supply. To include energy information into the pricing process, Cashflow needs data about the energy source and in the case that a finite source, like a battery, is used, also the charging level of the source. Another external component is the node's radio interface. From the radio interface, Cashflow can obtain information about the signal quality of a link, as well as the used transmission speed. The last external component in the given configuration is the MAC-interface. Using information provided by the MAC-layer, Cashflow estimates the usage of the shared medium. Additionally, Cashflow forwards frames to the MAC-layer for transmission because in the given configuration Cashflow is implemented

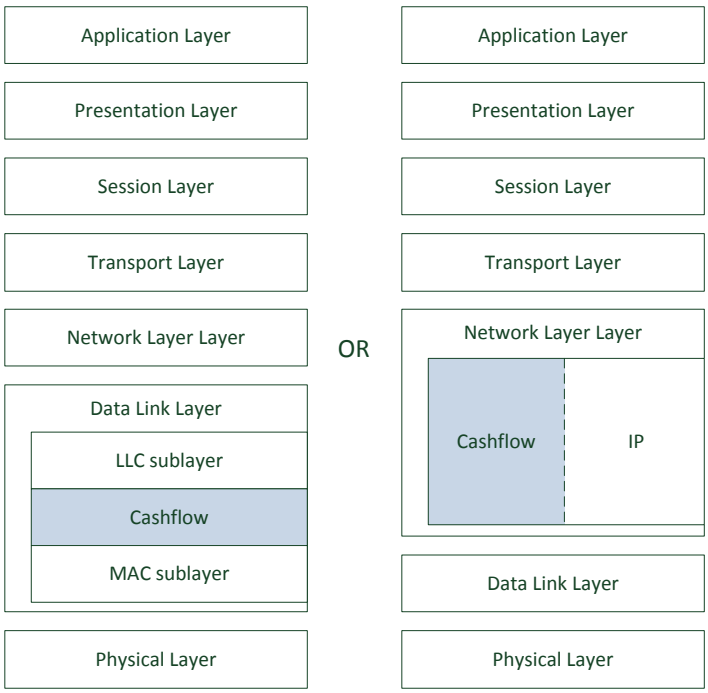


Figure 3.3: Comparison of layer 2 and 3 realization of Cashflow

directly in layer 2. If a layer 3 implementation in combination with IP would be used, Cashflow additionally would has to interact with the IP-layer.

As pictured in Figure 3.3, Cashflow could operate at layer 3 or layer 2. A layer 3 implementation could for instance be realized, by extending the IP-protocol by Cashflow's functionality. This approach would be similar to the approach presented in RFC4728 [Johnson07] for dynamic source routing. Using this approach, it is assumed, that to every node of the network an unique IP-address has been assigned. The routing protocol provides in this scenario the IP-address of the next node, which then gets translated for forwarding to the MAC-address of the next-hop node.

Using an implementation on layer 2, nodes would not act as router but as virtual switch. In this case, the routing algorithm provides the MAC-address of the next-hop node directly. From the view point of layer 3, all nodes of the network appear link-local and therefore the switching is completely transparent to layer 3. The benefit of this approach is, that every layer 3 protocol could directly use such a network. Additionally, there is no need for address translation, like from IP- to MAC- addresses,

at every node. Because of these benefits, the newest version of the routing protocol B.A.T.M.A.N [Johnson08] will be implemented in layer 2. Also for the design of Cashflow, a realization of layer 2 was assumed. Besides the already given benefits of a layer 2 realization, this approach allows an effective implementation of virtual channels by using label switching, as it will be described later.

Whether Cashflow is realized on layer 2 or 3 has mainly an influence on the interfaces of the shell module. Only some internal modules might have to be altered for optimization. However, the basic algorithms, like the route discovery process, stay the same. Hence, for the description and explanation of the routing process, the nomenclature of layer 3 is used. This is done due to didactical reasons, since most routing algorithms for mobile ad hoc networks are described using the layer 3 nomenclature.

To provide its routing and forwarding functionality, Cashflow inserts and additional header into frames, as pictured in Figure 3.4. In the top part of the figure an IEEE802.11 frame is shown using LLC in combination with SNAP for encapsulation packets of layer 3. Cashflow adds an additional header directly before the LLC protocol data unit for forwarding packets of higher layers. Additionally, there exist a number of frames for internal functions like route discovery and channel control. These frames will be explained together with the modules using them.

The basic protocol data unit of Cashflow is shown in Figure 3.5. It consists of a 1 bit sized type field and a data field carrying ether a routing or forwarding protocol data unit. The bit in the type field signalizes, if the protocol data unit included in the data field is of the type routing or forwarding protocol unit, and therefore should be handled either by the routing or the forwarding module.

3.3.2 Shell module

The shell module works as wrapper in Cashflow. It provides interfaces for external applications to use Cashflow's services and hides its internal structure. Additionally, it provides interfaces for internal modules to use functionality provided by external sources. This allows to adapt Cashflow to different environments, without changing internal modules. Only the shell module has to be adapted to the interfaces provided by external sources. For instance, the interface providing the batteries charging level might differ depending on the equipment manufacturer. Therefore, the shell has to be adapted to be compatible with different equipment. Since the internal

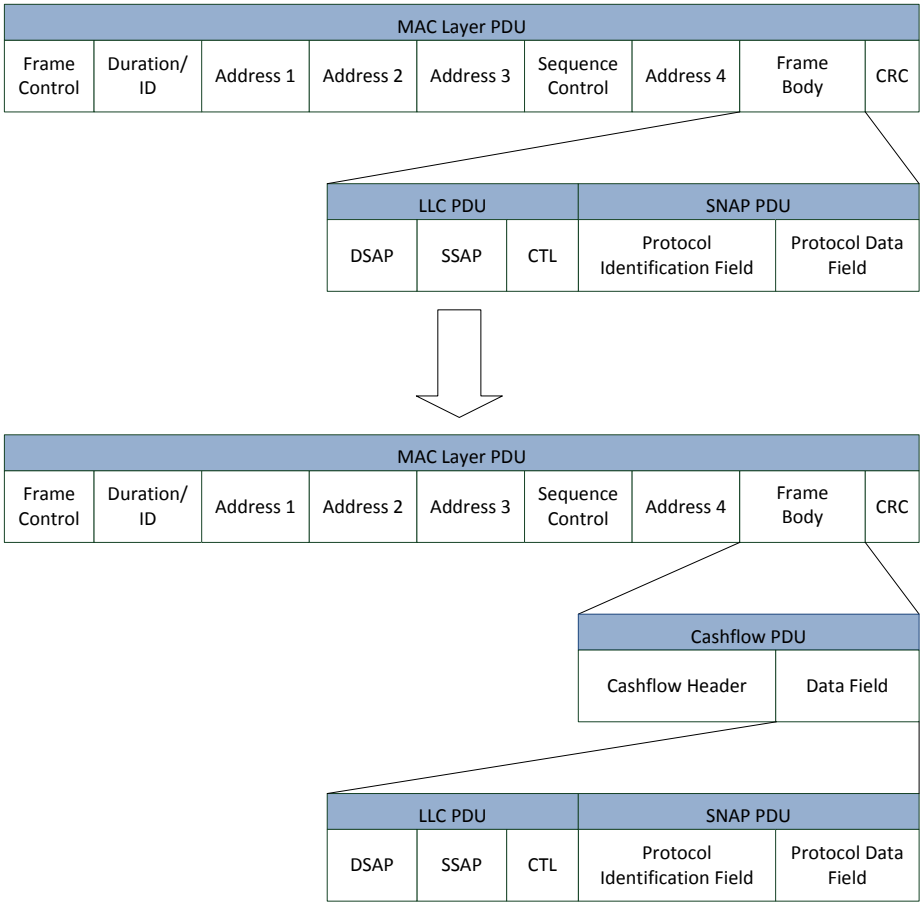


Figure 3.4: Insertion of Cashflow-PDU into MAC-PDU

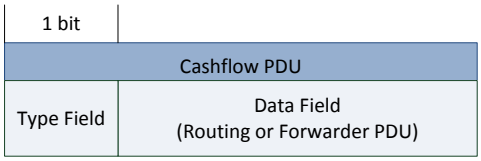


Figure 3.5: Cashflow's protocol data unit

modules request the needed information directly from the shell module, no adaption is needed. Consequently, the main functionality of the shell is to forward function calls from external to internal functions and vice versa.

Figure 3.6 gives an overview of the interfaces and functions the shell module implements. For every module, the shell provides a function, which

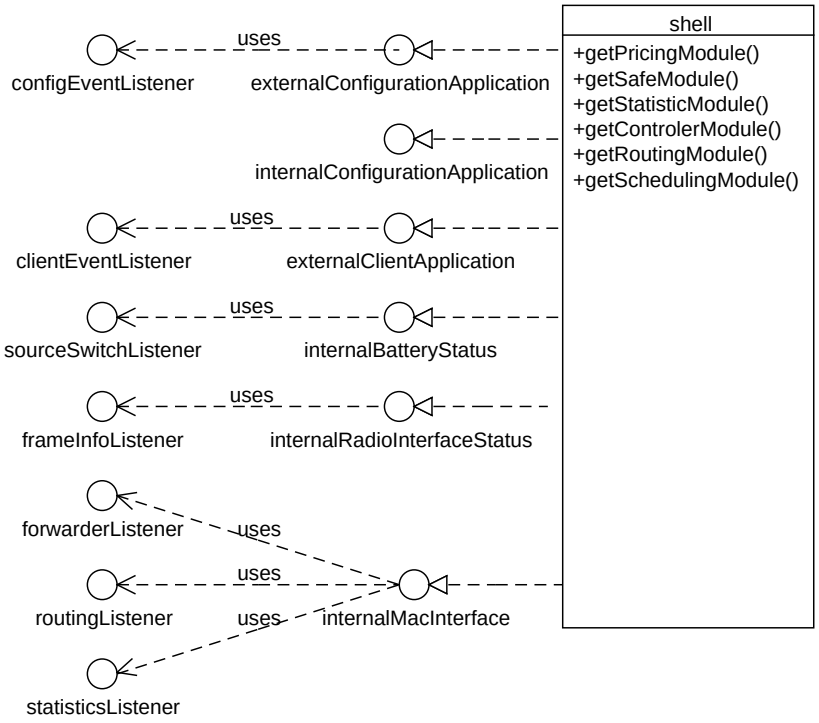


Figure 3.6: Interfaces implemented by the shell module

returns a reference to the module. This is done so that every one of the six modules can get a reference to every other module to communicate directly. For external applications, the shell offers two interfaces, namely *externalConfigurationApplication* and *externalClientApplication*. The interface *externalConfigurationApplication* was introduced to allow the development of external configuration applications. Using such an application, the user can control the behavior of the local Cashflow system, can define its participation degree, and get feedback from the system. By implementing additionally the *configEventListener* interface the configuration application can react on events triggered by Cashflow. The other external interface, *externalClientApplication*, is used for the development of applications using Cashflow. These applications request channels from Cashflow and transmit

data over these channels. They define the size and the price Cashflow is allowed to spend for specific channels. By providing sockets, these applications enable higher layers to use Cashflow. For example, an application providing a socket might receive an LLC-frame from the LLC-layer, opens a channel to the target node and transmits the frame. When the application receives frames to the same target continuously, it might opens additional channels to handle the load. The client application needs to implement an additional interface, the *clientEventListener* interface. Using the observer pattern [Gamma95] the shell uses this interface to inform applications about events like the reception of LLC-frames, the establishment of channels, or connection failures.

To allow Cashflow's internal modules to read configuration parameters provided over the *externalConfigurationApplication* interface by external applications, the shell implements the *internalConfigurationApplication* interface. Further three additional interfaces exist for the interaction with external systems. The*internalBatteryStatus* interface allows modules to request the power source type and its charging level. To react on sudden source changes, modules can implement the *sourceSwitchListener* interface. The *internalRadioInterfaceStatus* interface allows modules to register *frameInfoListener* listeners to gain information about the quality of received radio signals. The last interface is the *internalMacInterface* interface, which allows modules to interact with the MAC-layer. Additionally, it allows the registration of three types of listeners, to react on routing and forwarding frames, and to receive statistical data. These listeners are *forwardListener*, *routingListener*, and *statictics*Listener.

Figure 3.7 provides an overview over the *externalConfigurationApplication* interface. As described before, this interface is used for the configuration of different parameters of the system, as well as for monitoring the system's state. Most of the functions provided by this interface are used for storing values of parameters and read them back. By using other interfaces, internal modules can request these parameters. Other functions, providing status information, forward status requests to the responsible modules. The following list specifies the interface's functions:

setPrefThroughput(throughput)
: Using this function, a user can specify the preferred throughput rate for the node. This value is used for the price calculation, to adapt the node's fee according to the difference of preferred and actual throughput.

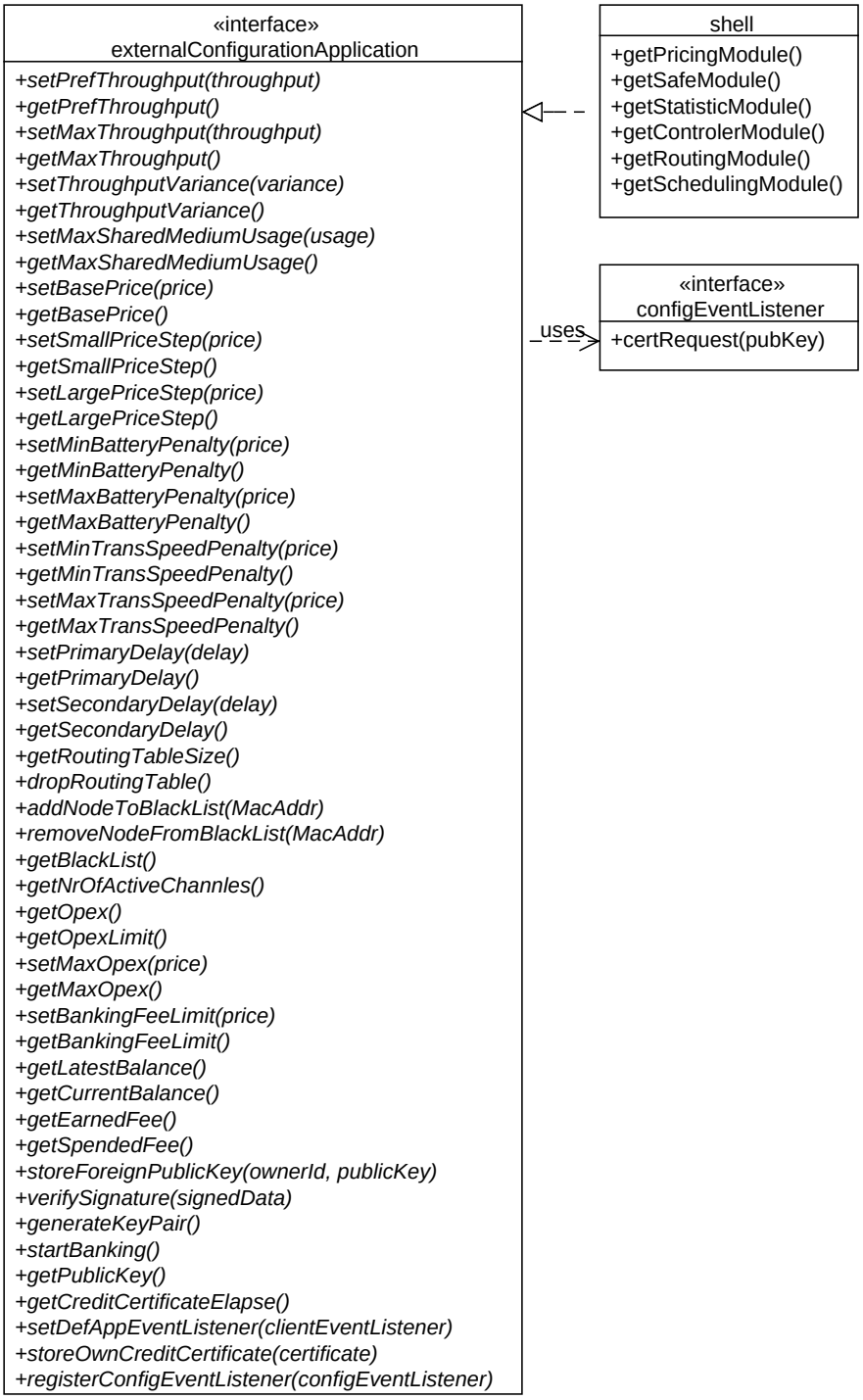


Figure 3.7: Diagram of the externalConfigurationApplication interface

getPrefThroughput()
This function returns the value set by *setPrefThroughput(throughput)*.

setMaxThroughput(throughput)
This function is used to set the maximum throughput rate. If a node receives a channel request, which would results in an overpass of the maximum throughput, the system rejects the request.

getMaxThroughput()
This function returns the value set by *setMaxThroughput(throughput)*.

setThroughputVariance(variance)
The value set by this function is used by the pricing function. It is used to define the variance of the throughput the pricing function accepts without adapting the fee.

getThroughputVariance()
This function returns the value set by *getThroughputVariance(throughput)*.

setMaxSharedMediumUsage(usage)
Using this function, the user can define the maximum usage of the shared medium. The system estimates the impact of every requested channel on the shared medium's usage. If a requested channel would result in an usage of the shared medium exceeding the usage ratio defined through this function, the channel gets rejected by the system. This is done to prevent overload situations. Additionally, a channel reject event influences the pricing.

getMaxSharedMediumUsage()
This function returns the value set by *setMaxSharedMediumUsage(usage)*.

setBasePrice(price)
With this function, the user can specify the minimum fee a node charges for packet forwarding.

getBasePrice()
This function returns the value set by *setBasePrice(price)*.

setSmallPriceStep(price)
The value provided by this function defines the adaption rate of the fee if the throughput over or underpasses the preferred throughput.

getSmallPriceStep()
This function returns the value set by *getSmallPriceStep(price)*.

setLargePriceStep(price)
This function is used to define the adaption rate of the fee if an overload situation occurs.

getLargePriceStep()
This function returns the value set by *setLargePriceStep(price)*.

setMinBatteryPenalty(price)
This function defines the minimum penalty added if the node runs on battery.

getMinBatteryPenalty()
This function returns the value of the minimum penalty charged if the node runs on battery.

setMaxBatteryPenalty(price)
Using this function, the user can define the maximum penalty charged if the node runs on battery.

getMaxBatteryPenalty()
This function returns the maximum penalty a node charges if it runs on battery.

setMinTransSpeedPenalty(price)
Using this function, the user can specify the minimum penalty charged if a link to a neighbor node uses a slow transmission rate compared to other links.

getMinTransSpeedPenalty()
This function is used to read the minimum penalty for transmission speed back.

setMaxTransSpeedPenalty(price)
This function is used for the setting of the maximum penalty charged for links with a slow transmission rate.

getMaxTransSpeedPenalty()
This function allows to read back the value set by *setMaxTransSpeedPenalty(price)*.

setPrimaryDelay(delay)
The parameter set by this function is used for the route discovery process to change the racing conditions depending on the fee nodes charge or the quality of links.

getPrimaryDelay()
This function allows to read back the value of the delay set by *setPrimaryDelay(delay)*.

setSecondaryDelay(delay)
Using this function, the user can specify the secondary delay parameter, used by the routing algorithm.

getSecondaryDelay ()
This function returns the value of the secondary delay parameter.

getRoutingTableSize()
This function requests the size of the routing table from the routing module and returns the value.

dropRoutingTable()
Using this function, a user can enforce a reset of the routing table in the routing module. The shell module calls the corresponding function of the routing module.

addNodeToBlackList(MacAddr)
This function is forwarded from the shell to the routing module. It is used to add nodes to the blacklist managed by the routing modules. Cashflow avoids routing over nodes listed in the blacklist.

removeNodeFromBlackList(MacAddr)
This function is used to remove an entry from the blacklist. The actual functionality is implemented by the routing module.

getBlackList()
This functions returns the current blacklist stored in the routing module.

getNrOfActiveChannles()
This function returns the number of active channels from the controller module.

getOpex()
This function returns the current operation costs of the open channels from the controller module.

getOpexLimit()
This function returns the value of the maximum fee the system is allowed to spend for currently open channels.

setMaxOpex(price)
With this function, the user can specify the maximum fee Cashflow is allowed to spend for channels per time unit.

getMaxOpex()
This function returns the value set by *setMaxOpex(price)*.

setBankingFeeLimit(price)
Using this function, the user can specify the maximum price a node is allowed to spend for connections to the bank. If there is a connection the bank available, but the channels fee overpasses the limit set by this function, the transmission to the bank is rescheduled.

getBankingFeeLimit()
This function returns the value set by *setBankingFeeLimit(price)*.

getLatestBalance()
This function returns information about the balance of the bank account from the moment, when the node had the last connection to the bank. The safe module implements the actual functionality.

getCurrentBalance()
This function returns the extrapolated account balance including changes since the last connection to the bank. The actual functionality is implemented into the safe module.

getEarnedFee()
This function returns the fee earned since the last connection to the bank. The actual functionality is implemented by the safe module.

getSpendedFee()
This function returns the fee spent by the node since the last connection to the bank. The actual functionality is implemented by the safe module.

storeForeignPublicKey(ownerId, publicKey)
This function allows to store of a public keys. The keys are managed by the safe module.

verifySignature(signedData)
This function allows to verify the validity of signatures using the corresponding function of the safe module.

generateKeyPair()
This function causes a new generation of a key pair by the safe module.

startBanking()
: This function enforces the safe module to open a connection to the bank and to perform banking.

getPublicKey()
: This function returns the public key managed by the safe module.

getCreditCertificateElapse()
: This function requests the time until the node's current credit certificate elapses from the safe module.

setDefAppEventListener(clientEventListener)
: Using this function, the controller module can register a client module as default application listener. When a channel is remotely opened due to a channel request of another node, Cashflow informs this listener about the new channel and additionally forwards received data to it. However, using the shell's *externalClientApplication* interface, the client application can register other listeners to handle channels.

storeOwnCreditCertificate(certificate)
: Using this function, the controller module can pass a new credit certificate to Cashflow, which get internally forwarded to the safe module.

registerConfigEventListener(configEventListener)
: With this function, it is possible to register configuration event listeners. Using these listeners, Cashflow can actively inform the control application about relevant events.

Figure 3.7 additionally presents the *configEventListener* interface. This interface only possesses one function, *certRequest(pubKey)*. Using this function, Cashflow's safe module can request a new credit certificate from the configuration application. Using the public key passed to the function, the configuration application can request the credit certificate from the bank and afterwards pass it back to Cashflow using the *storeOwnCreditCertificate* function.

The second external interface implemented by the shell module is the *externalClientApplication* interface. Figure 3.8 gives an overview over the functions it provides. It is assumed that client application using this interface implement additionally the *applicationEventListener*, interface, pictured in the same figure, to react on channel and routing events. The functions provided by the *externalClientApplication* interface can be divided into three

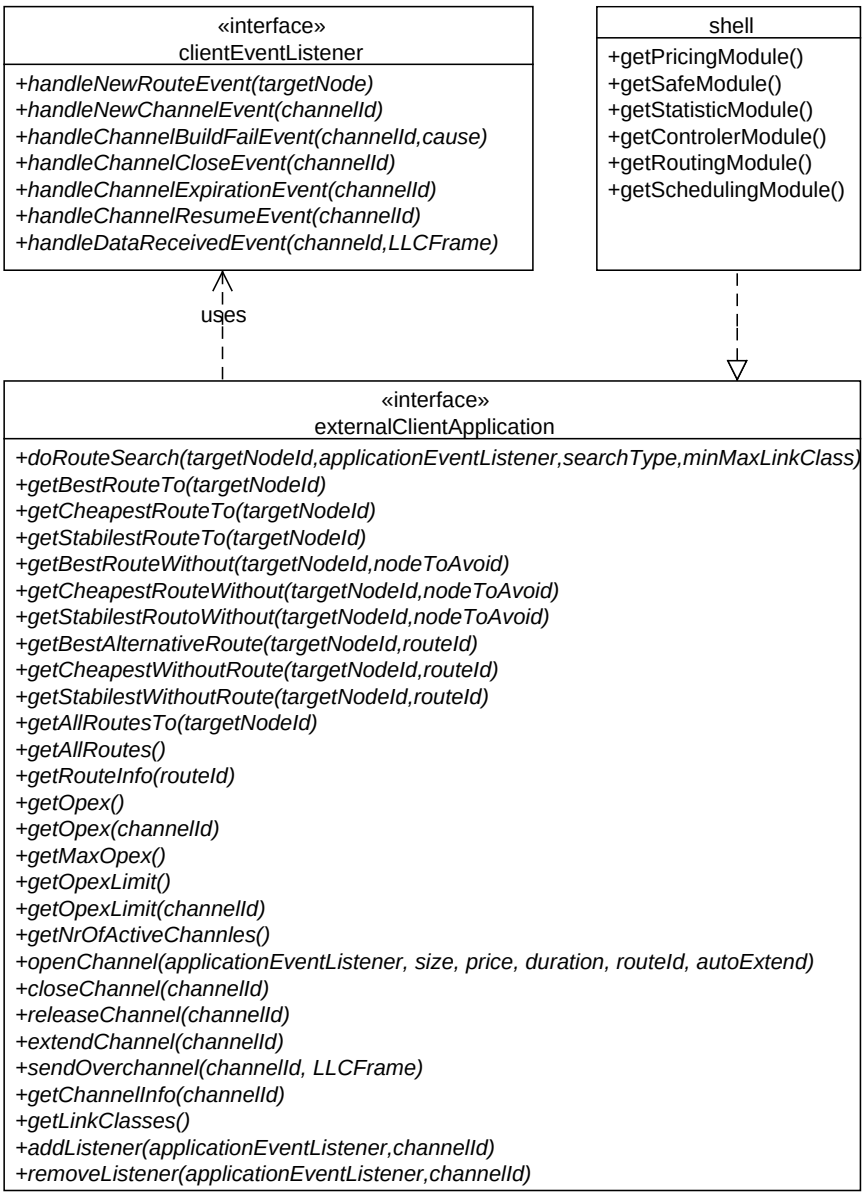


Figure 3.8: Diagram of the externalClientApplication interface

categories. The first class of functions is for interactions with the routing module. The shell module forwards calls for these functions directly to the corresponding functions of the routing module. These functions allow triggering of route searches as well as for route selection. The second class of functions interacts with the controller module. Consequently, functions belonging to this category are internally executed by the same named functions of the controller module. These functions allow the opening and closing of channels as well as the transmission of data over open channels. The last class of functions is the class of status functions. Again, the actual functionality of these functions is not implemented by the shell module, but by different internal modules. These functions allows to request information about the number of open channels, price relevant information, like the maximum price allowed to spend by the system, and information about the fee currently spend by Cashflow for open channels. The following list describes the functions provided by the interface:

doRouteSearch(targetNodeId,applicationEventListener, searchType,minMaxLinkClass)) With this function, client application modules trigger route search to a given target node. Since it is assumed that Cashflow is implemented in layer 2, the target node will be specified by its MAC-address. However, also other addressing schemes could be used. The search type specifies, if the routing algorithm should prefer stable over cheap routes or vice versa. The last parameter specifies the maximal relevant stability if the search type indicates the search of stable paths. Links, belonging to more stable classes than indicated by this factor, are threaded as if they belong to the indicated stability class. If the function is used for the detection of the cheapest path, the last parameter is used to indicate the minimum link quality the route search should consider meaning that the algorithm tries to find the cheapest path including only links belonging to the same or a better link class. If a route to the target node is found, the object passed as listener will get informed using a call of the *handleNewRouteEvent(targetNode)* function, which will be described later in more detail. Like for other functions in the context of the routing, the function-call is forwarded to the corresponding function of the routing module by the shell module.

getBestRouteTo(targetNodeId)
This function is used to request the best route to the specified target node from the routing table, managed by the routing module. In this context, the best route is defined as the cheapest route that fulfills

certain stability criteria.

getCheapestRouteTo(targetNodeId)
This function returns the cheapest route to a target node as known to the node, without considering stability.

getStabilestRouteTo(targetNodeId)
This function returns the most stabile route to a target node as known to the node.

getBestRouteWithout(targetNodeId,nodeToAvoid)
This function returns the best route, as known to the node, to a target node, whereby the stated node is not part of the path.

getCheapestRouteWithout(targetNodeId,nodeToAvoid)
This function returns the cheapest route, as known to the node, to a target node, whereby the stated node is not included in the path.

getStabilestRouteWithout(targetNodeId,nodeToAvoid)
This function returns the most stable route, as known to the node, to a target node, whereby the stated node is not part of the path.

getBestAlternativeRoute(targetNodeId,routeId)
This function returns the best route, as known to the node, to a target node, whereby no node from the other route is part of the path. This functionality is needed to build channels along braided paths.

getCheapestWithoutRoute(targetNodeId,routeId)
This function returns the cheapest route to the target node, whereby no node belongs also to the given route.

getStabilestWithoutRoute(targetNodeId,routeId)
This function returns the most stabile route to the target node, whereby no node of the path belongs also to the given route.

getAllRoutesTo(targetNodeId)
This function returns a list with all known routes to a target node.

getAllRoutes()
This function returns the complete routing list as stored in the routing module.

getRouteInfo(routeId)
Using this function, a client application module can request detailed information about a specific route.

getOpex()
This function returns the operation-costs for all currently open channels. This function corresponds to the same named function as specified in the *externalConfigurationApplication* interface.

getOpex(channelId)
This function returns the current operation costs of a specific channel.

getMaxOpex()
This function returns the maximum price the system is allowed to spend for to communication. This function corresponds to the same function of the *externalConfigurationApplication* interface.

getOpexLimit()
This function returns the maximum price the system is allowed to spend for maintaining the currently open channels. This function corresponds to the same function of the *externalConfigurationApplication* interface.

getOpexLimit(channelId)
This function returns the maximum fee the system is allowed to spend for the specified channel.

getNrOfActiveChannles()
This function returns the number of channels currently open.

openChannel(applicationEventListener, size, price, routeId, duration, autoExtend) With this function, a client application can open a channel. It has to specify a channel listener for the channel, which is in most cases the client application itself, provided if it implements the *applicationEventListener* interface. The listener is needed to react on channel events. With the parameter *size* the client specifies the number of packets the channel forwards per second, which corresponds to the channel's maximum throughput. The parameter *price* specifies the maximum price the system is allowed to spend for the channel. If the channel's fee overpasses this value, the channel will be close after it elapses. However, in most cases the effective fee a node spends for a channel will lie beneath this maximum fee. The parameter *routeId* specifies the route, along which the channel should be established. Since a route defines a specific path between the node and a target node, the target node is given implicitly. With the parameter *duration* it is possible to specify the channel's duration. The duration can vary between 1 and 256 seconds. The last parameter,

autoExtend, states, if Cashflow should extend a channel after its duration time has passed automatically. Otherwise the channel gets closed after the elapse of the duration time.

closeChannel(channelId)
With this function it is possible to signalize the system to close the given channel after the elapse of the duration time.

releaseChannel(channelId)
This function closes a channel immediately, even if the node has already paid for a longer channel duration period.

extendChannel(channelId)
With this function it is possible to activate Cashflow's functionality to extend the duration of the channel automatically, if the duration time of the channel elapsed. The same behavior of the system can be achieved by setting the corresponding flag during the channel's set up.

sendOverchannel(channelId, LLCFrame)
Using this function, the client application can transmit data like LLC-frames along the channel. However, for Cashflow it makes no difference what kind of data gets transmitted. Therefore, every data type can be transmitted using this function.

getChannelInfo(channelId)
This function returns detailed information about the specified channel.

getLinkClasses()
This function returns the different link classes used by the statistics module to characterize the quality of links. The link classes are needed by the route discovery algorithm for route selection and artificial delaying.

addListener(applicationEventListener,channelId)
Using this function, it is possible to bind additional *applicationEventListener*s to a given channel. This function is needed by the client module, working as default application listener, to start and register additional application listeners to handle remotely opened channels.

removeListener(applicationEventListener,channelId)
With this function a client module can unsubscribe *applicationEventLis-*

*tener*s. Again, this function is used by the default application listener to unsubscribe itself after registering other listeners.

As already stated before, the observer pattern is used to inform client application about events. Therefore, the client application has to register an object, implementing the *clientEventListener* interface, as pictured in Figure 3.8, with the following functions:

handleNewRouteEvent(targetNode)
This function is called by the routing module, if a new path to a node was discovered.

handleNewChannelEvent(channelId)
This function is called by the controller module, if a new channel as successfully been established.

handleChannelBuildFailEvent(channelId,cause)
This function is called by the shell module, if it was not possible to open a channel. The parameter *cause* indicates the cause of the failure, for instance, if the received offer overpasses the price the client application is willing to pay.

handleChannelCloseEvent(channelId)
This function is called, if a channel was closed.

handleChannelExpirationEvent(channelId)
This function is called a short time before the channel expires so that the client application is prepared for channel closing.

handleChannelResumeEvent(channelId)
The call of this function indicates that the duration of the channel has successfully been expanded.

handleDataReceivedEvent(channeld,LLCFrame)
This function is used to pass received frames to the client application.

Besides the external interfaces, the shell module implements a number of internal interfaces to allow modules to interact with external components or access configuration data. To access configuration variables, as set by the configuration application over the *externalConfigurationApplication* interface, internal modules can use the *internalConfigurationApplication* interface. Figure 3.9 gives an overview over the functions provided by this

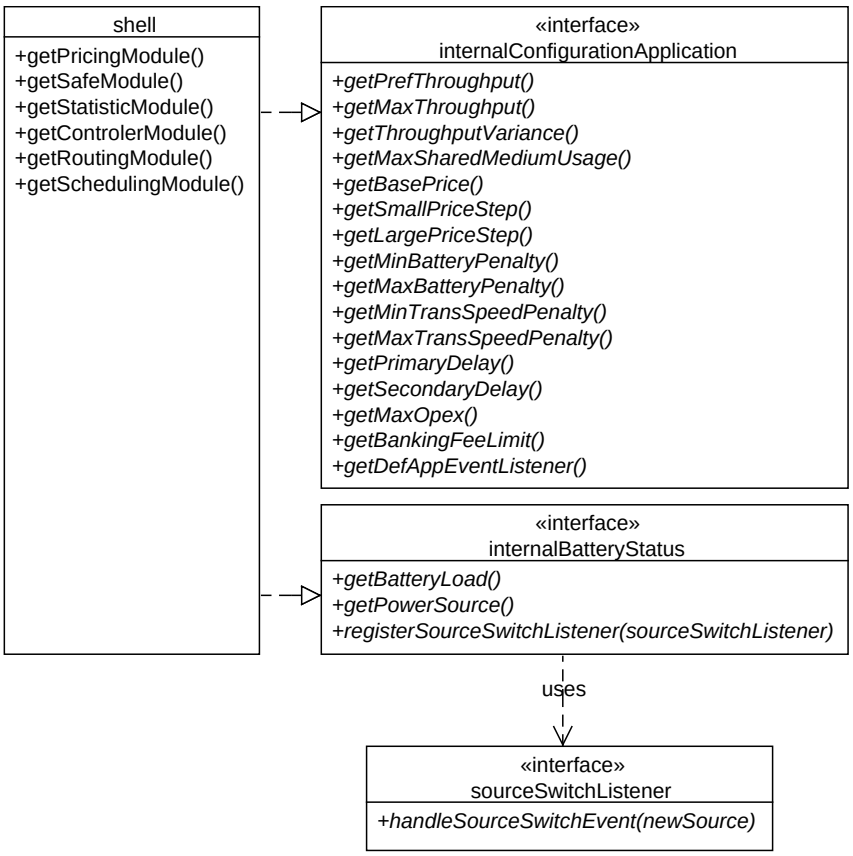


Figure 3.9: Diagram of the internalConfigurationApplication and internalBatteryStatus interface

interface. Since each of these functions is also specified in the corresponding external interface, all these functions have already been described before. Therefore, they will not be described separately.

Figure 3.9 also visualizes the *internalBatteryStatus* interface. This interface is used by internal modules to gain information about the power source and the battery's charging level. The function *getBatteryLoad()* returns a float value between 0 and 1 reflecting the battery's load. The pricing module uses this information to add an additional penalty to the fee depending on battery's charging state. This penalty gets only added, if the node uses the battery as energy source. If the node is running on AC power, no

penalty is added. To determine the power source, the interface provides the function *getPowerSource()*. Additionally, internal modules implementing the *sourceSwitchListener* interface can register them-self as listener, by using the *registerSourceSwitchListener(sourceSwitchListener)* function. If the source changes from battery to AC power or vice versa, the textitsourceSwitchListener's *handleSourceSwitchEvent(newSource)* function is called by the shell module to inform internal modules about the occurred source switch event. The parameter *newSource* of the handling function provides information about the new power source.

Figure 3.10 pictures the *internalRadioInterfaceStatus* interface and the *internalMacInterface* interface implemented by the shell module. Using the *internalRadioInterfaceStatus* interface, internal modules can gain information about the physical radio interface, by calling the function *getInformation*. In addition, internal modules implementing the *frameInfoListener* interface can register them self as listener using the function *registerFrameInfoListerner(frameInfoListener)*. When the physical interface receives a frame, if calls the *handleFrameInfoEvent(sourceMac,Info)* function of the listener. The parameters passed to this function include the MAC-address of the transmission source as well as additional informations, like signal strength and transmission speed. The statistics module uses this information to classify links to other nodes.

The second interface shown in Figure 3.10 is the *internalMacInterface* interface. This interface provides access to the MAC-layer for internal modules. In contrast to the other presented interfaces, which forward function-calls to other functions or stores variables, the implementation of the MAC-interface provides more functionality. The interface provides two functions for sending frames, namely *sendRoutingFrame(routingFrame, targetMac)* and *sendForwarderFrame(forwarderFrame, targetMac)*. With these functions, modules are able to transmit routing- or forwarder-frames using the node's radio interface. However, these frames are not forwarded directly to the MAC-layer, but are encapsulated into Cashflow-frames, as pictured in Figure 3.5, before the Cashflow-frames are encapsulated into MAC-frames. The type field of the Cashflow-frame is set depending on the encapsulated frame: 0 for routing-frames and 1 for forwarder-frames.

When the MAC-layer forwards a received frame to the shell, the shell can use the type field of the Chashflow-frame to identify the encapsulated frame type. The shell extracts the encapsulated frame and passes it to the *handleForwarderEvent(forwarderFrame,sourceMac)* or the *handleRoutingEvent(routingFrame,sourceMac)* function of the corresponding regis-

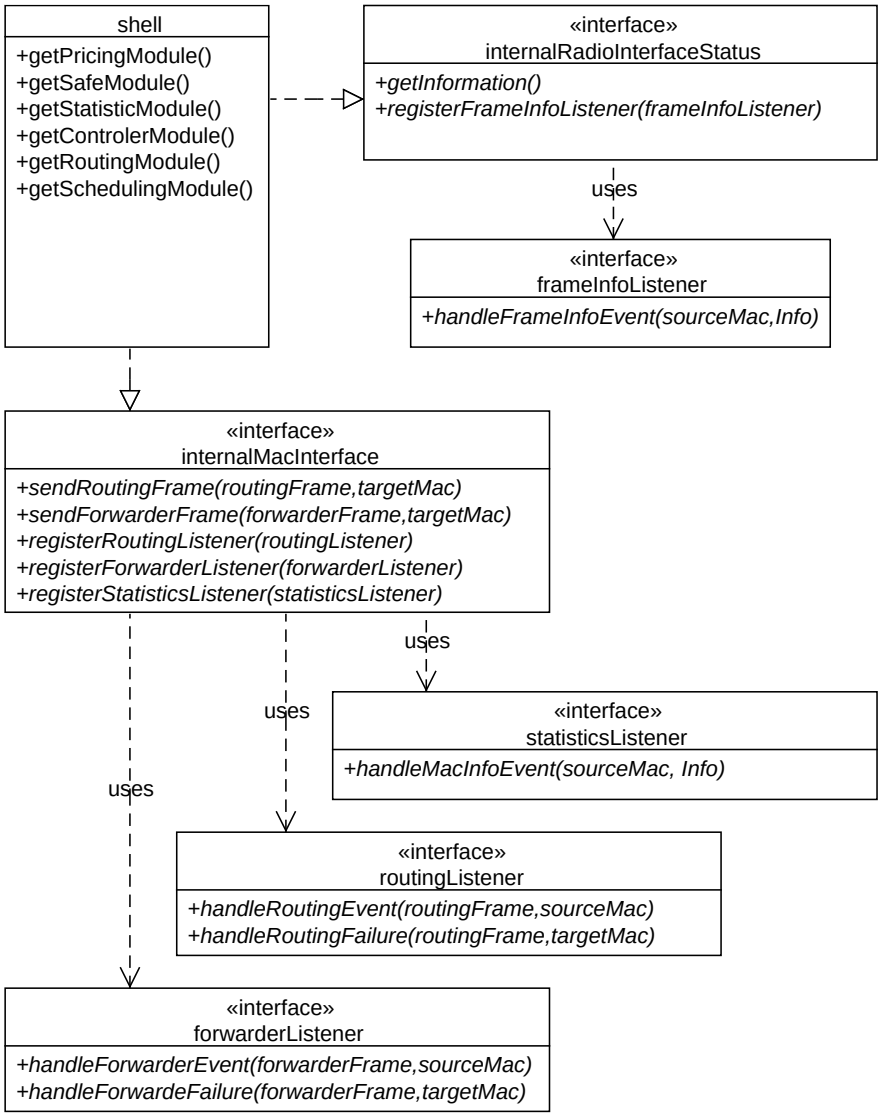


Figure 3.10: Diagram of the internalRadioInterfaceStatus interface and internalMacInterface

tered listeners, depending on the frame type. The same is done for frames returned by the MAC-layer to Cashflow, because of an occurred error. Such an error could for instance be caused by frame-loss due to transmission fail-

ures. Again, the shell analyses the type of the encapsulated frame and passes it to the *handleForwarderFailure(forwarderFrame,targetMac)* or the *handleRoutingFailure(routingFrame,targetMac)* function.

The *internalMacInterface* interface allows the registration of three different listener-types by the functions *registerRoutingListener(routingListener)*, *registerForwarderListener(forwarderListener)*, and *registerStatisticsListener(statisticsListener)*. The *routingListener* is used to handle received routing-frames or failure events caused by the transmission of routing-frames. Similar the *forwarderListener*, which is called by the shell module, if forwarder-frames have been received or have not been transmitted due to transmission failures. The *handleMacInfoEvent(sourceMac, Info)* of the *statisticsListener* is called by the shell after sending or receiving frames, to provide MAC-layer related statistical data like the number of retries needed for frame transmission.

Since the main purpose of the shell module is to act as a wrapper, no additional functionality is implemented into it. This module only provides a number of interfaces to allow Cashflow's modules to use functionality of external components on the one hand and, on the other hand, to hide the internal structure of Cashflow from external applications using the Cashflow system. The actual functionality provided by Chashflow is realized by the inner modules.

3.3.3 Routing module

The main purpose of the routing module is to maintain routing information needed for data transmission through multi-hop networks. It provides functionality to discover routes to nodes and to transmit data using routing information gained by route discovery. Besides maintaining routing information, the module also keeps records of malign nodes in a blacklist. Nodes stored in the black lists are not used by the routing module for data forwarding and no service will be provided to these nodes. This functionality allows to exclude malign nodes from the network and consequently works as motivator to prevent malign behavior.

Every routing algorithm for mobile ad hoc networks can be implemented by the routing module. However, regular routing algorithms are not designed to incorporate specific user requirements caused by the use of virtual currency systems in ad hoc networks. As stated in Section 3.1, it is assumed that users are interested to pay as little as possible for data transmission. Therefore, the routing algorithm is required to find the cheapest route to

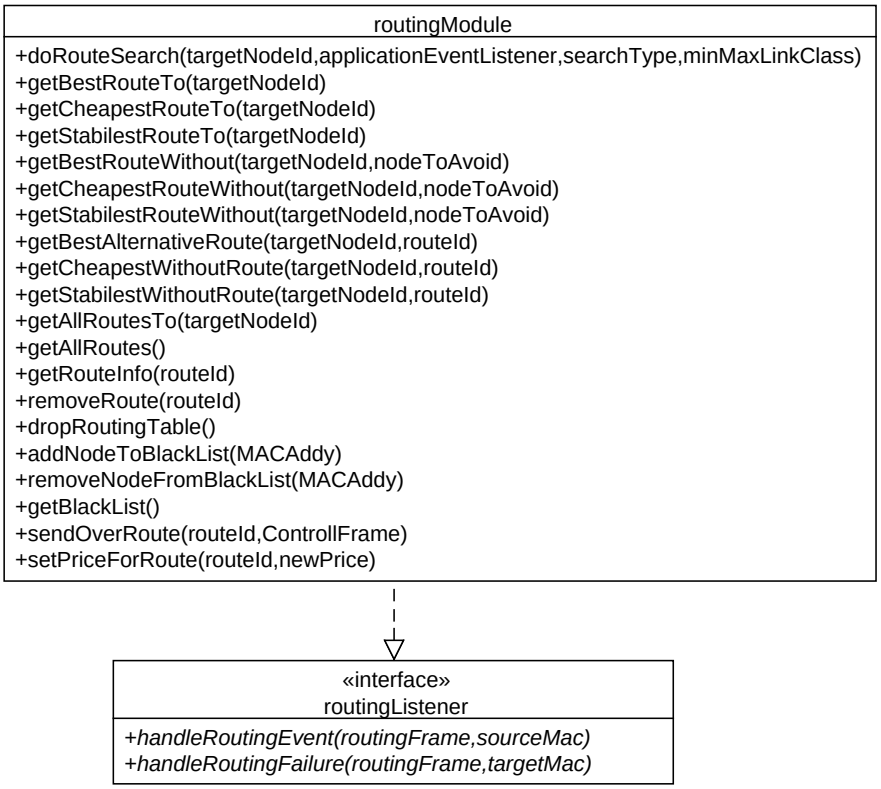


Figure 3.11: Overview over the routing module

a target node. Most routing algorithms are optimized to find the shortest route, which however might not be the cheapest one. Targeting this issue, a special reactive routing algorithm was designed to fit the needs of virtual currency systems in mobile ad hoc networks. This algorithm will be presented in this section, after the description of the routing module's functionality and interfaces.

Figure 3.11 gives an overview over the functions implemented by the routing module. With the exception of the last two functions listed in the routing module class, all functions are called by the same named functions of the *externalConfigurationApplication* or the *externalClientApplication* interface and have therefore been explained before. In short, the function *doRouteSearch(targetNodeId, applicationEventListener, searchType, minMaxLinkClass)* triggers a route search. The parameters define the target

node and the listener, whose *handleNewRouteEvent(targetNode)* function is called as soon as the routing protocol detects a new route to the specified target node. The last two parameters state if the routing algorithm should consider link quality or fee as criteria for route search and define restrictions concerning link quality. The following eleven functions shown in Figure 3.11 are used to request routes from the routing list. The function *getRouteInfo(routeId)* allows to receive detailed information for a specific route, while *removeRoute(routeId)* allows to remove routes from the routing table. If the complete routing table should be deleted, *dropRoutingTable()* provides the needed functionality. The functions *addNodeToBlackList(MACAddy)*, *removeNodeFromBlackList(MACAddy)* and *getBlackList()* allow to manipulate the black list.

All these functions are called by external applications using the corresponding interfaces. This is not true for the last two functions. The functions *sendOverRoute(routeId, ControllFrame)* and *setPriceForRoute(routeId)* are only available for Cashflow's internal modules. The first function is used by the controller module to transmit frames to other nodes to establish channels. This function encapsulates the given frame into a routing frame and forwards it along the specified path to another node. The shell module of the target node receives the routing frame and forwards it to the routing module using the *handleRoutingEvent(routingFrame, sourceMac)* function of the routing listener implemented by the routing module. If the transmission fails, the *handleRoutingFailure(routingFrame, targetMac)* of the routing listener is called, to inform the routing module about the event. The routing module reacts on this event by forwarding this information to the controller module.

The last function of the routing module, *setPriceForRoute(routeId, newPrice)*, is also used by the controller module. It is used to update the routing table if the controller module detects a change of the route's price.

As stated before, a special routing algorithm was designed, able to find the cheapest route between two nodes, whereas it considers restrictions on link stability. The algorithm belongs to the class of reactive routing algorithms. This means that the routing algorithm only searches for routes to other nodes when required. This has the drawback compared to proactive routing algorithm, that route discovery has to be performed before data transmission, if the source node knows no route to the target node. However, proactive routing protocols maintain routing-tables by flooding routing information, as soon as changes in the network topology occur. If used in mobile ad hoc networks using virtual currency systems, besides

topology information also the fees nodes charge have to be flooded, to allow nodes to use the cheapest route. Since this has to be done for every fee change, which occurs regularly, proactive routing schemed would cause a large overhead. Therefore, the reactive routing concept was chosen, since routing information is only collected if it is really needed, which results in a smaller overhead compared to proactive routing schemes.

The reactive routing algorithm, designed to find the cheapest route through the network, uses the following concept: When a node wants to communicate with another node, it broadcasts a route request packet. When receiving the request, every node except the source and the destination node, analyses if the request contains information about a new path, which is more preferable than the best path the node has learned about before. If this is the case, the node calculates a delay for the packet. The packet is broadcasted after the delay, if during the delay time no additional route request has been received with information about a more preferable route. If a routing packet including a more preferable route has been received, the delayed route request packet will be discarded and the new route request packet will be delayed accordingly. Otherwise, if the received route request contains no information about a better route, it will be discarded immediately. When the destination node receives a route request, it answers the request by sending a route reply packet immediately without any artificial delay. If the target node learns about a better route, it sends an additional route reply packet, which is the functional equivalent to the re-broadcasting function of intermediate nodes.

The main idea behind this algorithm is to delay the route request packets depending on the fee the intermediate node charges and on link quality parameters. Therefore, route request are forwarded faster along better paths, in terms of price and link quality, than along expensive or instable paths. This concept reduces the route discovery overhead. From the point of view of an intermediate node, by broadcasting a route request packet, it makes an offer to the successive nodes, to route over it, for delivering packets between source and destination node. Depending on the network topology, some intermediate nodes will receive route request packets for the same route request over different paths. When an intermediate node gets a route request packet over a cheaper or more stable path, this can be seen as a better offer, depending on the path selection criteria, from his precedent nodes. As consequence, the node can reproduce the offer it made to its successive nodes, by sending again a route request packet. It is the interest of the node and of the entire network, to reduce the number of reproduced offers, since previously broadcasted route request packets become irrelevant once

a reproduced offer was sent. As stated before, by delaying route request packets depending on the offer, better offers are forwarded faster, which decreases the probability that offers have to be reproduced. This leads to an increase of the route discovery performance.

This route discovery algorithm can be realized using concepts from different reactive routing protocols. However, the following realization of the routing protocol is based on source routing similar to the DSR-protocol [Johnson96]. Figure 3.12 gives an overview over the frames defined for Cashflow's routing module. These frames present also Cashflow's packet format if realized in layer 3 and not in layer 2 as in the presented in this thesis. For the route discovery process the request and response frames are of interest, which correspond to request and response packet format if implemented in layer 3. The routing data unit contains a subtype field, indicating the type of the routing frame. Frames used for route discovery are indicated by a subtype field set to 100 or 101. The last bit of the subtype field indicates, if the route discovery algorithm should prefer cheap routes over stabile routes or vice versa. The target node field holds the MAC-address of the target node, followed by the min/max quality field, which contains information about link quality constrains. The sequence number field is used together with the address of the route request's source as identification feature, to differentiate between concurrently route requests. The fee parameter field is used to store the cumulative fee the nodes along a route charge. Similar, the quality parameter stores information about the link quality between nodes along a path. The node counter is used for hop counting implicitly indicating the size of the node list. The last field holds all MAC-addresses of the path the frame has been forwarded along.

To perform route search, a node broadcasts a route discovery frame. At the first broadcast, the fee and the quality field is set to 0, the node count field to 1, and the node list of the frame holds the source node's MAC-address. Therefore, each node receiving a route discovery frame can determine the source, using the first MAC-address of the node list. By combining the sequence number and the source's MAC-address, each node receiving a route request can make a new entry in its route discovery table or relate the request to a previously made entry. Figure 3.13 gives an overview over the fields of the route discovery table. The first column contains the MAC-address of the route request's source. Together with the second column, holding the sequence ID of the route request, this tuple is used for the identification of the route request. The next two columns store the price and the quality parameters of the best route discovered during the route discovery process so far. The following field might contains a pointer to a

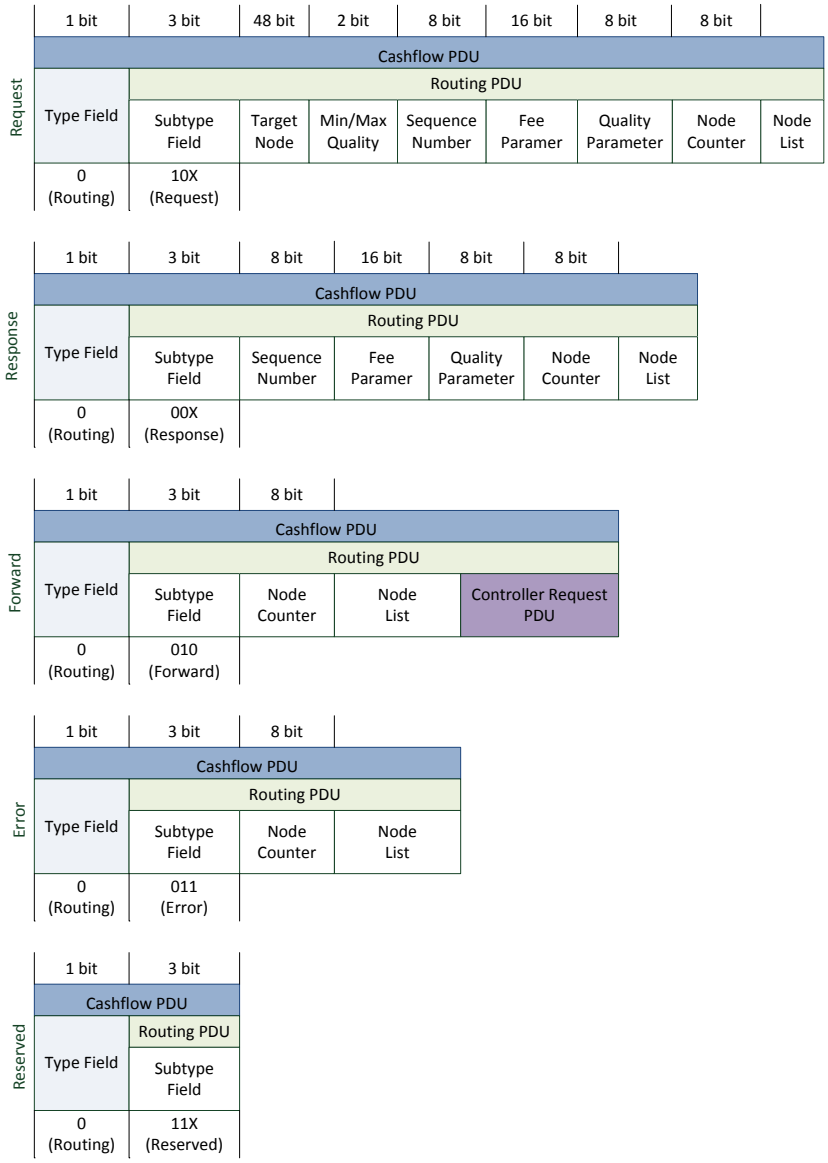


Figure 3.12: Overview over routing frames

source ID	sequence ID	best fee	best stability	pointer to packet	timestamp
00:0B:6A:5D:B6:31	134	21	10	(empty)	1251808076
48:2C:6A:1E:59:3D	146	14	9	0x401bef29	1251808078
00:0B:6A:5D:B6:34	467	23	10	0x3104608b	1251808079

Figure 3.13: Visualization of the route discovery table

route discovery frame, if the transmission of a route discovery frame is currently suspended by the artificial delay functionality. The last field contains the timestamp of the scheduled transmission time of the frame the pointer points to. The routing module implements a function, which is activated when a timestamp in the route discovery table matches the current time. This function broadcasts the route request frame to which the pointer of the corresponding entry refers. Afterwards it deletes the pointer. Additionally it deletes entries of the route discovery table where the timestamp is older than a certain value. This is done to prevent a continuous growth of the table and additionally it allows reusing sequence IDs.

When a node receives a route discovery frame, and the node is not the target node, it performs the algorithm depictured in Figure 3.14. After the reception of the frame in step 1, the algorithm searches the node list included in the route discovery frame for the own MAC-address, which would indicate that the node has already processed the currently received route discovery frame. This is done to exclude cyclic paths during route discovery. If the node has received the current route request frame before, the frame gets dropped as indicated in step 5 of the figure. If the received route discovery frame is new to the node, meaning that the frame has been forwarded along a new path to the node, the node checks in step 3, if its route discovery table already posses an entry for the route search the fame belongs to. This is done by searching for a corresponding source MAC-address and sequence ID tuple, indicating that the node has already received a route discovery frame belonging to the same route discovery process. If such an entry does not exist, the node performs step 8 of the algorithm, which will be explained later. Otherwise, is such an entry exists, the algorithm compares the fee and link quality parameters included in the route discovery frame to the values of the corresponding entry in the route discovery table, as shown in step 4. Based on these parameters, the type of route search, as indicated by the subtype field of the route discovery frame, and the link quality restrictions, the algorithm decides if the new route is better than the best previously found route. When the route search type indicates that the most stabile

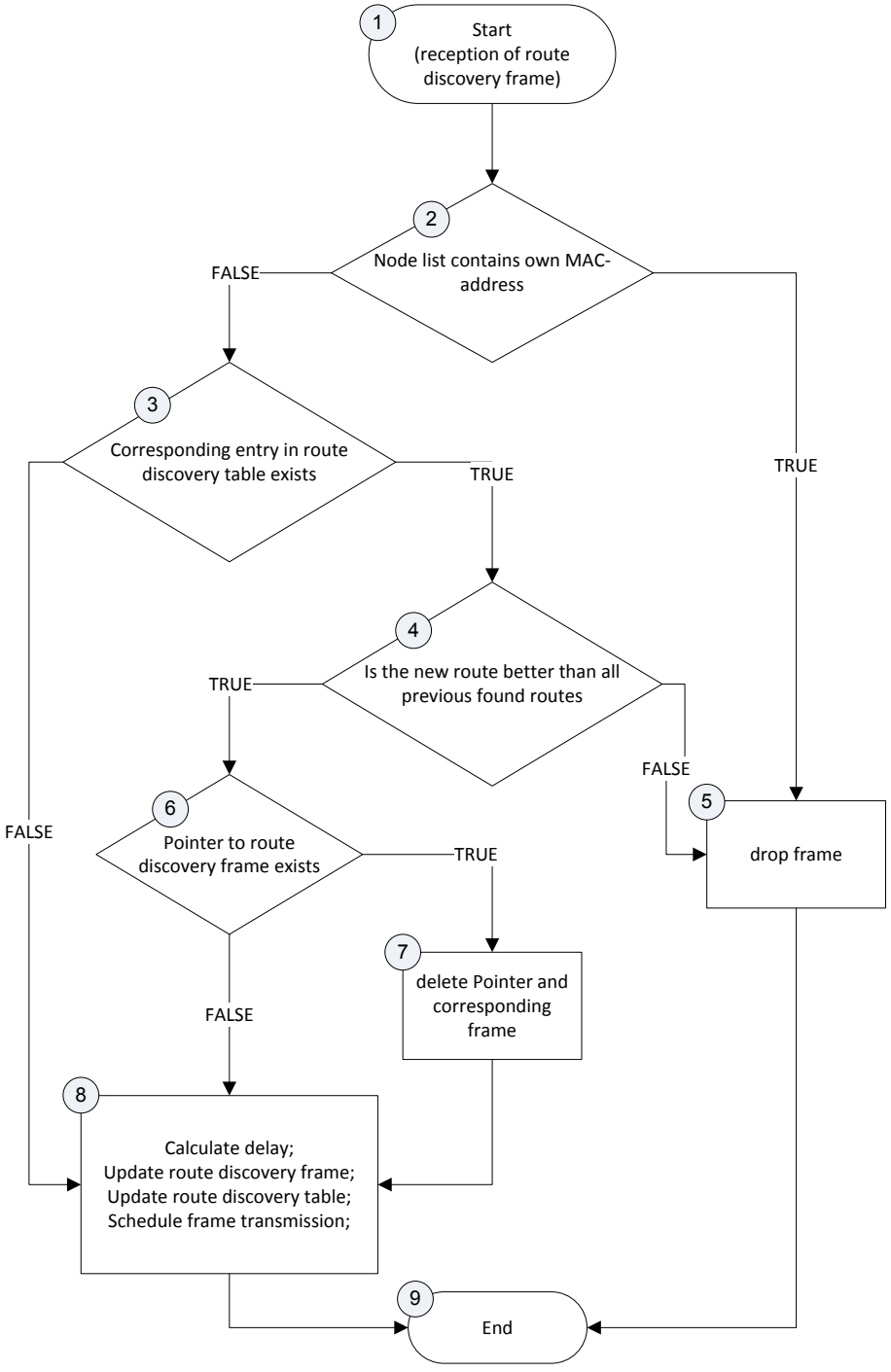


Figure 3.14: Diagram of the route discovery algorithm

route to a target is needed, a route is considered as better compared to a previously found route if the new route is more stable or if it is as stabile as the previous route but cheaper. Additional the quality restriction is considered. This is done by treating links belonging to a higher link class as indicated by the min/max quality field as if they would belong to the indicated class. If the cheapest route is preferred, a route is considered as better if it is cheaper, or, if the price is the same than the previous best route, if the stability is higher, provided that the route meets the minimum quality restrictions as indicated by the min/max field. This means that even if a route is cheaper compared to a previous found route, the previous route is kept if the cheaper route's quality is insufficient.

If the algorithm has come to the conclusion, that the new discovered route is not as good as the best route previously found, the route discovery frame gets dropped as pictured in step 4. If however the received route discovery frame includes a more preferable route, the route discovery algorithm verifies in step 6, if in the corresponding entry of the route discovery table a pointer to another route discovery frame exists. The existence of a pointer means that currently a route discovery frame is scheduled for broadcast in the future but the newly received frame contains a more preferable path. Therefore, if such a frame is scheduled, it has to be deleted as pictured in step 7 before the algorithm continues with step 8. If no such frame exists, the algorithm can directly continue with step 8.

$$df(f) = \begin{cases} D_{\text{maxFee}} & \text{if} \quad f > F_{\max} \\ \dfrac{(f - F_{\min}) \times D_{\text{maxFee}}}{F_{\max} - F_{\min}} & \text{if} \quad F_{\min} < f \leq F_{\max} \\ 0 & \text{if} \quad f \leq F_{\min} \end{cases} \tag{3.9}$$

$$dc(c) = \begin{cases} \dfrac{D_{\text{maxClass}} \times 10^{(c-1)}}{10^{(C_{\max}-1)}} & \text{if} \quad c > 1 \\ 0 & \text{if} \quad c \leq 1 \end{cases} \tag{3.10}$$

$$d(f, c) = df(f) + dc(c) \tag{3.11}$$

In step 8, the system prepares the rebroadcast of the route discovery frame. The algorithm appends the node's MAC-address to the node list the frame contains, increments the node counter field, and updates the fee and signal quality fields. After the preparation of the frame, the node has to calculate

the delay for scheduling the rebroadcast. Formula 3.9 to 3.11 describe the calculation of the route discovery delay suggested for the implementation of Cashflow. Formula 3.9 is used for the calculation of the delay caused by the fee the node charge. The parameter D_{maxFee} represents the maximum delay, which can be caused due to the fee f. If the route discovery frame indicates that the fee is the main route selection criteria, the route discovery protocol uses the function *getPrimaryDelay()* from the *internalConfigurationApplication* interface, implemented by the shell module, to determine the value of D_{maxFee}. Otherwise, if stability is the main route selection criteria, the value of the function *getSecondaryDelay()* of the same interface is used. As mentioned in Section 3.3.2, these functions are used for accessind the user configuration. The parameters F_{min} and F_{max} in Formular 3.9 are used to determine the range of fees. In other words, they are use to differentiate between high and low fees. To gain values for these parameters, the routing module calls the functions *getMinHopFee()* and *getMaxHopFee()* from the controller module. These functions return the value of the minimum and the maximum fee other nodes charge. Using all those parameters, the algorithm calculates a delay based on the fee nodes charge.

To calculate the link-quality depending delay, Formula 3.10 is used. The parameter D_{maxClass} is similar to the D_{maxFee} parameter of Formula 3.9. It determinates the maximum delay than can be caused due to the link quality. If the main routing decision parameter is the link quality, the value of D_{maxClass} is the response value of the *getPrimaryDelay()* function of the shell module, or the value of *getSecondaryDelay()*, if the fee is the main routing criteria. The parameter C_{max} holds the number of classes used by the system. The statistic module, which is used for the classification of links, provides the number on available classes over the function *getMaxLinkClasses()*. Based on the class c of the current link, the system can calculate the link quality dependent delay. The sum of the fee and the link quality dependent delay, as pictured in Formula 3.11, is the delay used for the scheduling of the frame's broadcast. It is worth noting that this is only an example for the calculation of the delay. The only requirement for the calculation of the delay is, that frames or packets, depending on the implementation layer, are delayed longer if it is expected that there exists better routes than the route included.

Using the computed delay, the algorithm calculates the moment when the frame should be broadcasted, by adding the delay to the current time. To schedule the broadcast of the frame, this timestamp is inserted into the route discovery table together with a pointer to the frame. As stated before, as soon as the current time matches a timestamp from the route

discovery table the corresponding frame gets broadcasted by the routing module. Additionally, the fields of the best price and the best link quality of the corresponding entry in the route discovery table gets updated. Then the algorithm finishes.

As mentioned before, the described algorithm is only used if the node is not the target node of a received route request frame. If the node is the target, a slightly different algorithm is used. Most of the algorithm corresponds to the algorithm shown in Figure 3.14. Only step 8 is different. In step 8 the target node generates a response frame using the information included in the received request frame. Instead of broadcasting the request frame like intermediate nodes do, the response frame is scheduled for transmission along the path the request frame has taken. After the generation of the response frame, the target node discards the request frame. Similar to intermediate nodes, which might broadcast route discovery frames for the same route discovery procedure multiple times each time including a better path, the target node might transmit multiple response frames, including different routes, back to the source node.

The structure of the response frame is shown in Figure 3.12. Like every other routing frame, the response frame possesses a sub type field, indicating that the frame is a response frame and additionally, if it is a response to a route request focusing on cheap or stable routes. The next field is the sequence number to identify to which specific route request the response frame belongs. The two following field include information about the fee and the quality of the links along the path. The last two fields store the number of hops of the path and a node list with all nodes along the path. For forwarding this type of frame, the node list is used. When a node receives a response frame, it searches for its own MAC-address in the node list and forwards the frame to the node with the MAC-address, which is stored next to the nodes own MAC-address. This mechanism is basically the same source routing mechanism as used by DSR [Johnson96]. When the route response frame finally reaches its target, the source of the route request, the information the frame contains is stored in the source node's routing table.

Figure 3.15 shows an example of a routing table. Each route entry possesses an unique ID. The ID of each route is used as return value for route request functions like *getBestRouteTo(targetNodeId)*. Additionally, the routing table contains fields indicating the target node of a route as well as fee and link quality parameters. These parameters are used to select the cheapest, the most stable, or the best route to a target, whereas in this context the

ID	target node	fee	stability	discovery	last update	last usage	nodelist
134	00:0B:6A:5D:B6:31	21	10	1251806014	1251808076	1251808413	00:0B:6A:5E:B7:54 - 08:00:69:02:01:FC - ...
135	48:2C:6A:1E:59:3D	14	9	1251806073	1251808078	1251808459	00:0B:6A:5E:B7:54 - 00:0A:34:67:EF:A2 - ...
136	00:0B:6A:5D:B6:34	23	10	12518071092	1251808079	1251808578	03:2C:76:13:39:62 - 0A:AF:F8:7E:87:30 - ...

Figure 3.15: Visualization of the routing table managed by the routing module

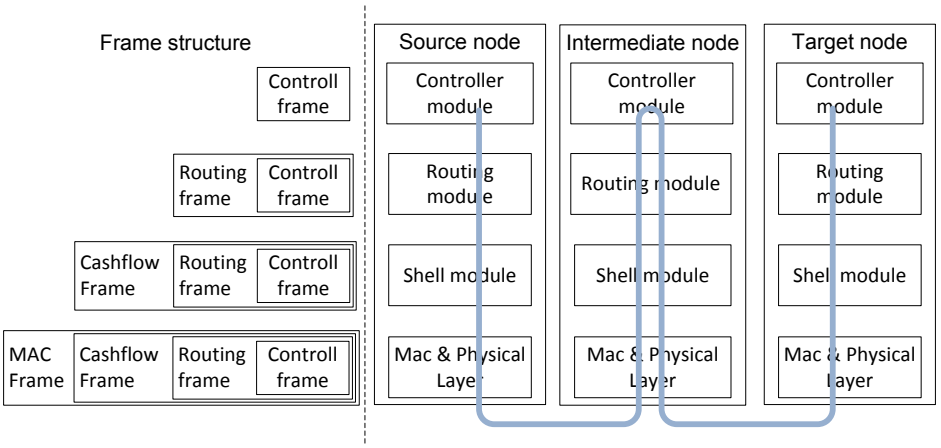


Figure 3.16: Visualization of the forwarder-packet's path

best route is defined as the cheapest route fulfilling certain quality criteria. The next three fields contain timestamps, marking the moment of route discovery, the last update of the entry, and the last usage of the route. The update field marks the moment when either the route's fee was updated by the channel module using the *setPriceForRoute* function, or when the same route was discovered again during a later route discovery process. The last field contains the actual routing information, a list of all nodes of a path.

Using the routing tables information, the routing module can forward frames for the controller module to any node of a network. This is done by using the forward-frame as pictured in Figure 3.12. This frame consists of a subtype field to identify itself as forwarder frame. Additionally, the frame includes the list of nodes of the path the frame should be forwarded along, and a field with the actual number of nodes in the node list. Since the number of nodes vary for each path, the number of nodes included in the path field is needed to determine the end of the node list and the begin of the encapsulated data. Since in Cashflow only the controller module is allowed to use this forwarding mechanism, the encapsulated data always belongs to

the controller module. It is important to note that routing modules from intermediate nodes does not directly forward forward-frames. They extract for every hop the encapsulated control frame, forwards the received frames together with information about the source and the target node of the frame, as well as information about the next and previous hop, to the controller module, by calling the controller module's *handleControllInfoOutlineEvent* function. The controller module returns an altered version of the controller frame, which is encapsulated again into the forward-frame before it gets transmitted to the next node. Only when the routing module detects that it is the target of a forward-frame, it extracts the included controller packet and passes it to the controller module without information about the next hop, since there exists no next hop. In this case, the controller module does not return a controller frame. Using this mechanism, channel modules of intermediate nodes can read and edit bypassing channel frames as visualized in Figure 3.16.

Because of the dynamic nature of mobile ad hoc networks, detected paths might break. When the routing module tries to transmit a routing frame along a broken path, the path's intermediate nodes forward the frame until it reaches a node, which is not able to forward the frame to the next hop. Using the error frame presented in Figure 3.12, the intermediate node informs the source node about the broken route. The corresponding mechanism is pictured in detail in Figure 3.17. In this picture, the controller module of the source node calls the *sendOverRoute* function of the routing module, passing a frame and the ID of the route. To keep the picture simple, parameters past to functions are not always pictured in the figure. The routing module embeds the control-frame into a forward-routing-frame and transmits it to the next node using the MAC-layer. The MAC-layer of the intermediate node receives the routing frame and forwards it to the node's routing module using the *handleRoutingEvent* function. The routing module extracts the control-frame out of the forward routing-frame and passes it to the controller module, which returns the frame after analyzing it. The routing module embeds the control-frame into a routing-frame and tries to forward it to the next hop, which in this case is the target node. In this scenario, it is assumed that the target node has moved out of the range of the intermediate node, with the consequence that the transmission is not possible. After several retries the MAC-layer gives up and returns the routing-frame to the routing module, using the *handleRoutingFailure* function to indicate that the transmission was unsuccessful. The routing module informs the controller module about this event and additionally transmits an error frame to the previous node, which in this scenario is also the source

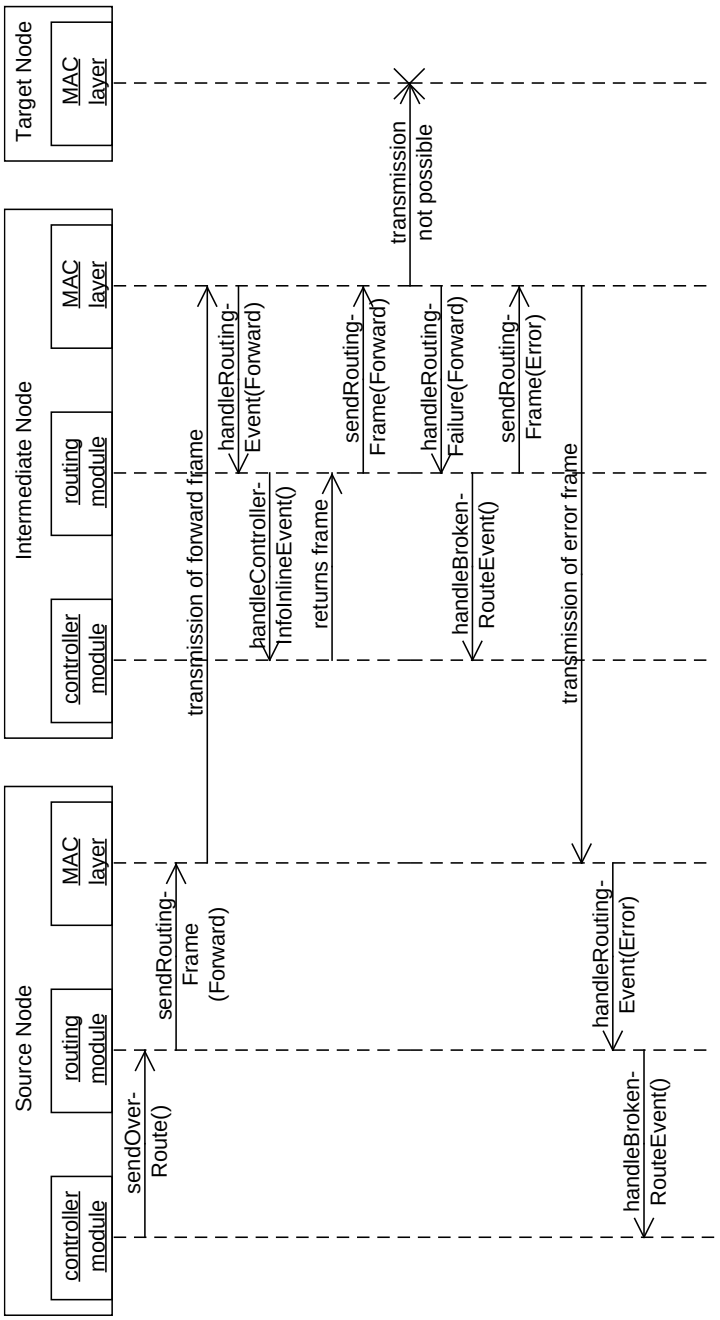


Figure 3.17: Flow diagram describing the broken route handling

node. When the MAC-layer of the source node receives the error frame, it forwards it to the routing module of the node. The routing module informs the controller module about the event and removes all routes running over the broken link from the routing table. The information, which link is broken, can be extracted from the node list of the error frame, since as the error frame's source the node, which could not be reached, is registered in the node list.

The described functionality is needed for the usage of Cashflow in mobile ad hoc networks. However, as pictured in Figure 3.12, two subtypes of routing-frames are reserved. These frames can be used to extend Cashflow, for instance to integrate it into the Internet, as it will be discussed in Section 3.5.

3.3.4 Forwarder module

The purpose of the forwarder module is to forward frames along channels with specific rates. It provides functionality to manage channels and forward frames along them. It implements label switching functionality. When receiving a frame, it uses the channel ID stored in the frame to identify the frame's next hop using a routing list. Additionally it implements a scheduler to limit the throughput along channels to previously negotiated values.

Figure 3.18 gives an overview over the functions provided by the forwarder module. The function *registerChannel(channelId, nextHop, lastHop, size)* allows to register a new channel. The channel ID is used to identify channels. It consists of the source node's ID and a channel number. The next and the last hop specify the next and the previous hop along the channels path as seen from the source node, relatively to the local node. If the local node is the source node, the previous hop value is NULL. Correspondingly, the next hop value is set to NULL at the target node. The size specifies the throughput over a channel in terms of frames per second.

If the *registerChannel* function is called, the forwarder module makes a new entry in its channel list, as pictured in Figure 3.19. It stores the ID of the channel, which consists of the channel's source and the channel number. Additionally it stores the next and the previous hop of the channel from the source node's viewpoint, and the channel's size. Using the size parameter, the module can calculate the next transmission time, as pictured in the figure, which will be explained later in detail. Additionally the module allocates two queues for the channel. These queues are used to buffer frames depending on their type, before they are transmitted to the next hop.

After a channel is registered at the forwarder module, the channel can be used for frame forwarding using the function *sendOverChannel(frame, channelId, direction, isControlFrame)*. The parameter *frame* holds the frame which should be forwarded along a channel. The *channelId* specifies the channel, which should be used for forwarding. Using the parameter *direction*, it is possible to specify the direction in which a frame should be transmitted. If the source of a channel wants to transmit a frame to the target node, it needs to send the frame in the forward direction, meaning that the forwarder module will pass the encapsulated frame to the node, which is stored as next hop in the channel list. The target node however needs to transmit frame in the backward direction to reach the source node. In this case, the forwarder module uses the node stored as previous hop in the channel list as next hop. The last parameter indicates what type of frame should be forwarded. The forwarder module distinguishes between data and control frames. The frametype influences the frame's handling. Data frames, which in the proposed solution are LLC-frames from applications using Cashflow, are embedded in forward frames, in contrast to frames of the type control, which are embedded into control frames.

Figure 3.20 gives an overview over the different kind of frames used by the forwarder module. Besides of the already mentioned forward- and control-

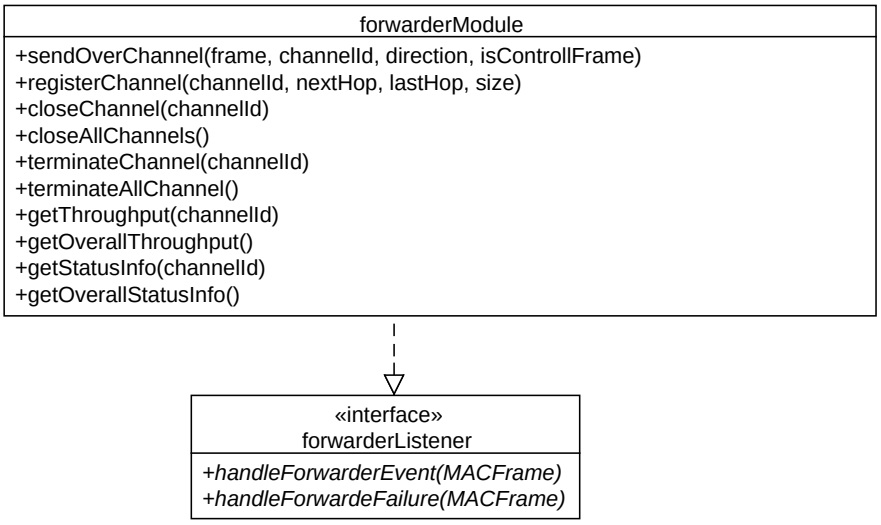


Figure 3.18: Diagram of the forwarder module's interfaces

Channel ID		*Routing information*		*Scheduling infomation*	
Init node	**Channel Number**	**next hop**	**previous hop**	**next transmission**	**size**
00:0A:34:67:EF:A2	89	00:0B:6A:5D:B6:31	08:00:69:02:01:FC	1251808413	12
03:2C:76:13:39:62	11	NULL	48:2C:6A:1E:59:3D	1251808459	5
00:0B:6A:5E:B7:54	35	00:0B:6A:5D:B6:34	NULL	1251808578	8

Figure 3.19: Overview over the channel list

frames, the exists also a failure-frame which is used to signalize broken channel events to the nodes involved in the channel. The first field of each forwarder frame is the subtype field. This two-bit-sized field defines the type of the forwarder frame: 00 for forward frames, 01 for failure frames, and 10 for control frames. The combination 11 is not used. The next two fields of each frame specify the ID of the channel, followed by a field specifying the direction. Using this information, the forwarder can relate frames to channels and consequently can determine, under the consideration of the direction field, the frame's next hop. In forward frames, the data field follows the direction field, containing a frame from an application using Cashflow. Consequently, if Cashflow is implemented in layer 2, in most cases a LLC-frame will be embedded in forward frames. The failure frame contains no additional field to encapsulate other frames, in contrast to the control frame, which possesses a field to embed other frames, since control frames are used by the controller module for the transmission of controller frames containing information for channel management.

After the *sendOverChannel* function has embedded the delivered frame into the correct forwarder frame, the forwarder frame is buffered in one of the channel's queues. Each channel possesses two queues: a priority queue, which is used for the transmission of control frames, and a default queue for the transmission of forward frames. Additionally, each channel possesses a scheduling process, which periodically removes frames from the queues, looks up the address of the next hop using the channel list, and transmits the forwarder frame using the *sendForwarderFrame* function, implemented by the shell module. The time between the periodic transmissions is determined by the size of the channel. For instance, given a channel has a size of ten. This means that the channel has a throughput of ten frames per second. After the transmission of a frame, the channel's scheduler calculates the next transmission time by adding a tenth of a second to the

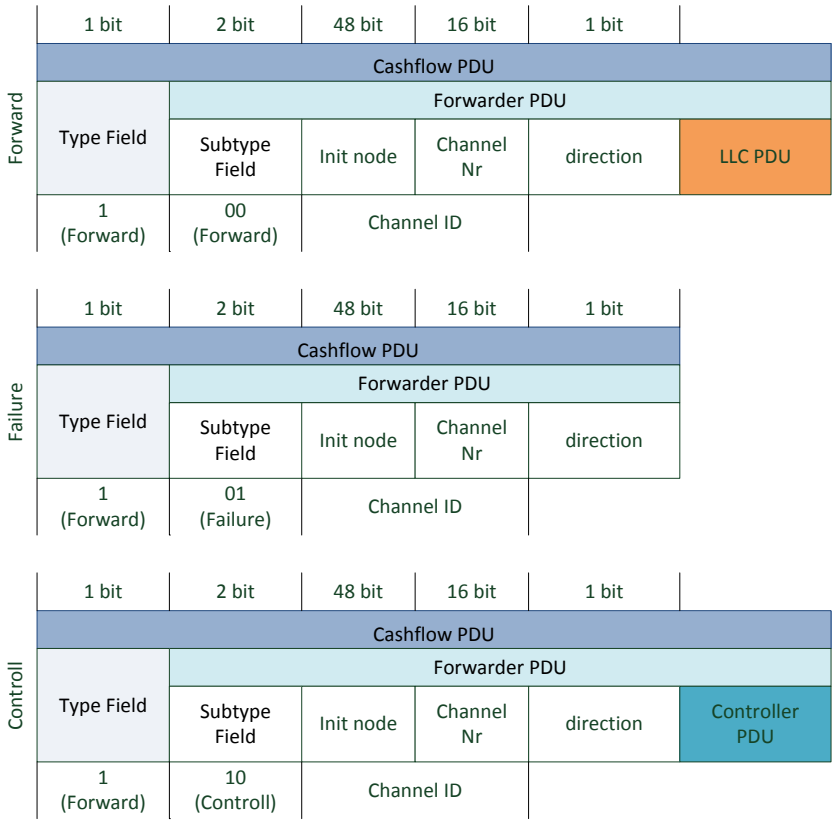


Figure 3.20: Overview over the forwarder frames

current time, and stores the calculated timestamp in the corresponding column of the channel list. If the moment stored in the channel list occurs, the scheduler removes the next frame from one of the queues and transmits it. However, if there is no frame in any of the two queues, the transmission process is delayed until the arrival of a new frame in one of the queues, which is then transmitted instantly. The scheduler transmits only frames from the normal queue if there is no frame in the priority queue left.

Using the function *closeChannel(channelId)*, it is possible to mark a channel for deletion by the scheduler. If the deletion mark is set, the scheduler deletes the channel's queues and its entry in the channel list as soon as there is no frame left in any of the channel's queues. Afterwards, the scheduler instance of the channel terminates itself. The function *closeAllChannels()* triggers the *closeChannel* function on every channel. This is useful if a node quits its participation in a shared wireless network.

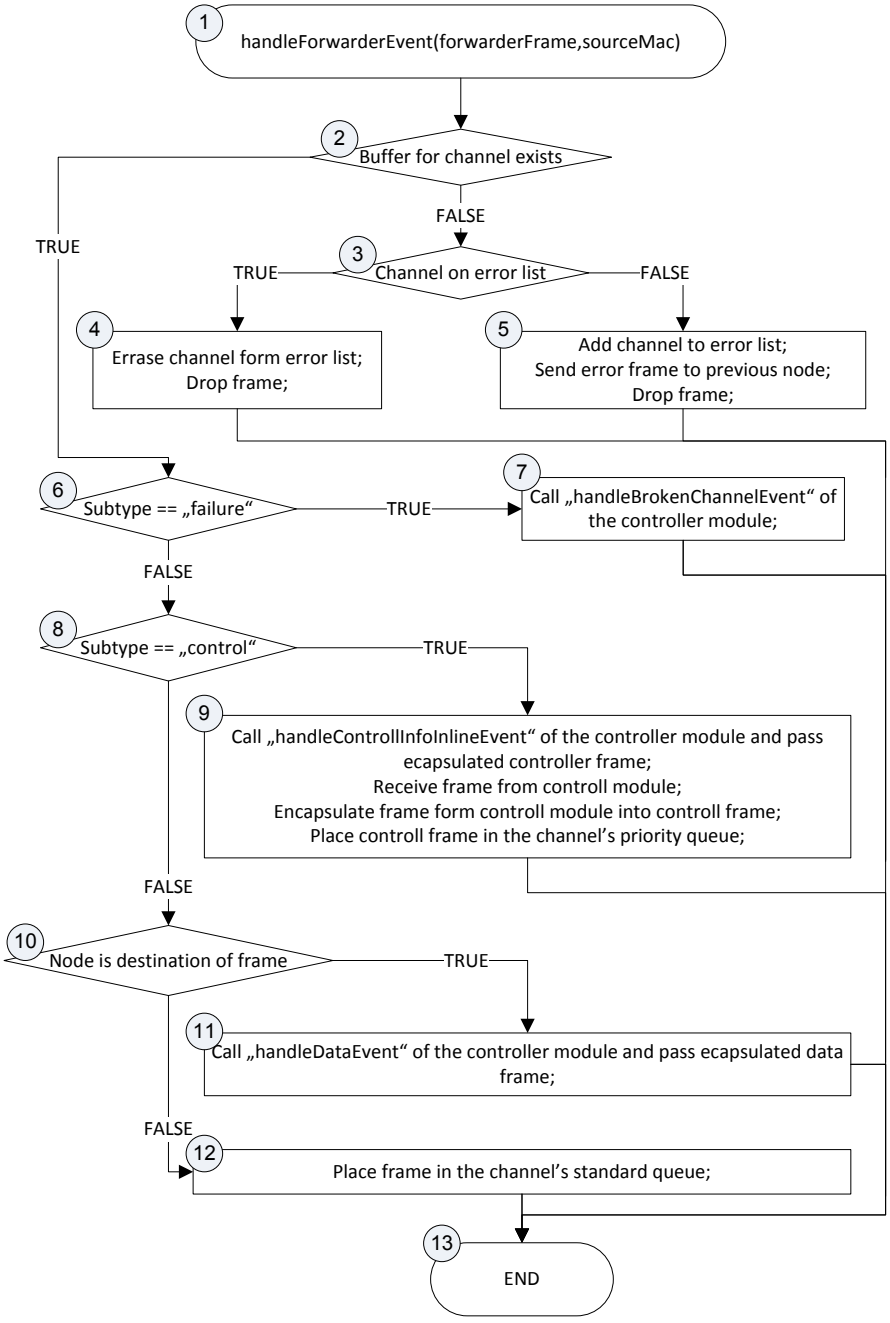


Figure 3.21: Overview over the handleevent function of the forwarder module

This is not the only way to close a channel. Also the function *terminateChannel(channelId)* can be used to delete a channel. The call of this function results in an instant deletion of a channel. The frames in the channel's queues get dropped before the queues are deleted. Additionally the function removes the corresponding entry from the channel list and deletes the channel's scheduler. Again, to perform this function on all channels, the function *terminateAllChannels()* can be used.

The last four functions shown in Figure 3.18 are used to gain additional information about channels. The function *getThroughput(channelId)* returns information about the size of the channel and the real throughput the channel causes. The real throughput might lies beneath the size of the channel, because the channel's source node might note use the complete capacity of the channel, or because of lost frames. The function *getOverallThroughput()* returns the same information for all open channels. To gain further statistical information, the functions *getStatusInfo(channelId)* and *getOverallStatusInfo()* can be used. They return information like the number of frames transmitted along a channel, the direction of frames, meaning how many frames are transmitted from the source node to the target node and vice versa, and the ratio between control frames and forwarder frames. The difference between this function is that *getOverallStatusInfo()* returns the information for active channels and not only for one like *getStatusInfo(channelId)*.

The forwarder module also implements the *forwarderListener* interface. After start up, the forwarder module registers itself as listener at the shell module, using the shell's *registerForwarderListener* function. When the shell module receives a forwarder frame, it calls the *handleForwarderEvent* function, passing the received frame and the address of the physical sender of the frame. Figure 3.21 visualizes the functionality of the *handleForwarderEvent* function. After receiving the frame in step 1, in step 2 the function verifies that the frame belongs to an active channel. If this is not the case, as shown in step 3, an error has occurred, meaning that a node tried to transmit a packet along a non-existing channel. This could be caused for instance due to an abrupt channel deletion as result of a connectivity loss. The function verifies if the error has already occurred before by searching for the channel ID in the error list. If no entry exists for the channel, it adds the channel to the error list and sends a failure frame to the previous node, to inform the node about the non-existing channel. Additionally it drops the received frame as shown in step 5. However, if the channel is already on the error list, as pictured in step 4, the module does not send a failure frame but erases the entry from the error list. This means

that if the node receives a frame for the same non-existing channel a third time, the node will transmit a failure frame again. This mechanism was introduced to avoid a loop, which could occur if multiple link errors along a path occur nearly simultaneous. In this case, a failure frame could result in another failure frame, resulting again in a failure frame and so on, which is avoided by this algorithm.

If the frame belongs to a channel, the handle function analyses the type of the frame. If the frame is of the type failure-frame, indicating that the corresponding channel is broken, the function calls the *handleBroken-ChannelEvent* function of the controller module in step 7. This allows the controller module to react on broken channel events. If the frame is of the type control, the function calls the *handleControlInfoInlineEvent* function of the controller module in step 9 and passes the encapsulated control frame to the module. The controller returns another control frame, which is encapsulated in a forwarder frame of the type control. Afterwards the control frame is inserted in the priority queue, so that the scheduler can transmit it to the channel's next hop.

If the received frame is neither a failure frame nor a control frame, the frame has to be a forwarder frame of the type forward containing an LLC frame. The function checks if the local node is the final target of the frame, meaning that the channel ends in the local node. If so, the LLC frame is encapsulated and forwarded to the controller module using the *handleDataEvent* function. If the node is not the target of the frame, the frame gets inserted in the default queue for transmission to the next hop.

The second listener function the forwarder module implements is the *handle-ForwarderFailure* function. The shell calls this function, if the MAC-layer was not able to transmit a forwarder frame. This can be seen as a local broken channel event. Therefore, the forwarder module reacts similarly as if it receives a forwarder frame of the type failure. It forwards the event to the controller module by calling its *handleBrokenChannelEvent* function.

Summarizing, the forwarder module manages the forwarding process of the channels. High-level management of channels is provided by the control module, which will be discussed in the next section.

3.3.5 Controller and channel controller module

The task of the controller module is to manage and maintain channels. It implements protocols needed to negotiate channel properties, to set channels up and to operate them. It assigns to each channel a specific channel

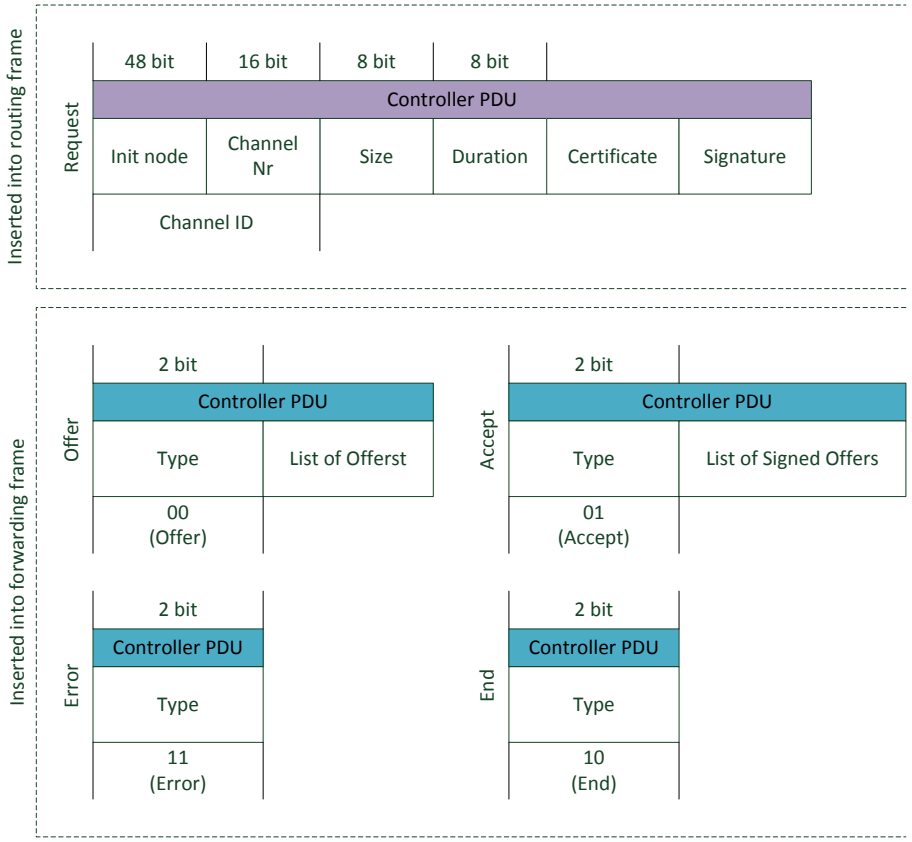


Figure 3.22: Overview over controller frames.

controller, which is responsible for the management of a single channel. The control module and the channel controller module are tightly coupled and therefore presented together. Additionally only the controller module communicates directly with instances of the channel controller module. Therefore, other architectures within the controller module are possible, where no specific channel controllers are used. However, the usage of channel controller modules for channel management allows better structuring of the system.

To perform its tasks, the controller module uses a number of frames, which are sent to controller modules of other nodes, using the routing or the forwarder module, depending on the tasks. Figure 3.22 gives an overview over the five frames used by the controller module grouped depending on the used transmission module. The first pictured frame is the request frame of the controller module, which is exclusively transmitted using the routing

module. Consequently, the only frame type the controller module receives from the routing module is the request frame. Therefore, this frame is the only frame which does not need a specific field to mark its type. The request frame is used to establish channels. The first two fields carry the ID of the channel, which should be established. As already stated before, the channel ID consists of the source's MAC-address and a 16 bit large channel number. The next two fields carry channel parameters. The first parameter is the size of the channel. The size states the number of frames, which should be transmitted over the channel per second. Due to the field size of 8 bit, the maximum throughput of each channel is limited to 256 frames per second. The second parameter field is the duration field. The value of this field states the duration of the requested channel in seconds. Again, due to the field size of 8 bit, the maximum duration of a channel is limited to 256 seconds. This means, in Cashflow up to 65536 frames can be charged in a single step, in contrast to other virtual currency system, where each frame is charged separately. The field before the last field of the frame holds the credit certificate, which was discussed in detail in Section 3.2.3. The credit certificate is used to prove the node's credit-worthiness. The last field is used to sign the channel parameters as well as the credit certificate, as already described in Formula 3.4 in Section 3.2.3

All other frames pictured in Figure 3.22 are transmitted over the forwarder module using a channel. To distinguish between the different frames, the first field of each field is used as type field, specifying the frame's identity. Frames with the type field set to 00 represent offer frames. When a node receives a request frame as target node, it sends an offer frame back to the source using a temporary established channel. While passing intermediate nodes, each intermediate node includes its offer in the frame. Therefore, the frame contains a field for the inclusion of the offers into the frame. Finally, the source of the route request receives the offer frame and might decides to accept the offer. If so, the source node transmits an accept frame along the temporary channel. The type field of the accept frame is set to 01 for identification. The accept frame possesses a second field carrying the signed offers. With the reception of the signed offers, nodes change the status of the channel from temporary to active. The offer and accept frames are also used for the extension of the channel's duration, as it will be described later.

The frames presented so far are used for the establishment and extension of channels. If a node wants to close a channel before it elapses, the node can use an end frame to perform this task. The end frame consists solely of a type field, set to 10 to indicate its type.

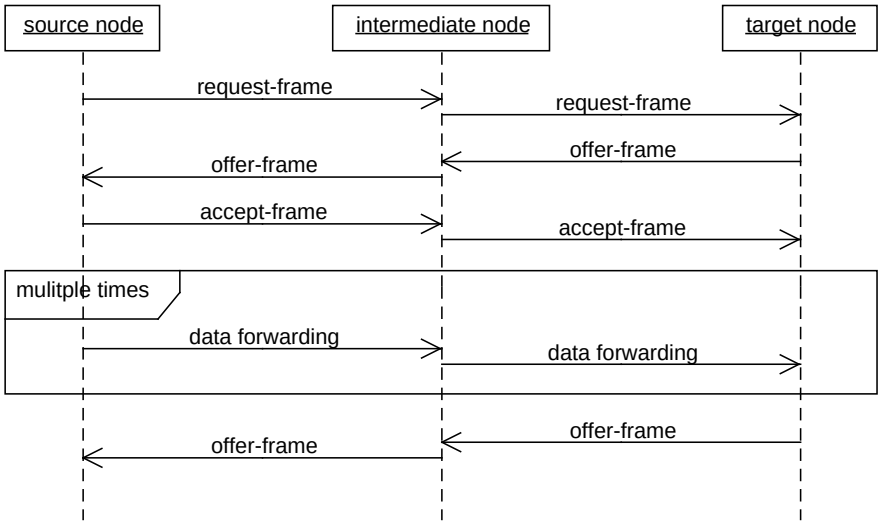


Figure 3.23: Flow chart of the channel establishment.

The only frame left is the error frame. This frame is used to indicate that the channel is broken. Similar to the end frame, it only consist out of a type field, set to 11 for type indication.

Figure 3.23 visualizes the establishment of a channel. The source node transmits a channel request over the intermediate node to the target node. The target node responses with an offer frame, which is transmitted back over the intermediate node to the source node. The source node receives the offer and decides to accept it by transmitting an accept frame. After the establishment of the channel, the source and the target node can exchange multiple frames. A short time before the channel's contract ends, the target node sends a new offer to the source node for an extension of the channel with, the same characteristics like the original one. Again, the intermediate node inserts its offer and the source node receives an offer for channel extension. If the source node is not interested in an extension of the channel, it can drop the offer. Otherwise, the source node can accept the offer by sending an accept frame and thereupon continues to transmit data along the channel.

Figure 3.24 visualizes the path of frames through the different modules during channel establishment. As stated before, the controller module starts

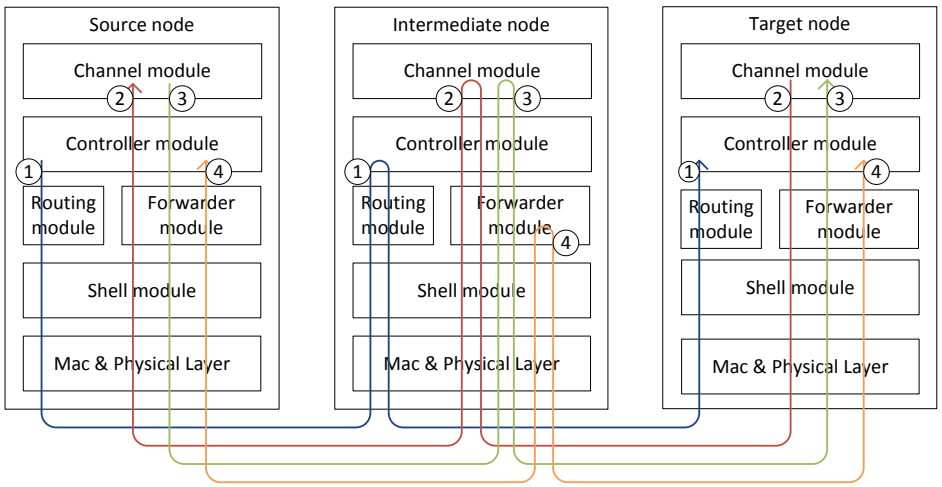


Figure 3.24: Path of control frames.

the establishment of a channel by transmitting a request frame along a given route through the network, including the node's credit certificate and the channel parameters. The line marked with 1 visualizes the request frame's path through the different modules. Besides transmitting the frame, the controller module additionally starts an instance of the channel controller, which manages the channel. One of the first things the channel controller does is to provoke the establishment of a channel at the forwarder module. The request frame is transmitted using the routing module's *sendOverRoute* function and reaches, after the physical transmission, the routing module of the next hop, which is in the given example an intermediate node. The routing module passes the route request frame to the controller module, which starts a new instance of the channel module, which provokes the establishment of a channel by the forwarder module after verifying that the content of the frame was not altered. This is done by checking the signature included in the frame. Additionally the module safes the credit certificate included in the request and verifies its validity. Then the controller module passes the request back to the node's routing module, which transmits the request frame to the next hop of the path, which in this case is the target of the request. The target node starts an instance of the channel controller module, which now has to perform a number of tasks. Like the other channel controllers before, it registers a new channel at the forwarder module after verifying the frame was not altered. Additionally it verifies and stores the included credit certificate using the safe module's *storeCertificate* function. Then it requests the fee the node charges for the channel

by calling the *getPriceForChannel* function of the pricing module. As next step, it generates an offer frame and calculates its entry for the offer list. The calculation of this entry has been discussed in Section 3.2.3 and can be seen in Formular 3.6. In short, it is a signed quadruple, consisting out of a hash value, calculated out of the request, the node's account number, the channel ID and the calculated fee.

After inserting the offer into the frame's offer list, the frame gets transmitted back to the source node. The line marked as 2 in Figure 3.24 visualizes the path of the offer frame. In contrast to the request frame, the offer frame is transmitted by the forwarder module, using the newly established channel. By setting the *isControlFrame* parameter of the forwarder module's *sendOverChannel* function, the frame gets transmitted within a control frame. When the control frame, which includes the offer frame, reaches the forwarder of the next hop, the forwarder can identify the included frame as frame belonging to the control module and therefore forwards the offer frame over the controller module to the channel controller by calling the controller module's *handleControlInfoInlineEvent* function. Like the target node, the channel controller of the intermediate node calculates an offer for the requested channel, adds the offer to the frame's offer list, and transmits it to the next hop, which is in this scenario the source node of the channel.

The source node's channel controller, belonging to the current channel, receives the offer and evaluates it. If the fee of the channel is too high, the channel controller module drops the offer frame. Additionally, it informs the forwarder module that it should delete the channel. The channel controllers, belonging to the requested channel and running on other nodes, wait for the accept frame for a certain time. If they do not receive the accept frame within the time span, they assume that the source node has rejected the offer and delete the corresponding channel within the forwarder module.

If the source node accepts the offer, it calls the *handleNewChannelEvent* function of the registered application listener to inform it about the new channel. Additionally it signs all the offers included in the offer list and sends an accept frame, which includes all the signed offers, to the target node. The corresponding line is marked as 3 in Figure 3.24. This frame is again transmitted by the forwarder module, using the corresponding channel so that it reaches the intermediate node. The channel controller of the intermediate node extracts the included offer, which is now signed also by the source node, and deletes the timer, which would cause the deletion of the channel in the case that the offer has not been accepted by the source node. Then the accept frame is again forwarded to the next hop, which in

this scenario is the target node.

The target node extracts the signed offer. Additionally, it calls the shell's *getDefAppEventListener* function to call the listener's *handleNewChannelEvent* function. With this call, the external application gets informed that there exists a new channel, which could be used for data transmission. To automatically send a new offer before the channel is closed, a timer is started by the target node's channel controller.

After the establishment of the channel, the applications, running on the source and the target node, can use the controller's *sendOverChannel* function to exchange data. The path of the corresponding frame is visualized as line 4 in Figure 3.24. It is important to node, that the frames are directly routed using the forwarder and not over the controllers of intermediate nodes.

A short time before the channel life time elapses, the target node transmits a new offer to the source node, based on the parameters of the original channel. To distinguish the offer from the original one, the signed offer gets extended by an additional extension number, resulting in the following offer format: (compare with Formula 3.6)

$$\{RH, accountNrSN, channelId, feeIntNode1, extensionNr\}_{\text{sigIntNode1}} \tag{3.12}$$

Again, intermediate nodes insert their offers into the offer list of the offer-frame. When the source node receives the offer, it can again decide to accept it, by sending an accept frame as pictured in Figure 3.25, or reject it, by dropping the frame. If the source node does not extend the channel, all nodes involved in the channel close the channel, after the pass of the channel's duration, by deleting the channel from the forwarder module. Independent if the channel gets extended or not, as soon as the regular end of the channel is reached, all nodes transmit the stored offers, which they had received during the channel's establishment, to the safe module, together with statistical information about the channel. Finally, if the source node did not extend the channel, the controllers belonging to the deleted channel terminate themselves.

Besides the reject of an offer, there are additional events, which lead to a channel's closing. As pictured in Figure 3.25, a link failure can lead to an abrupt end of channels. When the forwarder module detects a link failure, it informs the controller module about the event, which forwards the event

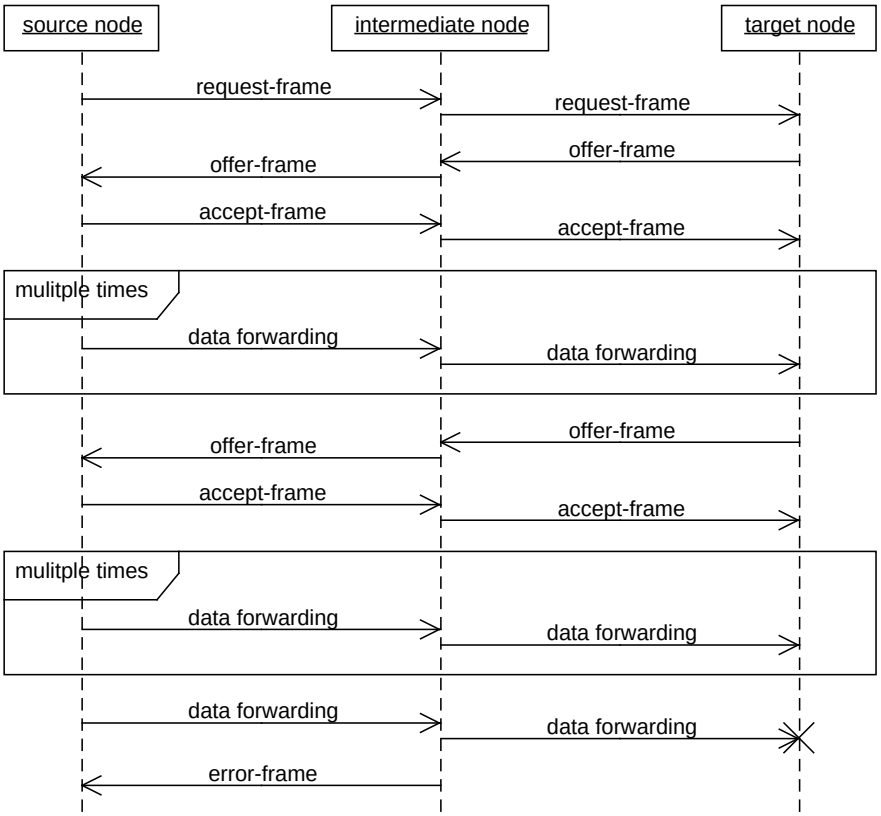


Figure 3.25: Extending a channel by sending a new offer.

to the corresponding channel controller. The channel controller reacts on this event with the transmission of an error frame, to inform the other nodes involved in the channel that the channel has been broken. Alternatively, if the broken channel event has occurred at the source or the target node, the application listener gets informed about the event, but no failure frame is send, since this would make no sense. Afterwards the channel controller informs the forwarder that the channel can be deleted and terminates itself. The channel controller shows the same reaction if it receives an error frame.

The last alternative of a channel's ending is pictured in Figure 3.26. In this scenario the source node wants to terminate the channel before its duration time elapsed. When channel controllers receive an end-frame, they

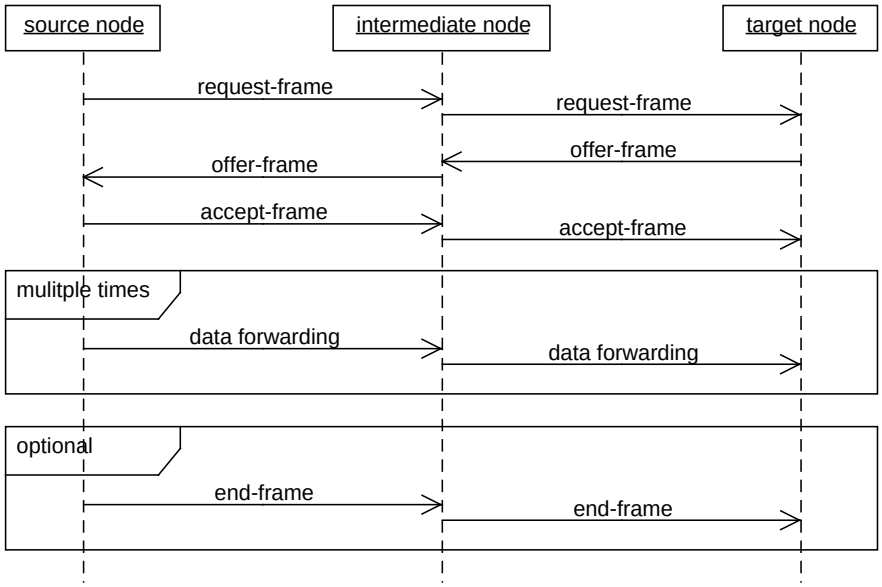


Figure 3.26: Visualization of a premature channel ending.

immediately discard the corresponding channel, store statistical data about the channel into the safe module, and terminate themselves.

To interact with other modules, the controller and the channel controller module provide a number of functions, as pictured in Figures 3.27 and 3.28. Most of the functions of the controller module are used indirectly by external applications, using the corresponding interfaces of the shell module. The first function of the controller module, the function *getOpex*, is an example of a function used by external applications. It returns the sum of the current operation costs of all active channels. This is done by calling the function *getOpex(channelId)* for each open channel and sum up all return values. The *getOpex(channelId)* function returns the operation cost of a single channel by calling the *getOpex* function of the channel controller associated with the corresponding channel. Similar the function *getOpexLimit*, which returns the maximum costs all current open channels could cause. Again this function is realized by calling the *getOpexLimit(channelId)* for each open channel, which calls again the *getOpexLimit* function of the corresponding channel controller. The function *getNrOfActiveChannels()* counts all active

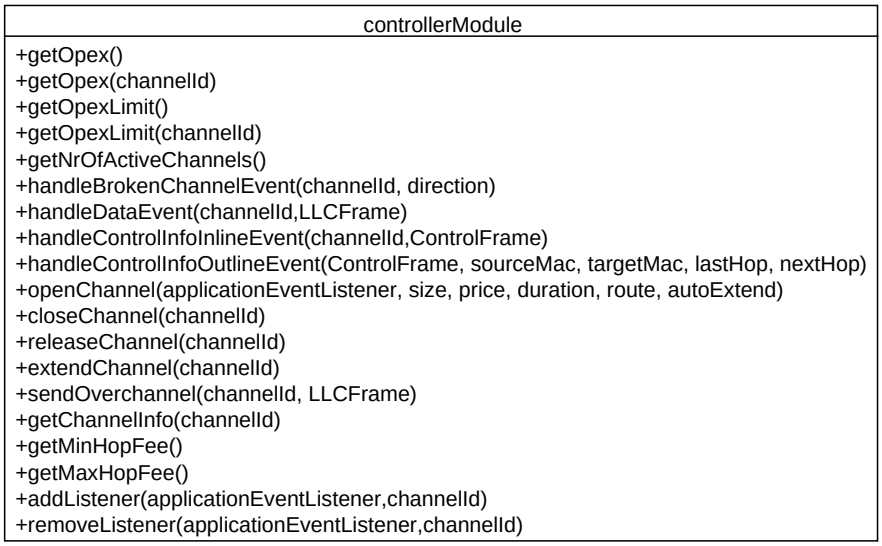


Figure 3.27: Interfaces of the controller module

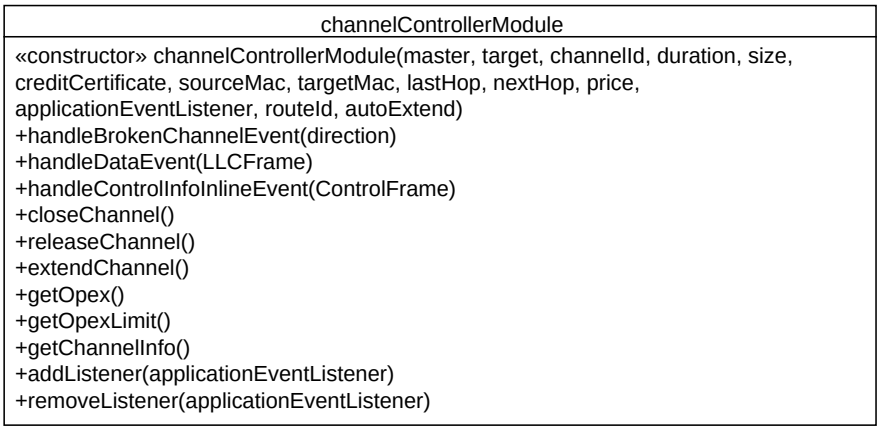


Figure 3.28: Interfaces of the channel controller module

channel controllers. The number of active channel controllers corresponds to the number of open channels, since each controller is responsible for exactly one channel and each channel possesses exactly one channel controller.

The functions *handleBrokenChannelEvent*, *handleDataEvent*, and *handleControlInfoInlineEvent* are used by the routing and forwarding modules to inform the controller about different events. Since these events belong to specific channels, it is the task of the corresponding channel controllers to react on these events. Therefore, these functions only forward the function calls to the same named functions of the channel controllers.

The function *handleControlInfoOutlineEvent* is used by the routing module to pass newly received control frames to the controller module. The included control frame is always of the type request, since the controller module is only allowed to use the routing module to forward controller frames of this type. As stated before, request frames are used to initialize the channel's establishment. Using the parameters passed to the function and the information included in the frame, the controller module starts a new instance of the channel controller by calling the channel controller's constructor. In Figure 3.28 all parameters of the constructor can be seen. The first parameter specifies, if the controller should act as master of the channel, meaning that the controller is responsible for the initialization of the channel. Since this is not the case if the channel controller is created as reaction of a channel request, this parameter is set to false in the given case. The next parameter states if the current node is the target of the channel, which is the case if the parameter *nextHop* of the event function was set to *NULL*. If so, the parameter is set to true. Otherwise, if there exists a next hop, the value of the target parameter is set to false. The parameters *channelId*, *duration*, *size* and *creditCertificate* are extracted from the request frame. The parameters *sourceMac*, *targetMac*, *lastHop* and *nextHop*, are passed through from the call of the *handleControlInfoOutlineEvent* function. The rest of the parameters are only relevant if the controller acts as master. Therefore all these parameters are set to *NULL* in the given case. Besides starting a new instance of the channel controller, the controller module verifies the included credit certificate and passes the included public key of the source node to the safe module.

Simmilar to the *handleControlInfoOutlineEvent* function, the function *openChannel* results in the creation of a channel controller. This function can be called indirectly by external applications using the same named function provided by the shell module. In this case, the channel controller acts as master, therefore the master parameter is set to true and the target parameter to false, since the channel's source node cannot be also the target of the same channel at the same time. The channel ID is created by the controller module. The controller module keeps a list of the channel numbers of the currently open channels and creates a channel ID for the new

channel, by combining its own MAC-address with an unused channel number. The values of the *size* and *duration* parameter are passed through from the *openChannel*'s call. With the *creditCertificate* parameter, the function passes the node's credit certificate, which is stored at the safe module, to the new instance of the channel controller. The parameter *sourceMac* holds the MAC-address of the current node. Using the route ID passed with the call of the *openChannel* function, the controller can request the target of the channel as well as the address of the next hop to set the corresponding parameters of the channel controller's constructor accordingly. The value of the parameter *lastHop* is *NULL*, since in this case there exists no previous hop. The values of the rest of the parameters are also passed through from the call of the *openChannel* function to the constructor. The *price* parameter states the maximum fee the channel controller is allowed to spend to open the channel. If the channel is more expensive, the controller has to reject the offer. The value of this parameter is also the return value of the *getOpexLimit* function of the controller. Using the *applicationEventListener* parameter of the constructor, the application requesting the channel can register itself as listener. The *routeId* parameter is used to update the price field of the corresponding entry in the routing table managed by the routing module. The last parameter of the constructor states, if the channel controller should try to extend the channel if the duration time of the channel has passed.

The following functions which follows in Figure 3.27 are all directly forwarded to the corresponding functions of the channel controller module. They are used to close channels immediately, to release the channel after the duration time has passed, to activate the automatic channel extension if the corresponding parameter was not set when the channel was opened, to send data over a channel, and to request detailed information about a channel.

The functions *getMinHopFee* and *getMaxHopFee* are needed by the routing module for the calculation of the fee depending artificial delay. These two functions return the minimum and the maximum fee nodes in the network have offered in the last time for data forwarding.

The last two functions are again directly forwarded to the corresponding functions of the channel controller module. Using these functions, external applications can register additional event listener or remove them. These functions are needed by the default application to hand over channels to other applications.

The functions of the channel controller module are pictured in Figure 3.28.

The controller module of each node involved in a channel creates a new instance of the channel controller to manage the channel. Using the channel controllers constructor, the properties of the controller are defined. The first two parameters define the functionality of the controller along the channel. A controller could act as master or target of a channel, or as intermediate node. The behavior of the channel controller depends on its role in the channel. If a channel controller is initialized as channel master, it requests an offer by transmitting a request frame along the given route and initializes the channel at the forwarder module. Channel controllers running on intermediate nodes only have to initializes the channel at the forwarder module after their creation. The controller acting as channel target creates like all other controllers the channel at the forwarder module but additionally creates a response frame including the fee the node charges for the channel, as calculated by the pricing function.

The next parameter, the duration, is needed by all channel controllers along the channel to trigger the elapse of the channel. Additionally the target channel controller uses this time value to calculate the moment when it transmits the offer for channel extension to the source node. The size parameter is needed for the creation of the channel at the forwarder module, since this module needs the size value to schedule the data transmission along the channel. The credit certificate is needed for the extension of the channel, since the channel controller module is not allowed to generate an extension offer if the validity of the certificate has past. Additionally, the credit certificate is needed by the channel controller acting as master, since it is needed for the creation of the request frame. The parameters *sourceMac*, *targetMac*, *lastHop* and *nextHop* are needed for the channel configuration at the forwarder module. As stated before, the remainder parameters are only relevant if the channel controller is acting as channel master. The price parameter is needed to decide if the controller is able to accept an offer for a channel or if it has to reject it because the fee of the channel surpasses the value of the price parameter, stating the maximum fee the controller is allowed to spend. The application listener is needed by potential receivers of data from Cashflow's viewpoint, which are the source and the target node, since the channel controller passes received data to the listener using the listeners *handleDataReceivedEvent* function. The *routeId* parameter is needed to update information about the fee and last usage of the corresponding entry in the route list, managed by the routing module. The last parameter is used to activate automatic channel extension. This can be also done by calling the *extendChannel* function.

The next function of the channel controller module as shown in Figure

3.28 is the *handleBrokenChannelEvent* function. This function is called indirectly by the forwarder module, if it is not possible to transmit a frame to the next hop because of lost connection. To react on the broken channel event, the channel controller builds an error frame and sends it back in the opposite direction as stated by the direction parameter of the function call, to inform other nodes about this event. Then the controller deletes the channel at the forwarder module using the modules *terminateChannel* function. If the channel controller is acting as target channel controller, it informs the registered application listeners about the occurrence of the event. This is also done by the controller acting as channel master, which additionally calls the *removeRoute* function of the routing module, to delete the broken route from the list of detected routes. After these tasks have been fulfilled, the involved channel controllers delete themselves.

The *handleDataEvent* function is called by the forwarder module, again indirectly using the corresponding function of the controller module, to pass received data frames to the channel controller. Since data is always transferred from one end of the channel to the other, this function as only called if the channel controller acts as master or target of a channel. The controller passes the received data to all registered application event listeners by using their *handleDataReceivedEvent* function.

The *handleControlInfoInlineEvent* function is the most complex function of the channel controller module. Figure 3.29 and 3.30 visualize its functionality. As pictured in Figure 3.29, the function has to handle four different control frame types. After the function's call in step 1, the function checks if the frame is of the type *offer*. If this is the case, the function performs in step 3 the offering procedure as pictured separately in Figure 3.30. If the control frame is not of the type *offer*, the function checks in step 4 if it is of the type *accept*. Such a frame signalizes that the source of a channel has accepted the received offer. As pictured in step 5, a channel controller receiving such a frame starts or updates a timer for the channel, depending if the accept frame is related to the first offer or to an extension of the channel, which gets active when the channel time has elapsed. The timer itself is used to trigger the deletion of the channel. At the target node, the timer is additionally used to trigger the generatrion of a new offer for channel extension, shortly before the channel elapses. Additionally, in step 5, the offer is extracted from the accept frame and stored locally for accounting after channel's end. Depending if the channel controller acts as target of a channel or not, which in this context could only mean that the channel controller acts as intermediate node, since a channel controller acting as master never receives an accept frame, the functionality varies. If the chan-

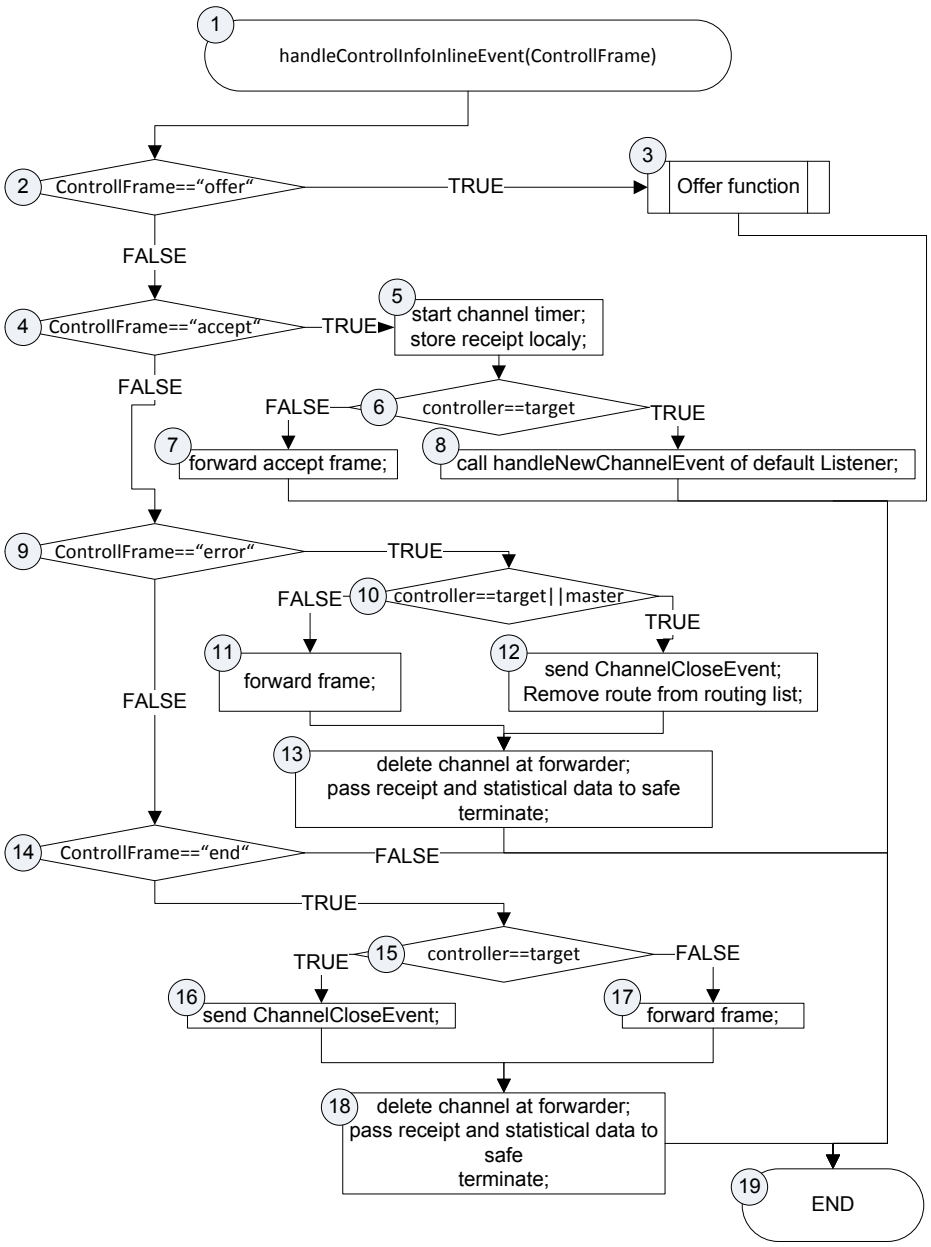


Figure 3.29: Diagram of the channel controller module's handleControlInfoInlineEvent.

nel controller acts as target as pictured in step 8, the channel controller informs the default listener, which he can access using the corresponding function of the shell module, about the new channel. Otherwise, if the chan-

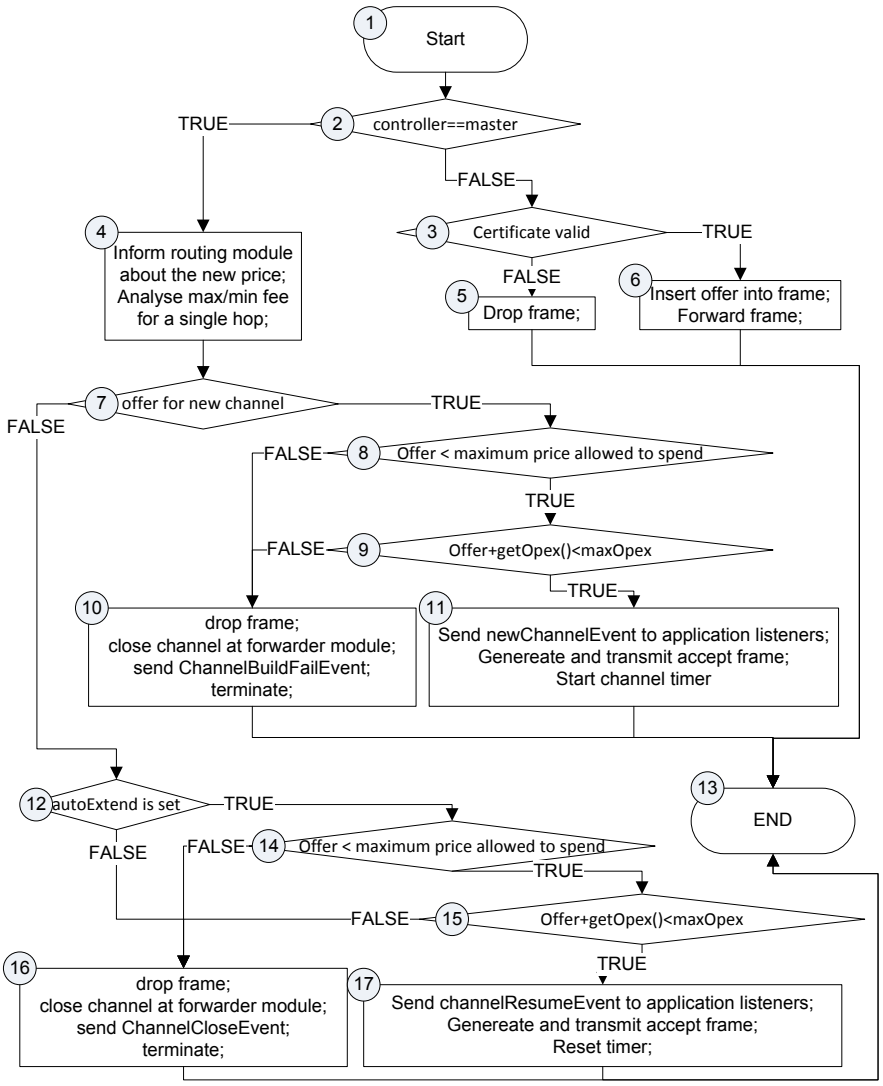


Figure 3.30: Diagram of the offer functionality of the handleControlInfoInlineEvent function

nel controller is located in an intermediate node of the channel, as pictured in step 7, it forwards the accept frame to the next hop by returning it to the function's caller, which in this case is the forwarder module. Independent of the channel controller's type the function finishes in the next step.

If the function as received an error frame, as pictured in step 9, the functionality depends if the channel controller is acting as intermediate node

of the channel or if it handles an end of the channel, meaning it has the role of the channel's master or target. If the channel controller is located on an intermediate node of the channel, it has to forward the error frame. Otherwise, it has to inform the registered channel listeners that the channel no longer exists. Additionally it procures the deletion of the route the channel has used from the routing module's routing list using the route ID the channel controller has received during its creation. Independent of the channel controller's role, the controller additionally deletes the channel from the forwarder module, passes the stored receipt together with statistical information to the safe module, using the safe's *storeReceiped* function or, if the channel controller was the channel's master, the *storeStatistics* function. As last step, the channel controller terminates itself.

In step 14, the function checks if the frame is of the type *end*. If this is not the case, the frame is of an unknown type, resulting in the end of the function. Otherwise, if it is of the type *end* the behavior depends if the channel controller is acting as target or not. If the channel controller is running on the target node of the channel, it informs the registered listeners that the channel has been closed, using the corresponding function of the application listener interface. Otherwise, the channel controller forwards the frame. As pictured in step 18, independent of the channel controller's role, the channel controller deletes the channel from the forwarder module, performs the accounting with the safe module, and terminates itself.

Figure 3.30 visualizes the functions behavior if the passed controller frame is of the type *offer*. Again, the behavior depends on the channel controller's role. If the channel controller is not acting as master, which consequently means that the channel controller has the roller of an intermediate controller, the function verifies in step 3 if the certificate received during the channels request is valid. If so, the channel controller inserts its offer into the frame using the pricing module's *getPriceForChannel* function and forwards the frame. Otherwise, if the certificate is not valid, the frame gets dropped.

If the channel controller is acting as master, it informs the routing module about the fee the nodes charge for data transmission along the given path, as pictured in step 4. Additionally, it analyses the fees the intermediate node charges for data forwarding to update the information about the minimal and maximal fee currently charged in the system. The next operations depend if the offer is the answer to a channel request or if the offer was sent automatically by the channels target to extend the channel. If the offer is the answer to a request, the function verifies in step 8 that the offer is

smaller than the maximum price the channel controller is allowed to spend for the channel. In step 9, the function verifies that the current operation costs of all open channels, together with the costs caused by the new channel, lies beneath the maximum costs the system is allowed to spend. If this test results in the conclusion that the new channel is not affordable, the offer frame gets dropped as pictured in step 10. Additionally, the channel controller deletes the corresponding channel from the forwarder module, informs the registered application listener that it was not possible to open a channel by using the listener's *channelBuildFailEvent* function, and finally terminates itself. Otherwise, if the fee is acceptable, the channel controller informs the application listener that a new channel is available using the listener's *newChannelEvent* function, as pictured in step 11. Additionally, the channel controller generates and transmits an accept frame, which includes the signed offers, which were encapsulated in the received offer frame. As last step, the channel controller starts a timer, which is used to trigger the application listener's *handleChannelExpirationEvent* and the function performing the channel's deletion, if the channel's duration has elapsed and no new offer for extension has been received. This might happens if the offer, sent by the target node shortly before the channel elapses, gets lost. In this case, the system behaves as if the channel controller has received an unaffordable offer.

If step 7 results in the conclusion that the received offer was sent by the target node because of the imminent channel's ending, the function checks in step 12, if an extension of the channel is in the node's interest, which is the case if the auto extension flag is set. As stated before, this can be done directly at the request of a channel or later, by calling the channel controller's *extendChannel* function. If it is in the node's interest to extend the channel, the function verifies in step 14 and 15 that the channel is affordable. If the channel is not affordable or if it is not in the node's interest to extend the channel, the offer frame gets dropped, as pictured in step 16. Additionally, the channel gets deleted from the forwarder module and the registered application listeners gets informed about the channel's closing, using their *closeChannelEvent* function, before the channel controller terminates itself. Otherwise, if it is in the interest of the node to extend the channel and the channel's price is affordable for the node, the function informs the registered application listeners about the resume of the channel, as pictured in step 17. Additionally the function generates a new accept frame including the signed offers and transmits it along the channel. As last step the timer gets reset to trigger the application listener's *channelExpirationEvent* function.

The next function pictured in Figure 3.28 is the *closeChannel* function.

This function triggers the transmission of an end frame along the path, resulting in an immediate closing of the channel. After the frames transmission, the function stores statistical information about the channel at the safe module, deletes the channel from the forwarder module, and terminates the channel controller module.

The functions *releaseChannel* and *extendChannel* are used to set or reset the auto extension flag, resulting if set in an extension of the channel before the channel elapses. The next two functions, *getOpex* and *getOpexLimit*, return the current fee the node spends for the channel and the maximum fee the channel controller module is allowed to spend to keep the channel active. The *getChannelInfo* function returns informations about the channel, including throughput, the moment the channel was opened, the time until the channel elapses, and the channel's fee. The last two functions, *addListener* and *removeListener*, allows to register and to delete application listeners, which get informed by the channel controller about channel related events like data reception.

Summarizing, the controller module implements together with the channel controller module the protocols needed to establish and manage channels. For each channel an own instance of the channel controller is created, whose functionality depends if the channel begins or ends in the current node, or if the channel passes the node.

3.3.6 Safe module

The task of the safe module is to provide functionality for the public key infrastructure. Additionally it implements functionality to perform the accounting and the interaction with the bank. To perform the communication with the bank, the module implements the *applicationEventListener* interface. Using this interface, the module could use Cashflow to open a channel to the bank node and transfer data along the channel like any other application.

Figure 3.31 gives an overview over the functions provided by the safe module. The functions of the listener interface have already been described in Section 3.3.2 and will therefore not be described in this section. The first function of this module is the *getLatestBalance* function. This function returns the balance of the bank account as it was at the moment, when the node had the last contact to the bank. The function *getCurrentBalance* returns an estimated balance, based on the balance of the bank account at the moment of the last contact to the bank, less the fees spent since the last

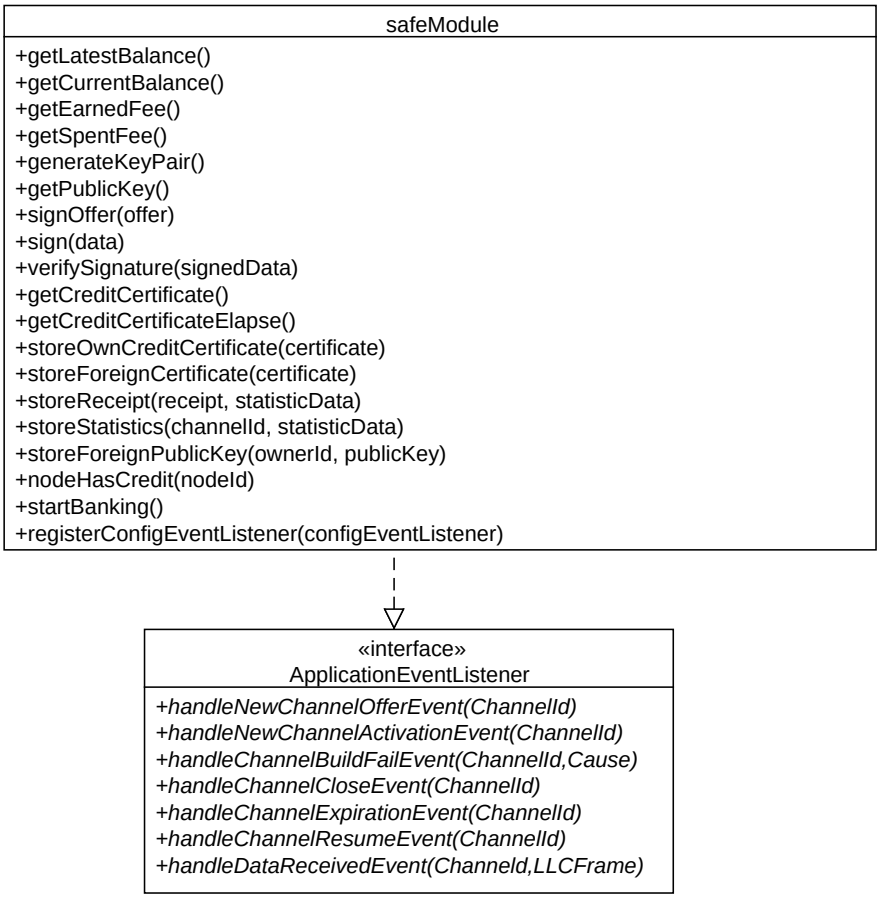


Figure 3.31: Overview over the safe module's functions

contact to the bank, plus the fees earned due to forwarding of other node's frames. If a user uses an account exclusively for one node, this value is relatively accurate. In this case, differences only occur if the user has externally added or removed credits form the account. However, if several nodes share the same account, the estimated value does not necessarily correspond to the actual balance. The next function, *getEarnedFee*, returns information about the credits earned by providing services to other nodes. This information includes the credits earned within certain time intervals, like the last hour or day, and additional information about the credits earned since the last connection to the bank node. Similar to this function, the function

getSpentFee returns information about the amount of credits spent. These four functions are accessible by the configuration application using the configuration application interface. Therefore the node's user can control the costs caused by the system.

Using the function *generateKeyPair*, it is possible to initiate the generation of a new key pair needed for the credit system. With the function *getPublicKey* it is possible to access the public key of the local node. A direct access to the private key is not foreseen in the current architecture due to security reasons. Instead, the safe module provides two functions for signing. The function *signOffer* allows signing offers received from other nodes. This function returns the signed offers and additionally adapts the current balance based on the fees included in the offers. The second signing function is the function *sign* which can be used to sign any type of data. To verify signatures, the function *verifySignature* can be used, which returns true or false depending on the outcome of the signature check. To gain access to the locally stored credit certificate of the node, the function *getCreditCertificate* can be used. If a function is only interested in the moment when the credit certificate elapse, the function *getCreditCertificateElapse* returns the corresponding timestamp. The shell provides the same function over an interface to the configuration application. Internally the shell forwards the function call to the safe module. The shell module forwards also the call of the function *storeOwnCreditCertificate* from the configuration application to the safe module. With this function, it is possible to store a credit certificate to the safe module, which is used by Cashflow to pay for the services of other nodes.

The following five functions shown in the figure are used by the controller module to handle payment. When the controller module receives a credit certificate as part of a channel request, it stores the certificate into the safe module using the *storeForeignCreditCertificate* function. When this function is called, the safe module safes the certificate and deletes previously stored certificates belonging to the same node. If a node has accepted the offer for a channel, it has to transmit an accept frame including the signed offers which so become receipts. When the controller receives the accept frame it extracts the signed receipt belonging to it, validates the certificate using the safe module's *verifySignature* function, and opens the channel. When the channel has elapsed, the controller module stores the receipt together with statistical data to the safe module using the *storeReceipt* function. The safe module uses the stored information to transfer the earned credits from the source node's, to the local node's account. Additionally the current balance is updated by adding the earned fee.

When a channel elapses, not only the intermediate and target nodes store information at their safe module. Also the source node stores statistical information at the safe module using the *storeStatistics* function. Similar to the receipts, this information is transmitted to the bank, since it is useful for fraud detection.

The *storeForeignPublicKey* function allows storing public keys of other nodes into the safe module. The safe module uses these keys to verify the signature of signed data. By using the function *nodeHasCredit*, the controller module can check if the safe module possesses a valid credit certificate of a node. This is important since Cashflow possesses a mechanism to extend the duration of channels, but it is not in the interest of the local node to extend the channel in the case that the source node's credit certificate has expired meanwhile.

Using the *startBanking* function, the control application can trigger Cashflow's banking mechanism, using the corresponding function of the shell module, if there is a connection to the bank. If this function is called, the safe module performs the banking independent if the fee the banking causes exceeds the limit set by the *setBankingFeeLimit* function over the shell module.

The safe module cyclically tries to perform banking. It verifies that there exists a connection to the bank and that the fee for the connection is below the limit set for banking. To connect to the bank, the safe module implements the application event listener. Therefore, the safe module acts as if it would be an application using the Cashflow system. After establishing a channel to the bank, the safe module transmits the collected receipts together with statistical data to the bank. The bank moves the earned fees from the different accounts to the node's account. Then, the bank sends the new balance to the safe module, which makes it available to other module by its *getLatestBalance* function. Additionally the bank node transmits update information for the black list to the safe module, which informs the routing module about the IDs of nodes, which are new on the black list.

The last function the safe module implements is the *registerConfigEventListener* function. Using this function, the control application can register itself as event listener using the same named function of the shell module. This event listener is used by the safe module to request a credit certificate in the case that it possesses no certificate, or the current certificate is not valid anymore, because it has expired or because the public key of the node has changed. After the control application has received a new certificate for the node, it uses the already described *storeOwnCreditCertificate* function to

pass the certificate to the safe module.

3.3.7 Pricing module

The purpose of the pricing module is to calculate offers for channel requests. As described in Section 3.2.2, a market concept is used for price calculation. So, the fee is basically based on supply and demand. However, a number of parameters influence the node's fee, which are discussed in this section.

One source of input for fee calculation is the node's user. The user can set a number of pricing relevant parameters by using functions provided by the shell for the control application. He can define the minimum fee a node charges for forwarding frames, the preferred and the maximum throughput, as well as the allowed variance of the throughput before the pricing function adapts the so-called base price. Moreover, the node's user can configure the maximum usage of the shared medium using the *setMaxThroughput* function provided by the shell module. The last two parameters the user can specify are the so-called minimum and maximum battery penalty. A battery penalty is added to the fee whenever the node is running on battery. The actual amount of the penalty depends on the battery's charging level. If the battery is fully charged, the minimum battery penalty is added to the fee. As the charge of the battery decreases, the penalty increases until it reaches the maximum penalty when the battery is nearly empty.

To realize such a charge dependent penalty, the pricing function needs to use the energy source as input. The result of the integration of the energy source as parameter in the fee calculation is that nodes using battery are less attractive as relay nodes for other nodes, especially if the battery penalty is high because of low charge, compared to AC powered nodes.

Concerning the network stack, the physical layer is another input source. It provides information about the transmission speed of links, which is used by the pricing function. An additional input source is the MAC layer. By collecting information about the shared channels usage [Davis04] [Davis05], the pricing function can react, if the preferred usage rate is in average exceeded for a certain time. If the average usage lies above the maximum usage of the shared medium, as defined by the user, the pricing function increases its fee. Additionally, a node can use this information to reject channel requests if it is expected that the additional load caused by the requested channel would result in an overload of the shared medium. Therefore, the usage of the information provided by the MAC layer results in an overload protection, since nodes in high load areas of a network tent to increase their

fee, with the consequence that routes through high load areas become less attractive.

The last input source is Cashflow itself. As stated before, the node's user can configure the preferred and the maximum throughput, as well as the tolerated throughput variance. Cashflow provides information about the number of currently open channels, as well as the resulting throughput to the pricing function. Using this information, the pricing function adapts the fee according to the difference between preferred and actual throughput.

These are the parameters used for pricing in the current version of Cashflow. However, also additional sources could be integrated into the pricing function if needed.

The fee itself is internally calculated by two functions. The first function calculates the basic fee a node charges. The basic fee reflects the current price level and gets adapt periodically depending on the current throughput and the shared medium's usage. Using this basic fee, the function *getPriceForChannel* calculates the fee a node charges for specific channels, depending on channel parameters and the node's context.

Figure 3.32 visualizes the algorithm for basic fee calculation. After the start of this function, in step 2 it initializes the variable *basicFee* to the minimum fee the user wants to charge for forwarding frames along a channel. This minimum price is set by the user, using the configuration application, and is accessible for the pricing module over the shell's *getBasePrice* function. Then the function enters a loop. After waiting for 1000 milliseconds in step 3, the function starts to update the fee. In step 4 and 6, the function checks if the current throughput lies above or beneath the preferred throughput, including the accepted variance, and increases or decreases the value of the basic fee accordingly in step 5 and 7. The nodes user defines the step size for the price change. The price module uses the shell's *getSmallPriceStep* function to gain the corresponding parameter, set by the user.

In step 8, the function checks if a request for a channel was rejected because the additional load would have exceeded the maximum throughput. If so, the basic fee is increase by the value the user has define over the shell's *setLargePriceStep* function. To gain information if a channel request was rejected because of the throughput, the pricing module possesses the function *eventChannelReject*, which is called by the controller module if the corresponding event occurs. The function sets a flag so that the occurrence of this event can be included in the calculation of the basic fee.

In step 10, the algorithm verifies that the current usage of the shared

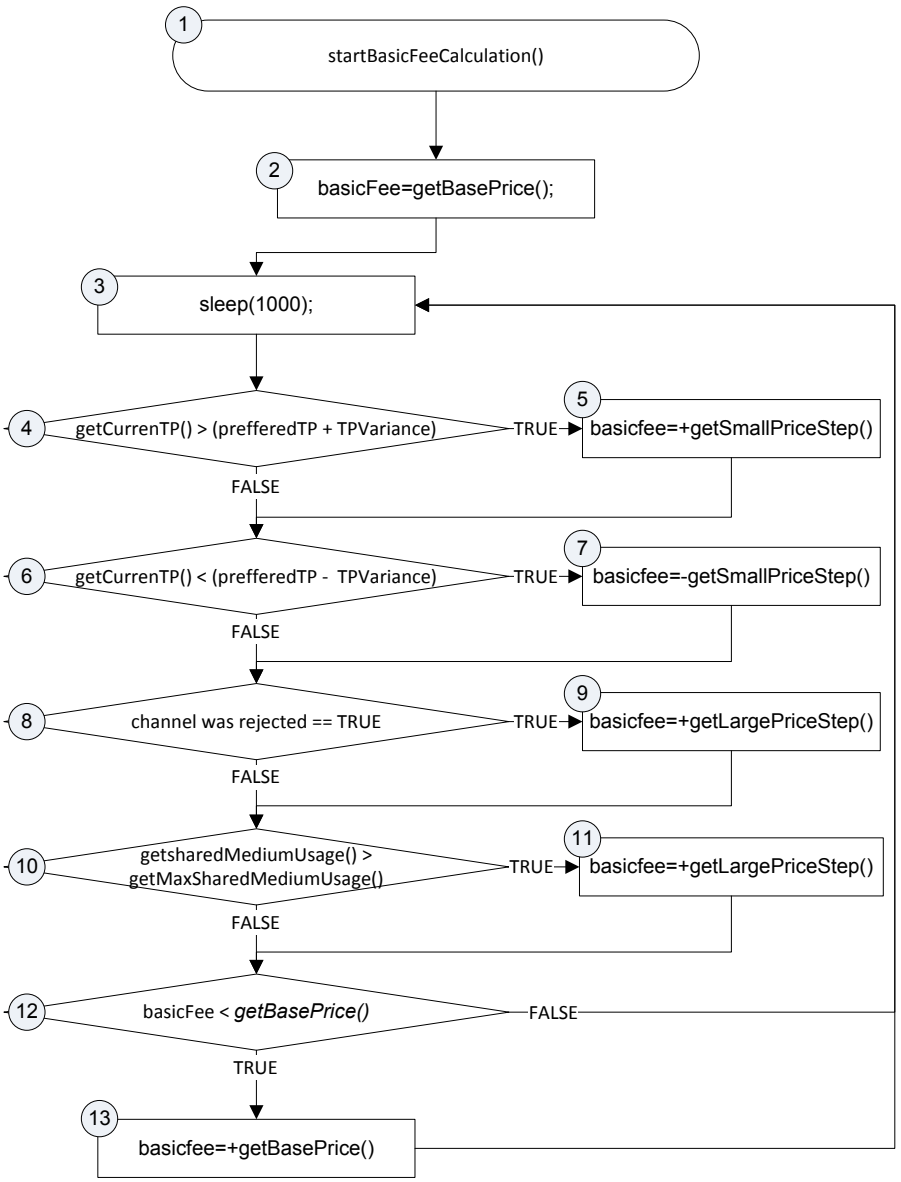


Figure 3.32: Visualization of the basic fee's calculation

medium is below the maximum usage preferred by the user. If the usage is higher, the fee is increased by the same value as if a channel has been rejected. After all these modifications on the basic fee, the algorithm

verifies in step 12 that the new fee is not smaller than the minimum fee. This can happen if there is a low load situation over a long time in the network. If the new fee would undercut the base price set by the user, it is set to the value of the base price. Summarizing, the algorithm pictured in Figure 3.32 adapts the fee depending on supply and demand.

The function *getPriceForChannel* uses the basic fee to calculate offers for concrete channel request. The offering function receives four parameters: the throughput in terms of frames per second, the duration of the channel in seconds, and the IDs of the next and the previous node. Figure 3.33 visualized the algorithm implemented by the *getPriceForChannel* function.

After the call of the function in step 1, the fee for the requested channel is calculated, depending on its size in terms of frames per second, its duration, and the basic fee. The variable *basicfee* contains the value for the basic fee calculated by the previous presented *basicFeeCalculation* function.

The remainder of the function adds additional penalties to the fee. The first penalty is the battery penalty, which is added if the node is running on battery. In step 3, the function verifies if such a penalty has to be added. If this is the case, in step 4 the penalty is added. The battery penalty consists out of a minimum penalty, which is added to the fee independent of the battery's charging level. Additionally a charge depended penalty is added, so that nodes running on low batteries charge more than nodes with full batteries. The actual amount of the penalty depends on two user variables stored in the shell module, which are accessible for the pricing module using the functions *getMinBatteryPenalty* and *getMaxBatteryPenalty*

In step 5 the function checks the speed of the incoming and outgoing links. If both links are classified by the statistics module as slow, the function adds the maximum transmission speed penalty as shown in step 6. This would be the case for instance, if a node supports the IEEE802.11g standard but both neighbor nodes, which represent the previous and the next hop, are equipped with IEEE802.11b technology. If only one of both links, which is verified in step 7, is slow, the minimum transmission speed penalty is added in step 8.

In step 9, the function estimates the cumulative throughput of existing and the requested channel, and verifies that the result lies beneath the maximum preferred throughput as configured by the user. If the estimated throughput exceeds the preferred one, the node rejects the request by setting the channels fee to infinite. Additionally it signalizes to the function calculating the basic fee that a request has been rejected by calling the pricing module's

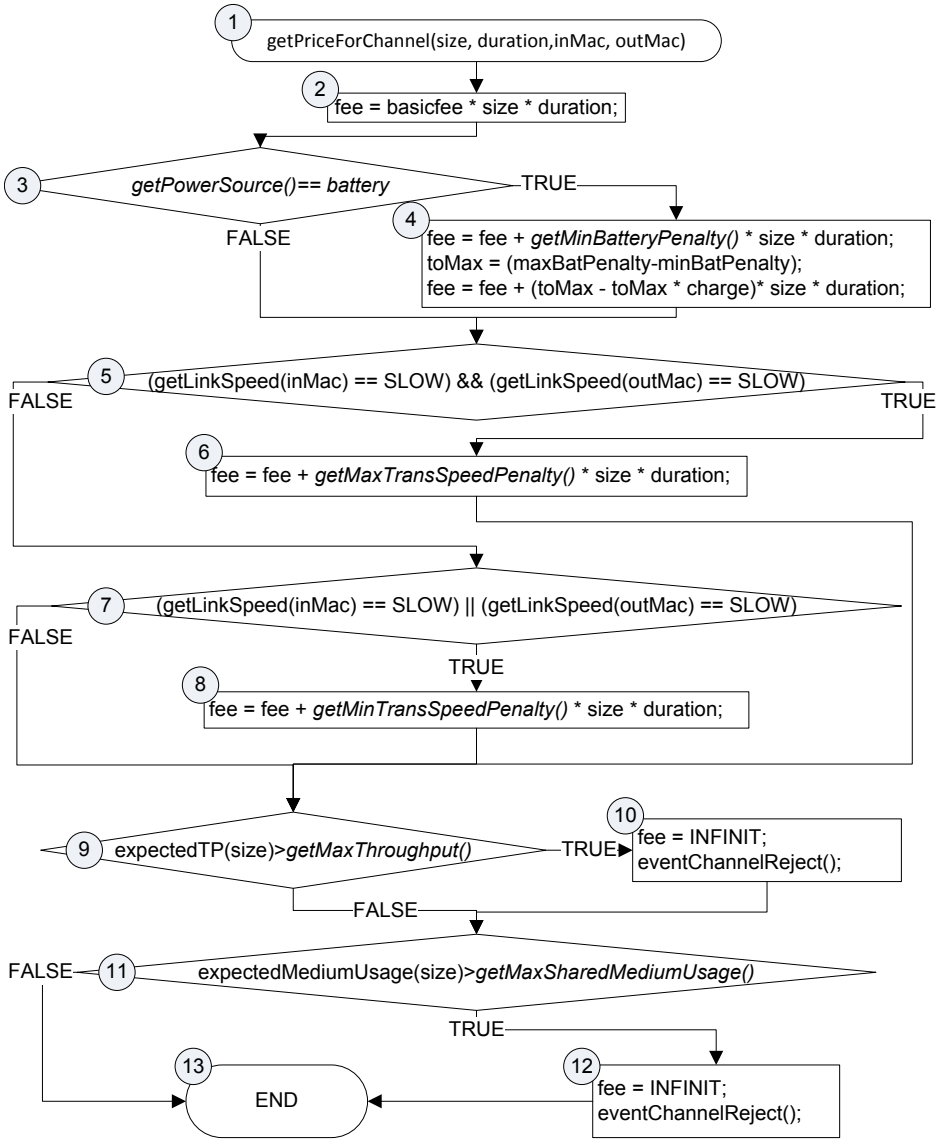


Figure 3.33: Visualization of the channel's fee calculation

eventChannelReject function, resulting in an increase of the basic fee on its next update. The algorithm likewise attempts to estimate the impact of the requested channel on the shared mediums usage in step 11. If the new

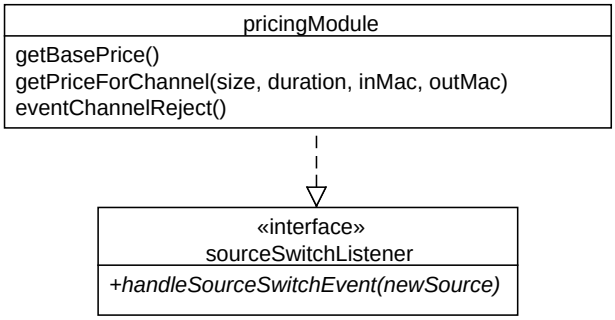


Figure 3.34: Overview over the pricing module's functions

channel would cause a local overload situation, the channel gets rejected. After adding all additional penalties to the fee, the function returns the fee.

It is worth noting that even if these two algorithms are used the evaluation of Cashflow, together they are only one example for pricing. The calculation of the basic fee reflects together with the calculation of concrete fees for channels the node's price policy. By adapting the pricing module, it is possible to implement different pricing strategies. Since the exploration of the optimal pricing strategy in marked based mobile ad hoc networks is not the focus of this thesis, only a simple algorithm was chosen as example, how the fee can be calculated in a way that it reflects the ratio between supply and demand, and additionally includes the nodes context as well as the user's preferences.

To complete the picture, Figure 3.34 gives an overview of the functions provided by the pricing module. The function *getBasePrice* returns the current value of the node's basic fee. The next function, *getPriceForChannel*, which has been discussed in details before, returns the price for a specific channel. This function is also used by the routing module to receive the fee for the so-called standard channel, a channel with fixed parameters, which are known to all network nodes. The fee for the standard channel is needed for the discovery of the cheapest route through the network. The last function of the pricing module is the *eventChannelReject* function, which is called to inform the module that a channel request has been reject. This information gets included in the periodical update of the basic fee. Besides these functions, the pricing module implements a *sourceSwitchListener* interface, which pro-

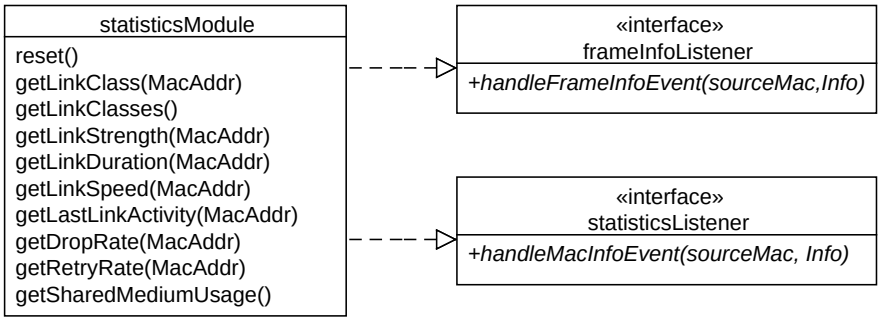


Figure 3.35: Overview over the statistics module's functions and interfaces

vides the function *handleSourceSwitchEvent*. Therefore, the pricing module can register itself as listener using the shell's *registerSourceSwitchListener* function, to get instantly informed by the shell module if the energy source changes.

3.3.8 Statistics module

The statistics module of Cashflow collects statistical data concerning links and the shared medium. Additionally it uses this information to classify links dependent on their stability and duration. The classification of the links is needed by the routing algorithm, which uses this information to find routes through the network fulfilling certain stability criterias.

Figure 3.35 gives an overview over the statistics module's interfaces. The statistics module implements the *frameInfoListener* interface, as well as the *statisticsListener* interface, to collect statistical information from the physical radio interface and the MAC-layer. Therefore, the statistics module registers itself as listener using the *registerStatisticsListerner* function of the MAC layer interface, implemented by the shell module, and the *registerFrameInfoListerner* function of the radio interface. Both interfaces posses a handle-event function, so that the radio and MAC layer interface can push information to the statistics module.

The statistics module itself mainly provides functionality to access collected statistical information. One exception is the first function of the module shown in Figure 3.35, the *reset* function. This function can be used by other modules to reset the statistics module, which means that all collected

information gets deleted. The next function, *getLinkClass* returns information about the quality of links to other nodes. As stated before, the statistics module classifies links depending on signal strength and duration. In the current version of Cashflow, three different classes are supported. When a new link to a node is detected, this link starts as link of class 3, the worst class. If the signal strength lies for a short time, in the current version of Cashflow this time is set to one second, above a certain radio interface dependent signal strength level, the link is promoted to class 2. However, if the signal strength level drops under a certain lower border, the link is downgraded to class 3 again. To become a class 1 link, which is the best link class supported by the system, the signal strength has to stay above the level needed to become class 2 for several minutes. Therefore, class 1 links refer to links between node, which have nearly no relative movement, and therefore reflects to the assumption that nodes show a nomadic mobility pattern. By preferring class 1 links over class 2 and 3 links, the routing algorithm tries to find routes over non moving nodes, which leads to a better stability of routes. If the function *getLinkClass* is called for information about a non-existing link, it returns 0 as link class. Like in other modules, this is only an example of a classification algorithm. To support other classification algorithms which might use more than three different classes, the function *getLinkClasses* was introduces. This function returns the number of link classes supported by the statistics module, whereas it is assumed that the class with the highest number reflects to the worst class and that class 1 is always used for the most stable links. Using the *getLinkClasses* function, the routing algorithm can adapt itself dependent on the number of classes supported by the statistics module.

The function *getLinkStrength* returns the current link signal strength for a given link. By using the function *getLinkDuration* other modules can request information about how long a link to another node exists. The *getLinkSpeed* function provides information about the speed of a link. This information is used by the pricing module to charge additional penalties for communication along slow links, since communication over slow links blocks the shared medium for a longer time compared to faster links. The next function shown in Figure 3.35 is the function *getLastLinkActivity*, which returns the moment of the last data reception over a link. The functions *getDropRate* and *getRetryRate* return information about the number of dropped frames and the average number of retries needed by the MAC protocol for the transmission of a data frame. The last function the statistics module implements is the *getSharedMediumUsage* function. This function returns information about the current usage of the shared medium. This

information is needed by the pricing module, which increases the basic fee if the shared medium's usage succeeds a user given value, and additionally by the control module, which drops route requests if the assumed cumulative usage of the current usage of the shared medium and the additional usage by the requested channel succeeds a certain value.

3.4 Attacks on Cashflow

Security is an important issue for ad hoc networks, especially because of their usage of an open shared medium. When payment is involved, security becomes even more important. However, a complete analysis of several possible attacks would go beyond the scope of this thesis. Therefore, this section focuses only on those attacks, which might bring an attacker financial advantages. Other attacks on mobile ad hoc networks, which are not related to Cashflow, have been discussed comprehensively in literature [Nahrstedt09], [Wu07].

3.4.1 Certificate misuse

To receive services from ad hoc networks using Cashflow without paying for it, a user might try to misuse credit certificates. Possible attacks belonging to this category includes forging and stealing of credit certificates, as well as the usage of elapsed certificates, and the acquisition of credit certificates from the bank by false pretenses. All these attacks are prevented by Cashflow using a cryptographic system. As discussed in Section 3.2.3, each node, each user, and the bank posses a public and a private key. Additionally it is assumed that there exists mutual thrust between users and the bank as well as between user and nodes belonging to the user. We argue that under the assumptions that no private key gets stolen, that the mutual thrust is not misused, and that the cryptographic system is save, the described attacks are not leading to the desired results of gaining free services.

There are two possible targets for an attacker to acquire a credit certificate by false pretenses. First, he could try to fake the identity of a node to persuade the user the node belongs to, to request a credit certificate from the bank for the attacker's node. If this attack would be successful, the attacker could use the credit certificate to pay for used services and the victim user would have to pay for the services the attacker has used. However, a node requesting a credit certificate from the user it belongs to, signs the request. Therefore, the user can verify that the request has not been altered and that the node is really the node it claims to be. To fake the signature the

attacker has to steal the nodes private key or has to break the cryptographic system. However, each node is responsible for the security of its private key and, as stated before, it is assumed that the used cryptographic system fits the state of art and is therefore not breakable in reasonable time. Second, an attacker could try to fake the identity of a user to persuade the bank to issue a credit certificate. Again, similar to the fake of a node's identity, the attacker would have to steal the user's key or break the cryptographic system. Again, this is prevented, because the user signs the request. The signature is used by the bank to verify the originality of the request and the identity of the user.

Another possible attack of a malign node is to steal and use the credit certificate of another node. Basically, a malign node does not even have to steal a credit certificate, since when a node requests a channel, it passes its credit certificate along the path of the channel. From the attacker's point of view the problem is reduced to the question, how to use the certificate of another node. The certificate includes the node's public key and the channel request, which includes the credit certificate, is signed by the node the credit certificate belongs to. Using the signature, each node receiving a channel request can verify that the request comes from the node to which the included certificate belongs. Therefore, to use the certificate of another node, the attacker has to fake the signature, which is not possible without stealing the victim node's private key or by breaking the cryptographic system.

The usage of elapsed certificates is prevented by a timestamp within the credit certificate. Since each certificate is signed by the bank, and each node in the system, which accepts the credit certificates signed by the bank, possesses the bank's public key, the time stamp cannot be altered without making the credit certificate invalid. The only way to alter the time stamp and keep the certificate valid is by faking the bank's signature, which could not be done without breaking the cryptographic system or stealing the bank's private key. The same is true for faking a complete credit certificate or alter another node's certificate. Without the private key of the bank, it is not possible for an attacker to sign the faked certificate. And a certificate, which is not correctly signed, is worthless for the attacker, because no node accepts such an certificate.

3.4.2 Channel misuse

Malign nodes might try to misuse channels of other nodes to gain financial benefits. To save resources, a node providing a channel might drop pack-

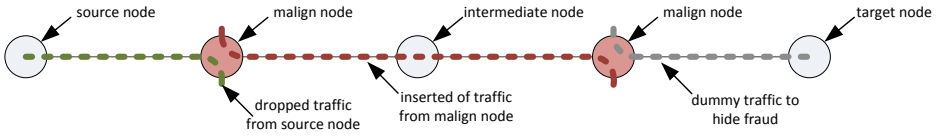


Figure 3.36: Visualization of a packet insertion attack on a channel

ets transmitted over the channel instead of forwarding them. Using this strategy, the node would earn a reward for the channel without actually providing a service. To prevent this fraudulent behavior, all nodes involved in a channel transmit statistical data, including the number of received packets, to the bank node together with the earned receipts. Using the statistical information, the bank detects fraudulent behavior and punishes malign nodes by blacklisting them. Since the blacklist managed by the bank is distributed to all nodes of the network, after some time all nodes refuse to provide services for the detected malign nodes. An alternative approach to prevent this kind of behavior would be to pay suspicious nodes only part of their reward, depending on the number of packets received by neighbor nodes. This concept has been discussed extensively in [Zhong03], where it was proven that from a game theoretical point of view, this concept prevents the drop of packets to gain financial benefits.

If there exists a number malign nodes in the network, which want to communicate with each other, they might try to take over channels form other nodes, which coincidental connect two or more malign nodes. Figure 3.36 visualizes this kind of attack. The source node, which is located on the left side of the figure, has opened a channel to the node of the right side of the figure, passing two malign nodes. When the malign node on the left side receives a packet from the source, it might drop the packet and replace it by another packet including data, which the malign node wants to transmit to the other malign node. The second malign node receives the packet over the intermediate node. To hide the fraud, the second malign node replaces the packets sent by the first malign node by other packets including dummy data.

The difficulty of this scenario is to detect, which nodes act malign and which do not. One way to solve this problem would be that the source node signs every packet and intermediate nodes forward only packets with valid signatures. However, this would increase the packet size and additionally would be computational expensive. As alternative, every node keeps a copy of a certain number of packets forwarded lately. If the source node has

the suspicion that his channel gets misused, it transmits a command to all nodes along the channel to generate a hash value out of the stored copies of the transmitted packets. When nodes have connection to the bank, the nodes transmit the hash values as part of the statistical data to the bank, which can use this information to detect the malign nodes because of the differences in the hash values. Consequently, the bank blacklists malign nodes, making this kind of fraud unattractive.

3.4.3 Routing manipulation

A node might not be willing to participate in route search, since this is a free service in Cashflow, and therefore the node does not receive a direct reward for participating in route discovery. However, in long term it is not profitable for a node to boycott other node's route discovery processes, since nodes would not find routes running over the uncooperative node. As consequence, no node establishes a channel over the uncooperative node. Therefore, by boycotting the route discovery process, the node drains the source for channels and consequently its revenue sources.

To transmit data to another node in the network without charging for this service, a node might try to append additional data to a route discovery packet, so that during route discovery the data gets transmitted to the target node. However, Cashflow prevents this attack, since for route discovery a strict and to all nodes known packet format is used. Therefore, nodes can detect suspicious route discovery packets and drop them.

Besides these presented attacks, there exist a number of additional attacks targeting the route discovery process, which are not specific to Cashflow. For instance, denial of service attacks and the altering of routing information in the route discovery packets are issues. However, a number of solutions for these attacks have been proposed in literature, which can be found besides other in [Hu05] and [Yang04].

3.4.4 Receipt misuse

The receipts used for charging provides an additional attack point for malign nodes. Malign nodes might try to fake or manipulate receipts to gain credits, or encash receipts multiple times. To fake or manipulate receipts, a malign node needs to fake the signature of another node, which can only be done by breaking the cryptographic system or by stealing the private key of another node. As stated before, it is assumed that nodes store their private keys safely and that the cryptographic system cannot be broken in reasonable

time. Additionally, a faked receipt would become suspicious for the bank, since in Cashflow all nodes involved in a channel transmit their receipts to the bank as soon as there exists a connection. Therefore, it would be suspicious, if no other node encashs a receipt belonging to the same channel as the faked receipt. A similar mechanism prevents that nodes encash the same receipt multiple times. Even if the probability that two receipts are the same is nearly zero and therefore the retransmission of a previous encashed receipt can be detected by the bank by comparing old receipts with the new one, the absence of other node's receipts belonging to the same channel indicates the fraud.

3.4.5 Conclusions

The analysis of possible attacks has shown the importance of the usage of a state of the art cryptographic system, which cannot be broken in reasonable time. Additionally, it is important that private keys stay private. If an attacker has stolen the private key of a node or a user, the attacker can use services of other nodes on the victim's costs.

3.5 Cashflow and hybrid networks

So far, Cashflow has been described for the usage in isolated mobile ad hoc network. It was assumed that there exists no connection to the Internet and that only wireless technology is used for communication. However, realistic scenarios fulfilling these assumptions are rare. Therefore, this section presents how ad hoc networks using Cashflow can be connected to the Internet.

As stated in Section 3.3.5, Cashflow uses channels for data forwarding. A previously not mentioned benefit of this concept is that it allows seamless integration into the Internet using mobile IP [Perkins02] [Perkins07] [Johnson04]. Figure 3.37 visualizes the connection of a mobile ad hoc network to the Internet. In this figure, the light and dark blue nodes form a mesh network. Wireless links are represented by doted lines, wired links by solid lines. The four nodes in the gray area are connected by wired links to give an example how hard wired mesh networks can be integrated into a wireless network. To support wired links, only Cashflow's statistics module has to be altered. The previously described version of the statistics module was designed to classify wireless links based on signal quality and duration, where class 1 links refer to links, which have a strong radio signal and are stable. Compared to wireless links, hardwired links are in most cases sta-

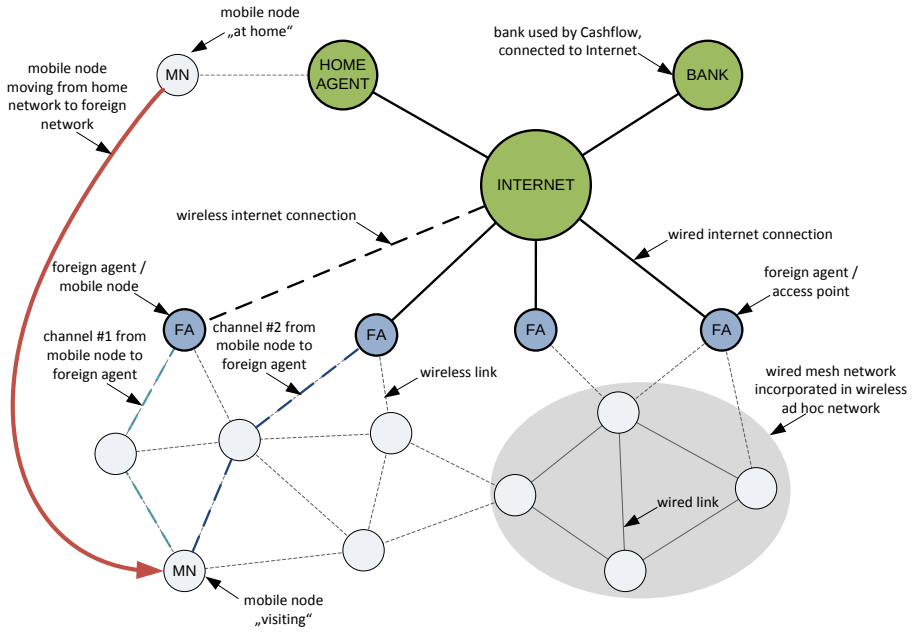


Figure 3.37: Internet connected ad hoc network using Cashflow and mobile IP

ble and have a relative low loss rate. Therefore, for hybrid networks, the statistic module is altered in a way that wired links are always categorized as class 1 links. An alternative approach would be to specify a specific class for wired links, which is preferred by the routing algorithm over other classes. For instance, the statistics module could use four classes, where class 1 refers to wired links and class 2 to 4 refers to class 1 to 3 wireless links of the original system. Using this approach, the routing algorithm automatically prefers wired over wireless links without any alteration of the algorithm. Summarizing, Cashflow is not limited to pure wireless ad hoc networks, but can also incorporate wired mesh networks. Even more, Cashflow can also be used in pure wired mesh networks.

In the scenario depictured in Figure 3.37, a wired mesh network is incorporated into the Internet-connected wireless ad hoc network. The dark blue nodes in the figure represent nodes using Cashflow, which have direct access to the Internet. For Cashflow it makes no difference how these nodes are connected to the Internet. They could have a wired connection like DSL or wireless like GPRS or HSDPA. These nodes act as foreign agent for mobile nodes of the network, which want to access the Internet. Alternatively, the nodes marked as foreign agent act as router if a care-of-address

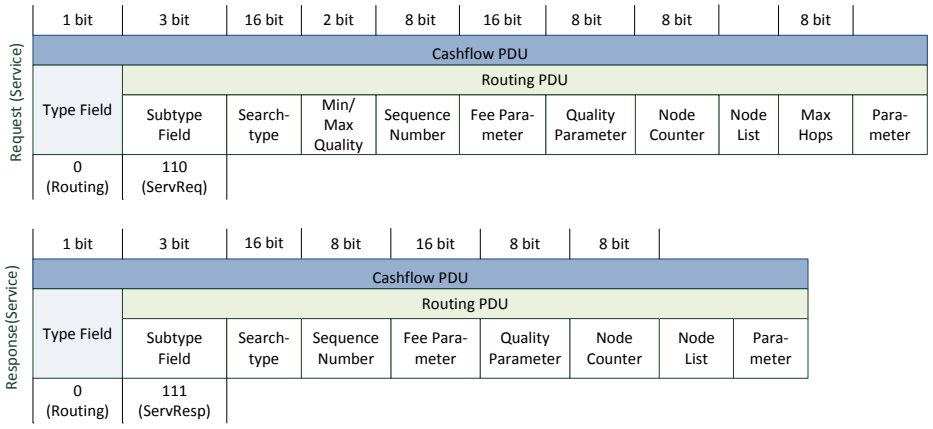


Figure 3.38: Extended frame structure for route discovery

approach is used. Figure 3.37 additionally depictures the bank used for charging by Casflow, which is also connected to the Internet, and a home agent, as defined by the mobile IP specifications [Perkins02] [Perkins07] [Johnson04].

Each mobile node has a fixed home IP-address over which the node is reachable, even if it is visiting a foreign network. When a mobile node is visiting a foreign network, which is using Cashflow, the node uses a special route request to find nodes providing access to the Internet. In Section 3.3.3 the routing protocol was described including the frame structure. As pictured in Figure 3.12 in Section 3.3.3, frames with the subtype 110 and 111 have been reserved for later usage, to implement functionality not needed for traditional routing in isolated mobile ad hoc networks. These previously reserved subtypes are now used to implement additional route discovery functionality. Figure 3.38 visualizes the two new routing frames. The new route request frame posses a search type field instead of a target field. Instead of using a specific MAC-address as target for route search, the search type allows to specify other search targets like services. Additionally, the request frame has been extended by two fields, the max hop and the parameter field. The max hop field allows to specify the maximum number of hops considered for route search. When a node receives a route request frame, it decrements the max hop field by one before it rebroadcasts the frame. If a node receives a route request frame where the max hop field is 0, it drops the frame. With this mechanism, the search area gets limited, which is important for the scalability of Internet connected ad hoc network. The idea behind this mechanism is to use the Internet as backbone to de-

crease traffic in ad hoc networks. Even from an economic point of view this makes sense, since with an increase of the path length between two nodes in the wireless network the probability increases that there exists a cheaper path over the Internet. The parameter field allows to append search related parameters to the route request frame. In addition, the response frame possesses a search type field instead of a target field. Likewise, a parameter field has been added.

When a mobile node is part of an ad hoc network and wants to connect to the Internet, it uses the new request frame to search for paths to nodes providing Internet access. For this purpose, the node sets the search type field to the value 1, which has been assigned to the service *Internetaccess*. Additionally, the maximum hop field is set for instance to the value 3, to limit the search to nodes within a range of three hops. The search for Internet access requires no additional parameters, therefore the parameter field is kept empty. Taking the configuration shown in Figure 3.37 as example, the route discovery algorithm returns paths to two nodes, which are in the range of three hops to the mobile node and posses access to the Internet. It is worth noting that the fee field of the response frame not only includes the price for a standard channel to the node providing Internet access but also the price for Internet access with the bandwidth of the standard channel.

So, the route discovery results in paths to nodes providing Internet access and additionally the source node of the route request receives information about the approximate fee of Internet access as well as the fee for the channels. To access the Internet, the mobile node opens a channel to a node providing access to the Internet. After establishing of the channel, the mobile node makes a contract with the node providing Internet access about the usage of its Internet access. Similar to the receipts for channels, this contract is signed by both parties, including the fee for the Internet usage as well as information about the provided bandwidth or data volume. For the payment, the infrastructure provided by Cashflow can directly be used. The next step depends on whether mobile IPv4 [Perkins02] [Perkins07] or mobile IPv6 [Johnson04] is used and in which mode.

Figure 3.39 visualizes the usage of mobile IPv4 in combination with a foreign agent and bidirectional tunneling. After establishing a channel to a node providing Internet access and willing to act as foreign agent, the mobile node binds itself to the foreign agent over path 1 as pictured in Figure 3.39. Additionally the mobile node registers its care-of-address, in this scenario the foreign agent care-of-address, to its home agent over path 2.

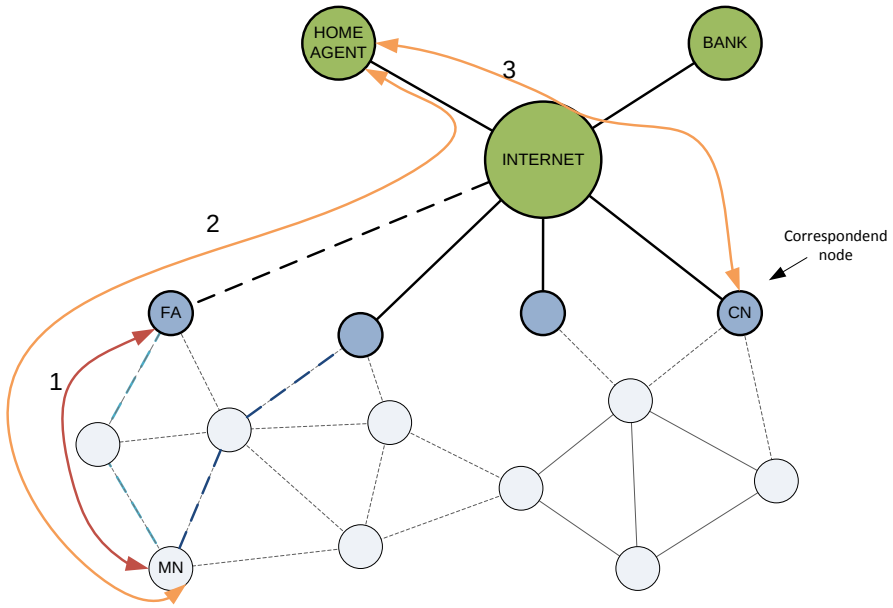


Figure 3.39: Internet connected ad hoc network using Cashflow and mobile IPv4

When a node, the correspondent node, sends a packet to the mobile node using the mobile node's home IP-address, the packet reaches the mobile node's home agent over path 3. The home agent looks up the previous registered care-of-address of the mobile node and tunnels the packet, using an IP-to-IP tunnel, to the foreign agent. The foreign agent encapsulates the packet and transmits it over the established channel to the mobile node. Since Cashflow operates in the presented configuration at layer two, the mobile nodes appears link local to the foreign agent. Therefore, the routing within the mobile ad hoc network is from the viewpoint of the IP-layer no issue. To transmit data from the mobile node back to the corresponding node, the mobile node transmits the packet over the channel to the foreign agent, which tunnels the packet to the home-agent. The home-agent finally forwards the packet to the corresponding node over path 3.

An alternative approach is visualized in Figure 3.40, using mobile IPv6 in combination with co-located care-of-addresses. In this scenario, the mobile node performs a search for nodes providing Internet access in the range of three hops like in the scenario described before. The route search results in paths to two nodes providing Internet access and therefore acting as gateways between the mobile ad hoc network and the Internet. The mobile

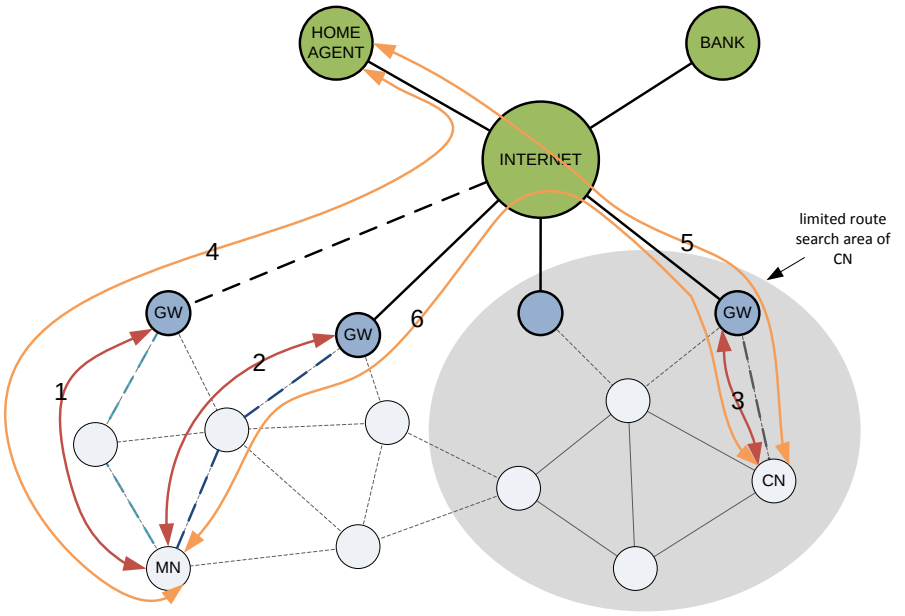


Figure 3.40: Internet connected ad hoc network using Cashflow and mobile IPv6

node opens a channel to each of the both gateway nodes and, in contrast to the scenario before, requests an IP-address from each of the gateway nodes. These addresses are used as co-located care-of-addresses, which are shown as connection 1 and 2 in the figure. In this scenario, also the correspondent node is part of the same mobile ad hoc network and possesses no direct Internet access. Therefore, this time also the correspondent node performs a search for nodes providing Internet services within a range of two hops, builds a channel to one of the nodes found during route search, and requests a care of address from it. After these procedures, the mobile node possesses three IP addresses: its fixed home IP address and two care-of-addresses. The corresponding node possesses two addresses, its fixed home address and one care-of-address, provided by the gate way node to which the corresponding node has opened a channel.

The mobile node uses one of its Internet gateways to bind itself to its home agent, by sending a binding update message, including one of its care-of-address. The home agent uses this care of address to forward packets to the mobile node. Even if it is not foreseen in the mobile IPv6 standard, the mobile node might sends its second care-of-address to its home agent as backup, if the channel to the first gateway breaks because of node move-

ment.

If the correspondent node, located in the ad hoc network, wants to communicate with the mobile node, it first performs a limited route search to verify, that the mobile node is not within a certain distance, in this scenario within a distance of two hops. To do so, the correspondent node uses the route request frame presented in this section, sets the search type field to 2, which is assigned to the search type *IPv6-address-search*, sets the maximum hop field to 2, to limit the route search to two hops, and places the home address of the mobile node into the parameter field to indicate the target node of the route search. If the mobile node would be in the range of two hops to the conrespondent node, the route discovery algorithm would return at least one path, which could be used for a direct communication within the mobile ad hoc network without using the Internet. However, in the given scenario the mobile node is not in the range of two hops to the correspondent node. Therefore, the routing algorithm returns no path. To transmit data to the mobile node, the correspondent node transmits packets with the home address of the mobile node as target over the Internet. The home agent of the mobile node receives the packets for the mobile node and forwards them along path 4 to the mobile node, using its care-of-address. The mobile node uses the care-of-address of the correspondent node to transmit packets directly to it. Additionally, to decrease the overhead, the mobile node performs a binding update with the correspondent node over path 6, which allows the correspondent node to send packets directly to the care-of-address of the mobile node. This allows communication between the mobile and the correspondent node, without using the home agent.

Summarizing, Cashflow allows to connect mobile ad hoc networks to the Internet using mobile IP. By allowing to be simultaneously connected to multiple foreign networks, the system improved the availability of the mobile nodes. Additionally, the credit infrastructure can be used to pay for Internet service. The introduction of limited route search makes the system scalable, if the Internet is used as backbone. Therefore, by connecting mobile ad hoc networks to the Internet, ad hoc networks with thousands of participants are possible. As site note, the newly introduced search frames allow in combination with Cashflow's credit system the introduction of a number of commercial services. For instance, printing services could be realized by defining a search class for printers, allowing to find printers nearby and pay for printing services using the credit system. To allow the integration of such services, a considerable large service type field is used by the newly introduced search frames.

3.6 Comparison of virtual currency systems

As mentioned in previous sections, Cashflow posses a number of features which are unique for virtual currency systems. Other features and concepts are also used by other virtual currency systems. Therefore there exists a number of similarities between Cashflow and other systems. For the sake of completeness, this section provides an overview over concepts and properties of virtual currency systems and compares them to Cashflow. These virtual currency systems include iPass [Chen04a], Nuglets [Buttyan01], Sprite [Zhong03] and Commit [Eidenbenz05].

Starting with the payment schemes, the virtual currency systems Sprite and Nuglets specify concrete solution for payment. Nuglets uses a real virtual currency called *nuglets* for payment. To pay for packet forwarding, nodes attach nuglets to packets. Before a node forwards packets for other nodes, it removes a certain amount of nuglets form the packet. The main benefit of this concept is that it needs no central instance. It adapts hard cash directly to mobile ad hoc networks. However, like in real life, the usage of hard cash has some disadvantaged. For instance, hard cash can be lost. In a mobile ad hoc network using Nuglets, the loss of a packet with attached nuglets automatically leads to the loss of the attached nuglets. Assuming that packet losses occur from time to time, the system could lose all nuglets, leading to a halt of the system if no compensation mechanisms are used by the system. However, Nuglets defines no mechanism to compensate lost nuglets. Another problem of this approach is that it requires tamper-proof hardware to prevent duplication of nuglets or other malign behavior like removing all nuglets from passing packets without forwarding them. The need of tamper-proof hardware limits the usage of Nuglets in scenarios where no special hardware is available. A further issue of this concept is that nodes have to estimate the amount of nuglets they need to attach to packets so that the packets receive their targets. If a packet is out of nuglets and has not reached the target, the packet gets dropped by intermediate nodes. If, on the other hand, the source node overestimates the needed amount, it donates nuglets to the target node. To overcome these issues, Nuglets introduced a buying scheme, where the target node pays for provided services. However, this leads to the question about the intention of the target node to pay for services used by the source node.

To overcome these issues, Sprite uses a credit system, which was adapted by Cashflow. As described in Section 3.2.3 the idea behind this system is that nodes keep receipts when they forward packets and transmit earned receipts to a central bank instance to get the earned reward. The drawback of this

concept is that special bank instances are needed for payment. However, this concept provides a number of benefits compared to the usage of virtual hard cash. The most important benefit is that this concept can be used without special hardware. Therefore, it allows to include mobile office equipment, like laptops and smart phones, into mobile ad hoc networks using a credit system for rewarding. Additionally, the loss of packets does not lead to a loss of units of the virtual currency like in Nuglets. Because of these benefits, Cashflow has adapted this concept, but in contrast to Sprite it charges for channels. This means that charging is done for a number of packets in bulk and not for every single packet separately. This reduces the overhead of the credit system, both in terms of transmission and calculation time. The virtual currency systems iPass and Commit does not explicitly propose payment schemes but they can be implemented using the credit concept.

Another difference between the virtual currency systems is the way they determine fees. Nuglets proposes two different mechanisms for pricing. One mechanism is a fixed price scheme, meaning that each node charges the same fix fee for packet forwarding. An alternative mechanism proposed by Nuglets uses a sealed bid second price auction to determine the next hop and the fee the next hop earns. Potential next hop nodes send a message including a bit to the forwarding node in a sealed way. The forwarding node selects the winner, which is the node with the lowest bid, as next hop. The price the winning node charges is the second lowest bid.

Sprite uses a kind of fixed pricing scheme, where the fee nodes earn for forwarding depend on the transmission success. Nodes which successfully have transmitted a packet, receive an fee α for packet forwarding, while if a node has not successfully transmit a packet, it only receives β as reward, whereby β is smaller than α. This is done due to game theoretical reasons, which are explained in detail in [Zhong03].

In contrast to Sprite, the virtual currency system iPass uses an auction scheme for pricing. Each node constitutes an auction market, where other nodes can bid for bandwidth. To do so, each packet contains a field with the transmission rate in bytes per second the source of the packet desires. Other fields hold the current rate assigned to the node and the node's bid. Each node has only a certain maximum bandwidth available, which gets assigned to the nodes with the highest bids. If the maximum bandwidth desired by a node is not available, the bandwidth, which is left for the node, gets inserted in the current rate field, if it is lower than the value inserted by previous nodes into this field. When the packet reaches its target, the

current rate field contains the rate of the bottleneck of the path. Using a feedback mechanism, the source node learns about the current rate along a path and, if the current rate does not match the desired rate, the source node can increase its bid to get more bandwidth assigned. Otherwise, if the current rate corresponds to the desired rate, the source node might reduce its bid, since this means that the source node is bidding higher than other nodes.

In Commit, the fee is determined by the energy used for data transmission along a path. When a node wants to transmit packets to a target node, a special routing algorithm is used to find the cheapest and the second cheapest path to the target. The fee of the second cheapest path is the fee the source node has to pay to transmit packets along the cheapest path to the target node. This is done to game theoretical reasons. However, before route search the source node defines the maximum fee it is willing to pay for data transmission and commits itself to transmit packets if the price lies beneath this value. So if the price of the second cheapest route lies above this value, the node does not have to use the path.

All of these schemes have in common, that the node's user has no direct influence on the price the node belonging to him charges for forwarding service. As described in Section 3.2.2, this issue is solved by Cashflow's pricing system, which allows to define a minimum fee the node has to charge for forwarding. The concrete fee a node using Cashflow charges gets determined using a market system, where the fee gets adapted based on supply and demand. Additionally the node's context influences determination of the fee.

The next distinctive feature is the ability of the virtual currency systems to support nodes to discover and use the cheapest path. The virtual currency systems Nuglets, Sprite, and iPass only allow the user to determine the price for a given path. They include no function to find the cheapest path between a source and a target node. Since it is in the interest of the user to pay the least fee for a service, this issue is problematic for user acceptance. In Commit the discovery of the cheapest route is part of the fee determination. Therefore, nodes always transmit packets along the cheapest paths. Also Cashflow allows nodes to use the cheapest path to other nodes. This is realized using a special routing algorithm, which additionally allows nodes to search for the cheapest path fulfilling certain stability criteria. In contrast to Commit, nodes are not bound to use the cheapest path. The selection of the path is completely in control of the nodes. Cashflow only provides the functionality to locate the cheapest path.

	Nuglets	Sprite	iPass	Commit	Cashflow
How to pay	+	+			+
How much to pay	+	+	+	+	+
Find cheapest route				+	+
Reacts on network load			+		+
Includes external fee parameters					+
variable participation degree					+

Table 3.2: Overview over provided functionality of different virtual currency systems

The reaction on network load is another distinctive feature. The load situation is not considered by the systems Nuglets, Sprite, and Commit. Therefore, it can happen that single nodes or parts of the network get overloaded. The virtual currency system iPass prevents local overload, since when a node has no bandwidth left, it drops packets from nodes with low bids, with the result that nodes have to increase their bid to get bandwidth. Therefore, the occurrence of local high load leads to an increase of the price level. However, since iPass provides no mechanism to find the cheapest route through the network, the increase of the price does not lead to the usage of alternative paths by nodes for data transmission. In Cashflow, the local load directly influences the fee nodes charge. Additionally, there exists a mechanism, which allows nodes to use the cheapest path to a target. The combination of these two mechanisms leads to a load balancing system, which will be analyzed in Section 4.3.4.

Additionally to the inclusion of local load, also the ability to include other external parameters is a distinctive feature of virtual currency systems. For instance, the ability of virtual currency systems to charge a higher fee if a node is running on battery. Only Cashflow allows the inclusion of external parameters into the fee calculation. Furthermore, because the system supports the discovery of the cheapest route though the network, also the routing system reacts indirectly on these external parameters.

The last characteristic, which is also unique to Cashflow, is its ability to allow users to decide to which degree they want to participate in the network. The user can define a preferred and a maximum throughput, which

is considered by the system. In other virtual currency systems, the user is not able to influence its participation degree, even if this is an important feature for the user acceptance of virtual currency systems.

To summarize this comparison, Table 3.2 gives an overview over the discussed characteristics.

3.7 Summary

In this chapter, we have presented Cashflow, a virtual currency system for mobile ad hoc networks. We started with an analysis of different usage scenarios for Cashflow by formulating assumptions and requirements. Based on this analysis, the concepts used by Cashflow were discussed. These include the channel concept for data transmission, the market concept for pricing, and the credit concept for payment. The main part of the chapter described the architecture of Cashflow, its modules, and its algorithms. Worth mentioning is the algorithm proposed for the routing module, which allows to search the cheapest path through the network considering link quality restrictions. The algorithm was optimized using artificial delaying to reduce the routing overhead. After the description of Cashflow different relevant security considerations were discussed. Then we proposed a concept based on mobile IP to integrate mobile ad hoc networks using Cashflow into the Internet. The last part of this chapter compared Cashflow to other virtual currency systems showing its strengths and weaknesses.

Chapter 4

Evaluation

This chapter [1] presents the simulation based evaluation of the proposed route discovery algorithm as well as the presented architecture. As simulation environment the IBKSIM [Wallentin10c] in combination with its wireless network simulation library [Wallentin08] was used.

The evaluation of the route discovery strategy focuses on the influence of the delay parameter on the overall performance of the algorithm. This includes the influence of the parameter on the number of packets send by network nodes during route search and on the time, the algorithm needs for route discovery. Additionally, the impact of the density of the network and the variance of the processing time nodes need to handle route discovery packets will be presented.

The architecture presented in this thesis is evaluated focusing on the effects of its load balancing and access regulation mechanism on the overall throughput of the network. Section 4.1 gives an overview over the simulation environment including a description of the environment parameters used for all simulations. Section 4.2 describes the simulation scenarios used for the evaluation of the route discovery strategy, followed by a detailed discussion of the simulation results. The architecture is evaluated in Section 4.3. Again, the simulation scenarios are described and the outcome of the simulations is analyzed. In Section 4.4 the chapter concludes with a summary.

[1] Parts of this chapter have been published in [Wallentin10b] and [Wallentin10a]

4.1 Simulation enviroment

For the evaluation of the route discovery algorithm as well as the proposed architecture, the IBKSim simulation enviroment was used. This discrete event based simulator was developed at the Institute of Broadband Communications of the Vienna University of Technology as successor of the IKNSim. The IBKSim distinguishes itself from other simulators like NS2 [Institute10] and OMNET++ [Omn10] by its usage of XML [Bray00] as configuration and logging language. Since XML is a textual data format, the usage of this format allows the user to develop simple scripts to generate and perform large simulation series, as well as to automatize the post processing of the simulation results. The IBKSim consists of a small simulation kernel and a number of libraries for specific simulation task. One of these libraries is the wireless network simulation library [Wallentin08]. This library was especially developed for the simulation of wireless networks. It supports the simulation of the physical environment of nodes using radio for communication, including the simulation of mobility and radio propagation. Additionally, it implements the 802.11 MAC format [IEEE99] and the 802.11b [IEEE00] standard on the physical layer.

The proposed architecture was developed and consequently enhanced using IBKSim and the wireless library. Therefore, the complete architecture of Cashflow is implemented as simulation module in the simulator. As a consequence, it is not only possible to investigate single components of the architecture, but also to evaluate the complete architecture as a whole, including the effects of the interplay between components.

In the IBKSim simulation environment, physical entities like mobile devices are represented by nodes. A node consists of at least one simulation object, a so-called component. The wireless library uses this concept to organize simulation components in a layered form, whereas components might include additional modules to simulate a specific behavior. Within a node, components can only communicate directly with other components representing the layer directly beneath or above of them. The simulation of the inter node communication is handled by the wireless library.

Figure 4.1 gives an overview of the different simulation components, including their modules, used for the simulation of mobile nodes running Cashflow. The *Wlan*, *Maclayer*, and *Shell* components represent the physical, the data, link and the network layer of the OSI model [OSI84]. The application container represents all layers above the network layer.

The main functionality of the application container in the simulation is to

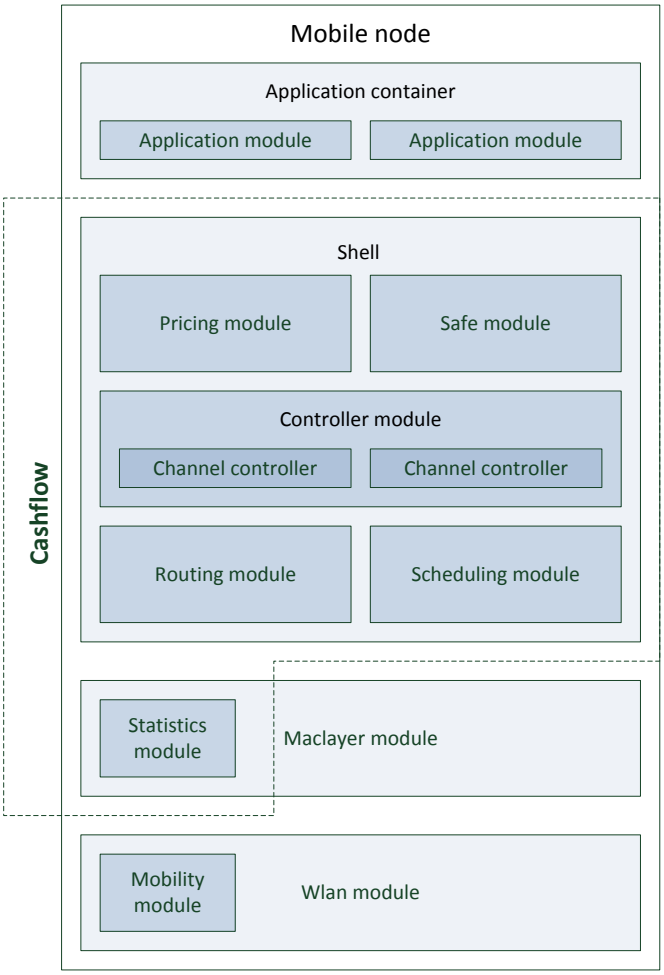


Figure 4.1: Architecture of a node in the IBKSim simulation enviroment.

start and configure different application modules, which trigger functions of Cashflow for evaluation. Depending on the scenario, they triggering route requests, request channels, or generate traffic.

The shell component simulates the core components of Cashflow. Therefore, the different modules included in the shell component correspond to the modules of the node's architecture as described in Chapter 3.3. Only the statistics module is implemented for the simulation as additional module directly into the maclayer module, since an existing module already provided nearly all the needed functionality. During the development of Cashflow, not only the final simulation modules have been implemented,

Parameter	Value
radio model	shadowing model
sensitivity	3.98107e-11 W
transmission power	0.079432 W
antenna gain	1
system loss	1
path loss exponent	3
shadowing deviation	4dB
radio frequency	2472 MHz (Channel 13 of the 802.11b standard)
bit rate	11 Mbit/s
transmission protocol	IEEE802.11

Table 4.1: Parameters of the radio simulation

but also a number of dummy modules. These modules posses not the complete functionality of the corresponding modules of Cashflow, but are useful for functionality testing.

The wireless library of IBKSim provides an implementation of the 802.11b protocol as maclayer component. This component has been used for the evaluation of the system. It was extended by an interface to allow the inclusion of statistic modules into the maclayer module. For evaluation, the statistic module suggested in Cashflow was implemented.

The wlan component simulates the radio channel. The wireless library provides two implementation of this module type [Wallentin08]. The basic component provides a simple model, where two nodes can communicate if they are within a certain distance and no other node is using the channel. The advanced component provides a more realistic model of the radio propagation by using different radio propagation and bit error models. The movement of the nodes is modeled by mobility modules. The wireless framework offers a number of different mobility modules, but for the evaluation only the random-waypoint and the no-movement module were used.

All simulations performed for evaluation have in common that the same parameters for the simulation of radio transmissions were used. Table 4.1 gives an overview over these parameters. To estimate the performance of the system using different parameter values, it is important to establish understanding of the influence of radio parameters on the simulation model. As radio model the shadowing model, as described in [Rappaport99], was used. This is the most realistic radio model implemented by the wireless simu-

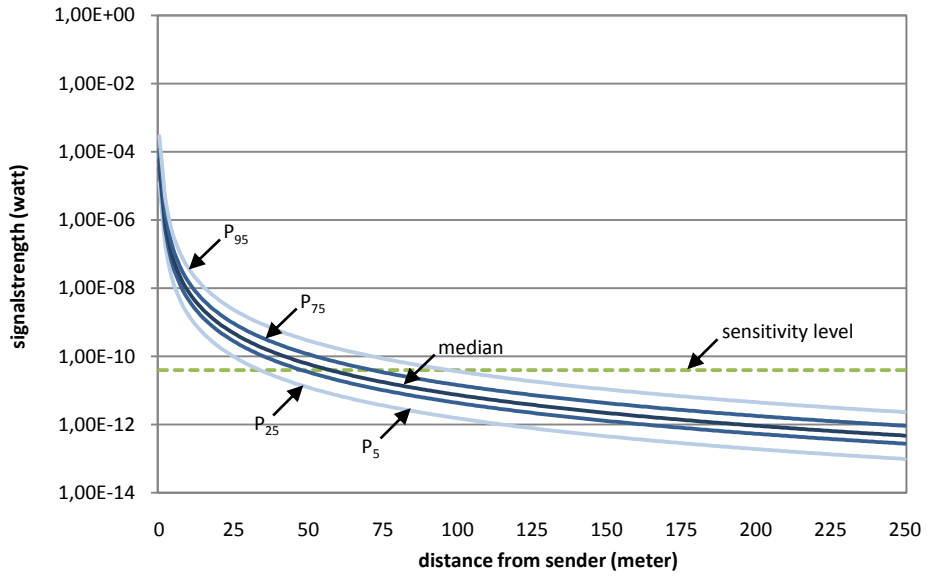


Figure 4.2: Statistical analyses of the signal strength as function of distance

lation library. For the evaluation of the architecture, it was configured to model the radio propagation in urban area. Therefore, a path loss exponent of 3 was used, as well as a variation parameter of 4 dB. For the configuration of the radio equipment, the parameters of the 3Com OfficeConnect Wireless 54Mbps 11g Compact USB Adapter [3Com-Corporation06] were taken. This radio unit has a sensitivity of 3.981071E-11 watt or -74 dBm and and an output of 0.079432 watt or 19 dBm for a 802.11b signal with a transmission speed of 11 Mbps. The gain and system loss parameters of the radio modules are set to 1, so no additional gain or loss is assumed. The last parameter is the radio frequency used for transmission, which is set to 2472 MHz, which corresponds with channel 13 of the 802.11b standard [IEEE00].

Figure 4.2 pictures on an exponential scale the signal strength distribution of a sender using the parameters described above, depending on the distance between the sender and the measurement point as. Additionally, the sensitivity level is depicted by a dotted line. The curves marked with P95 and P5 depict the 95th and 5th percentiles. P25 and P75 mark the 25th and 75th percentiles, which are also the first and third quartiles. The median, which is represented by the curve between the first and third quartile, crosses the sensitivity level at a distance of 61 meter. This means that at 61 meter

50 percent of the transmissions have a signal strength which lies above the sensitivity level of the receiver and can therefore be detected. It is worth noting that even if a receiver detects the signal of a transmission, it does not necessarily means that the signal is received correctly and consequently the carried message can be decoded.

To calculate the signal strength at a certain distance from the sender using the shadowing model, a reference signal strength at a known distance is needed. Using the free line of sight model [Rappaport99], the transmission power, the signal frequency, the system loss, and the gain are needed to calculate a reference signal strength. Consequently, the values of these parameters have an influence of the level of the curves, meaning that a change of the values result in a up or down movement of the curves. The path loss exponent has an influence on the tenor of the curves. The higher the exponent, the faster the curves decreases when the distance to the sender increases. The variation parameter has an influence on the distribution of the received packets signal strengths. An increase of the parameter results in an increase in the variation. This has an influence on the area in which the transmission probability switches from one to zero. It is worth noting that the simulation model prevents a distance between nodes beneath 0.5 meter by treating the nodes as if they were 0.5 meters away from each other. This is done to avoid unrealistic situation that two physical transmitter share exactly the same position. Therefore, the simulated signal strength measured at a distance of 0 meter differs from the value the shadowing model would lead to.

Figure 4.3 presents the transmission probability of data packets using the request to send / clear to send (RTS/CTS) method in comparison to broadcast, depending on the distance between sender and receiver for the given radio model parameters. The transmission probability for broadcast at a certain distance is similar to the probability that the signal strength of the transmission lies above the receiver's sensitivity level at the same distance. If this is the case, the signal can be detected. Due to the fact that a low signal to noise and interference ratio (SNIR) increases the bit error probability, the probability of receiving a valid broadcast frame with an encapsulated packet lies beneath the signal detection probability. In the given scenario, the difference between signal detection probability and the probability to receive a valid broadcast frame is minimal, since the background noise is small, and there is no additional interference caused by other nodes.

Using broadcast, the probability that a packet is transmitted correctly and the probability that the frame in which the packet is encapsulated is valid

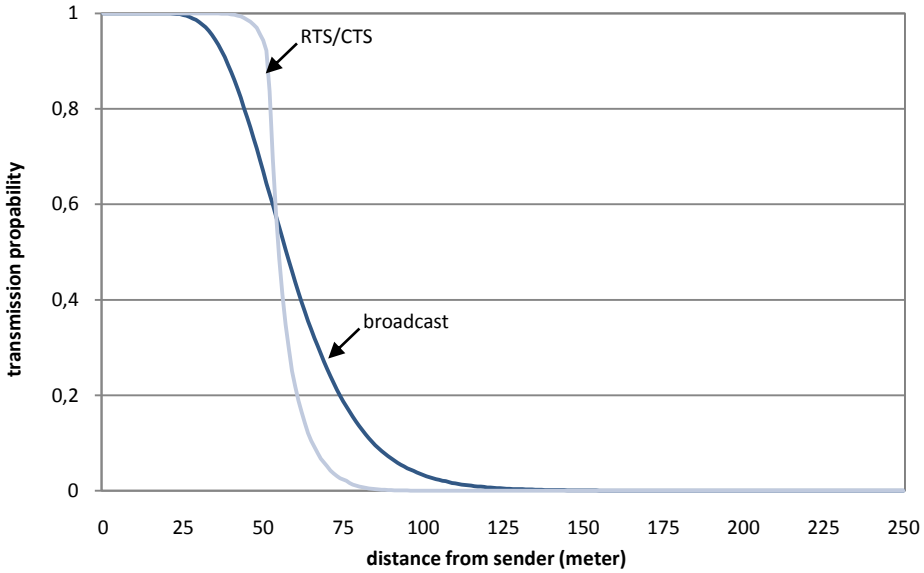


Figure 4.3: Transmission probability as function of distance

received is the same. This is not true if the packet is transmitted to a specific node using the acknowledgment mechanism of the 802.11 MAC protocol as well as the request to send / clear to send (RTS/CTS) method. In this case, at least four frames have to be exchanged for a complete transmission. Since the acknowledgment mechanism detects invalid transmissions and consequently retransmits frames, this mechanism tolerates if the target node does not correctly receive some frames. Consequently, the probability that a packet reaches its target node is higher compare to broadcast, if the transmission probability is sufficiently high. In the given scenario, until a distance of 45 meter the transmission probability is over 0.99 when the acknowledgment mechanism is used. At the same distance, the transmission probability for broadcasts has dropped to 0.86. When the distance increases, the transmission success probability of a packet using the acknowledgment mechanism drops faster than the transmission probability for broadcast. This leads to a crossing of the curves at the distance of 56 meter. At 79 meter, the transmission probability for the acknowledgment mechanism drops below 0.01, contrary to the transmission probability for broadcast, which lies above this value until the distance of 118 meter.

This information about the nodes transmission range formed the basis for the development of the simulation scenarios.

4.2 Evaluation of the route discovery protocol

The routing algorithm proposed in Chapter 3.3.3 uses artificial delay to reduce the flooding overhead. The value of the delay depends on the parameter D_{max}, stating the maximum delay used by the algorithm. This section presents the evaluation of the parameter's influence on the overall performance, as well as the influence of the network density and the nodes processing time. Additionally, the influence of the delay on the route requests propagation is visualized using series of heat maps.

4.2.1 Simulation scenarios

To examine the effects of the delay parameter as well as the influences of the network density and the variation of the nodes processing time on the overall performance, six different simulation scenarios were defined.

In the simulation scenarios 1a to 1c, 1000 nodes are uniformly distributed over an area of 1000 x 1000 meter, whereas in the simulation scenarios 2a to 2c the area is reduced to 500 x 500 meter. As result, the mean distance between nodes is smaller in scenario 2a to 2c, with the consequence that in average every node has more direct neighbors and the mean length of the shortest paths between nodes, in terms of hops, is smaller than in scenario 1a to 1c. In the simulation scenarios 1a and 2a, the nodes processing time is uniformly distributed between 1 and 2 milliseconds, for 1b and 2b between 1 and 4 milliseconds and for 1c and 2c between 1 and 10 milliseconds. Table 4.2 gives an overview over the different scenarios.

For each scenario, the delay parameter D_{max} was altered between 0 and 0.05 seconds in steps of 0.1 milliseconds which leads to 500 different configurations. When D_{max} is set to 0 the algorithm uses no artificial delaying and the simulation results of this specific configuration can be used as reference

		size of the area	
		1000 x 1000 m	**500 x 500 m**
processing time	**U(1,2) ms**	scenario 1a	scenario 2a
	U(1,6) ms	scenario 1b	scenario 2b
	U(1,10) ms	scenario 1c	scenario 2c

Table 4.2: Processing time and areal distribution used for the different simulation scenarios

to compare the delay's effect on the overall performance.

Using the described scenarios, two simulation series were performed. The first series focuses on the performance of the algorithm when used to find the cheapest route through a network. In the second series, the route discovery algorithm is used in combination with the statistics module of Cashflow, to find the most stable route between nodes.

4.2.2 Fee based routing

For the first simulation series, the nodes were configured to charge whole-number fees between 1 and 10 units for the forwarding packets. The uniformly distributed values were assigned randomly to the nodes.

For the calculation of the delay d the following formula was used:

$$d(f) = \frac{f \times D_{\max}}{10} \tag{4.1}$$

In this formula, D_{max} is the delay parameter and f the fee nodes charge. It is the simplification of Formula 3.11 presented in Section 3.3.3 where F_{min} is set to 0, F_{max} is set to 10 and $D_{maxQuality}$ is set to 0. D_{maxFee} in the original formula corresponds to D_{max} in the formula above.

From the 1000 nodes, which were uniformly distributed over the simulation area, one node was configured to hold an application module in the application container module. This application module triggered the route request to another node. All nodes in the network logged the number of packets sent and received, as well as the number of detected collisions. Additionally the time until the first and optimal route was discovered was recorded. For every one of the 500 configurations per scenarios, 20 runs have been performed, using different seeds for random variables.

Starting with the results of the simulation series using the 1000 x 1000 meter area, Figure 4.4 depicts a comparison of the average number of packets sent per node depending on the delay value for scenario 1a, 1b and 1c. In all scenarios, the average number of packets sent per node tends to decrease with increasing delay until the mean approximates the value 1. This means that in average for every route search approximately one packet is broadcasted per node. The rate by which the number of packets sent per node decreases as the delay increases varies depending on the scenario. Likewise, the average number of packets sent without the artificial delay varies depending on the scenario. In scenario 1a, if no artificial delay is used, every node sends 2,967 packets in average, in scenario 1b 5,456 packets

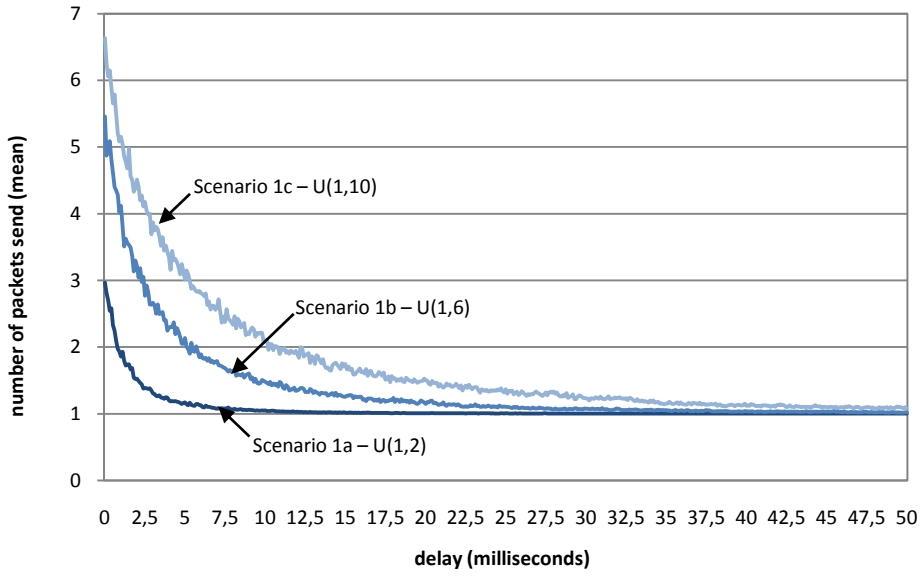


Figure 4.4: Influence of delay on average packet number send by nodes, using an area of 1000 x 1000 m

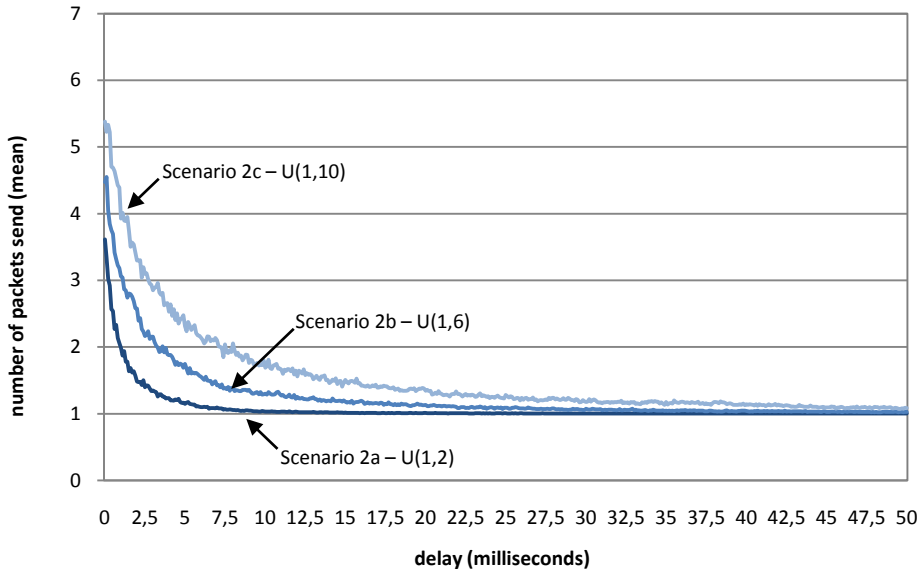


Figure 4.5: Influence of delay on average packet number send by nodes, using an area of 500 x 500 m

and in scenario 1c 6,635 packets. This difference is caused by the influence of the processing time on the collisions probability, which will be explained later in detail.

The effect that the number of packets sent in average approximates faster the value 1 in scenario 1a than in scenario 1c is also a direct result of the different node processing time variations. A high variance increases the probability that even if a packet from node A is delayed longer by the route discovery protocol than a packet from node B, node A might broadcast the packet before node B, even if this is not in the intention of the route discovery algorithm.

Figure 4.5 presents a comparison of the average number of packets send per node in scenarios 2a, to 2c, using an area of 500 x 500 meter. Similar to the series using a 1000 x 1000 meter area, the average number of packets sent per node decreases until it approximates the value 1. It is again observable that with an increase of the processing time variance, the average number of packets sent increases, as consequence of collisions. If no delay is used, in scenario 2c and 2b the average number of packets sent lies beneath the corresponding values of scenario 1c and 1b. This can be explained by the network density, which in these scenarios is four times higher than in the scenarios 1a, 1b and 1c.

Figure 4.6 pictures the average number of detected collisions for the scenarios using 1000 x 1000 meter area. In all three scenarios, an increase of the artificial delay results in a decrease of the number of detected collisions. It is important to note that the figure includes no information about the absolute number of packets lost due to these collisions, since it makes no difference from the receiver's point of view, how many packets are involved in a collision. When no delay is used, in scenario 1a 8.56 collisions are detected in average by every node of the scenario. This value increases to 13.62 for scenario 1b and 13.87 for scenario 1c.

The effect that the number of collisions increases when the node's processing time variance increases can be explained due to the effects the variance causes on the flooding process during route discovery. For a single packet, the probability to get lost due to a collision decreases as consequence of the increase of the processing time variance. Therefore one would might expect that the average number of collisions detected by the network nodes decreases with an increase of the processing time variance, which is exactly the opposite of the outcome of the simulation as stated above. The increase of collisions is the consequence of the traffic increase in the network, caused by the route discovery protocol as the processing time variance increases.

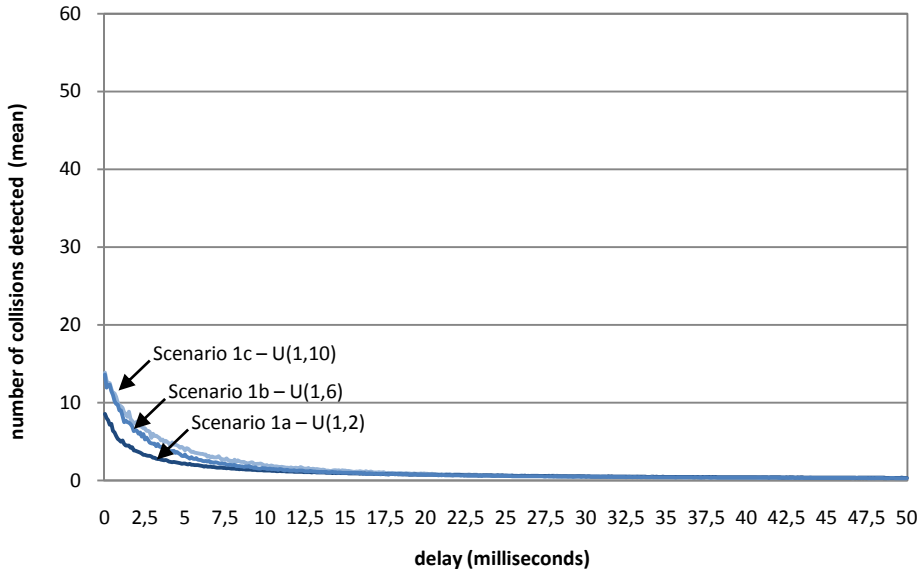


Figure 4.6: Influence of delay on average number of collisions detected by nodes, using an area of 1000 x 1000 m

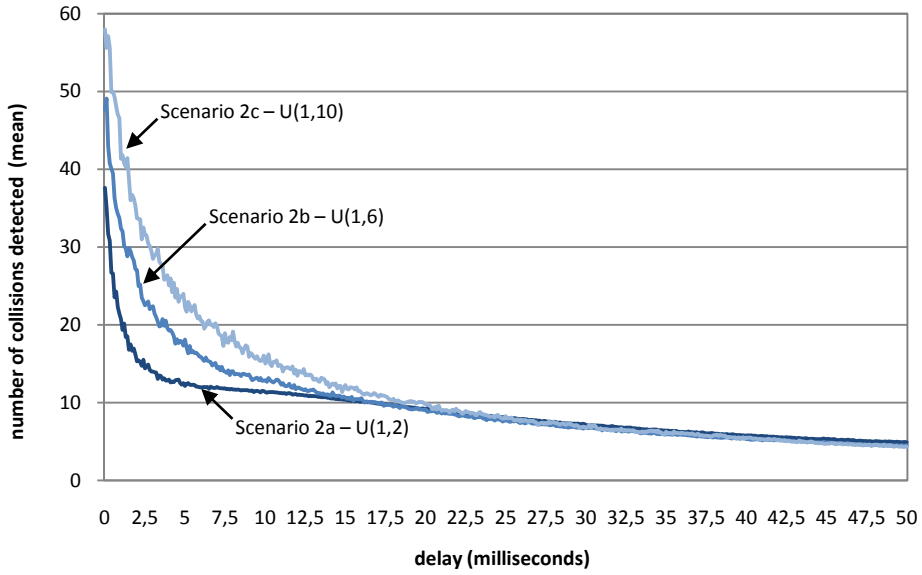


Figure 4.7: Influence of delay on average number of collisions detected by nodes, using an area of 500 x 500 m

This can be seen in Figure 4.4.

The processing time variance reduces additional to the backoff algorithm the probability that two nodes access the shared medium simultaneously, which results in collisions. This has an influence on the average number of nodes receiving a broadcasted packet of a node. If there are no collisions, all nodes within the footprint of a node receive the broadcasted packet. If collisions occur, part of the nodes does not receive the packet. For the route search however, the loss of a broadcasted packet is the same as discarding a potential path. Therefore, if a broadcasted packet is received by 15 instead of 40 nodes because of collisions, 25 potential paths are lost and consequently not considered during route search. Therefore, the quality of route search decreases if collisions occur, since only a subset of potential routes is considered. If the collision rate decreases and more paths are considered during route search, the probability that a node using the route discovery algorithm presented in Chapter 3.3.3 without artificial delaying has to rebroadcast route request packets increases. This causes additional traffic as it can be seen in Figure 4.4 comparing the average number of received packets for the three scenarios, when no delay is used. The consequence is an increase of the absolute number of registered collisions. The relative number of detected collisions in terms of detected collisions per detected packet decreases when the processing time variance increases.

As mentioned before, with an increase of the artificial delay the collision probability decreases. The artificial delay influences the collision probability in two ways. First, similar to the processing time variance, an increase of the artificial delay additionally scatters the points in time when nodes try to access the shared medium. Additionally, and this is the second effect of the artificial delay on the collision probability, an increase of the delay causes an decreases of the traffic, since nodes wait before they forward a route request. If a node receives a route request packet during the waiting time, it either drops the original route request packet in the case that the newly received packet contains information about a better route, or, in the case that the original packet which caused the begin of the waiting time includes information about a more preferable route than the newly received packet, the new packet gets dropped. Both cases result in a decrease of the traffic. With an increase of the delay, this effect gets amplified as described before.

Figure 4.7 presents the average number of detected collisions per node for scenario 2a, 2b and 2c. The collision number in these scenarios, using an area of 500 x 500 meter, lies over the values of the corresponding scenarios if

an area of 1000 x 1000 meter is used. The increase of the collision number is caused by the density increase. It is observable that the effect of the processing time variation becomes negligible when the delay becomes big enough. This is also true for the scenarios using the 1000 x 1000 meter area, but because of the smaller number of collisions, it is not as eye-catching as in these scenarios.

Figure 4.8 presents the influence of the artificial delay on the ratio between sent and received packets. The figure depicts the average number of nodes a node can reach using broadcast. The value depends on the transmission range of nodes, the node density, and the probability that packets get lost due to collisions. Since the average number of collisions approximates zero when the artificial delay increases, we can conclude for the scenarios using an area of 1000 x 1000 meter that in average every node has 11 other nodes in its radio footprint. If no or a small artificial delay is used, part of the packets get lost due to collisions, with the consequence that not all possible paths are considered during route search. By comparing the ratio between sent and received packets with the value the curve is converging to it is possible to estimate the number of possible connections not considered during route search per broadcast for a specific artificial delay value. When no delay is used, in scenario 1a 4.9 out of 11 connections are considered in average during route search per broadcast, 6.4 out of 11 for scenario 1b, and 7.5 out of 11 for scenario 1c. Again, the effect of different processing time variations on the collision rate is observable.

When 1000 nodes are distributed over an area of 500 x 500 meter instead of 1000 x 1000 meter, the average number of reachable nodes per node increases, as shown in Figure 4.9. As described before, because of the higher density the collision rate is higher than in the scenarios using 1000 x 1000 meter. We can conclude from Figure 4.9 that in the case when no collisions occur, in average every node can reach more than 35.17 nodes. It can be estimated by extrapolating data from figure 4.8 that the actual value of the average number of nodes in the footprint of a node is about 44. When no artificial delay is used, for scenario 2a in average 9.52 connections to neighbor nodes are considered during route search per broadcast, instead of 33.35 when a delay of 50 milliseconds is used. Similar in scenario 2b, where 11.16 connections in average are considered in contrast to 34.45 with the usage of the same artificial delay. In scenario 2c, 13.73 connections are considered per broadcast in average, 35.17 when an artificial delay of 50 milliseconds is used.

Figure 4.10 pictures the average number of received packets per node for

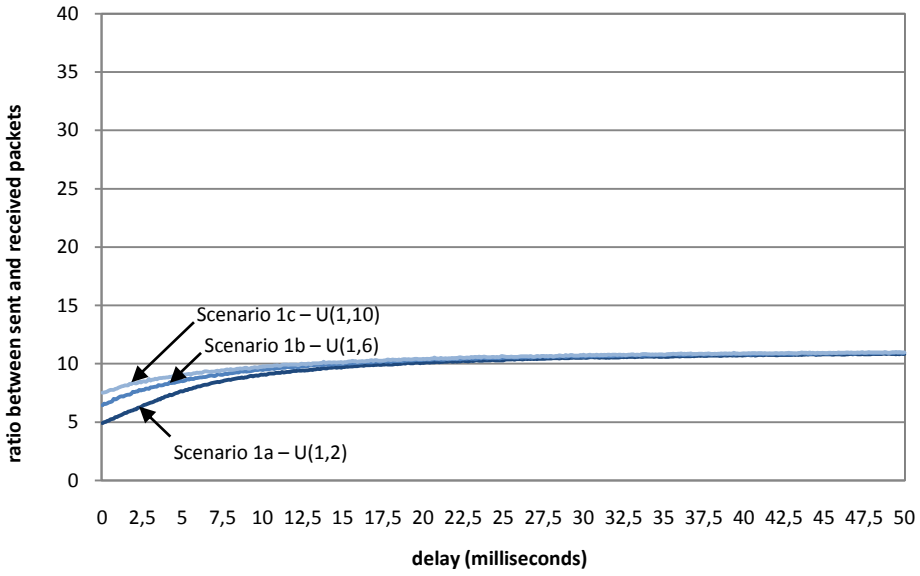


Figure 4.8: Ratio between average number of send and receives packets per node, using an area of 1000 x 1000 m

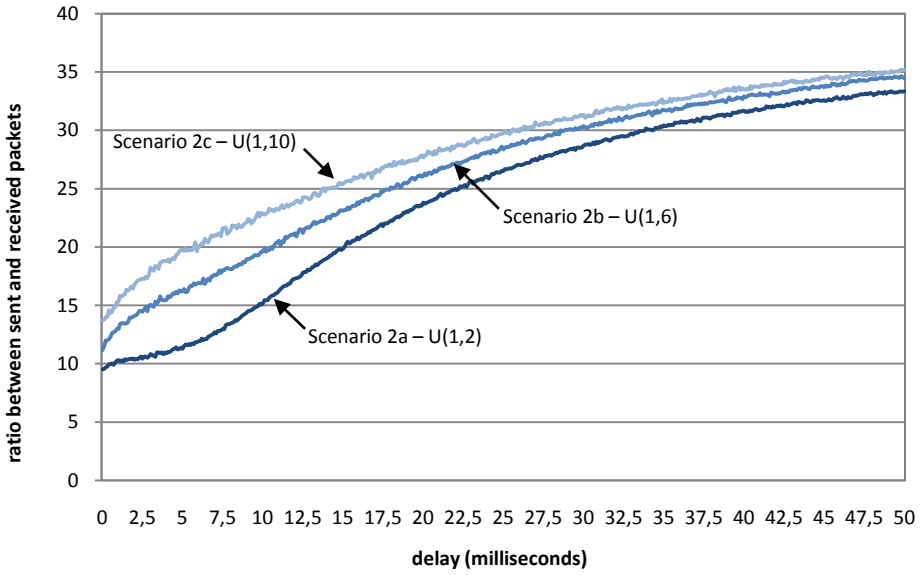


Figure 4.9: Ratio between average number of send and receives packets per node, using an area of 500 x 500 m

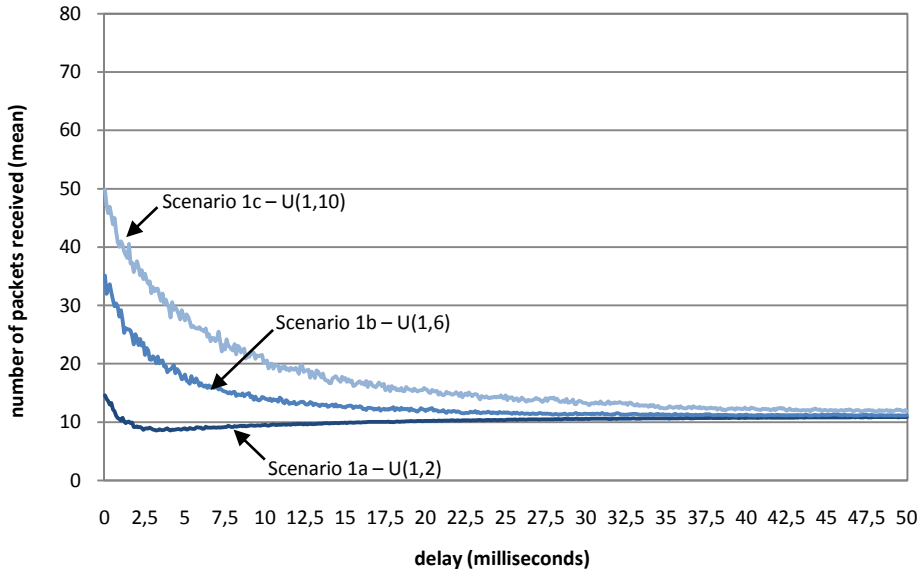


Figure 4.10: Influence of delay on average packet number received by nodes, using an area of 1000 x 1000 m

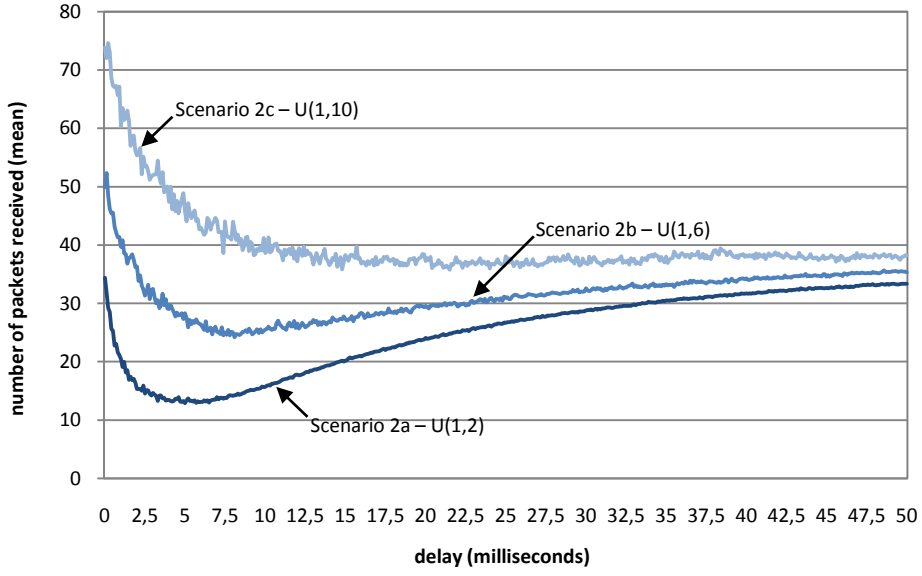


Figure 4.11: Influence of delay on average packet number received by nodes, using an area of 500 x 500 m

scenario 1a, 1b and 1c, depending on the delay. With an average of 49.81 received packets per node when no artificial delay is used, scenario 1c has the highest reception value, followed by scenario 1b with 35.1, and scenario 1a with 14.6 packets in average. Since with an increase of the delay, the average number of packets sent approximates the value 1 and the collision rate approximates 0 in all three scenarios, the average value of received packets approximates the value 11, which is the average number of nodes in the footprint of a node as described before. Also in Figure 4.11, picturing the average number of received packets per node when the nodes are distributed over an area of 500 x 500 meter, all three curves approximates the value of the average number of nodes in the footprint of nodes, even if in these scenarios it is not as visible as in the scenarios 1a to 1c. The effect of artificial delaying used by the route discovery protocol on the average number of packets send per node as well as on the collision rate is visible in all three curves, especially in the curve progression of the scenario 2a's curve. First the average number of packets received per node decreases, mainly as result of the decrease of the average number of packets send per node. When an artificial delay larger than 5 milliseconds is used the average number of received messages increases again, because the additional delay decreases the collision probability as described before.

The artificial delay also influences the time the route discovery protocol needs for route search. Figure 4.12 pictures the median of the time passed until the first route between a source and a target node was found for the scenarios 1a to 1c. Since the positions of source and target node are randomly chosen in these scenarios, the number of hops between these nodes varies greatly between the different simulation runs. This is the reason, why the curves pictured in this figure are not smooth. Never the less, it is observable that the average time until the first route is found increases linearly when the artificial delay increases. The offset between the curves is caused by the different processing time variations used in the scenarios. The higher the average processing time, the longer it takes the route discovery process to find the first route to the target node, independent if a artificial delay is used or not. As shown in Figure 4.13, the same is true for scenario 2a to 2c where the area is reduced to 500 x 500 meter. In addition, a linear increase of the route discovery time is observable, but this time the inclination is not as high as in scenario 1a to 1c. The reason for this effect is that the nodes are distributed over a smaller area and therefore the number of hops between source and target node is in average smaller.

Besides the time needed to discover the first route, the time needed to discover the best route is of interest. Since the target node only returns a

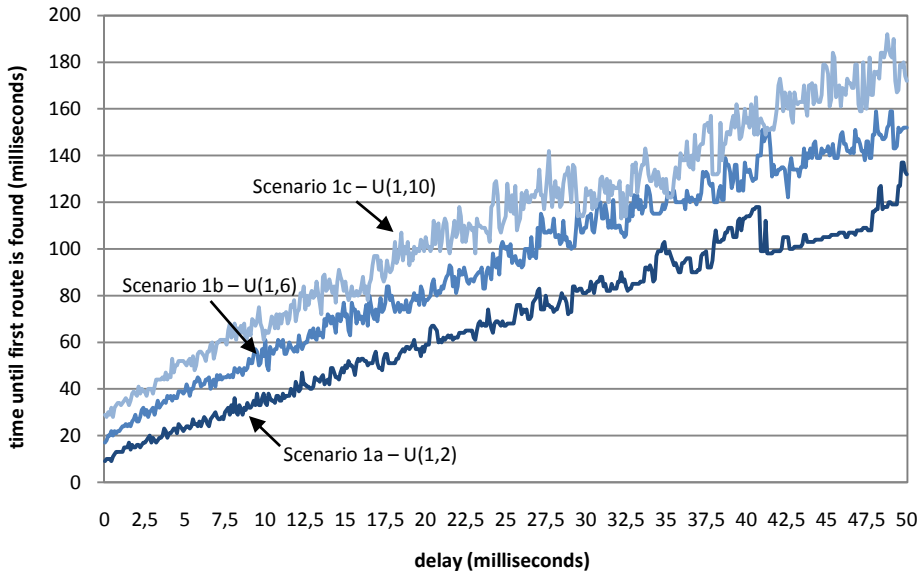


Figure 4.12: Time until first route to target is found, using an area of 1000 x 1000 m

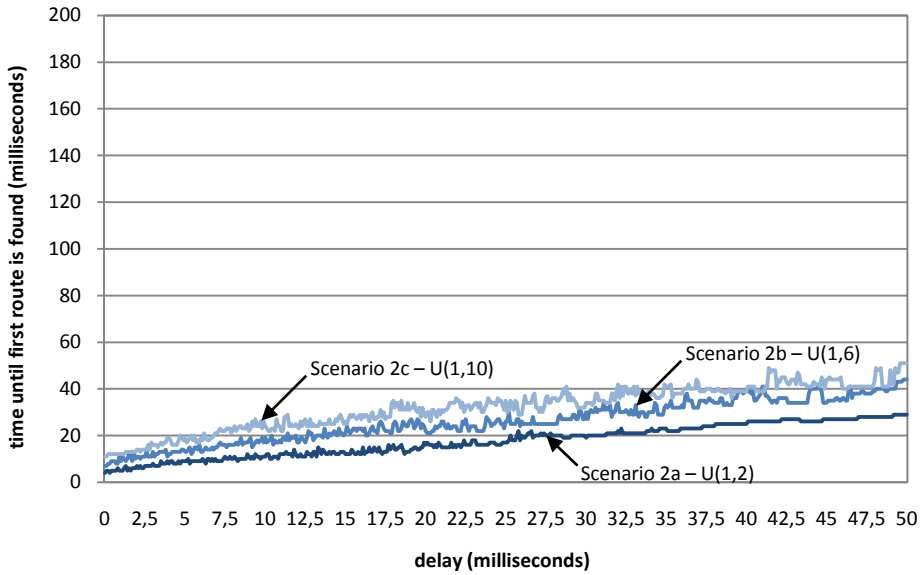
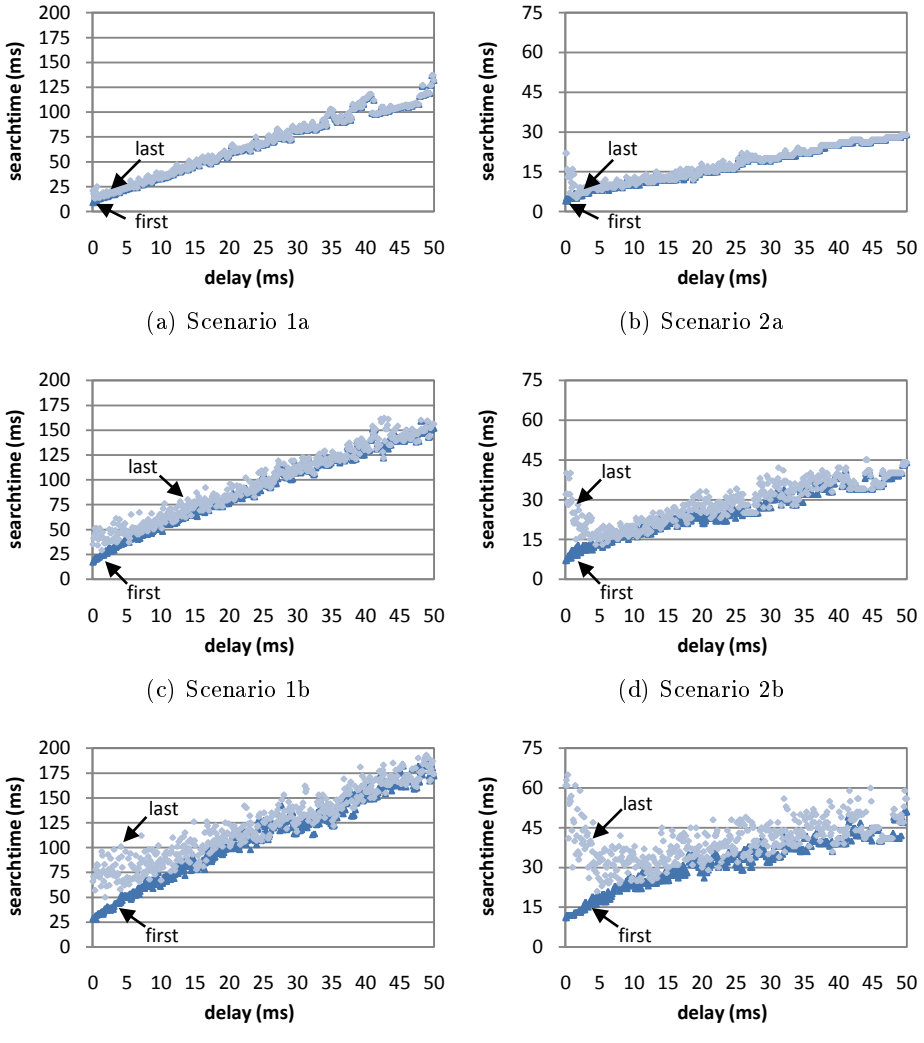


Figure 4.13: Time until first route to target is found, using an area of 500 x 500 m



(a) Scenario 1a

(b) Scenario 2a

(c) Scenario 1b

(d) Scenario 2b

(e) Scenario 1c

(f) Scenario 2c

Figure 4.14: Comparison of the time until the first and the last route was found for scenarios 1a to 1c and 2a to 2c. (scales for scenario series 1 and 2 differ.)

routing message to the source node if the first route or a better route than the previous found round is detected, the best route will consequently be the last route the route discovery protocol detects during a search from the source node's point of view. The diagrams in Figure 4.14 compare the time until the first and the best route was found in average for scenario 1a to 1c and 2a to 2c. In all six scenarios, the time until the first route was found approximates the time until the best route was found. That means, that if the delay is big enough, the route discovery algorithm returns only one path to the sender, and this one path is the best path available between source and target node. Additionally, the influence of the processing time variation is observable. The higher the processing time variance, the higher the variance of the time the algorithm needs to detect the best route available in the network.

Summarizing, the simulation series have shown the influence of artificial delaying on the route discovery process based on the fee nodes charge. The usage of artificial delay results in a decrease of the number of packets needed for route search and, at the same time, increases the quality of the route search. This is done by reducing the number of collisions during route search. Since collisions cause the exclusion of potential routes during route search, a reduction of collisions increases the route discovery algorithm's quality.

4.2.3 Quality based routing

After analyzing the influence of the artificial delay on the performance of route discovery using fee as main routing parameter, this section focuses on the artificial delay's influence if link quality is used as main routing parameter. Cashflow's statistics module monitors the quality of links to neighbor nodes and classifies the link depending on the average received signal strength, transmission failures and time. A detailed description of the classification algorithm is given in Section 3.3.8. In short, the statistics module divides the links intro three classes: When a new link is detected, the link automatically belongs to class 3, the bottom quality calls. If the signal quality of a certain number of received signals is above a certain level and if no transmission problems occurred for a certain time, the link's class is upgraded to class 2. However, if the signal losses quality over time, the link gets again classified as class 3 link. If a link belongs to class 2 for a relative long time, in this context for some minutes, the link's class is upgraded to 1, marking links with good quality and stability in the time domain, since a good connection over a long time indicates that the two

nodes connected by the link have nearly no relative movement or at least stay within a certain distance.

In general, it is in the interest of nodes to build channels along links of class 2 or 1, since links of class 3 are in most cases unstable, which could result in additional overhead caused by retransmission of frames, in frame loss, or in the worst case in channel disruption. Retransmissions and frame loss are also possible if class 2 or 1 links are used, but the probability is lower. Concerning channel disruption, the probability at class 2 links is lower than at class 3 links but higher than at class 1 links. The difference between class 2 and 3 can be explained by the difference of the transmission probability, since a dropped frame is interpreted by the system as channel disruption. A channel can also break because of node's movement. This explains the difference between class 2 and 1 links, since the long lifetime of class 1 links indicates nearly no relative movement.

The simulation series used to analyze the influence of artificial delay on quality based routing are nearly identical with the simulation scenarios used for purely fee based routing as described in Section 4.2.1. The only difference is that in contrast to the simulation scenarios used in 4.2.2 a warm up phase is needed. During the warm up phase all nodes randomly broadcast messages allowing the statistic modules of other nodes to categorize links. After the warm up phase, one node triggers a route search to another network node, which gets analyzed exactly like in the previously presented analysis of the route discovery strategy using fee as route decision criteria.

Figure 4.15 shows a graphical visualization of a simulated network with 1000 nodes distributed over 1000 x 1000 meter after the warm up phase. In this figure class 1 links are painted red, class 2 links green and class 3 links blue. It is observable that class 3 links are dominating. To give concrete numbers, in this scenario 38395 links belong to class 3, 2662 to class 2 and 2580 to class 1. This means only 13,6% of the links are actually interesting for routing.

Figure 4.16 shows the network graph after removing all class 3 links. It is observable that in most cases the usage of class 2 and 1 links is sufficient to reach most network nodes. However, some nodes, like the nodes at the position marked with C, are only reachable using class 3 links. Therefore, in some cases it is important to consider also class 3 links. Cashflow's route discovery protocol is able to handle these situations, which differentiates the presented protocol from other link strength sensible routing protocols like signal stability-based adaptive routing [Dube97].

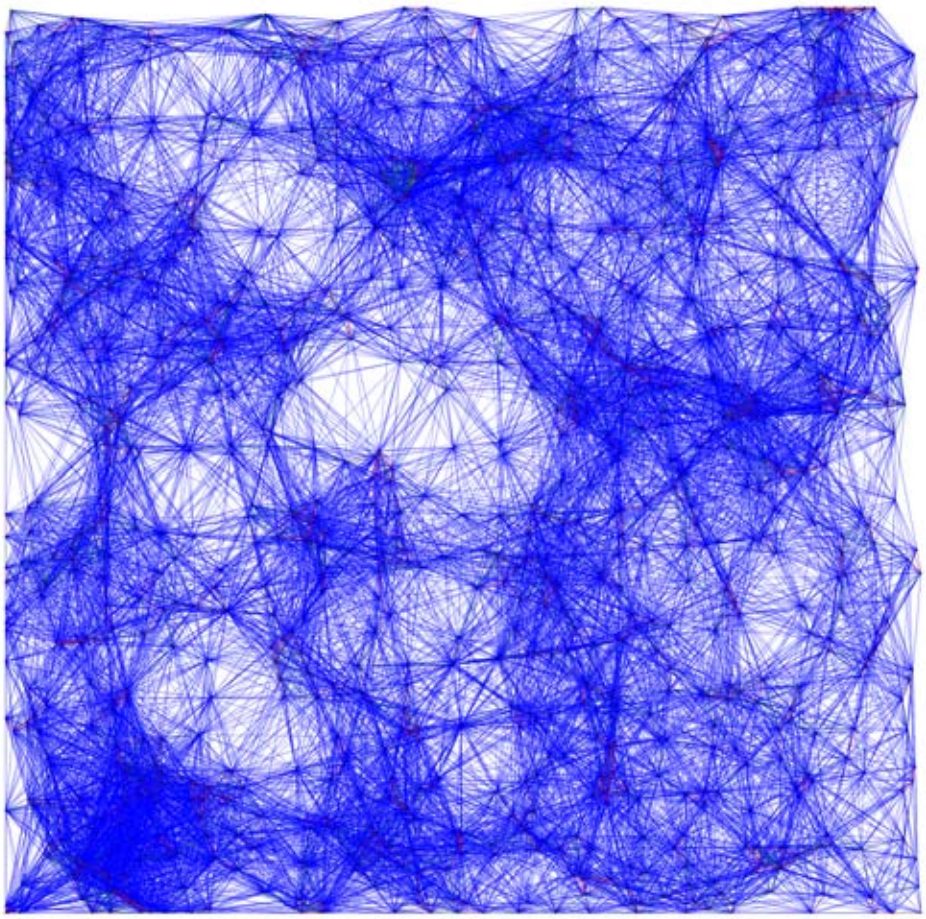


Figure 4.15: Visualization of class 1, 2 and 3 links of a network

The two shown figures additionally lead to the assumption that if class 2 and 1 links are preferred over class 3 links, the average path length increases. For instance, nodes at position A and B in Figure 4.16 are directly connected by class 3 links. However, if these links should not be used, the path contains several hops instead of a single one. Since the routing algorithm also considers class 3 links, in cases when the best path is much longer than a path including one or two class 3 links, the source node will learn about the path including class 3 links as well as the alternative more stable path. Therefore, in such cases, Cashflow leaves the decision which path to take to the node.

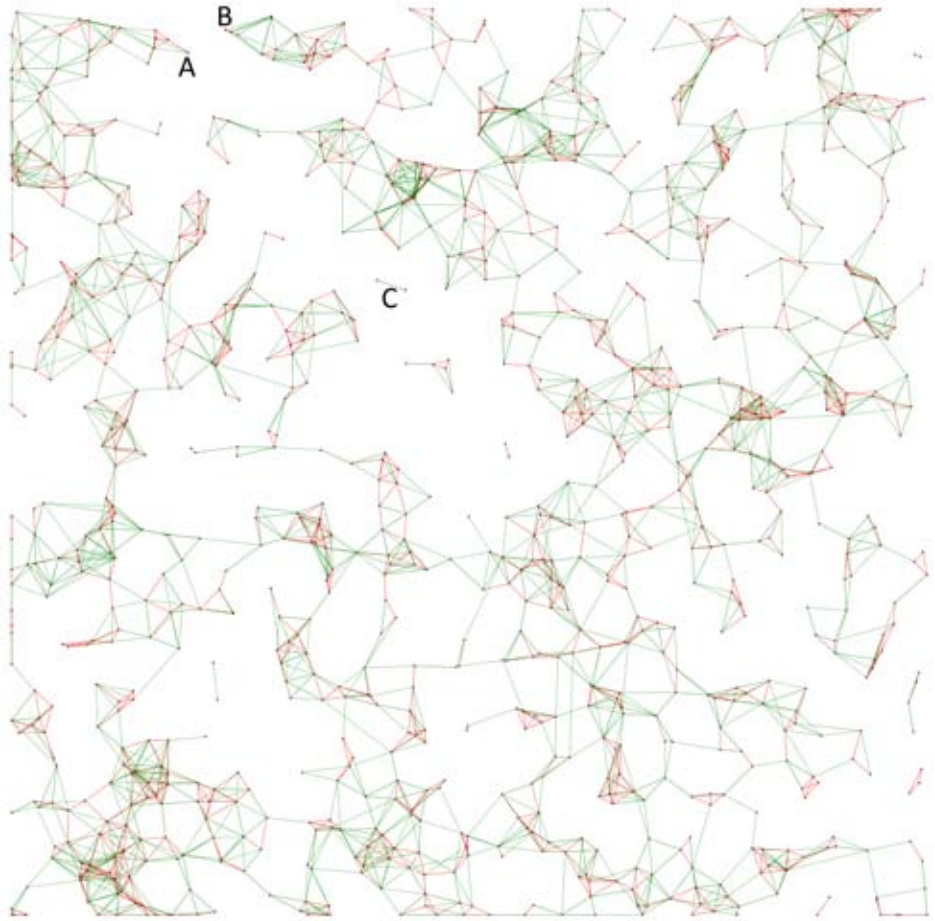


Figure 4.16: Visualization of class 1 and 2 links of a network

Figures 4.17 and 4.18 show class 2 and class 1 link separately. It is observable that while class 2 links span one single graph covering most of the network nodes, class 1 links form unconnected clusters. Even if this is not relevant for Cashflow's route discovery protocol, which is optimized to find the cheapest route with certain link quality properties, the clustering forms an interesting base for the development of purely quality based routing algorithms using Cashflow's delaying concept to optimize flooding. Since class 1 links are considered as relative stable within a class 1 link cluster, a proactive routing scheme could be used to update information if a new node is connected over a class 1 link or if a class 1 link is downgraded to class 2. This could be combined with a reactive routing scheme, using Cashflow's

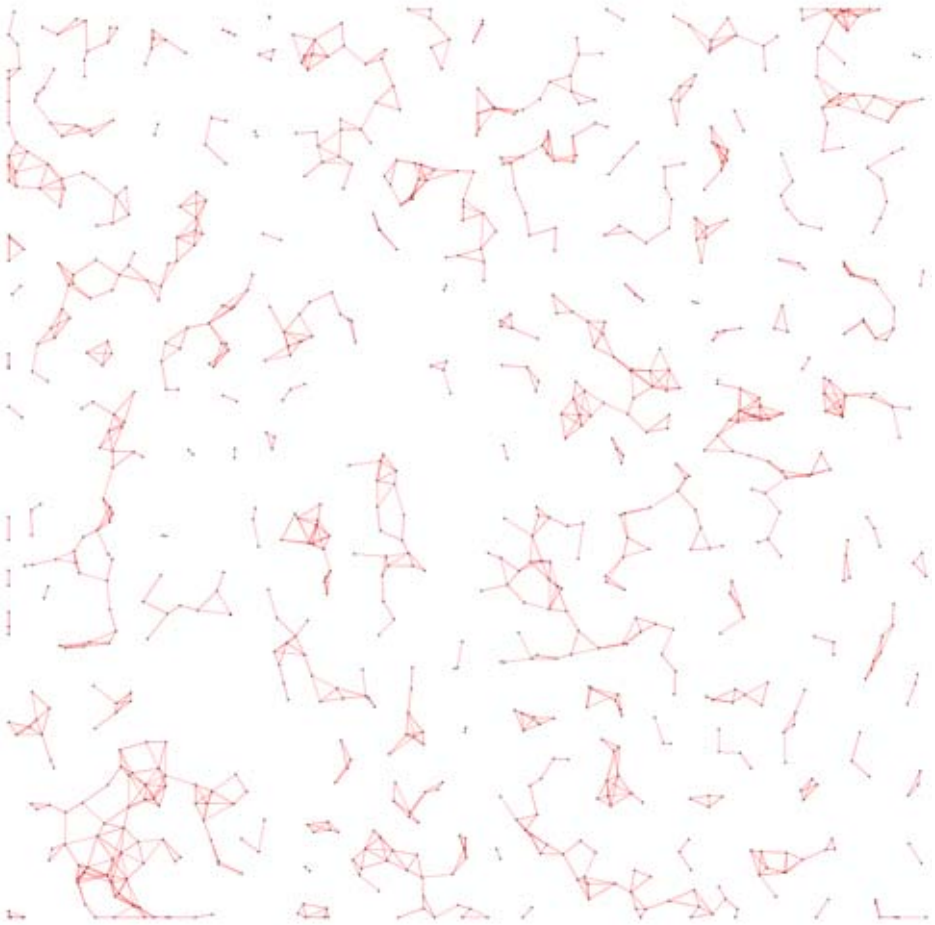


Figure 4.17: Visualization of class 1 links of a network

artificial delaying concept, to discover paths including class 2 and 3 links connecting class 1 clusters. However, since the consideration of fees brings an additional dynamic facture into the route discovery process, this scheme is not feasible for Cashflow.

The main outcome of this visual analysis is that it can be assumed that in average the algorithm prefers longer paths over short ones. This is on the one hand because class 3 links might connect part of the networks which are only connected over long paths of class 1 and 2 links, and, on the other hand, because class 3 links have a longer span than class 1 and 2 links, which means that several hops along class 1 and 2 links are needed to bridge the distance

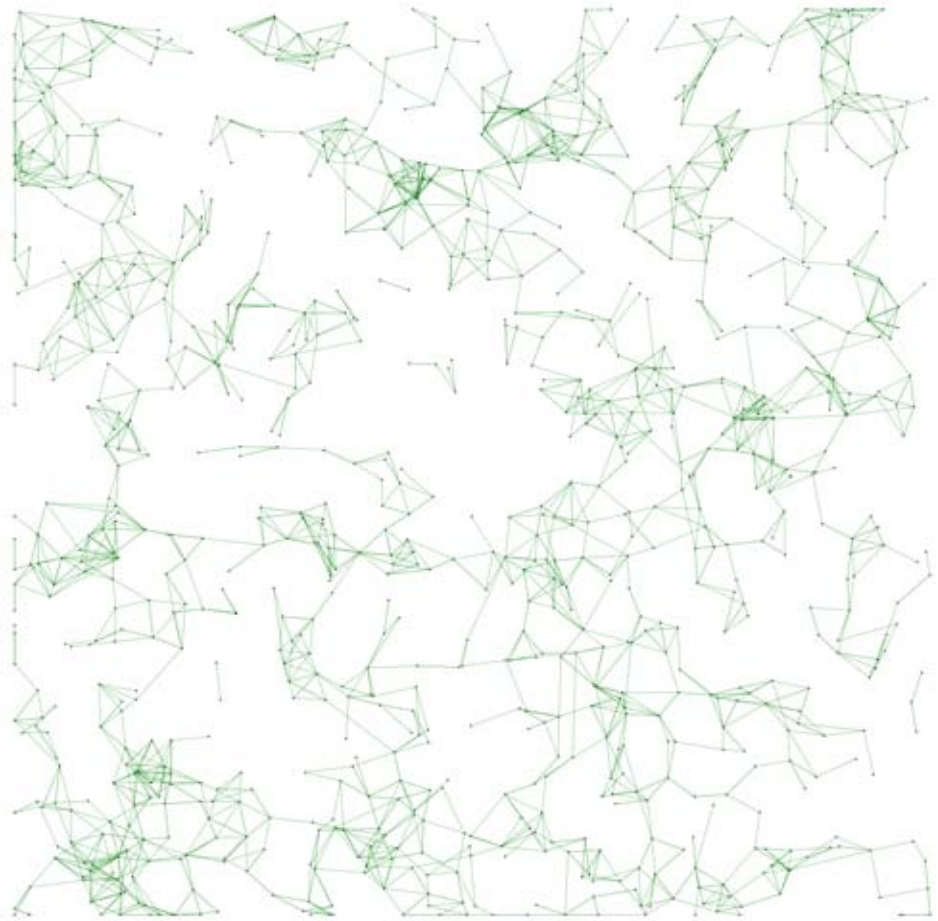


Figure 4.18: Visualization of class 2 links of a network

of one class 3 link. This is a difference compared to the simulation series, using fee as main routing decision criteria. In these series, the algorithm preferred in average shorter routes over longer once because of the lower costs. This leads to the assumption that if the routing algorithm prefers quality over price and consequently in average longer routes, the overhead, if no artificial delaying is, used will be larger compared to the simulation series presented in the last section. Additionally, the larger number of messages exchanged during route discovery will increase the number of collisions. In short, it is assumed that in absolute numbers the artificial delaying will decrease the overhead more than in the simulation series presented in the last section.

For the calculation of the artificial delay the nodes used the following formula:

$$d(c) = \begin{cases} \frac{D_{\text{maxClass}} \times 10^{(c-1)}}{10^{(2)}} & \text{if} \quad c > 1 \\ 0 & \text{if} \quad c \leq 1 \end{cases} \tag{4.2}$$

This formula is a simplified version of formula 3.11 presented in Section 3.3.3, where D_{maxFee} is set to 0, since the fee is not considered for routing, and C_{max} to 3, since in the used scenario the statistics module uses three classes for link classification. D_{maxClass} is the maximum delay used for artificial delaying. The actual value varies depending on the simulation. The parameter c represents the class of the current link. In short, nodes delay a frame for the time D_{maxClass} if the link is of class 3, and a tenth of D_{maxClass} if the link is of class 2. If the link belongs to class 1, no artificial delay is used.

Starting with the discussion of the results of the simulation using an area of 1000 x 1000 meter, Figure 4.19 shows the average number of packets send per node depending on the delay and on the processing time variation. It is observable that with an increase of artificial delay the average number of sent packets decreases. This is true for all three simulation series, independent from the processing time variation. That means, similar to the previously presented simulation series using node's fee as route decision parameter, the routing protocol's overhead is reduced by changing racing conditions using artificial delaying. With an increase of the delay the probability that the first discovered path to a node is also the best path increases. Therefore, the number of undesired rebroadcast is reduced, which results in a better performance. Additionally, the influence of the processing time variation is observable; the smaller the variation, the stronger the influence of artificial delaying. Comparing Figure 4.19 with the results of the previous simulation series, where the nodes fee was the main routing decision parameter, it is observable that in average more packets are transmitted per node during route search. The relation between delay increase and overhead decrease cannot be compared directly between these two simulation series, since different delaying mechanisms are used.

Figure 4.20 visualizes again the average number of packets send per node depending on the used delay and the processing time variance. For this simulation series, the nodes were distributed over an area of 500 x 500 meter, resulting in a four times higher density. Again, it is observable that with an increase of the delay the average number of transmitted packets de-

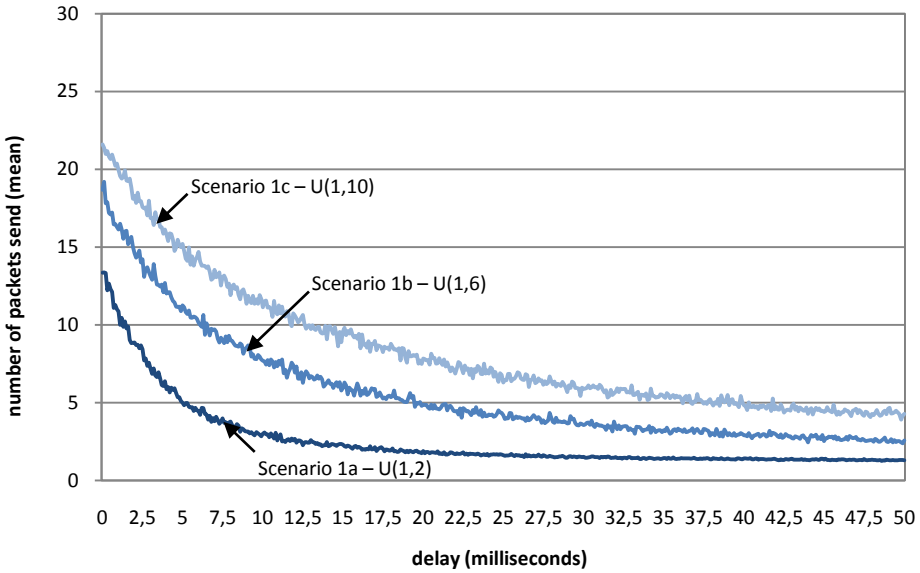


Figure 4.19: Influence of delay on average packet number send by nodes, using an area of 1000 x 1000 m

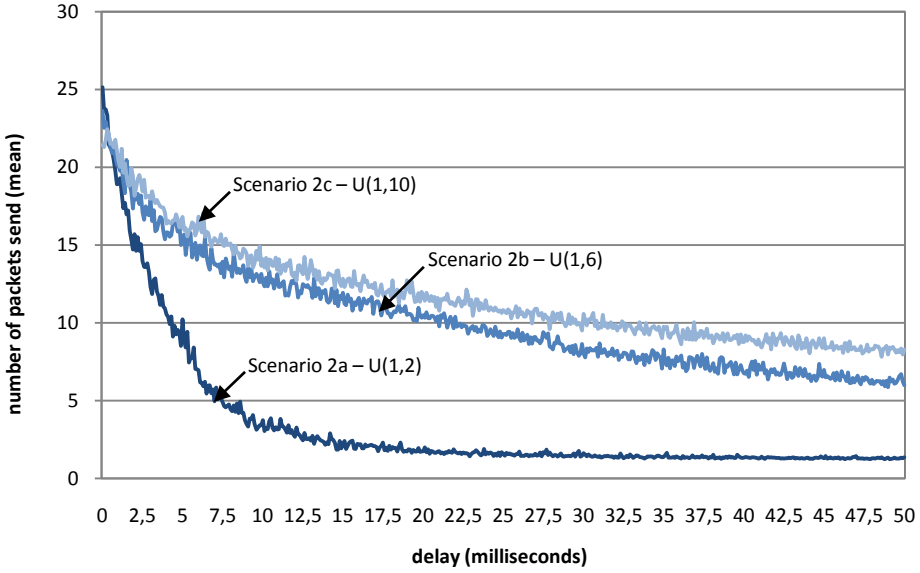


Figure 4.20: Influence of delay on average packet number send by nodes, using an area of 500 x 500 m

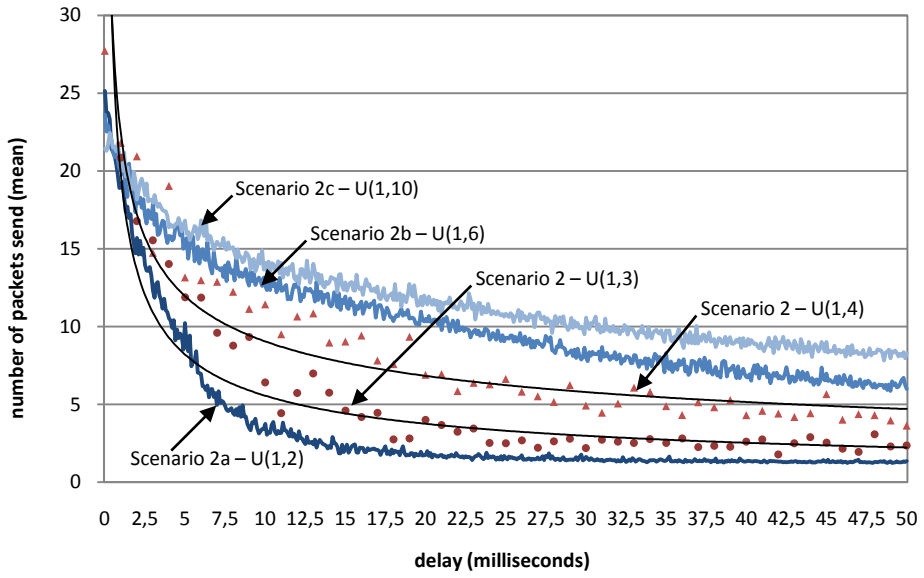


Figure 4.21: Influence of delay on average packet number send by nodes, using an area of 500 x 500 m overlaid by additional simulation results

creases. The curve of scenario 2a, using a processing time variation between 1 and 2 milliseconds, is striking in this figure. This effect can be explained due to the high collision probability. The processing time variance acts as kind of extension of the backup algorithm implemented by the 802.11 protocol. However, as mentioned before, the processing time variance has also a negative influence on the artificial delaying used by the routing protocol.

For verification, two additional simulation series have been performed, using delay steps of 1 instead of 0,1 milliseconds and only five reruns. These simulation series used a processing time variation between 1 and 3 milliseconds and 1 and 4 milliseconds. Figure 4.21 visualizes again the average number of sent packets for scenario 2, but additionally the results of the two additional simulation series are overlaid. The dots represent the values of the series using a processing variation time between 1 and 3 milliseconds, the triangles represent the values of the other series. Additionally, the corresponding power function based trend lines are shown. The results of these two simulation series lies between the outcome from series 2a and 2b, showing the influence of the processing time variation on the artificial delay's effect. Comparing the results to the outcome of the corresponding simulation series using fee as route decision parameter, again a higher

number of transmitted packets is observable.

Figure 4.22 shows the number of detected collisions depending on the artificial delay as well as on the processing time variation. It is observable that the number of collisions decreases when the artificial delay's value increases. This is mainly caused by the reduced traffic, which is the result of the artificial delay's influence on the racing conditions. The artificial delay itself has additionally a direct influence on the collision probability, since it delays the begin of the node's transmission depending on the link quality resulting in different retransmission times for nodes, which have received the same transmission simultaneously. Figure 4.23 shows the number of detected collision for the scenarios using an area of 500 x 500 meter. Again, a correlation between transmitted packets, shown in Figure 4.21, and collisions is observable and the positive effect of artificial delaying on the collision rate is visible. Comparing the collision rates between scenario 1 and 2, the influence of the density is observable, as described in Section 4.2.2.

Figure 4.24 and 4.25 picture the ratio between sent and received packets for scenario 1 and 2. With an increase of the artificial delay also the ratio between send and received packets increases. This means that in average a broadcast is received by more nodes. Since in each scenario the transmission power of all nodes is constant, this effect can be explained by lower traffic. Lower traffic influences the probability of receiving a packet in two ways. First, it reduces the collision probability as described before, resulting in a higher reception probability. Second, lower traffic also means less noise caused by interference at the radio channel, increasing the probability that even weak signals can be received. However, as shown in Figure 4.22 and 4.23, the collision probability does not reach zero in the simulated range of the artificial delay. Therefore, the ratio between sent and received packets is increasing continuously in all presented scenarios.

Even if with an increase of the delay the number of nodes receiving a single broadcast increases, the absolute number of received packets per node decreases, as pictured in Figure 4.26 for scenario 1 and Figure 4.27 for scenario 2. This is a direct result of the decrease of the retransmission rate, caused by the artificial delay's influence on the racing conditions. An abrupt dropped of the reception rate, caused by the reduction of the rebroadcast rate, followed by an increase due to the reduction of the collision probability, as it is observable at the simulation series using fee as route decision parameter, is not observable in the current scenarios.

Figure 4.28 visualizes the average time needed by the route discovery algorithm to detect the first route to the target node. With an increase of

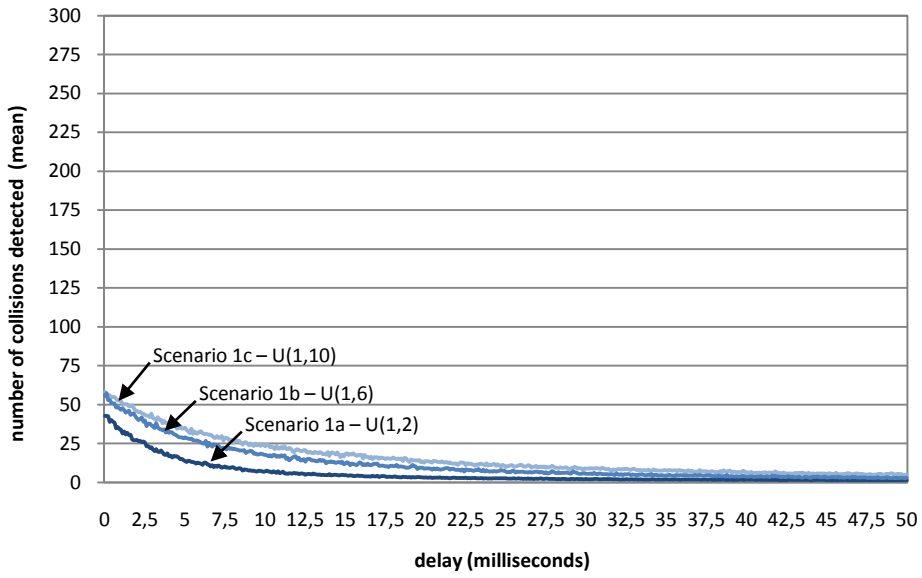


Figure 4.22: Influence of delay on average number of collisions detected by nodes, using an area of 1000 x 1000 m

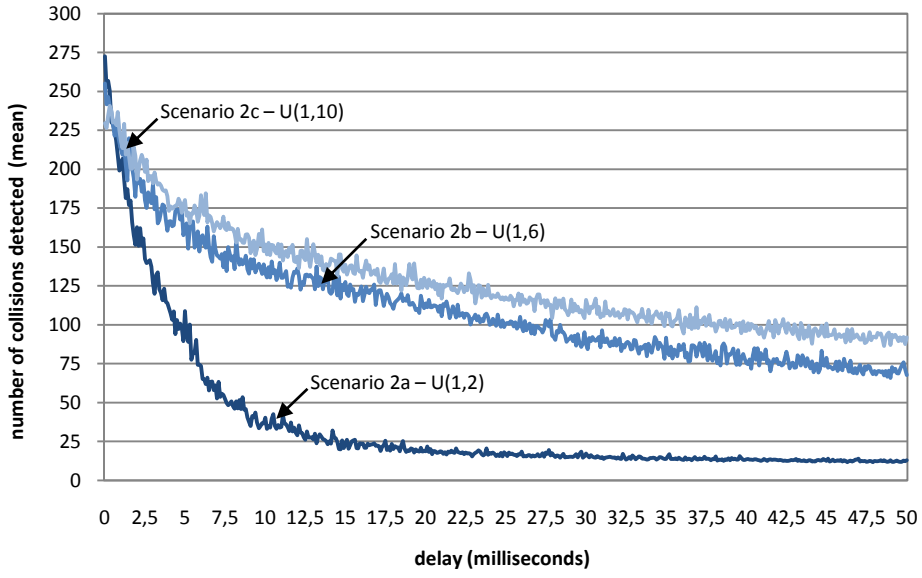


Figure 4.23: Influence of delay on average number of collisions detected by nodes, using an area of 500 x 500 m

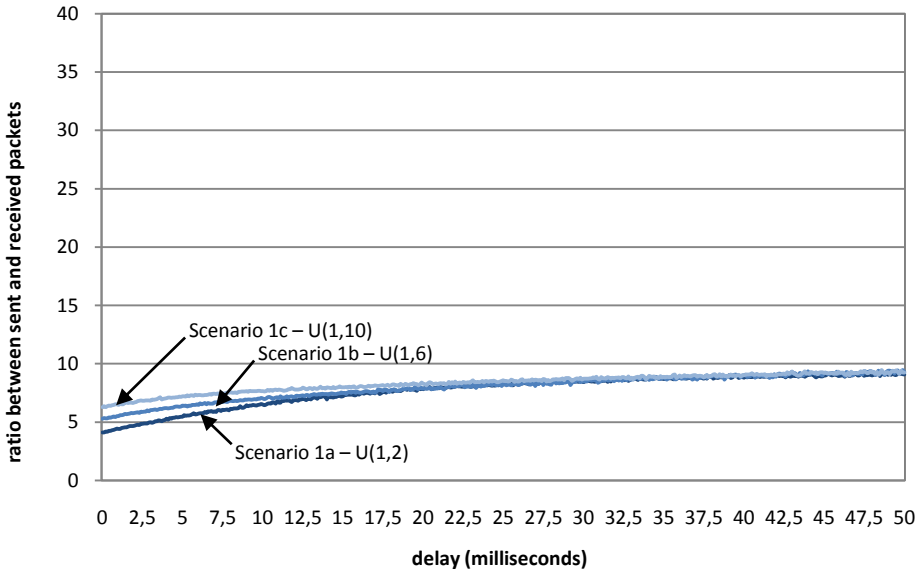


Figure 4.24: Ratio between average number of send and receives packets per node, using an area of 1000 x 1000 m

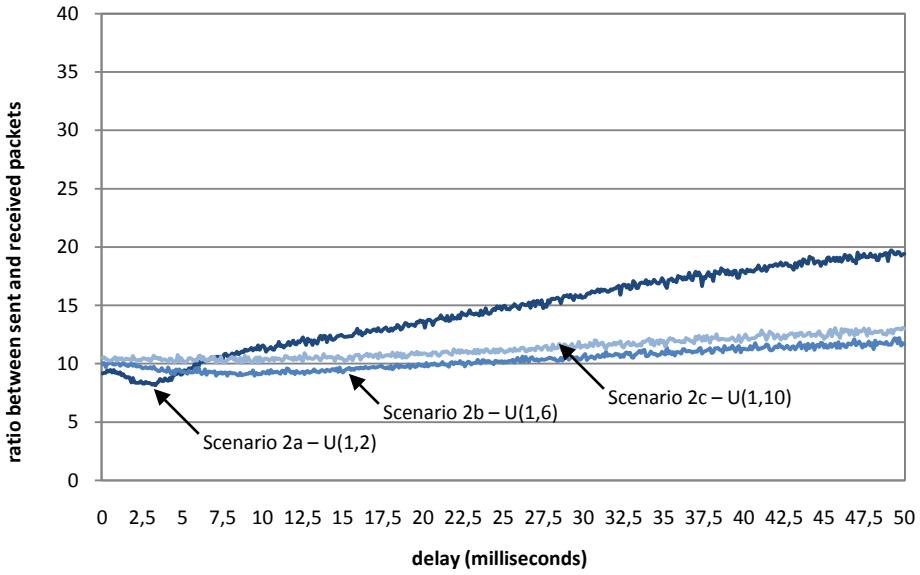


Figure 4.25: Ratio between average number of send and receives packets per node, using an area of 500 x 500 m

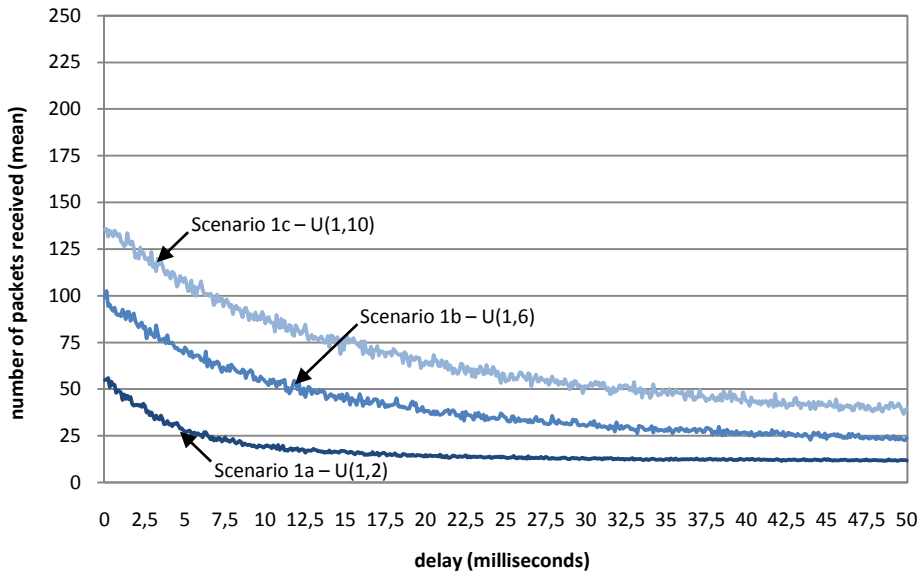


Figure 4.26: Influence of delay on average packet number received by nodes, using an area of 1000 x 1000 m

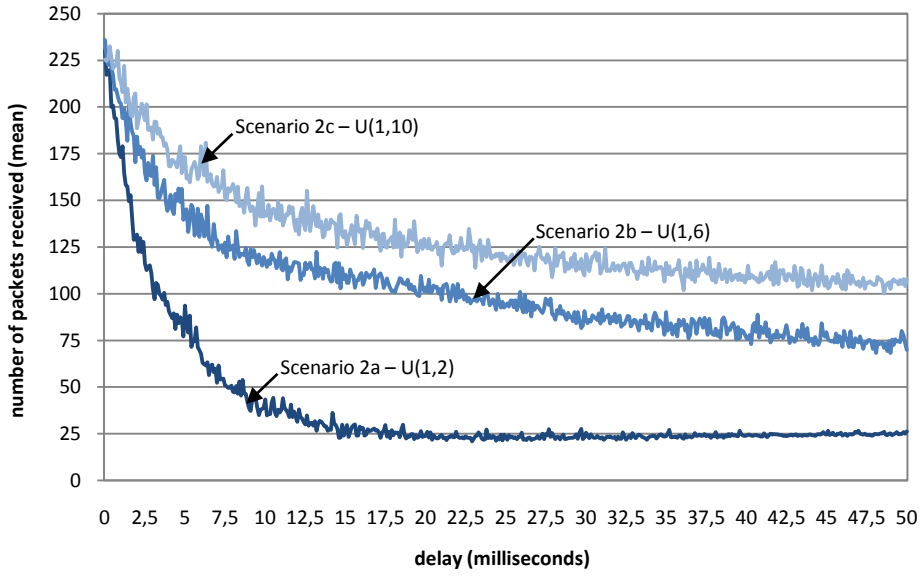


Figure 4.27: Influence of delay on average packet number received by nodes, using an area of 500 x 500 m

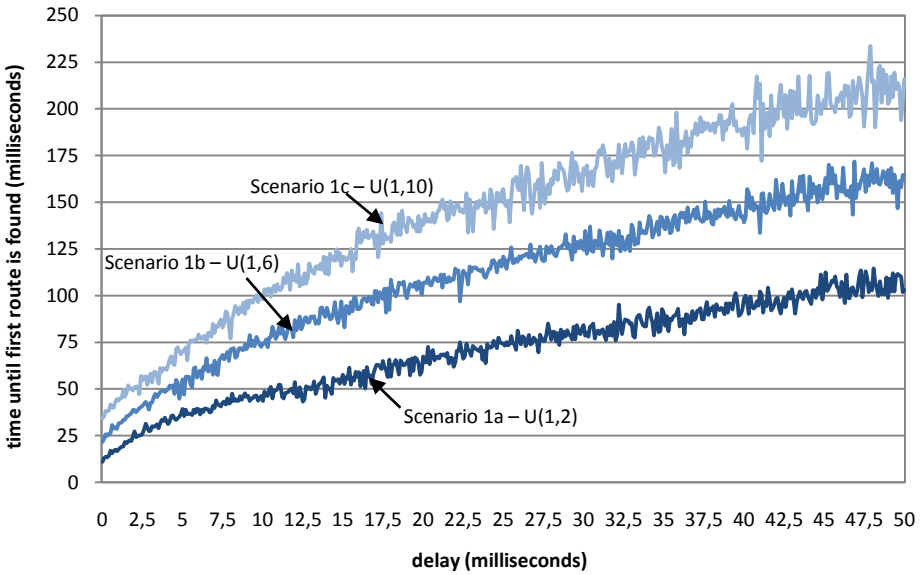


Figure 4.28: Time until first route to target is found, using an area of 1000 x 1000 m

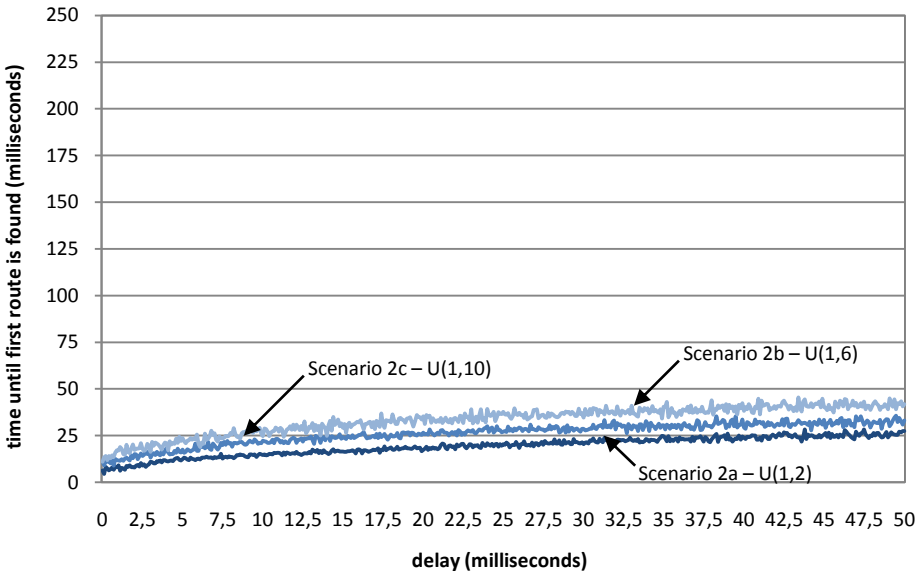


Figure 4.29: Time until first route to target is found, using an area of 500 x 500 m

the artificial delay, also the time needed for route discovery increases in all three scenarios, since the time needed for route discovery consists of the time needed for data transmission, the processing time, and the artificial delay. This explains the correlation of the artificial delay and the route discovery time. Additionally, this is also the explanation for the different route discovery times of the three scenarios. Assume a path of 20 hops. In scenario 1a, each packet gets in average delayed 1.5 milliseconds per hop, resulting in a cumulative delay caused by the processing time of 30 milliseconds. Comparing this to scenario 1c, where each node delays each packet in average 5.5 milliseconds resulting in a cumulative delay of 110 milliseconds, in scenario 1a the path is discovered 80 milliseconds earlier than in scenario 1b.

Furthermore, it is observable that the three curves for scenario 1a to 1c do not increase by the same rate, which is an indirect effect of the change of the racing conditions by the route discovery protocol. With an increase of the delay, the number of rebroadcasts needed to find the best route decreases, since preferable routes are detected earlier during the route discovery process. This has been discussed in detail before. The usage of link quality as main route decision criteria results in average in longer but stable routes, compared to the shorted possible routes. Consequently, with a decrease of the artificial delay the time needed for the discovery of the first route increases, but additionally the average length of the first discovered route increases. Since the increase of a routes length causes a scenario dependent increase of the route discovery time, which is caused by the different average processing times, the curves increase at a different rate depending on the delay.

Figure 4.29 pictures again the time needed for the discovery of the first route in the scenarios 2a to 2c. Similar to the results of scenario 1, an increase of the time needed for first route discovery as consequence of the artificial delays increase is observable. Likewise, the impact of the different processing times can be seen. Comparing the curves of scenario 2 with scenario 1, it is observable that the average time needed for the discovery of the first route is smaller compared to scenario 1. This is the result of the of the smaller area used for these simulation series and the resulting higher node density. Since the area is smaller, the average number of hops between two nodes decreases. This effect mainly causes the shorter discovery time. However, also the higher density has an influence. The higher density decreases the probability that some nodes or parts of the network are only long-winded reachable as shown in Figure 4.16, resulting in a faster route discovery.

Summarizing, this section has shown the positive effects of the proposed artificial delaying mechanism on the overall performance if the routing algorithm uses link quality as main route decision criteria. Further, it was shown that when link quality is used as main route decision criteria, in average nodes would perform more rebroadcast than if fee would be used as route decision criteria. Therefore, artificial delaying has an even stronger impact on the protocols performance in terms of saved broadcasts.

4.2.4 Propagation analysis

Chapter 4.2.2 and 4.2.3 already discussed the influence of artificial delaying on the overall performance of the route discovery algorithm. This chapter focuses on the distribution of network activity during route discovery, depending on the discovery strategy, the artificial delay, and the processing time variation. For the analysis of the network activity, eight simulation scenarios have been deployed. All eight scenarios have in common that the nodes use the same radio parameters as described in Chapter 4.1. Each scenario consists of a network of 1024 nodes. In contrast to the simulation scenarios used for the analysis of the routing algorithm's performance, the nodes are not randomly distributed. Instead, the nodes form a regular grid of 32 to 32 nodes, with a distance of 30 meter between them. This topology was chosen to allow a visual presentation of the network activity in the form of heat maps.

The scenarios differ from each other by the used route discovery strategy, the used delay, and the processing time variation. Table 4.3 gives an overview over the resulting eight scenarios. Scenarios of the type "Fee" uses the fee nodes charge as main routing decision parameter, in contrast to scenarios of the type "Quality", where the connection quality between nodes is taken as basis for routing decisions. Four out of the eight scenarios use an artificial delay of 10 milliseconds for $D_{maxClass}$ or D_{maxFee} to calculate the actual delay, depending on the route decision strategy (compare with Formula 3.9, 3.10 and 3.11). For the other four scenarios, no delay is used. The last parameter by which the scenarios differ from each other is the processing time variance. Again, four scenarios uses a processing time uniformly distributed between 1 and 2 milliseconds, the other four vary the processing time between 1 and 10 milliseconds.

In all eight scenarios, the node in the lower left corner triggers a route search with the node in the upper right corner as target. In scenarios using the fee nodes charge as decision parameter, the route search is started immediately at the start of the simulation. This cannot be done in the scenarios using

link quality as decision parameter, since the statistics module, which evaluates the connection between a node and its neighbor nodes, needs a number of transmissions to get enough data for the statistical analysis. Therefore, a route search immediately started at the beginning of the simulation would not be representative. Hence, these scenarios start with a warm up phase, where nodes perform a number of broadcasts to allow the statistic modules to valuate connections. After the start up time follows a short waiting time to guaranty that there is no network activity any more. Finally, the node in the lower left corner triggers a route search like in the scenarios where the fee is used by the routing algorithm as decision parameter.

The moment, when the route search is triggered by the application, was defined as time 0 for the propagation analysis. From this time on, every five milliseconds in the simulation a heat map of the network activity was created. The heat maps are based on the number of received packets per node during the last five milliseconds. This was done for the first 95 milliseconds during the route search, resulting in 160 heat maps, 20 for each scenario. The complete series of heat maps are included in Annex A

Figure 4.30 pictures an example of a heat map displaying the network activity during route discovery at a certain moment. In this heat map, areas with no network activity are colored dark blue. In contrast, areas with network activity are colored in shades of light blue, green and yellow, depending on the amount of packets detected per node during the last 5 simulated milliseconds. Yellow marks areas with the highest activity, which correspond to 10 or more detected packets. Areas with no network activity are called inactive areas in contrast to active areas, which reveres to areas where at least one packet is detected. In this heat map the position of the source node, which triggers the route search is marked. The triggering of the

Scenario	**Type**	**Delay**	**Variance**
FeeD0U2	Fee	0 ms	U(1,2) ms
FeeD1U2	Fee	10 ms	U(1,2) ms
FeeD0U10	Fee	0 ms	U(1,10) ms
FeeD1U10	Fee	10 ms	U(1,10) ms
QualityD0U2	Quality	0 ms	U(1,2) ms
QualityD1U2	Quality	10 ms	U(1,2) ms
QualityD0U10	Quality	0 ms	U(1,10) ms
QualityD1U10	Quality	10 ms	U(1,10) ms

Table 4.3: Parameters of the simulation scenarios used for propagation analysis

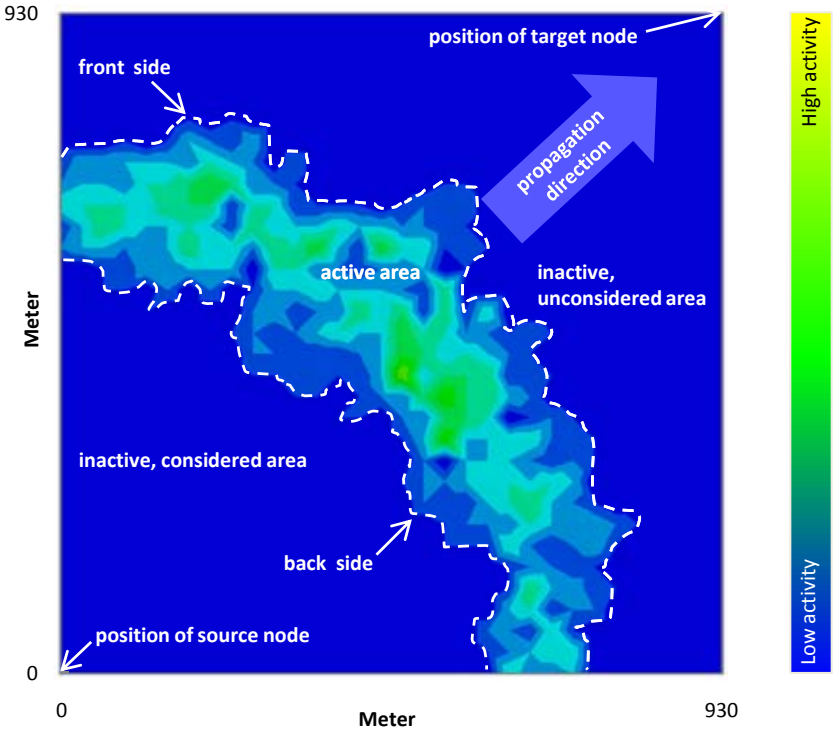


Figure 4.30: Explanation of heat maps

route search causes a flooding of route request packets. The flooding starts at the position of the source node and propagates over the whole network until eventually a route to the target node, which is also marked in the heat map, is found. Since flooding causes network activity, the flooding generates an active area, which propagates over the complete network in direction to the target node. Depending on the scenario and the time, the active area eventually divides the network in two inactive areas, as shown in the heat map. The inactive-considered area is the inactive area, which includes the source node. Nodes located in this area are not involved in the current route discovery process any longer. These nodes have already discovered the best route, depending on the route selection criteria, between themselves and the source node. Since the active area has already passed the nodes located in the inactive-considered area, the border between active and inactive-considered area is named back side. The inactive-unconsidered area is the inactive area, which includes the target node. The active area expands into this area. Therefore, the border between the inactive-unconsidered area

and the active area is named front side of the active area. Since all nodes located in the inactive-unconsidered area have received no route discovery packet so far and consequently not broadcasted a route discovery packet themselves, they possess no current routing data to the source node. At the moment, when the active area finally expanses over the target node, the first route between the source and the target node is detected by the algorithm. This first path does not have to be the best path necessarily. As long as the target node is located in the active area, the source node might learn about better routes. After some time the active area will dissolve and the route discovery process is finished.

The size and the propagation speed of the active area are influenced by the artificial delay, the route selection criteria, and the processing time variation of nodes. Starting with an analysis of the artificial delay's influence on the network activity during route discovery, Figure 4.31 presents a comparison of a heat map selection for the scenarios FeeD0U2 and FeeD1U2. The heat maps on the left side present the network activity of scenario FeeD0U2 at the time 0.01, 0.02, 0.03, and 0.04. The right column presents the activity of FeeD1U2. The same routing selection criteria and the same processing time variance is used in both scenarios but only in scenario FeeD1U2, artificial delaying is used. By comparing the heat maps, it is observable that the propagation speed of the active area is not as fast in scenario FeeD1U2 as if no artificial delay is used. When no artificial delay is used, the active area reaches the target node at sometime between 0.01 and 0.02 milliseconds after the start of the route discovery. At 0.02, the active area in scenario FeeD1U2 has not even passed the half way between the source and the target node. Since the moment when the active area reaches the target node is also the moment when the first route is discovered as described before, this observation approves the influence of the delay on the route discovery time, as discussed in Section 4.2.2.

Besides the propagation speed, the artificial delay also influences the size of the active area. When no artificial delay is used, a larger area is active, in contrast when an artificial delay is used. This is the case, because the route discovery protocol prevents a fast expansion, as long as the collective knowledge of the nodes in the active area about the topology is low. The benefit of this strategy is that it reduces the number of nodes involved in the route discovery process simultaneously and consequently reduces the probability that node have to renew their offer. The route discover algorithm is design in a way that as long as nodes frequently discover new paths to the source node, they extends their delay period without broadcasting route request packets. Only when the duration of the artificial delay time

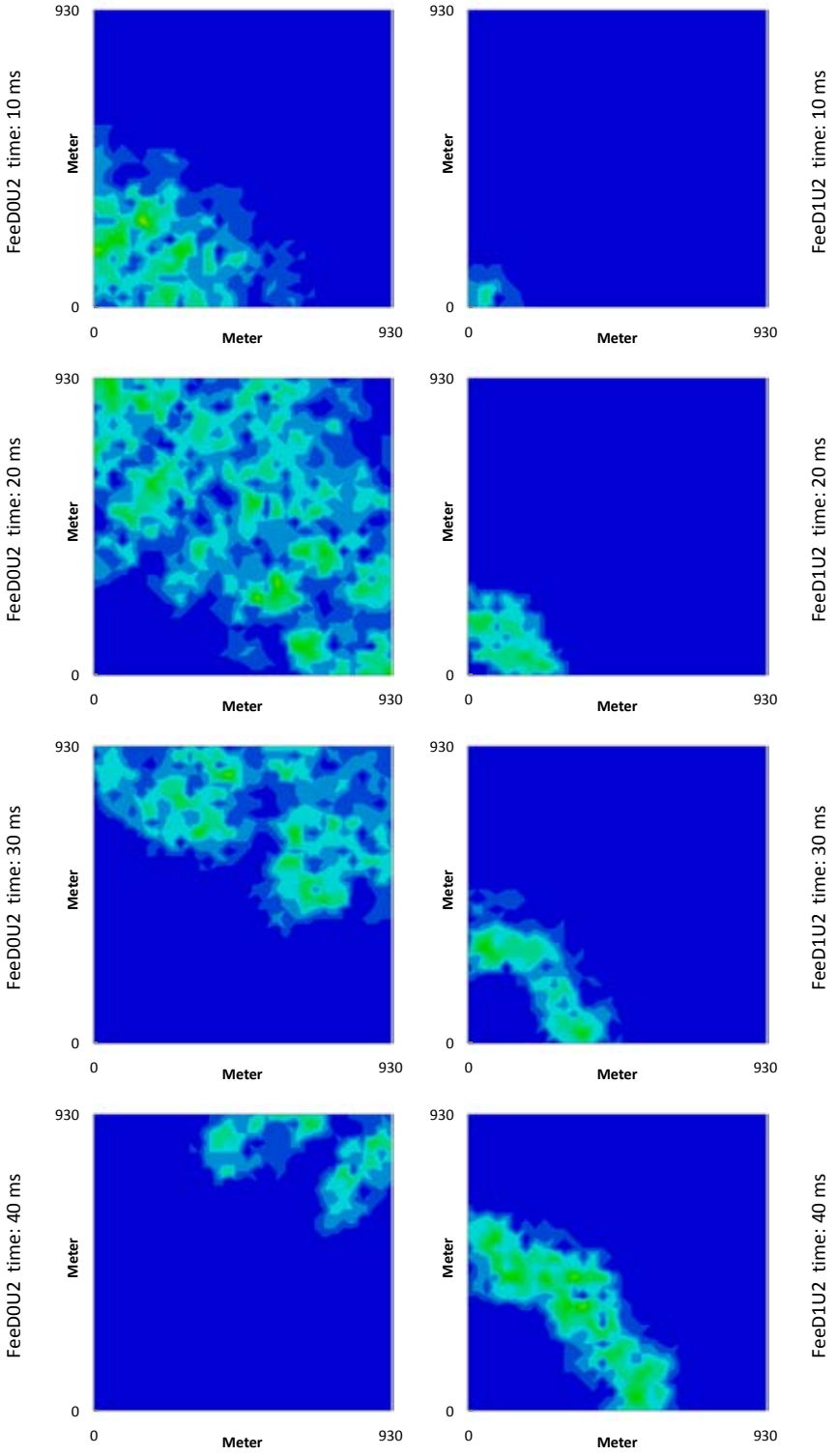


Figure 4.31: Network activity at the time 0

has passed without the discovery of a better path during the waiting period, a node can broadcast a route discovery packet including the best path it has learned so far. This mechanism especially affects nodes along the front side of the active area, which are new to the route discovery process. When the active area expanses over a node, this node will learn about new routes frequently and consequently will delay the forwarding of the received route discovery packets, until the time between two new discoveries exceeds the artificial delay the node uses. As consequence, an undesirable fast expansion of the active area is prevented.

These two effects, the reduction of the propagation speed and the decrease of the size of the active area, are not limited to the processing time variance and route selection strategy used in these two scenarios. Figure 4.32 visualizes the influence on the propagation speed for the other scenarios. The four heat maps on the left side represent scenarios where no artificial delay is used. On the right side, the heat maps of the corresponding scenarios using artificial delay are depicted. The heat maps on the right side represent the moment, when the first route was found. Because of the influence of the network density as well as the processing time variation, the actual point in time when the first route is found varies, depending on the scenario. Therefore, the heat maps representing scenarios with a processing time variation between 1 and 2 milliseconds, visualize the network activity at 25 milliseconds after the route search's start. The heat maps representing scenario FeeD0U10 and the corresponding scenario FeeD1U10 visualize the activity at 55 milliseconds. The last two heat maps in Figure 4.32 are taken from scenario QualiD0U10 and QualiD1U10 at 65 milliseconds after the start. In all four cases, it is observable that at the moment, when the active area reaches the target node in scenarios where no delay is used, the active area has not even passed the half area if the artificial delay is activated. It is worth noting that the heat maps showing scenarios with artificial delay look similar to each other, even if they represent the network activity at different moments. The activity area covers nearly the same area in all four heat maps. This means, that the artificial delay slows the propagation down by a factor, which is independent from the processing delay variation and the route selection strategy.

Besides the influence of the artificial delay on propagation speed, also its influence on the active area is observable, as depictured in Figure 4.33. Again, the heat maps on the left side represent scenarios without the usage of artificial delay, and on the right side the corresponding heat maps with the usage of artificial delay are depictured. Each heat map pair represent the network activity at a moment, when in the heat map of the scenario

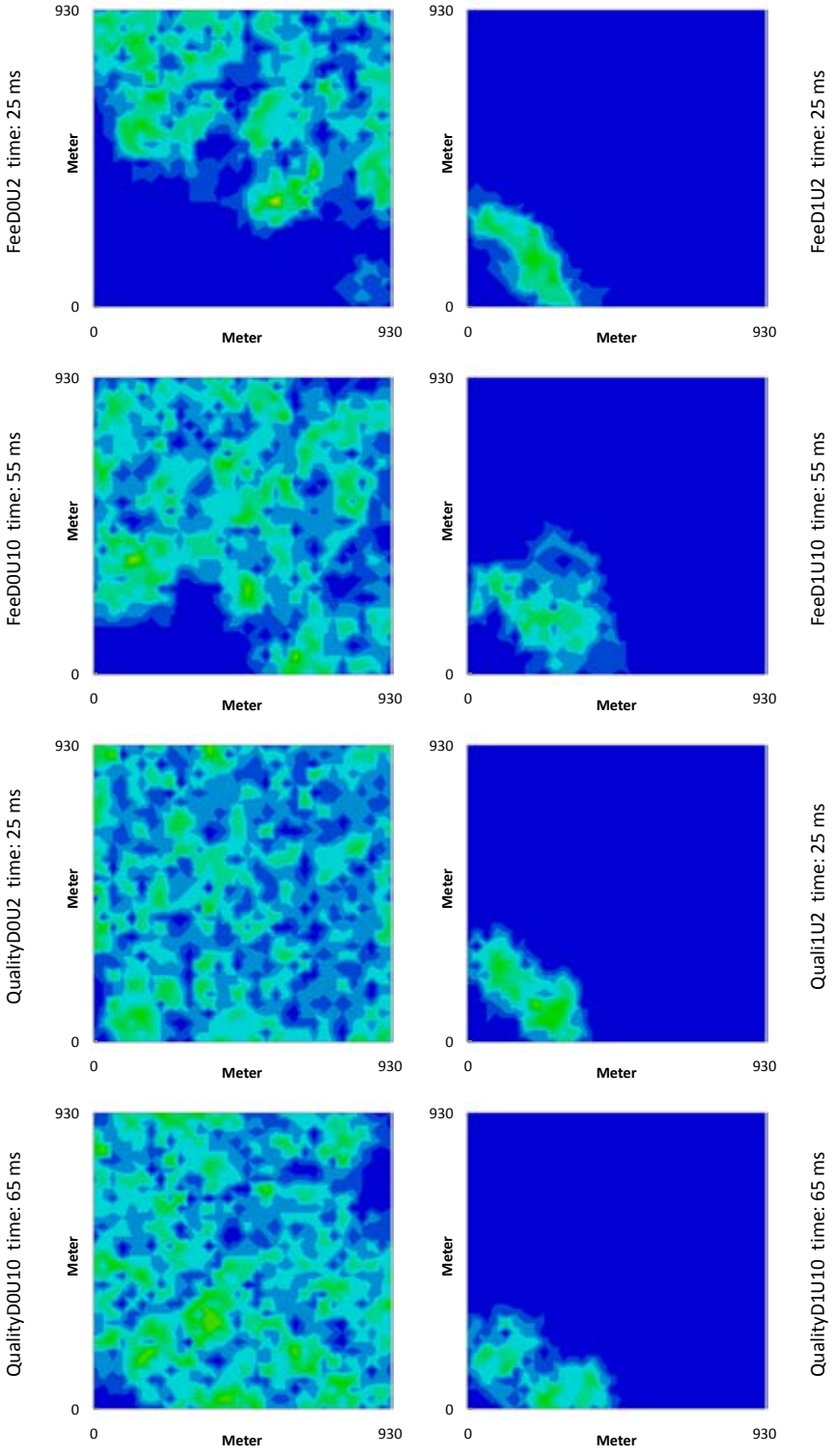


Figure 4.32: Influence of artificial delay on propagation speed

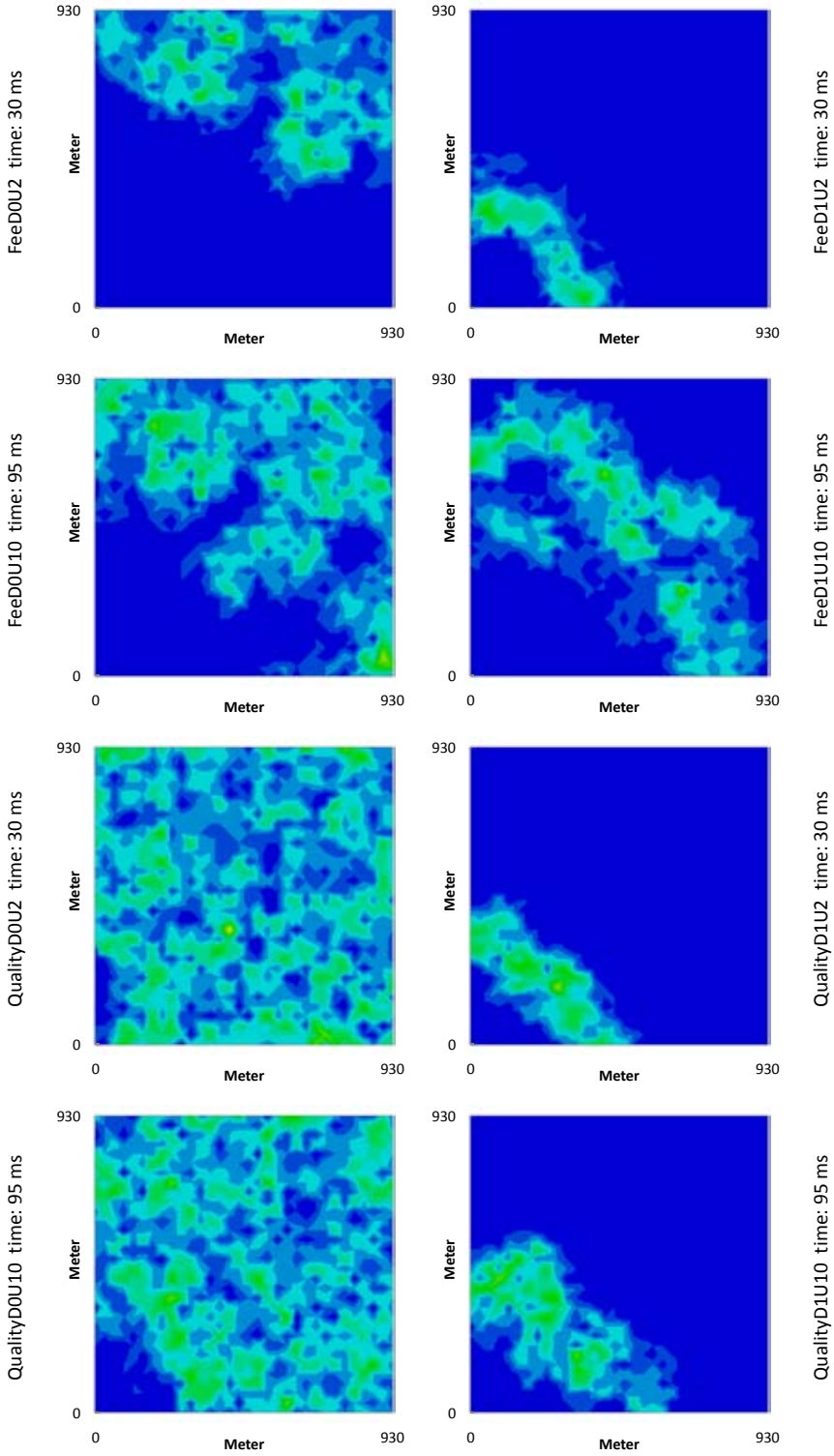


Figure 4.33: Influence of artificial delay on active area

using artificial delay the active area is clearly bordered by two inactive areas. At the same time, when no artificial delay is used, a larger area is active. In some cases, the active area covers nearly the whole area. Summarizing, it is observable that the artificial delay decreases the active area's size in all presented scenarios.

Continuing with the influence of the processing time variation on the expansion speed and the active area, Figure 4.34 shows heat maps of the scenarios using a processing time variation between 1 and 2 milliseconds on the left side. On the right side, the heat maps of the corresponding scenarios with a processing time variation between 1 and 10 milliseconds are depictured. For the analysis of the propagation speed, the heat maps on the left side of the picture visualize the moment, when the route discovery protocol discovers the first route to the target node. Comparing the heat maps on the left side with the corresponding heat maps on the right side, it is observable that in scenarios using a lower processing time variation, the first route is found earlier, compared to scenarios using a larger processing time variation. With an increase of the processing time variation, the average time a node delays a broadcast increases, resulting in a slower propagation of route discovery packets. This observation coincide with the results of the statistical analysis in Section 4.2.2 and 4.2.3.

A direct influence of the processing time variation on the size of the activity area is not clearly visible. Only the comparison of the scenarios FeeD1U2 and FeeD1U10 as pictured in Figure 4.33 gives an indication on the influence of the processing time on the artificial delay. The active area in the heat map of scenario FeeD1U10 is larger than the heat map depicting the activity of the corresponding scenario using a lower processing time. As described before, the route discovery algorithm prevents nodes from broadcasting route discovery packets, as long as the time between the detection of new, more preferable routes is smaller than the used artificial delay. With an increase of the processing time variation the effectively of this functionality gets extenuated, because the increase of the processing time variation slows the route discovery process down and consequently increases in average the time between the last route detection and the detection of a better route. Therefore, if the processing time variation increases, the probability increases that the artificial delay time passes without the detection of a new route, resulting in a broadcast of the route request before the actual optimal route to the node is found, causing a faster propagation of the activity area. This causes an increase of the routing algorithm's overhead, since the routes included in these prematurely broadcasted route discovery packets are more likely to become obsolete than later ones.

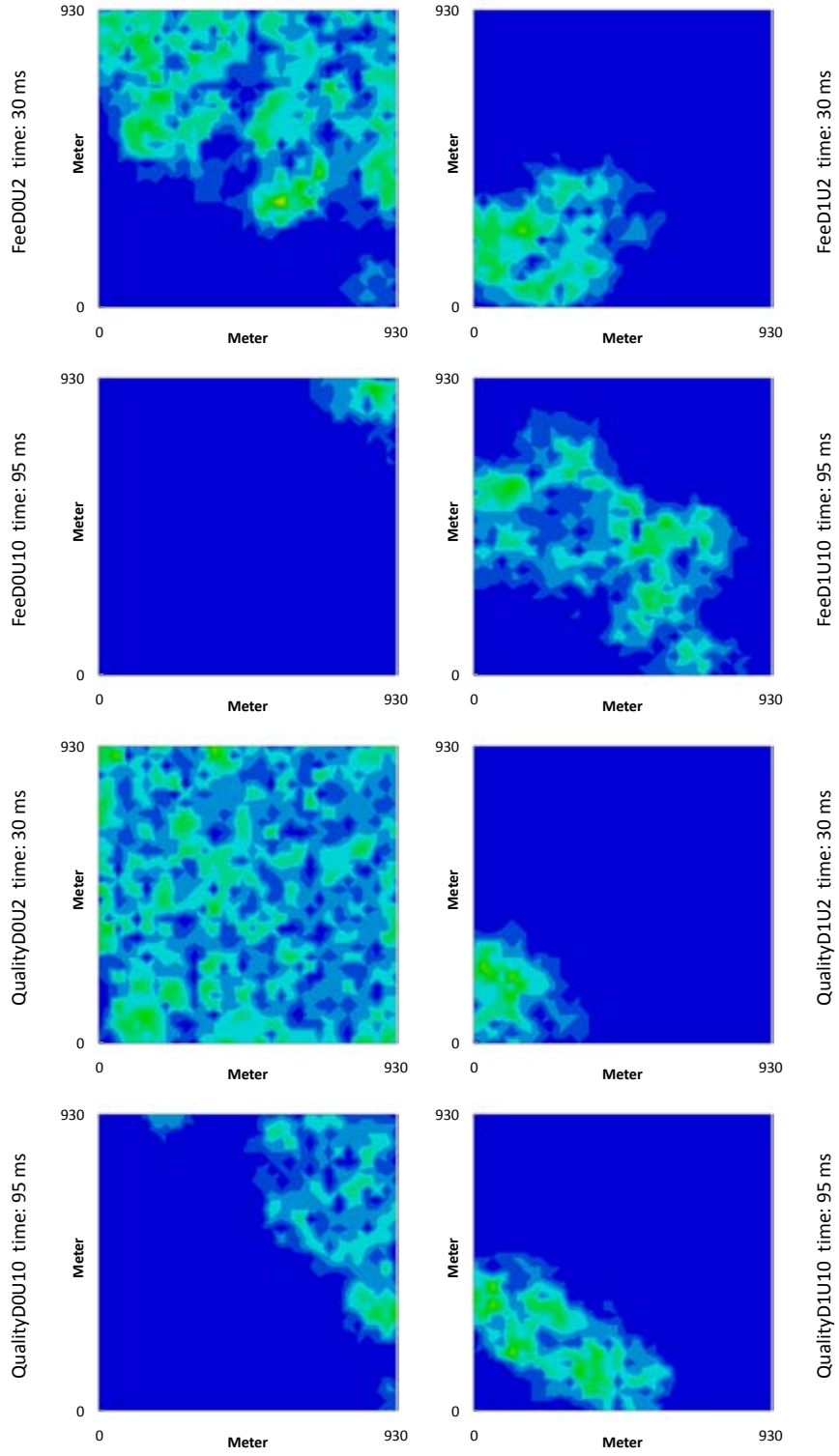


Figure 4.34: Influence of processing time variation on propagation speed

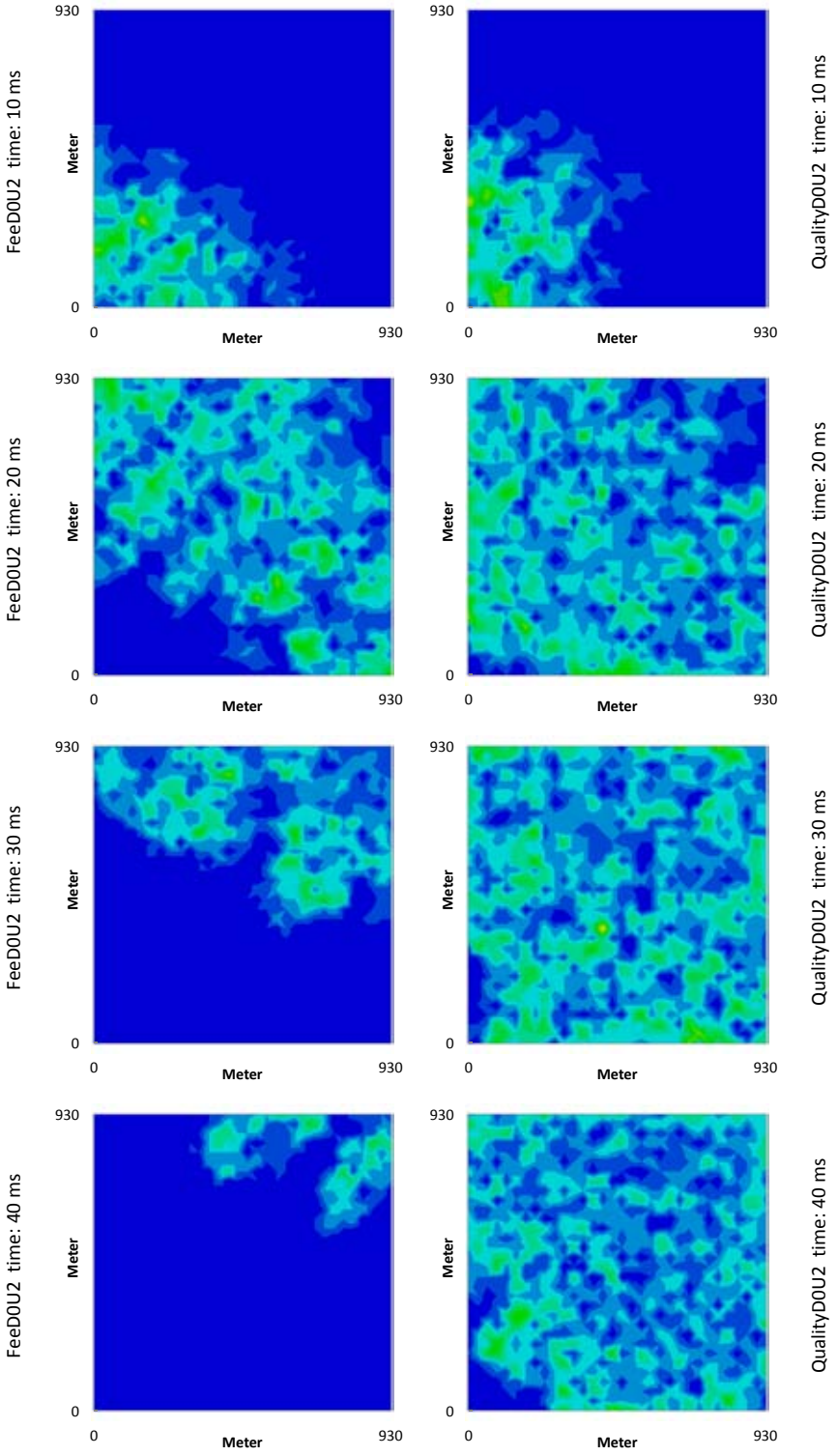


Figure 4.35: Comparison of scenario FeeD0U2 and QualityD0U2

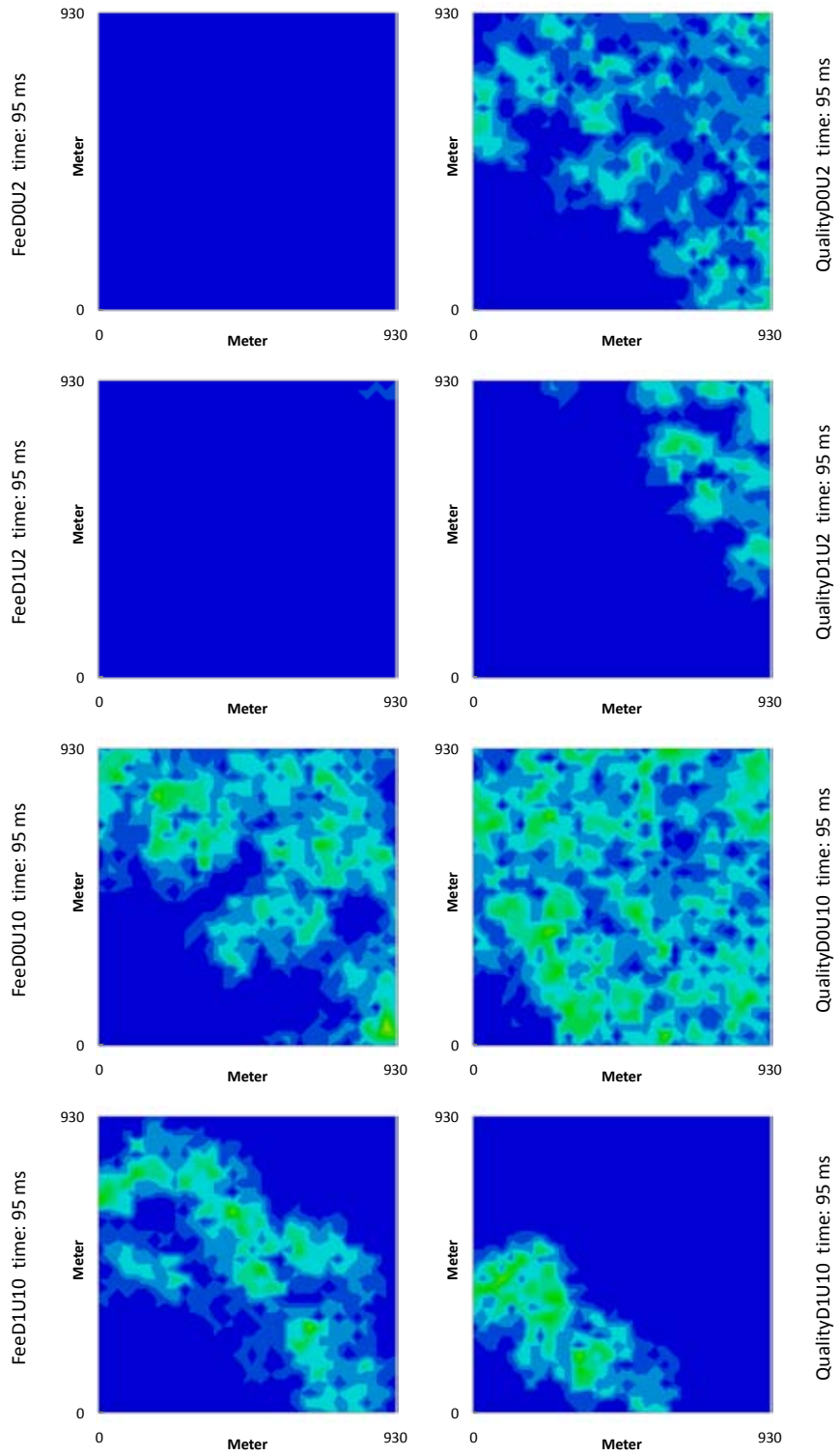


Figure 4.36: Comparison of scenarios based on route decision strategy

Besides different values for artificial delay and processing time variation, different route selection strategies where used in the investigated scenarios. Figure 4.35 depictures heat maps of scenario FeeD0U2 and QualityD0U2, visualizing the network activity at 10, 20, 30, and 40 milliseconds after the start of the route discovery process. The four heat maps on the left side picture the activity of scenario FeeD0U2, the heat maps of scenario QualityD0U2 can be seen on the right side if the figure. Even if there is no obvious difference in the propagation speed, in terms of expansion of the active area along the front side in this scenario, there is a noticeable difference in the speed of the back sides movement. Comparing the heat maps representing the moment of 30 milliseconds after start of the route discovery process, when the fee nodes charge is the main route selection criteria, the inactive considered area covers half of the complete area. In contrast, the activity area covers the whole network when link quality is used as main route selection criteria.

The same effect can also be observed in the other scenarios as pictured in Figure 4.36. The heat maps on the left side represent scenarios using fee as route decision criteria. The maps on the right side visualize the network activity of the corresponding scenarios using quality as route decision parameter. All heat maps in this figure represent the network activity at 95 milliseconds after the route search's start. Comparing the heat maps of scenario FeeD0U2 and FeeD1U2 with scenario QualityD0U2 and QualityD1U2, it is observable that while there is nearly no network activity any more in the scenarios using fee as route decision parameter. In the other two scenarios, the route discovery process is still in progress. Also in the heat map representing scenario FeeD010, the inactive considered area already covers a larger area than in the corresponding scenario QualityD010. A similar difference concerning the active area can also be observed comparing the last two heat maps. Comparing these both scenarios, again a difference in the propagation speed of the active area is clearly visible. This accords to the statistical analysis of the time needed for the first route discovery as pictured in Figure 4.12 and 4.28.

The difference in the movement speed of the active area's back side, caused by the different route decision strategies, can be explained by the effect that when connection quality is used as route decision parameter, the route discovery algorithm prefers longer routes than if fee is used for the route selection. Consider the following example. A number of nodes are arranged in a line and are numbered consecutively. Every node has a good connection to his direct neighbor nodes and a weak connection to the neighbor's neighbor. So node 1 has a good connection to node 2 and a weak connection

to node 3. Node 2 has a good connection to node 3, a weak one to node 4, and so on. Additionally, it is assumed that every node charges the same fee and no artificial delay is used. When a node triggers a route search to find a path to another node in the line, there are two optimal outcomes depending on the route decision strategy: If the fee is the route selection criteria, the route discovery algorithm will return a path, where only every second node is used as hop. This path consists only out of week connections, but it is the cheapest possible path. The other case is when the quality is used as route selection criteria. In this case, the route discovery algorithm will return a path with every intermediate node along the line between sender and target node is included. This path consists only out of strong connections, but it is also the most expensive path. The route discovery process using the fee as route selection criteria will now work like follows: Node 1 broadcasts a route request, which is received by node 2 and 3. Node 2 and 3 now re-broadcast the request. The broadcast from node 3 is received by node 4 and 5, the broadcast from node 2 by 3 and 4. Node 3 ignores the route discovery packet from node 2, because the fee for the path 1 - 2 - 3 is more expensive than the direct path 1 - 3. Therefore, after the first broadcast of node 2 and 3, the front side of the activity area lies between node 5 and 6, the back side between 3 and 4. If quality is used as route decision parameter, node 3 would not have ignored the received route discovery packet from node 2 because path 1 - 2 - 3 has a higher quality than the direct path 1 - 3. Therefore, in this case node 3 would try to rebroadcast the route discovery packet, this time including the path 1 - 2 - 3. Consequently, the back side of the activity area lies between node 2 and 3. Because of the rebroadcast, the back side of the activity area moves slower than in the case that fee is used as route selection criteria, where no rebroadcast is needed. During the route discovery, this effect accumulates leading to the described effect.

Summarizing, the heat map analyses presented in this section support the outcome of the statistical analysis laid out in the previous two sections. Additionally, the heat maps give insight on the influence of artificial delay, processing time variance, and route selection criteria on the propagation of route request packets. In short, the usage of artificial delay slows down the propagation of route request packets, which accords to the statistical analysis, and addition has an influence on the area in which the route discovery process is active. The value of the processing time variance has an influence on the propagation speed. If in average nodes need longer for the processing of route request packets, the propagation slows down. The last parameter, the route selection criteria, has mainly an influence on the retraction of the active area. When quality is used as route selection criteria, the active area

propagates faster then it retracts, leading to a lager active area compared to fee as route selection criteria.

4.2.5 Conclusion

The simulation results presented in this section lead to the conclusion that the change of racing conditions using artificial delaying decreases the routing protocol's overhead. This comes at the cost of an increase of the time needed for discovering the first route to the target node. However, in most cases, especially if only a small delay is used, the first discovered route is not the best route in terms of fee or link stability, but rather the last route a node learns about. One of the outcomes of the simulation was that even if the route discovery process is slowed down by the artificial delay, the time needed for the detection of the best route to a target does not have to increase necessarily comparing to no delay usage. If the delay is conveniently chosen, the time the algorithm needs to detect the best route to a target is shorter compared to scenarios without usage of artificial delay.

The artificial delay also increases the quality of the route discovery, since it reduces the collision probability. Since flooding techniques used for route discovery are usually based on broadcasts, the correct reception of the transmitted data by all nodes in the radio footprint of a node is not verified. Consequently, if during the discovery phase route discovery packets get lost, the corresponding links are not considered by the routing protocol. Since the usage of artificial delay reduces the number of packets broadcasted during route discovery, the collision probability decreases resulting in a lower probability that links are not considered.

4.3 Evaluation of the Cashflow architecture

In Chapter 3 we presented the architecture of the virtual currency system Cashflow, including algorithms for the realizations of the proposed modules. This section focuses on the evaluation of the architecture using the suggested algorithms, by analyzing the access regulation mechanism as well as the load balancing functionality of the system. Cashflow provides an interface for applications to request channels with specific properties to other nodes. It is in the reasonability of the applications to decide what connection they need and how much they are willing to pay for it. To evaluate Cashflow, such an application was implemented and will be presented in the following subsection. After the description of this evaluation application, the access regulation mechanism of Cashflow, realized by the pricing functionality, will

be analyzed. Before this section concludes, the evaluation of the balancing functionality is presented.

4.3.1 Model of the source

The evaluation application is an example of a simple application using Cashflow. This application is used for the evaluation of Cashflow. Its basic functionality is to open a specific number of channels to a target node and use these channels to transmit data to the target node. The number of channels the application opens depends on a parameter stating the maximum number of open channels and on the fees intermediate nodes charge for forwarding.

The application can be configured by the following parameters:

- *target*: Using this parameter, it is possible to specify the target node.
- *number of channels*: This parameter is used to configure the number of channels the application should open to the target node. It is important to note, that it is not always possible for the application to open exactly as many channels as specified in this parameter because of the fee forwarding nodes charge. This parameter can be also seen as an upper limit, since the application will not open more channels than specified.
- *maximum price*: This is the maximum price the node is allowed to pay for channels. If the application request a new channel and the cumulative price including this new channel lies above this value, the node must reject the offer. In this case, the application is not able to open as many channels as specified by the parameter *number of channels* .
- *channel size*: This parameter states the number of packets that should be transmitted through a channel.
- *packet size*: This parameter specifies the maximum size of packets used for the transmission through the channel. The product of channel and packet size gives the maximum throughput through a channel.
- *channel type*: This parameter states if the fee or the stability of the channel is more important. In all following scenarios, stability is preferred.

The evaluation application performs three tasks in parallel as pictured in Figure 4.37: route management, channel management and data management. The task of the route management function is to trigger the route discovery process if needed. Figure 4.38 pictures the flow chart of this process. After the start of this process, the algorithm immediately triggers a route search. The target of the route search is the node specified by the parameter *target*. After the triggering of route search, the variable *lastRouteSearch* is set to zero. This variable is used to store the time when the last route search was performed. Besides this variable, also the variable *priceOpex* is initialized with the value zero. It is used to store information about the operation costs of the currently active channels. Then this function enters the sleep state. After 0.5 seconds, the function gets active again and checks if at least 10 seconds have passed and the value of *priceOpex* has changed since the last route search. If not, it enters the sleep state again, else, the function triggers another route search and refreshes the values of the variables *lastRouteSearch* and *priceOpex*. Using this mechanism, the node has to wait at least 10 seconds between route searches. The comparison of the current *Opex* value, provided by the *getOpex* function, to the *priceOpex* value stored at the last route discovery time is used to prevent unnecessarily route searches. The function *getOpex()* returns a value, which is the cumulative price of all currently open channels. A change of this value can be caused by the opening or closing of a channel or by an intermediate node, which has changed the fee it charges during the last expansion of the channels activity time. So if a node has already established its channels over a network with low load, causing no changes in the fee the nodes charge, no route search is triggered. This is done to reduce the number of

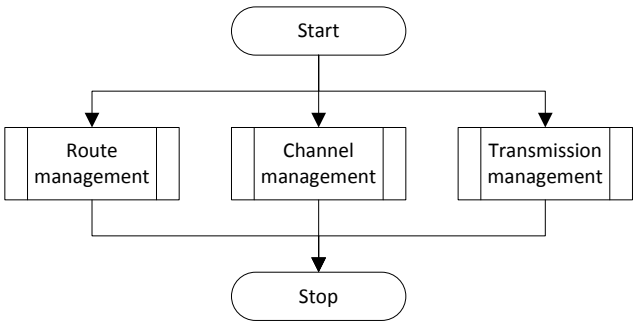


Figure 4.37: Overview over evaluation application

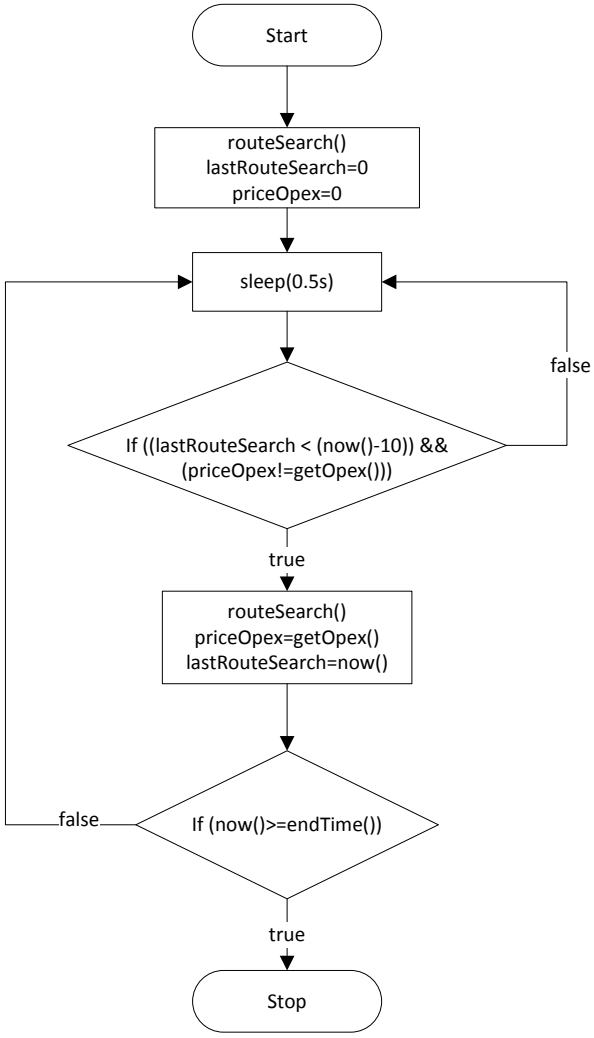


Figure 4.38: Visualization of the route management function

route searches to a minimum, as long as there is no indicator that there was a change in the fee nodes charge, making the current routes obsolete. This process, like the other two management processes, is active until the simulations end.

Figure 4.39 pictures the flow diagram of the channel management tool.

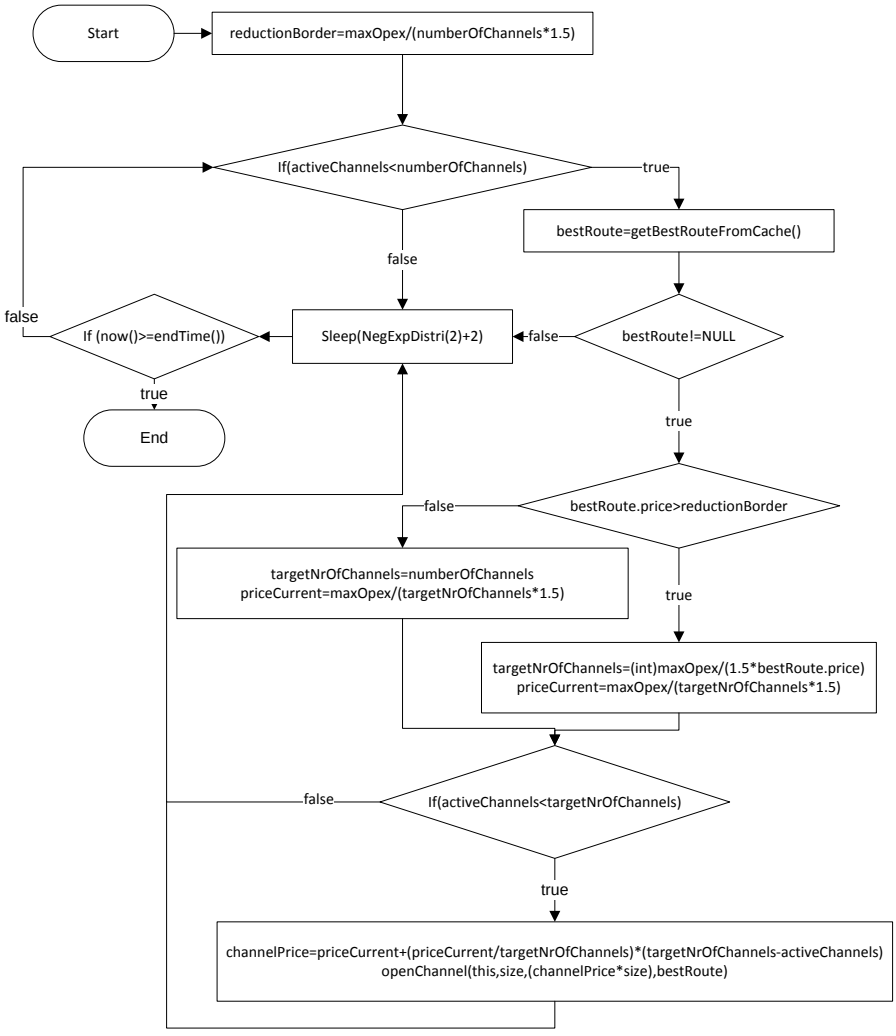


Figure 4.39: Route management function

The purpose of this module is to open the optimal number of channels to the target node. After the start, it calculates the value of the variable *reductionBorder*. This is the quotient of the maximum price the management module is allowed to spend divided by the product of the number of channels this module should open times 1.5. The value of *reductionBorder* is used to decide if the number of channels the module should open can

be reached or if only a smaller number of channels can be opened. The function used for opening a channel requires a price value, which states the maximum fee the channel is allowed to cost. The actual price of the channel can lie beneath this value, but as soon as the price level reaches the value of the maximum fee, the channel is closed. The multiplication of the number of channels with 1.5 during the calculation of *reductionBorder* allows the module to keep channels alive, even if the price increases in average up to 50% compared to the original price. After the calculation of the *reductionBorder*, the route management function checks if additional channels should be opened. If the current number of open channels equals the number of channels the module should open, the module enters the sleeps state and rechecks the number of channels later. If the number of active channels is smaller than the number of channels the module should open, the module requests the cheapest route to the target node from the routing module. If the routing module cannot deliver a route to the target node, the module enters the sleep mode and tries to open channels later again. If a route is found, the price of this route is compared with the value of *reductionBorder*. In the case that the route's fee is larger than the value of *reductionBorder*, the algorithm recalculates the target number of channels. In the other case, if the route's fee is lower, the target number of channels is set to the number of channels the module should open as given by the corresponding parameter. Additionally, in both cases, a basis price for opening channels is computed. The next step verifies if the number of active channels lies beneath the number of target channels. If the target number of channels corresponds to the number of active channels, no additional channels need to be opened. Therefore, the algorithm enters the sleep state again. Otherwise, it calculates the price for the new channel and opens it. The price is based on the base price calculated before and additionally on the ratio between active channels and the target number of channels. The idea behind this price assignment strategy is to assign different maximum prices to the different channels with the result that in the case that all channels are using the same path and the intermediate nodes start to increase their fees, the fees of the open channels will not reach the corresponding maximum fee values at the same time. Considering the case that all channels using the same value for the maximum fee and the actual price reaches this value, all channels would be closed within a minor time span, which is not in the intention of the user. Using staggered maximum prices, an increase of the fee causes also a staggered release of channels, giving the network time to react on the resulting drop in the network load to which the pricing function of the nodes is correlated. Additionally, the

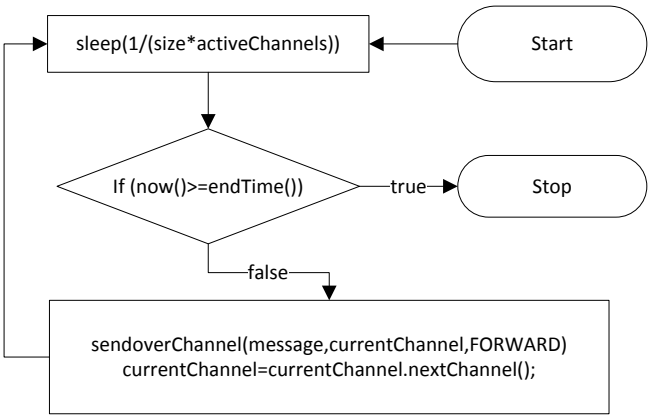


Figure 4.40: Visualization of the transmission management function

source node does not lose all channels at nearly the same time. After this state, the algorithm returns to the sleep state as pictured in the flow chart.

The last function depicted in Figure 4.40 is the transmission management function. It periodically transmits messages over the open channels. The sending rate depends hereby on the number of open channels and their size, as given through the configuration parameters.

In the following scenarios, this application is used to open channels depending on the fee and generate traffic over these channels. Additionally, it can be seen as a prototype of an application using Cashflow.

4.3.2 Evaluation of fee calculation

The regulation of access is tightly coupled with the pricing mechanism. In regions with high network activity nodes increase the fee they charge, making routes through these regions less attractive. The high fee results in a decrease of open channels, since it is not affordable for nodes to maintain all open channels. Additionally, nodes might reschedule parts of their communication until communication is cheaper. This section focuses on the analysis of the pricing function and its ability to regulate the access to the network.

For the analysis, a chained network with eight nodes, as pictured in Figure 4.41, is used. The distance between nodes amounts to 25 meter. Using the

configuration parameters for the physical and MAC layer as described in Section 4.1, every node has strong connections with his direct neighbors and weak connections to two-hop neighbors. In all following scenarios, the evaluation application is executed on node 1. The application running on this node tries to open channels and transfers packets with a size of 2250 byte to node 8.

The pricing module used in the simulations presented in this section corresponds to the pricing module suggested in Section 3.3.7. Recapitulating, the pricing function is configured by the following parameters: *basePrice*, *batteryPenalty*, *maxUsage*, *stepSize*, *preferedThroughput*, *maxThroughput*, *throughputTolerance*. The sum of *basePrice* and *batteryPenalty* form the minimum fee a node charges for forwarding packets. Every node updates its price once every second. If the current throughput of a node lies above the sum of *preferedThroughout* and *throughputTolerance*, the fee is increased by the value stated in the parameters *stepSize*. In the case that the current throughput lies beneath the difference of *preferedThroughout* and *throughputTolerance*, the fee is reduced by the value of *stepSize*. In the case that the node had to reject a channel request because the load of the new channel would lead to a throughput larger than stated in *maxThroughput*, or if the current usage of the shares medium exceeds the value of *maxUsage*, the fee is additionally increased by two times the value of *stepSize*. This summarizes the functionality of the used pricing module. A detailed description can be found in Section. 3.3.7.

Table 4.4 shows the basic configuration of the application and pricing modules. Some of these values change depending on the simulation scenario but most values are the same in all simulation scenarios. The first simulation scenario, scenario C1, uses exactly these parameters. This scenario represents a low load situation. As depictured in Figure 4.42, the evaluation

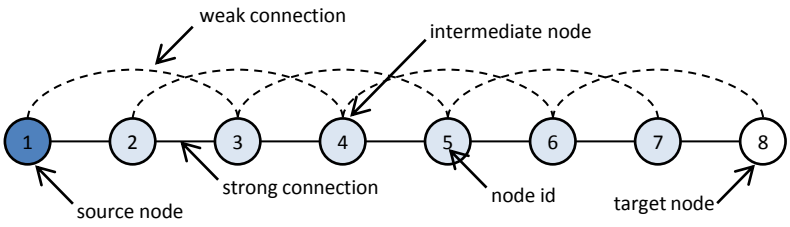


Figure 4.41: Chain topology used for the evaluation of the pricing function

Parameter	Value
evaluation application	
target	node8
number of channels	5
maximum price	700
channel size	5
segment size	2250 byte
channel type	stabillity
pricing module	
basePrice	1
batteryFee	0
maxUsage	0.7
stepSize	0.01
preferedThroughput	540 kbit/s
maxThroughput	900 kbit/s
throughputTolerance	54 kbit/s

Table 4.4: Default parameters of evaluation application and pricing function

application can afford to open five channels as ordered by the corresponding parameter and keeps these channel alive during the whole duration of the simulation of 2000 seconds. Since the cumulative throughput of 450 kbit/s is lower than the preferred amount of 540 kbit/s, and the usage of the shared medium is beneath 0.7, as pictured in Figure 4.44 and 4.45, nodes keep their fee at the minimum level during the whole simulation, as pictured in Figure 4.43 and 4.46. In this and all following scenarios, the shared medium's usage curve is similar for all nodes in the same scenario, with the difference of an offset. Therefore, only the shared medium's usage curve of node 4 is depictured for this and all other scenarios. Figure 4.45 compares the average usage of the shared medium as detected by the nodes. The differences between the values allow drawing conclusions about the actual curve for other nodes. The shared medium's usage measured by nodes is the ratio between active and inactive phases on the radio channel, including the time nodes have to wait because of a set NAV-timer as active phase. Therefore, it is not surprising that nodes with two two-hop neighbors senses a higher usage of the shared medium than nodes having only one two hop neighbor like node 2 and 7.

In the next scenario, scenario C2, node 1 tries to open eight channels with a cumulative throughput of 720 kbit/s, which lies over the preferred amount of

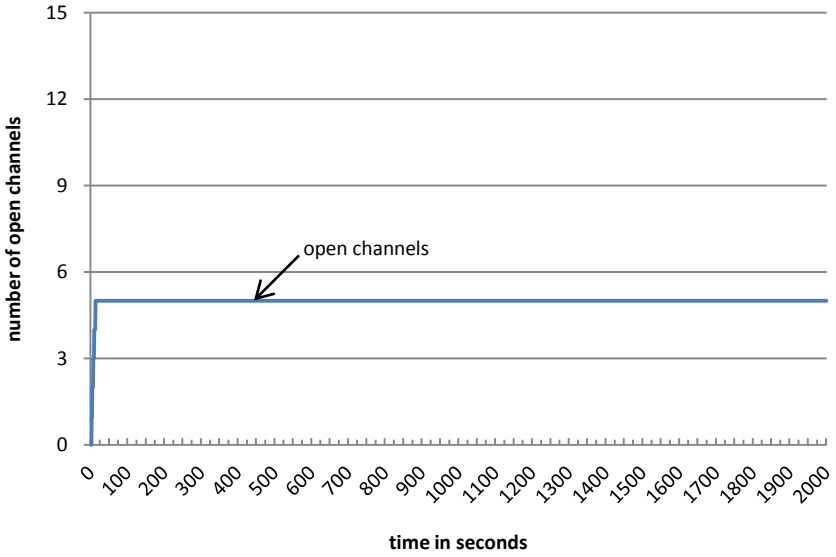


Figure 4.42: Number of open channels between node 1 and 8 in scenario C1

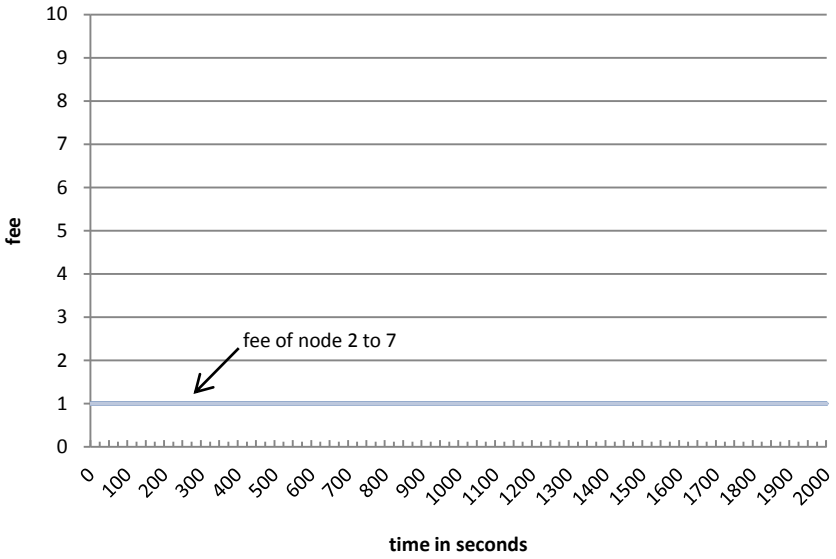


Figure 4.43: Comparison of nodes fees in scenario C1

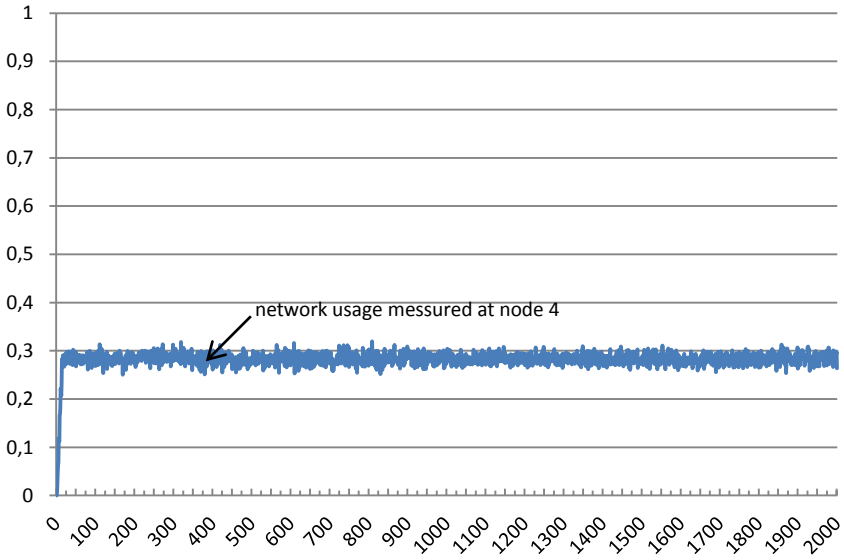


Figure 4.44: Shared medium's usage as monitored by node 4 in scenario C1

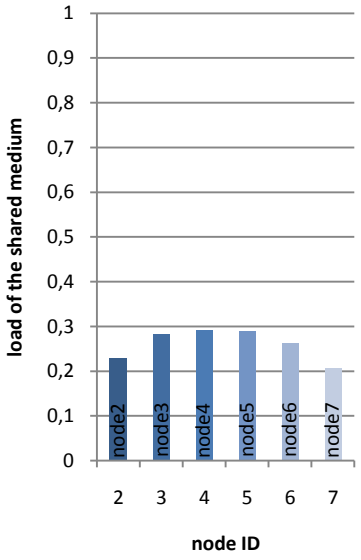


Figure 4.45: Comparison of nodes average load in scenario C1

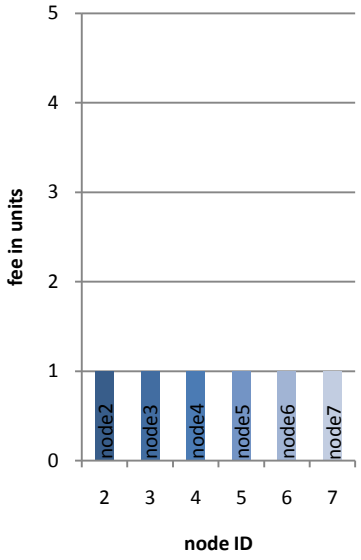


Figure 4.46: Comparison of nodes average fees in scenario C1

540 kbit/s but below the maximum throughput of 900 kbit/s. When node 1 requests eight channels, the intermediate nodes grant all those channels but react with an increase of the fee they charge, as it is pictured in Figure 4.48. All nodes increase their fee by a similar rate, until the fee reaches a value whereby it is not possible for the application function to maintain all open channels without exceeding the maximum amount the application is allowed to spend. Assuming an average price of 2.3 units per packet per channel charged per node, the fee for eight channels would amount to 828, which exceeds the maximum amount of 700 units. Also seven channels would exceed the maximum price. Therefore, at this price level and with the given parameters, only six channels are affordable for the evaluation application as pictured in Figure 4.47. Those six channels produce a throughput of 540 kbit/s, which fits exactly to the preferred throughput of the intermediate nodes. Consequently, they do not increase the fee anymore leading to a stable network state. As pictured in Figure 4.49, the network load never exceeds the value of 0.7, which is the maximum allowed load on the radio channel as given by the corresponding parameter. Therefore, the shared medium's usage has no influence on the fee calculation in this scenario.

Scenario C3 is the first overload scenario in this simulation series. The preferred throughput of each intermediate node is set to 540 kbit/s, the maximum throughput to 630 kbit/s. Node 1 tries to open nine channels corresponding to a throughput of 810 kbit/s. As pictured in Figure 4.52, the number of open channels fluctuates between seven and eight channels, until the number drops to six channels and stabilizes at this value. Comparing the number of open channels with the activity level, as pictured in Figure 4.54, it is observable that the activity level is constant during the whole simulation. This effect is caused by the exchange of control information inside channels during the channel's establishment. When a node wants to open a channel, it sends a channel request packet to the destination node. This request packet is submitted outside a channel causing the intermediate nodes to open a channel. The target node return a packet including information about the fees charged for the channel or information that the channel could not be used and consequently will be closed immediately because of an overload situation. After forwarding a control packet including information about an overload situation, the channel is closed by the node. This leads to the fluctuation in the number of open channels but causes no significant traffic. The benefit of forwarding the request packet to the target node, even if it is clear for some intermediate nodes that the channel will never be used because of a local overload situation, is that nodes along the path, which are not in an overload situation, learn about the demand

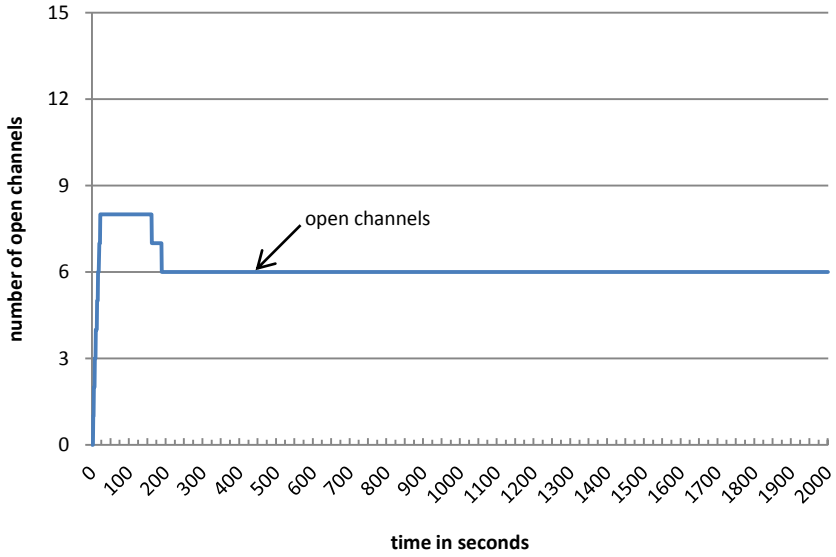


Figure 4.47: Number of open channels between node 1 and 8 in scenario C2

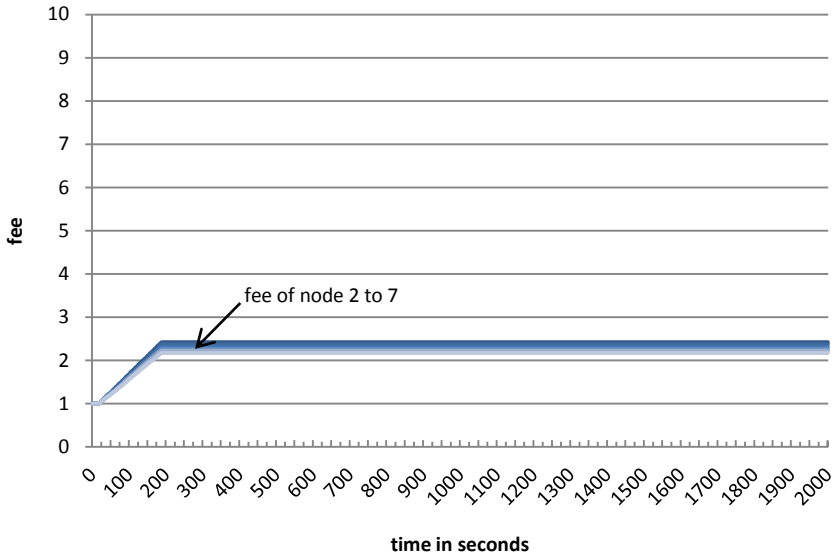


Figure 4.48: Comparison of nodes fee in scenario C2

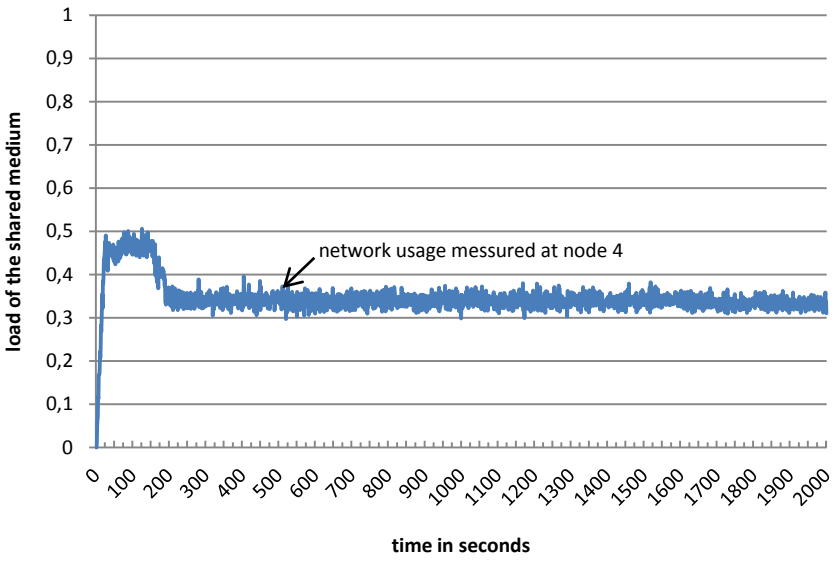


Figure 4.49: Shared medium's usage as monitored by node 4 in scenario C2

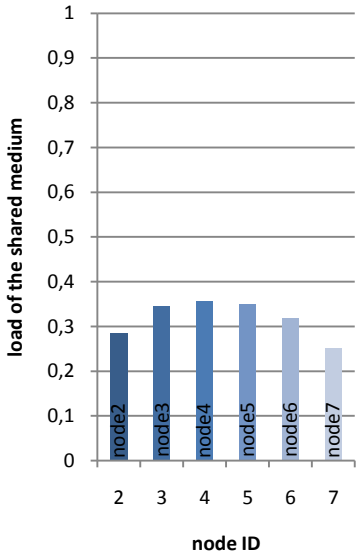


Figure 4.50: Comparison of nodes average load in scenario C2

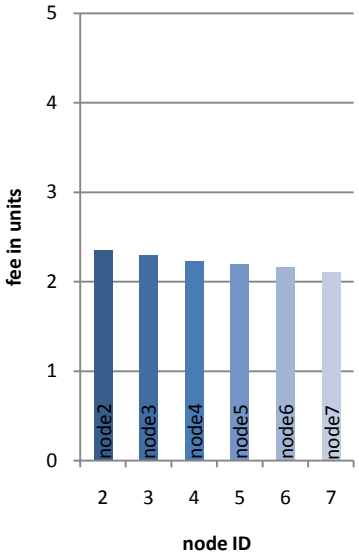


Figure 4.51: Comparison of nodes average fees in scenario C2

and can use this information to adjust their fees. Additionally, this leads to a faster adjustment of the fee for a channel if more nodes along a path are in overload situations. As pictured in Figure 4.53, all nodes adjust their fee and not only node 2, which is the first node observing that more channels are requested as it can handle. Comparing the fee development of this scenario with the fees from scenario C2 as pictured in Figure 4.48, the nodes increase the fee faster when overload is detected.

The next scenario, C4, is again an overload scenario. This time the maximum throughput accepted by nodes is set to such a high value that is does not have any influence on the simulation. Additionally, the maximum usage of the shared medium is reduced to 0.5 and the number of channels the evaluation application tries to open is increased to 20. Furthermore, the price the application is allowed to spend is set to 1500 units. The consequence of this configuration is that the evaluation application tries to produce a load resulting in a usage of the shared medium above 0.5. In Figure 4.57, presenting the number of open channels, a similar fluctuation of open channels is observable like in the last scenario, also stabilizing at the value where the throughput produced by the channels matches the preferred throughput. Figure 4.59 shows the usage of the shared medium as observed by node 4. It is worth noting that even during the overload situation the node manages to keep the usage of the shared medium nearly all the time beneath the maximum value of 0.5. This is the case because nodes estimate the additional load a channel would cause and prevent the establishment of channels if the sum of the measured and the estimated load lies above the maximum value. In this scenario, the development of the fee is especially interesting as shown in Figure 4.58. Nodes 3 to 5 increase their fee with a higher rate than node 2, 6 and 7 for the first 250 seconds. Then all nodes increase the fee for the same rate until the systems reaches a stabile state. This behavior can be explained using Figure 4.60. Nodes positioned in the middle of the chain topology are in the footprint of more nodes than nodes near an end of the chain. Therefore, they detect more network activity meaning that the shared medium is used more than the shared medium around nodes near an end of the chain. Consequently, these nodes are confronted with a high load situation earlier, in terms of open channels, and prevent the establishment of additional open channels. At the same time, the detected overload causes a higher increase of the fee in comparison to the other nodes. When the overload situation is over, all nodes in the network increase their fee at the same rate, because for all nodes the throughput caused by the channels lies above the preferred value but below the maximum throughput value and additionally, the load on the shared medium is also below the maximum

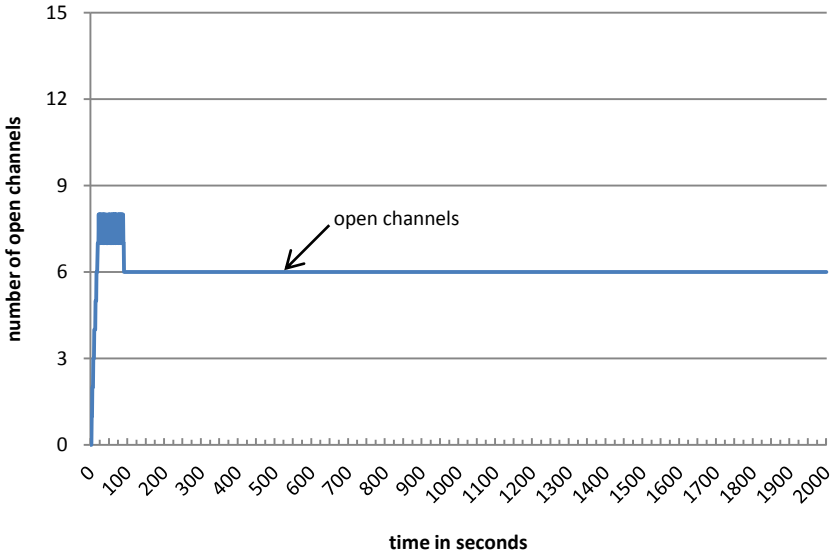


Figure 4.52: Number of open channels between node 1 and 8 in scenario C3

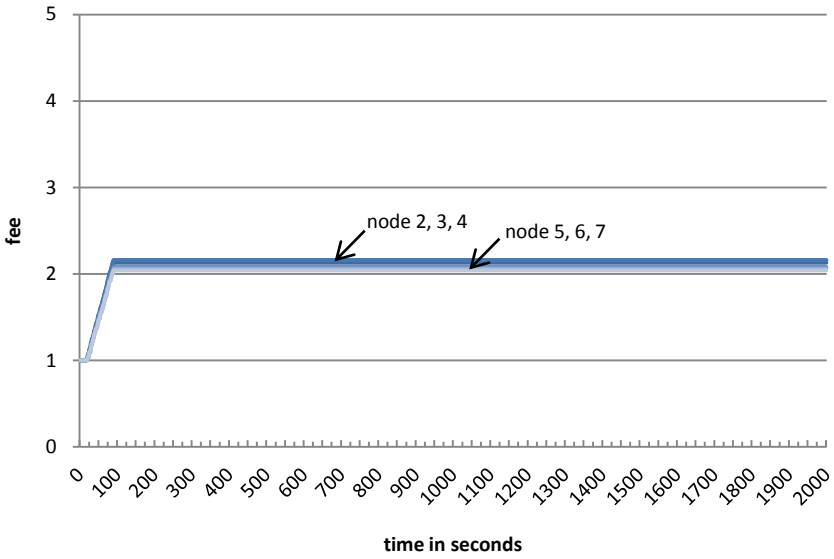


Figure 4.53: Comparison of nodes fee in scenario C3

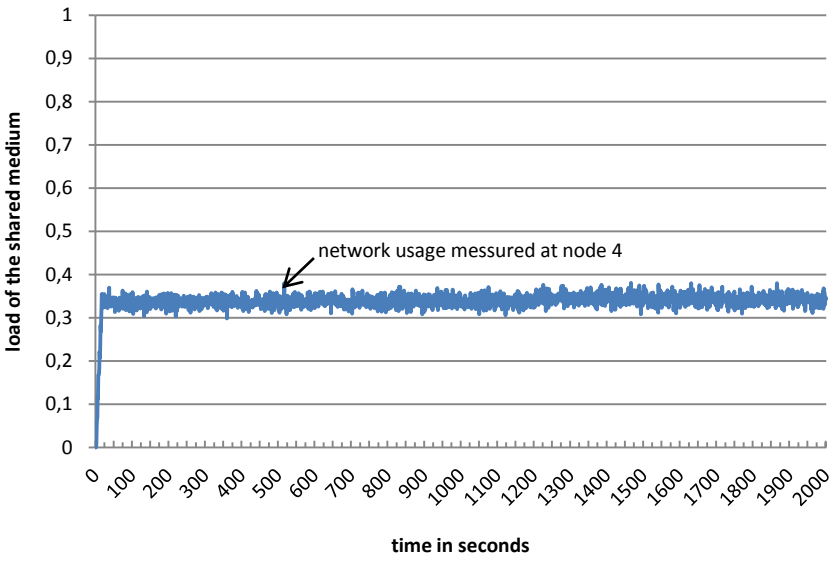


Figure 4.54: Shared medium's usage as monitored by node 4 in scenario C3

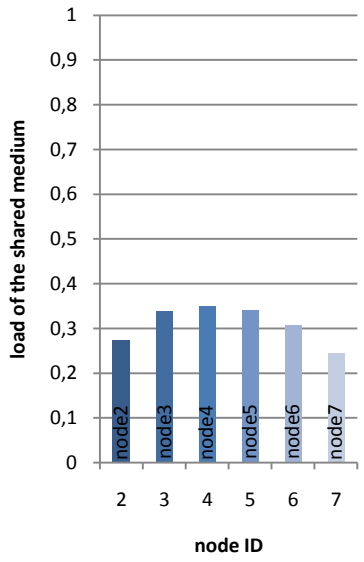


Figure 4.55: Comparison of nodes average load in scenario C3

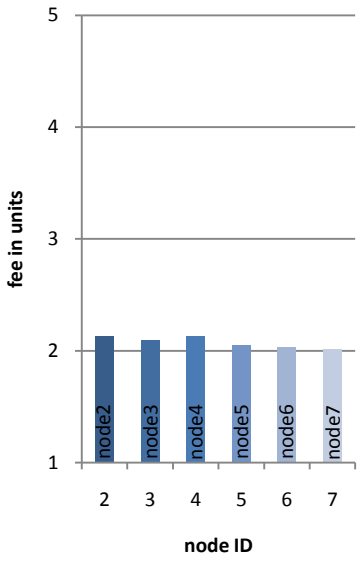


Figure 4.56: Comparison of nodes average fees in scenario C3

value.

In all the previously presented scenarios, all nodes use the same parameterization. In scenario C5 one node, node 4, uses different parameters than the other nodes. Therefore, the influence of the node's parameterization on each other is observable. In this scenario, all nodes with the exception of node 4 uses the standard parameters as described in Table 4.4. Node 4 differs from the other node by the value of the preferred throughput, which is set to 360 kbit/s in contrast to 540 kbit/s used by the other nodes in this simulation. The evaluation application running on node 1 tries to open eight channels, as pictured in Figure 4.62, with a cumulative load of 720 kbit/s. This value lies between the value of the preferred and the maximum throughput for all nodes. Therefore, as pictures in Figure 4.63, all nodes increase the fee they charge, forcing the evaluation application to reduce the number of channels. After 189 seconds, the evaluation application reduces the number of open channel to six, which corresponds to a throughput of 540 kbit/s. For all nodes, with exception of node 4, this value correlates to the preferred value and therefore they stop to increase their fee. Since node 4 prefers even a lower load, it continues to increase the fee, forcing the evaluation application to reduce the number of open channels to five after 353 seconds, resulting in a throughput of 450 kbit/s. Since this load lies beneath the preferred throughput of all the other nodes, these nodes start to decrease their fee to motivate the evaluation application to open more channels. Since these nodes together can adjust their fee faster than node 4 alone, the cumulative price reaches a value where the evaluation application is able to open a sixed channel again. After some time, node 4 has increased the fee to a level so that the evaluation application is forced again to close a channel, resulting in a drop of the fee the other nodes charge. This interplay between node 4 an the other nodes continue, until the other nodes charge a fee of 1 and are therefore not able to reduce their fee any further, because this value is the minimum fee as given by the parameters in this scenario. Node 4 continues to increase its fee until it reaches a value where the evaluation application reduces the number of channels to four, which corresponds to the preferred throughput value of node 4.

Summarizing, the results of the presented simulations show that the fee's adaption, according to the usage of the shared medium and the throughput trough nodes, provides a feasible mechanism to regulate the access to the network. Because of the used chain topology, in the presented scenarios the evaluation application was forced to transmit the data along only one path. Therefore, it had no possibility to avoid nodes with limited capacity left and use traffic balancing to avoid overload scenarios, as it will be presented

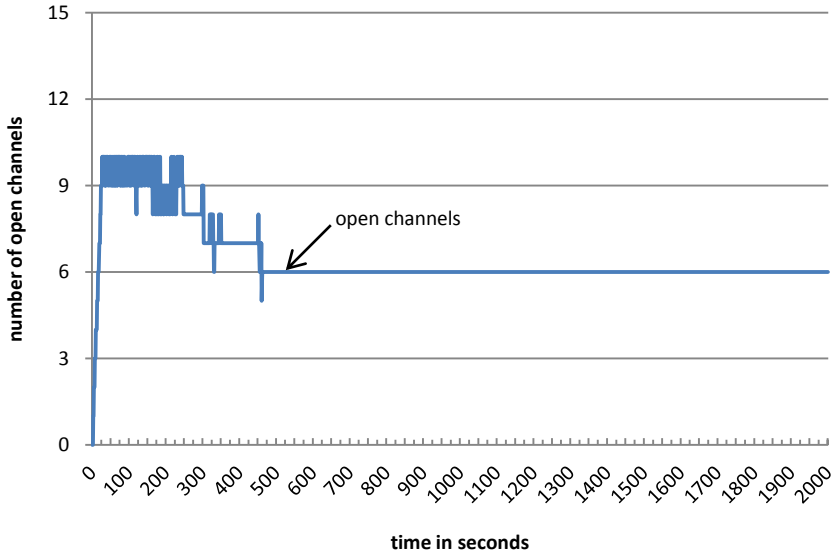


Figure 4.57: Number of open channels between node 1 and 8 in scenario C4

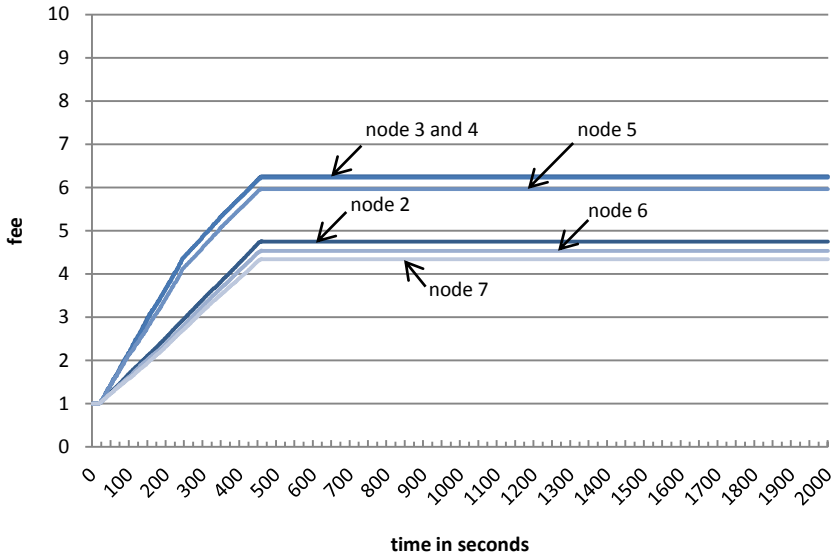


Figure 4.58: Comparison of nodes fee in scenario C4

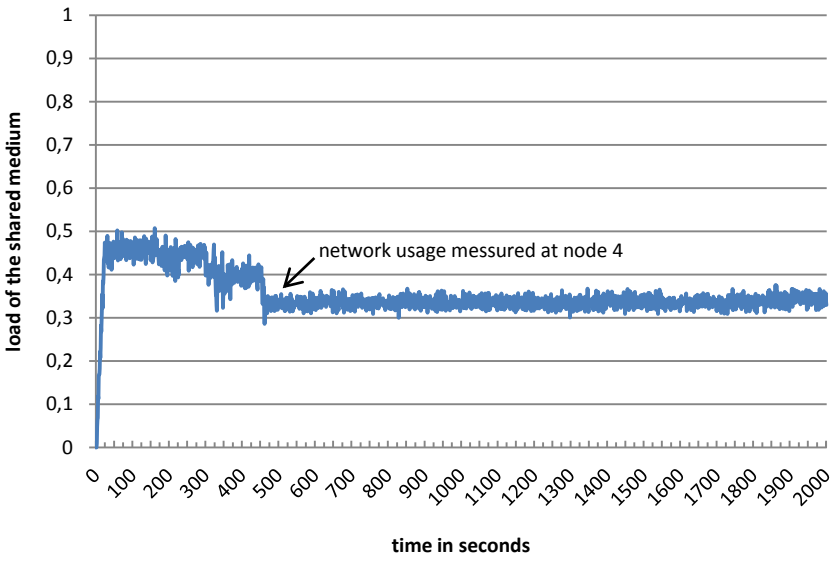


Figure 4.59: Shared medium's usage as monitored by node 4 in scenario C4

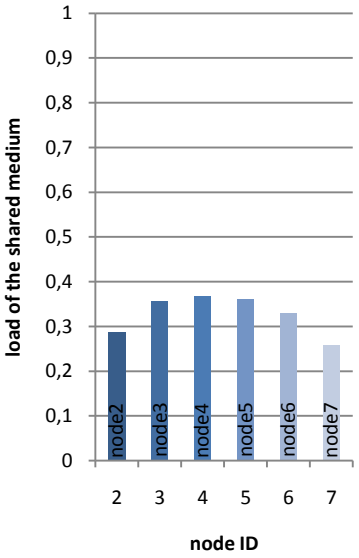


Figure 4.60: Comparison of nodes average load in scenario C4

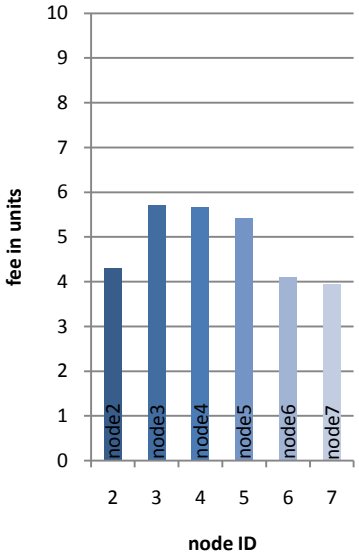


Figure 4.61: Comparison of nodes average fee in scenario C4

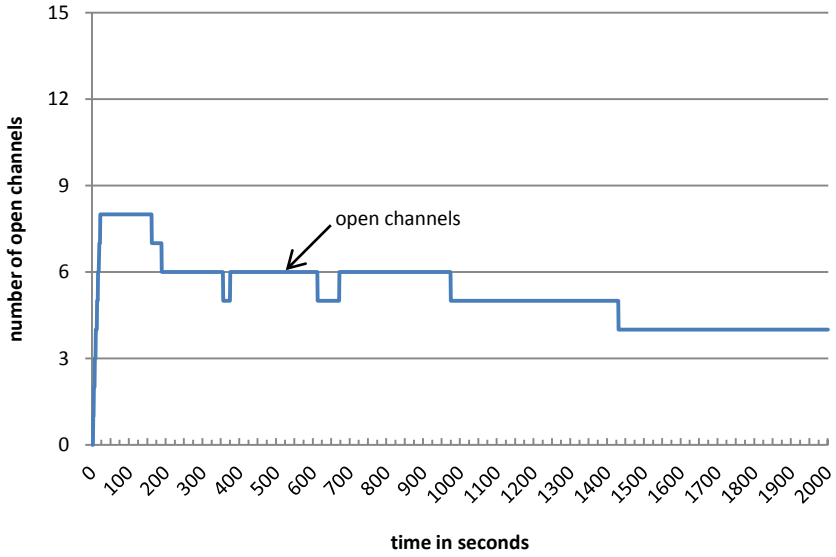


Figure 4.62: Number of open channels between node 1 and 8 in scenario C5

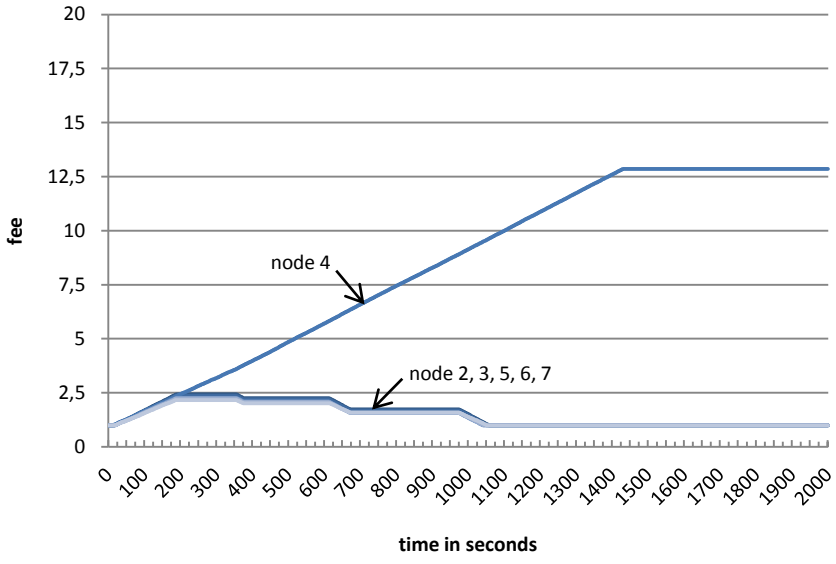


Figure 4.63: Comparison of nodes fee in scenario C5

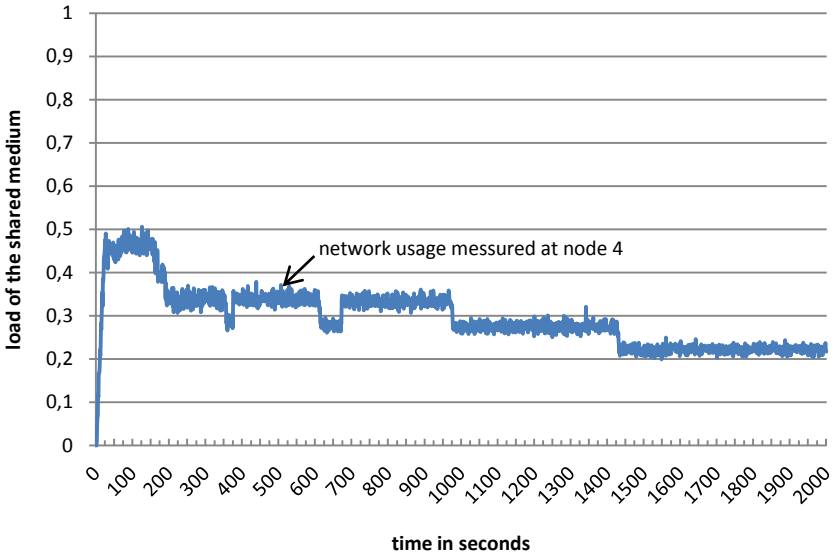


Figure 4.64: Shared medium's usage as monitored by node 4 in scenario C5

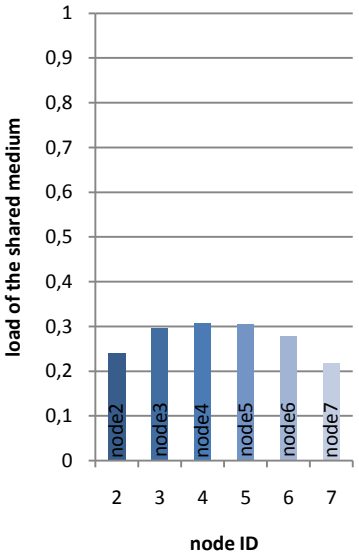


Figure 4.65: Comparison of nodes average load in scenario C5

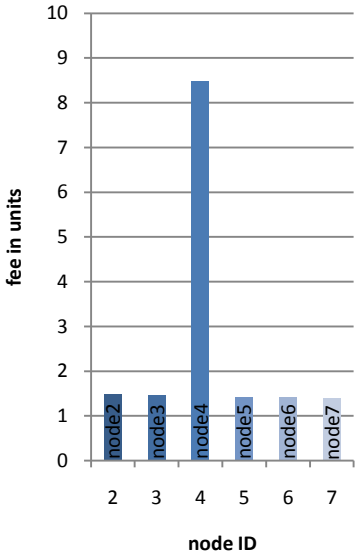


Figure 4.66: Comparison of nodes average fee in scenario C5

in the following section.

4.3.3 Evaluation of balancing functionality

The combination of channel and market concept used by Cashflow results in an implicit load balancing function. To visualize the effect of the load balancing function, this chapter compares heat maps from scenarios with activated load balancing functionality to scenarios with deactivated load balancing. Since load balancing is an implicit function of Cashflow, a direct deactivation is not possible. However, by deactivating the pricing function, also the load balancing function becomes inactive.

Similar to the propagation analysis, for the evaluation of the balancing functionality a grid topology with 32 to 32 nodes was used, with a distance of 30 meter between nodes. Table 4.5 gives an overview over the used parameters for the pricing and application modules. It is worth noting that the application modules used in the following scenarios only open channels to target node but, in contrast to previous simulation series, do not send dummy data over these channels. This was done to execute the simulations in reasonable time. If data traffic would have been simulated like in the previous scenarios, the execution of a single simulation would have taken

Parameter	**Value**
evaluation application	
number of channels	50
maximum price	25000
channel size	5
segment size	1000 byte
channel type	stabillity
pricing module	
basePrice	1
batteryFee	0
maxUsage	0.7
stepSize	0.01 or 0
preferedThroughput	400 kbit/s
maxThroughput	960 kbit/s
throughputTolerance	54 kbit/s

Table 4.5: Default parameters of evaluation application and pricing function for the evaluation of Cashflows load balancing functionality

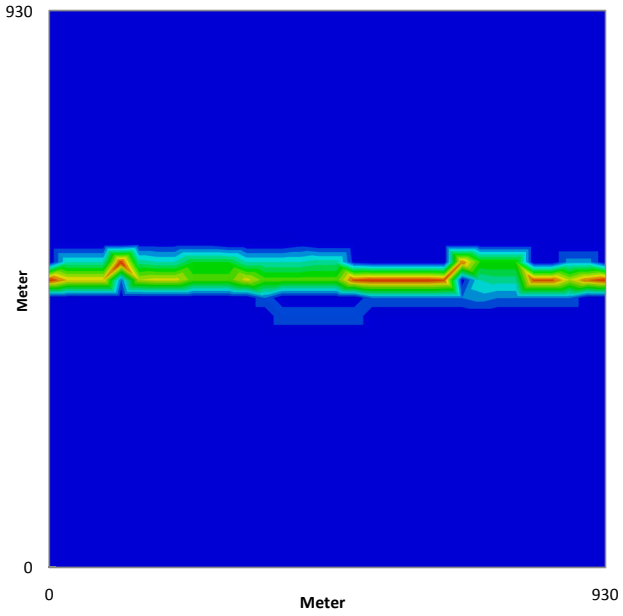


Figure 4.67: Scenario with one source using no load balancing

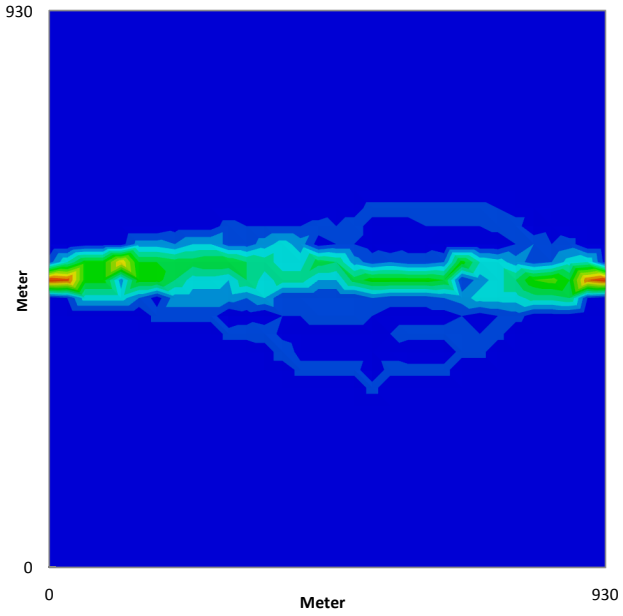


Figure 4.68: Scenario with one source using Cashflow for load balancing

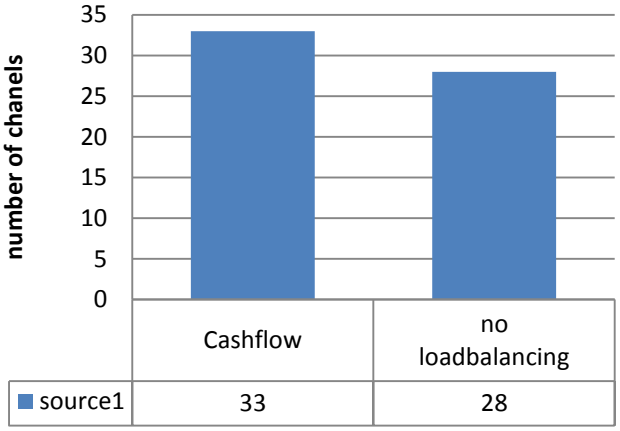


Figure 4.69: Comparison of open channels depending on the usage of load balancing

months instead of days. As a consequence of this configuration, local overhead is detected by an overpass of the maximum throughput value and not by an overpass of the shared medium's usage. This reflects to scenarios where nodes allow only a small throughput, with the consequence that an overload of the shared medium is not possible.

In the first scenario, one node located on the left side of the network tries to open 50 channels to the opposite node on the right side. The heat maps in Figure 4.67 and 4.68 represent the network activity after 500 seconds, whereas in Figure 4.67 the pricing function and consequently the load balancing function was deactivated. Areas with high loads are colored in yellow and red, whereas areas in green and blue shades mark areas with medium or low load, as preferred by nodes. It is observable that in both scenarios, the nodes around the source and target node are very active because of the small number of possible paths. However, the effect of Cashflow's load balancing functionality is observable by comparing the intermediate parts of both scenarios. If no load balancing functionality is used, as pictured in Figure 4.67, the source node prefers the shortest path, resulting in a number of high loaded areas. If the load balancing function is activated by activation of the pricing function, the shortest path becomes unattractive as soon as it gets crowded. Therefore, the system uses alternative paths resulting in a distribution of traffic. Figure 4.69 compares the number of open channels. Using load balancing, the source node can open more channels. Since the routing algorithm prefers stable route, the source node has three potential neighbor nodes to route the traffic over

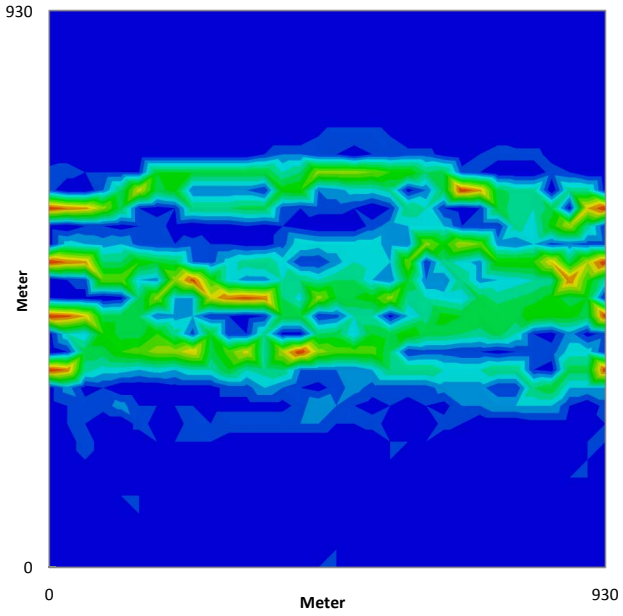


Figure 4.70: Scenario with four source nodes using no load balancing

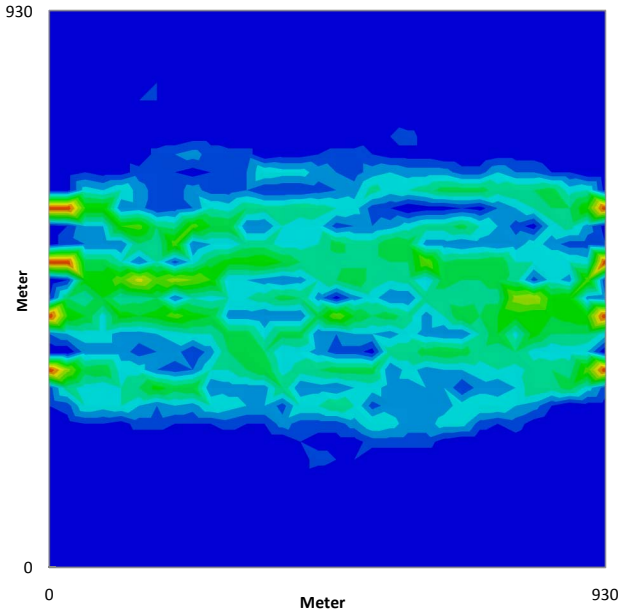


Figure 4.71: Scenario with four nodes using Cashflow for loadbalancing

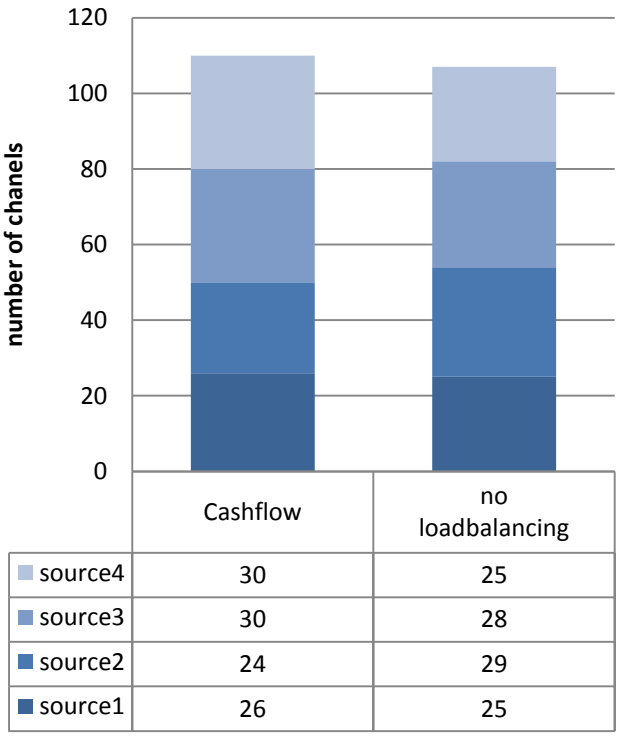


Figure 4.72: Comparison of open channels depending on the usage of load balancing

them. Together, these three nodes preferred in the given configuration 1200 kbit/s of throughput, which corresponds to 30 channels. Consequently, by regulating the price, the nodes will limit the number of open channels to nearly 30 channels. Therefore, in this scenario, the fee is the limiting factor. When the load balancing is deactivated, the maximum throughput of the one-hop neighbor nodes becomes the limiting factor. Even if the source node could in this case theoretically transmit up to 2880 kbit/s, the node can use only a fraction of this bandwidth, since the node is not able to balance the traffic over different paths. Because the source node prefers the shortest route to the target, it prefers one one-hop neighbor node to the other two one-hop neighbor nodes, resulting in an overload of the node, which consequently rejects additional channels.

In the next scenario, four nodes on the left side of the network try to open 50 channels to opposite nodes on the right side. Figure 4.70 and 4.71 show the heat maps of the network activity for this scenario, whereas in Figure 4.70 no load balancing is used in contrast to Figure 4.71. Similar to the last

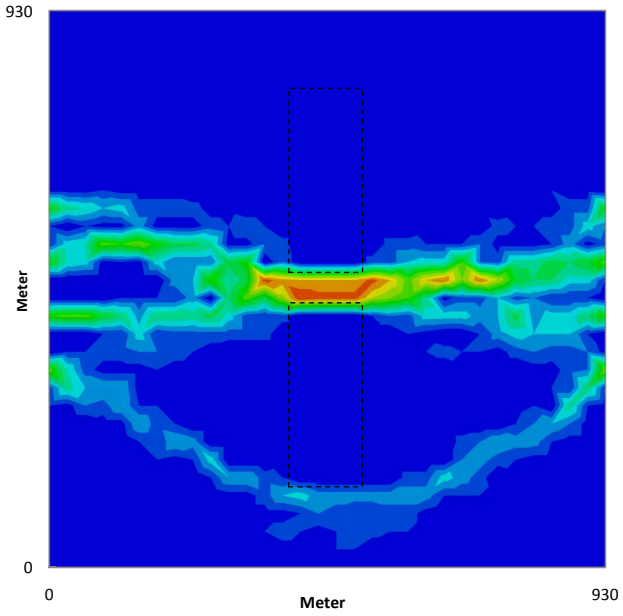


Figure 4.73: Scenario with four source nodes using no load balancing and two obstacles

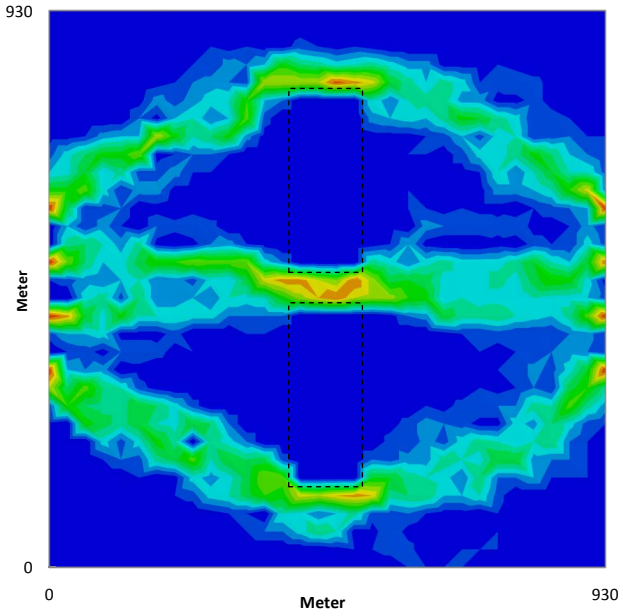


Figure 4.74: Scenario with four sources using Cashflow and two obstacles

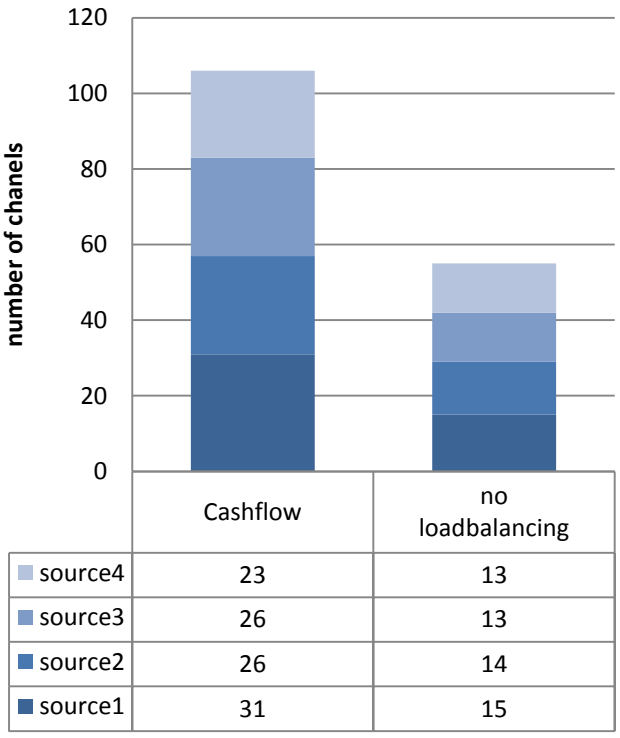


Figure 4.75: Comparison of open channels depending on the usage of load balancing

scenario, the existence of high loaded areas is observable if no load balancing is used. Additionally between high active areas, inactive areas exist. This is not the case if the load balancing functionality is activated. As pictured in Figure 4.71, Cashflow is able to avoid local overload situations. This also reflects in the number of channels the nodes could open. As pictured in Figure 4.72, when the load balancing function is used, the cumulative number of open channels overpasses the cumulative number of open channel when the load balancing function is inactive. Again, this is caused by the rejection of channels by overloaded nodes.

The last scenario is similar to the scenario described before, with the difference that an artificial bottleneck was inserted by deactivating a number of nodes. The result is a small corridor in the middle of the network with a width of two nodes and a length of four nodes. In the heat maps pictured in Figure 4.73 and 4.74, the inactive nodes are located in the boxes with dotted borders. The nodes within the bottleneck are not able to provide the bandwidth requested by the four nodes on the left border of the network.

However, in these scenarios the nodes can avoid the bottleneck by transmitting data along alternative routes. These alternative routes are longer and therefore in most cases not detected by routing algorithms preferring the shortest path. This is not the case if fee sensitive routing is used. If a pricing system is used, which increases the fee nodes charge in high load situations, the long routes become feasible, if the price level of the node in the bottleneck is high. Therefore, these routing protocols will prefer the longer routes as soon as there is an overload situation in the bottleneck. This can be seen in Figure 4.73 and 4.74. If no load balancing is used, all nodes prefer the shortest route through the bottleneck. This results in an overload and consequently the nodes within the bottleneck start to reject channels. If the pricing system is activated and therefore the load balancing system, the system stabilizes after some time. At the beginning, when all nodes charge the same fee, all four nodes will also prefer paths through the bottleneck because these are the cheapest routes. Because of the resulting high load situation, nodes within the bottleneck start to increase their fees. After some time, the fees reach a level where the longer bypass paths become cheaper than the paths through the bottleneck. As result, nodes start using the bypass and reduce the load in the bottleneck. This stabilizes the pricing level within the bottleneck.

The usage of the bypasses allows the nodes to open more channels to the target nodes, compared to the case when only the path thought the bottleneck is used. Figure 4.75 shows a comparison of the number of open channels. In the given scenario, the usage of Cashflow's load balancing system allows an increase of the throughput of 92,73%. This shows that in scenarios where alternative routes between nodes exist, the load balancing system is not only useful for the prevention of overload situations, but additionally allows increasing throughput.

4.3.4 Conclusion

Summarizing, the combination of pricing and channel concept results in an implicit load balancing and access regulation mechanism. By increasing their fee, high loaded nodes make them self less attractive as intermediate node. Therefore, nodes prefer paths along low loaded nodes, resulting in a balancing of the load. Additionally, since high loaded nodes can reject channel requests in case the additional channels would cause a local overload, the access to the network can be refused, if the transmission through the network is temporarily not feasible.

4.4 Summary

In this chapter we have presented the evaluation results of Cashflow. Starting with a detailed analysis of the routing protocol, it was shown that influencing racing conditions during the flooding phase of route discovery, by using artificial delay, allows to increase the performance of the route discovery strategy. To analyze the positive effect of the artificial delay, and to evaluate the impact of network density and node's processing time on the performance, a number of simulation series have been performed. The obtained results indicate that with an increase of the artificial delay, the overhead caused by the routing protocol decreases to the point where each node only broadcasts one packet for route discovery instead of n^2 packets, which is the theoretical maximum for the proposed route discovery strategy in a network with n nodes. (Compare with [Anderegg03] and [Eidenbenz05]). Additionally, it was shown that the processing time variance of nodes have a negative influence on the effectiveness of artificial delaying, meaning that in systems were the algorithm is challenge by large processing time variations, higher artificial delay values are needed. Concerning the network density, the results indicate that in dense networks, where collisions occur frequently, the usage of the delay increases the quality of the route discovery, by reducing the collision probability.

The second part of this chapter focused on the interaction of Cashflow's components. Using an exemplary application, the functionality of the proposed pricing function was analyzed. It was shown how the pricing function makes nodes unattractive for other nodes in situations where local load exceeds the user's preferred load or if the shared medium is locally overloaded. In a next step, Cashflow's ability to balance load and restrict access to the network in overload situation, as result of the least-cost routing scheme in combination of Cashflow's rescheduling ability of data transmissions, was presented. The main outcome of this analysis is that by combining the load and context sensitive pricing function with the least cost routing scheme, an effective load balancing mechanism is implicitly integrated into Cashflow, resulting in a better network usage.

Chapter 5

Summary and future work

In this thesis, we presented Cashflow, a virtual currency system to prevent selfishness in mobile ad hoc networks and to encourage nodes to participate. Cashflow distinguishes itself from other virtual currency systems by a number of unique features. In contrast to other virtual currency systems, Cashflow is bases on a channel concept. If a node wants to transmit data to another node in the network, it has to request a channel from the nodes along the path to the target node, specifying the channel's size and duration. The intermediate nodes return an offer, which the source node can accept or decline if the claimed fee is too high. This concept has a number of advantages. First, the source node could decline the channel and reschedule the data transmission if the current price level is too high. This is an important feature for the user acceptance since in most other virtual currency systems the user does not know how much a transmission will cost before he starts data transmission. Second, this concept decreases the charging overhead, since it allows charging packets in bulk so that the system has not to perform charging for every single packet. A third benefit of the channel concept is that it prevents local overload. If nodes receive a channel request, which would overpass the available bandwidth, they can decline it. This also result in a more deterministic behavior, since open channels does not get interrupted by new channels.

Another important characteristic of Cashflow is its pricing system. In contrast to other virtual currency systems, it implements a market system, which determines the fee based on supply and demand. The supply is regulated by two parameters. First, the maximum supply is determined by

the maximum available bandwidth, which depends on the used radio technology. Second, the node's user can specify, which ratio of his bandwidth can be used for forwarding packets for other nodes. Therefore, the user is in control of his participation degree, which is an important factor for user acceptance, but neglected by other virtual currency systems. The fee for packet forwarding gets periodically adapted based on the demand. If the demand lies over a certain threshold, the pricing function increases the fee. Otherwise, if the demand is low, the fee gets decreased. Additionally, the pricing function allows to include the node's context as additional parameter into the pricing. This is done by adding penalties to the fee, depending on the nodes context. For instants, nodes running on battery add a battery penalty to the fee. The actual amount of the penalty depends on the charging level of the battery. Using this concept, nodes get less attractive for other nodes as forwarding node while running on battery.

Cashflow uses a special routing algorithm, which is optimized to support the needs of virtual currency systems. The algorithm allows to find the cheapest route to a target node while considering restrictions related to the quality of the used links. This is also a feature of Cashflow, which distinguishes this virtual currency system from other systems. Most virtual currency systems solve the problem how to determine the fee and pay for packet forwarding along a given route, but they do not provide functionality to determine the cheapest path to the target.

The combination of market based pricing and the ability to detect the cheapest route to a target node result in a load balancing system, which is another important feature of Cashflow. Since nodes increase their fees as the traffic increases, other paths with low traffic gets more attractive. Therefore, nodes automatically balance their traffic to profit from the low price of alternative routes.

Like other virtual currency systems, Cashflow can be used in pure mobile ad hoc networks. However, Cashflow additionally supports the integration of mobile ad hoc networks into the Internet. It provides a special search function, which allows nodes within an ad hoc network to search for nodes providing Internet access. Using Chashflow's payment system, nodes can pay other nodes for the usage of their Internet connection. By adapting the mobile IP concept, nodes within an ad hoc network are also reachable over the Internet. In this context, Internet access is seen as a service for nodes. Since the search and the payment function are implemented in a generic way, Cashflow can be used as platform to for other pay-services, which makes it interesting for business scenarios.

Potential future work on Cashflow could focus on the support of multipoint communication. Currently, Cashflow supports bidirectional communication between two nodes. However, multipoint communication could be interesting for a number of applications, which however is not optimal supported by the proposed route discovery protocol. Another interesting issue is the support of gossiping [Friedman07] as data distribution method on top of the proposed channel concept.

As result of this thesis, there exists an implementation of Chashflow within IBKSim. However, the examination of the behavior of Cashflow in a real environment might lead to new findings, which could be useful for the improvement of the system. Therefore, the development of a kind of adapter, which adapts the simulated MAC interface of IBKSim to a real MAC interface, would be interesting. This would additionally allow developing new modules for Cashflow using IBKSim and port it for examination to real environments without modifications.

Annex A

Propagation analysis heat maps

This annex presents all heat maps generated for the analysis of the network activity distribution during route search. For the analysis of the network activity during route discovery, eight simulation scenarios were developed, using different parameters for route discovery. Depending on the used route selection criteria (the fee nodes charge or the link quality between nodes), the used artificial delay, and the processing time variance, the activity during route discovery varies, as it is observable in the heat maps presented in this annex. A detailed discussion of the heat maps as well as a precise description of the simulation scenarios can be found in Chapter 4.2.4.
The aim of this annex is to give the reader the opportunity to comprehend in detail the influence of the route decision criteria, as well as the artificial delay and the processing time variation on the route discovery algorithm, as discussed in Chapter 3.3.3. For the sake of completeness, all heat maps are presented in this annex, even if they have already been presented before. Heat maps presenting the network activity of the same moment but for different scenarios are grouped together to highlight the differences between the network activities of the different scenarios. Each heat map pictures the network activity of 5 milliseconds. Since during simulation for every 5 milliseconds a new heat map was generated, the 20 heat maps per simulation scenarios picture the first 95 milliseconds of route discovery.

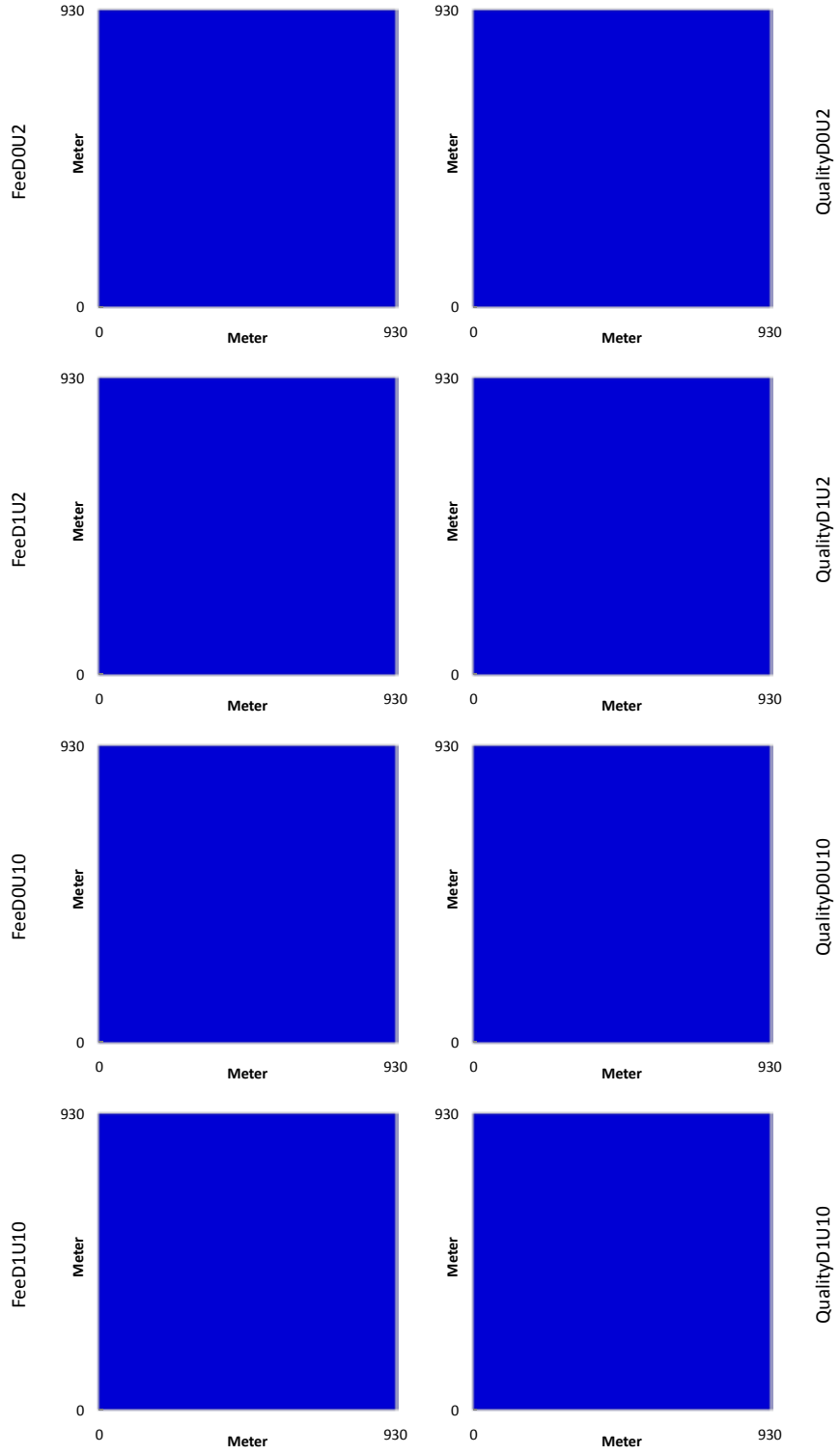


Figure A.1: Network aktivity, time:0

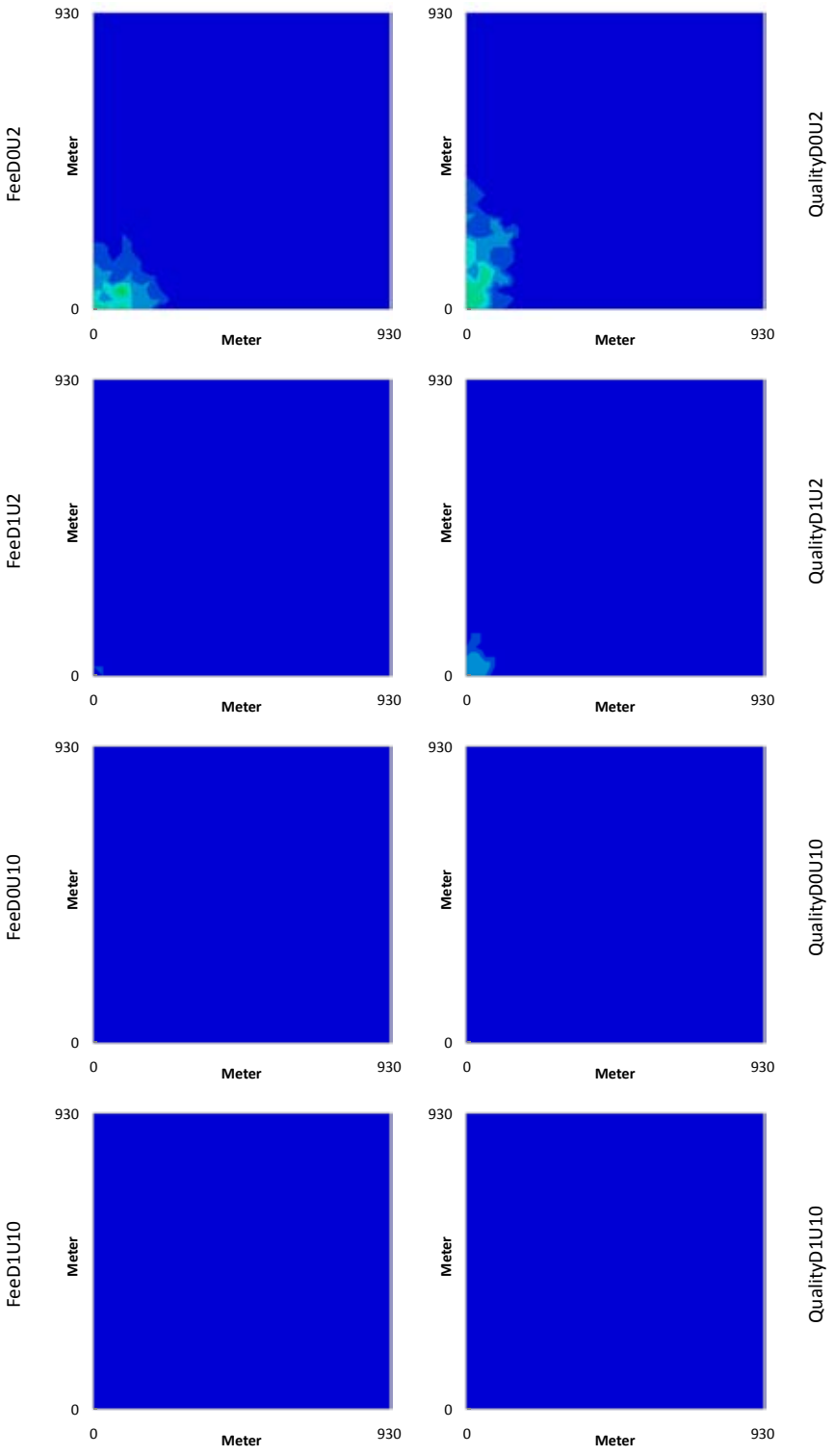


Figure A.2: Network aktivity, time:0.005

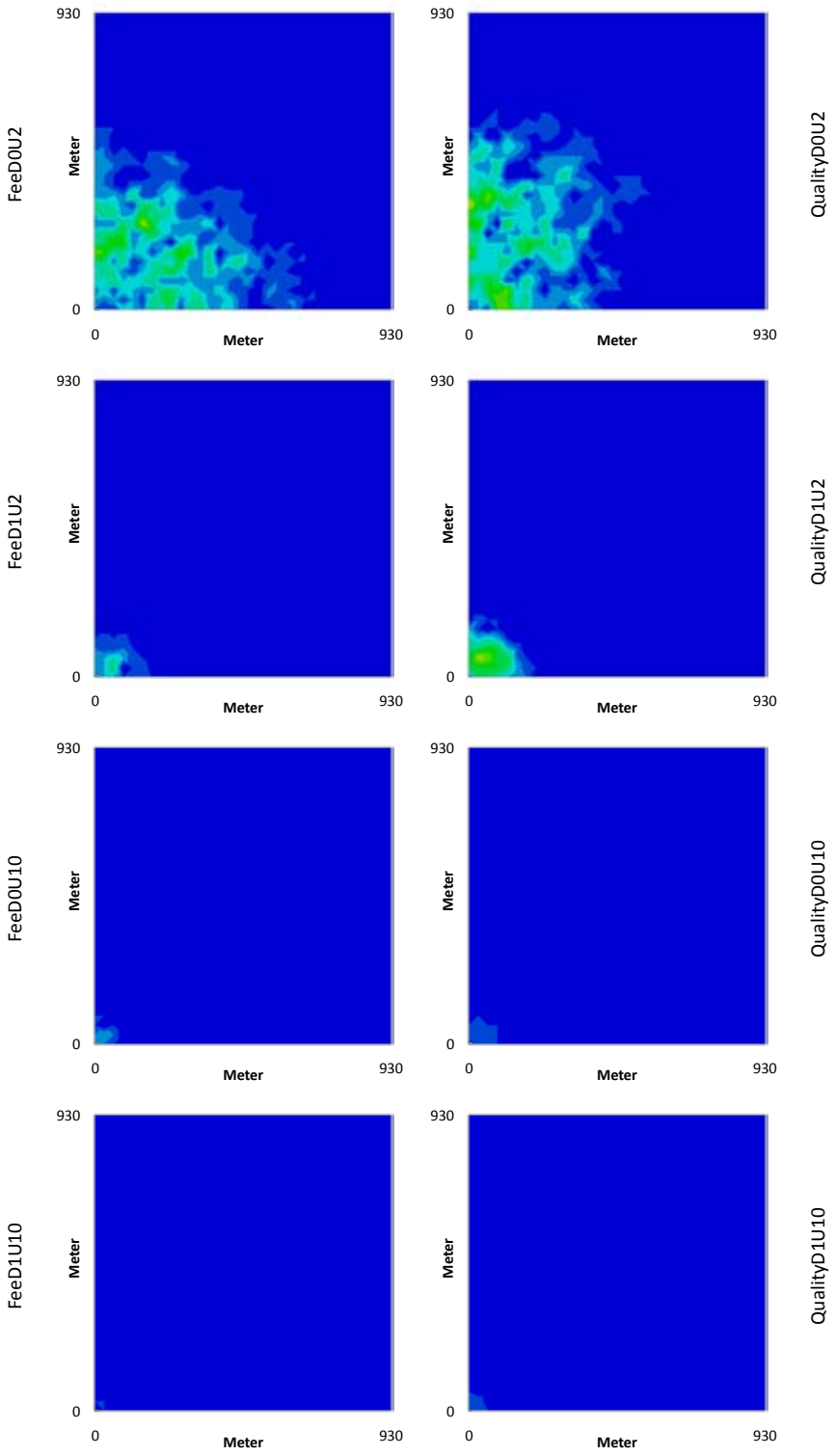


Figure A.3: Network aktivity, time:0.010

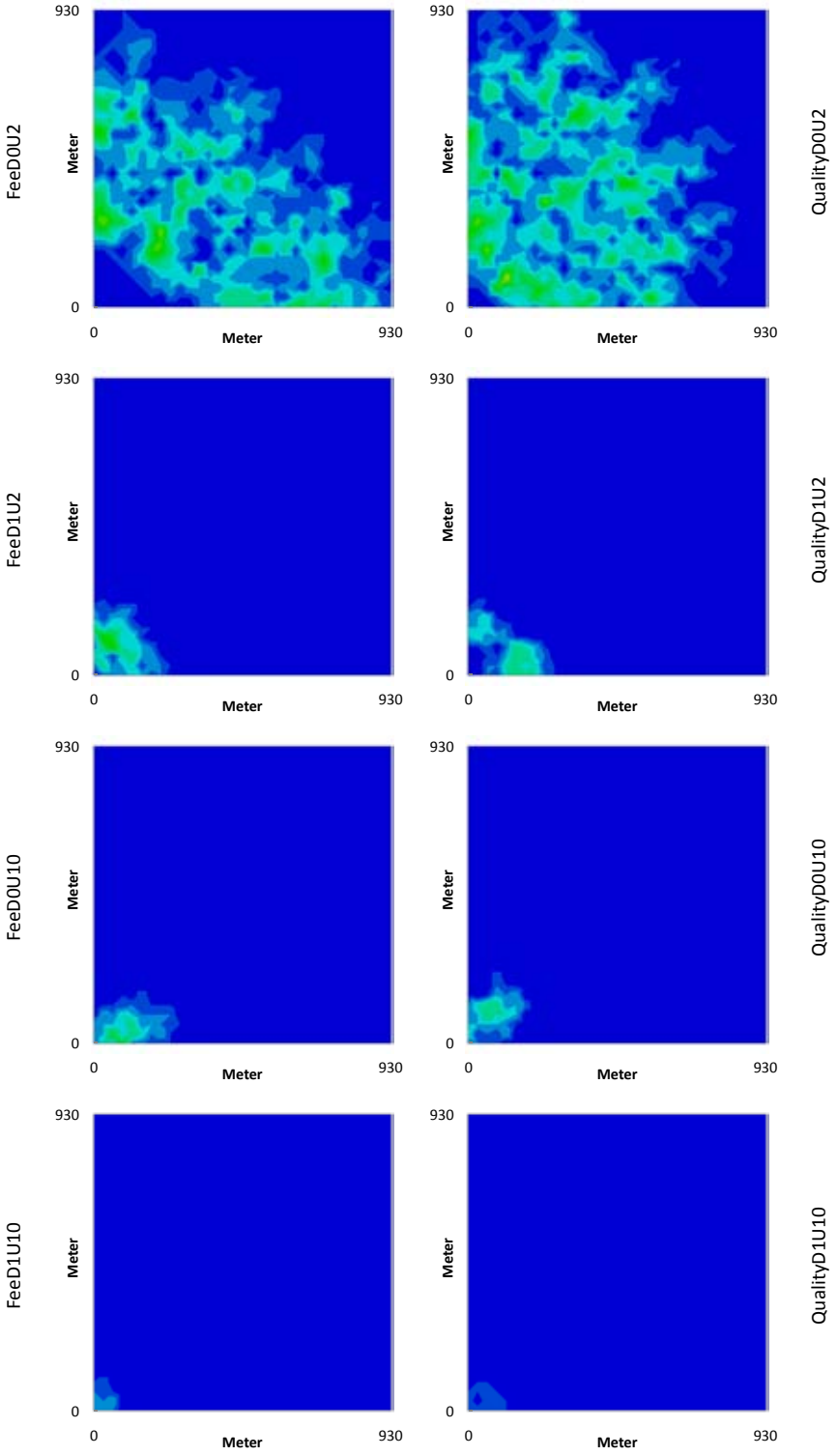


Figure A.4: Network aktivity, time:0.015

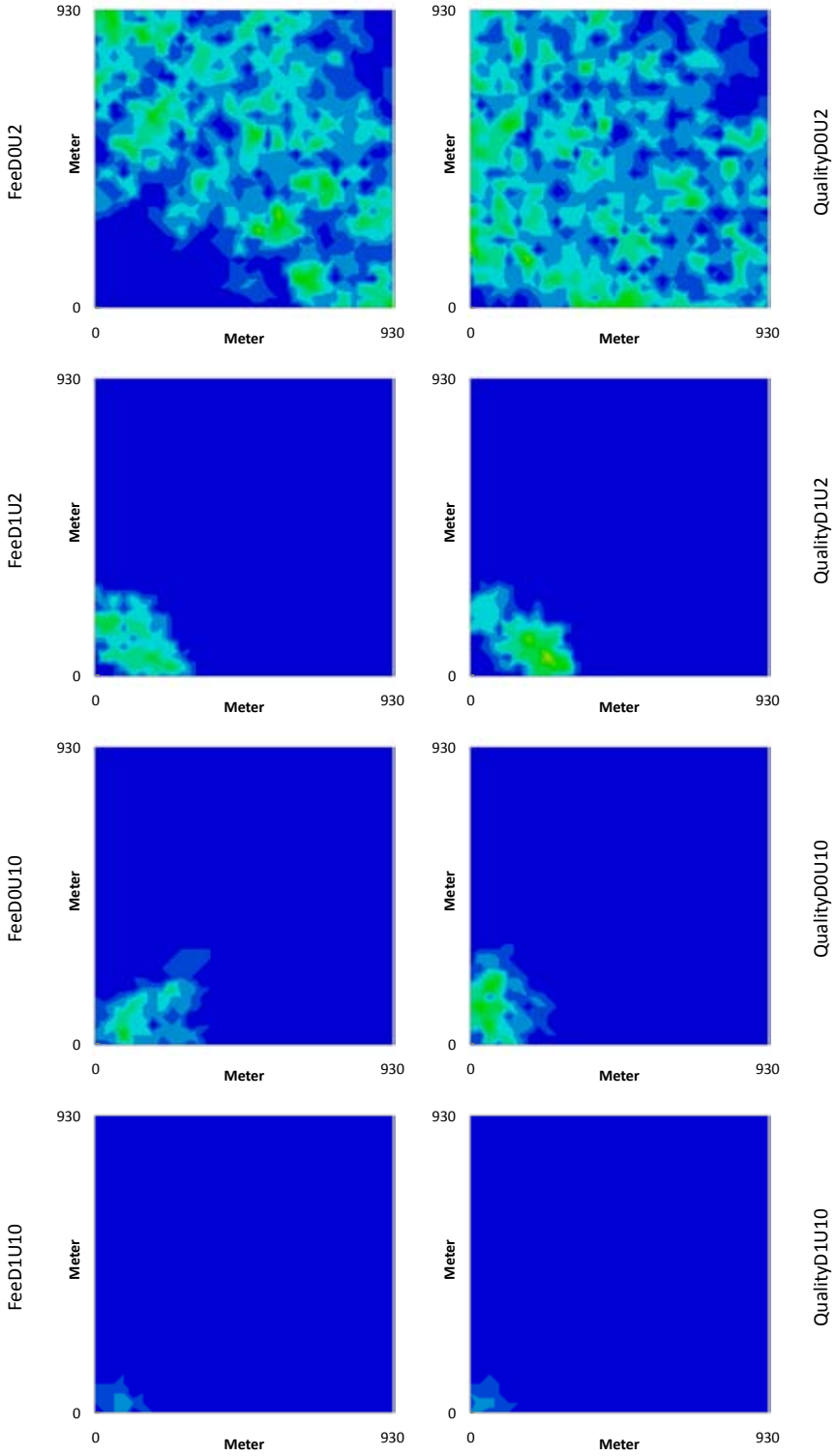


Figure A.5: Network aktivity, time:0.020

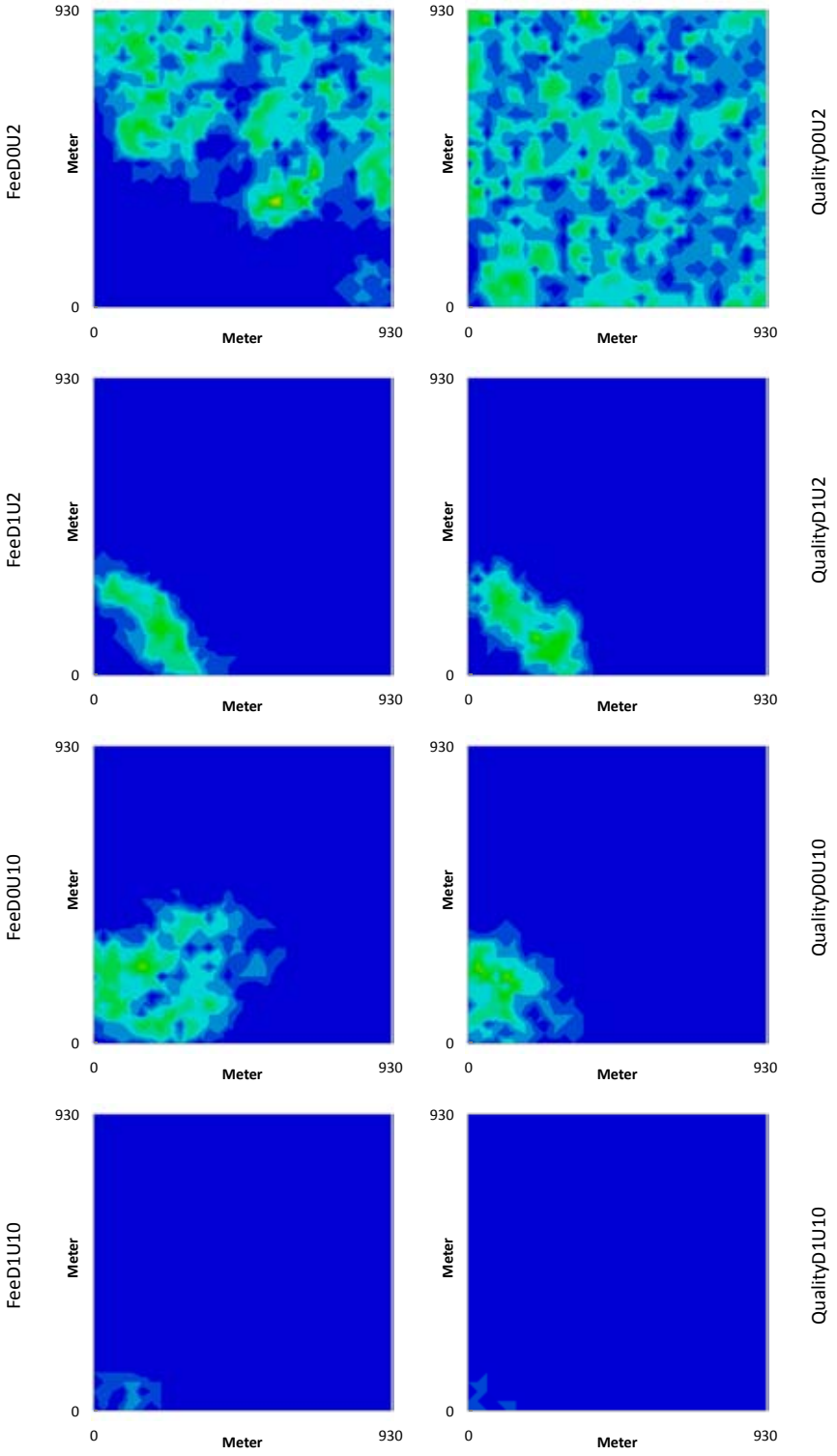


Figure A.6: Network aktivity, time:0.025

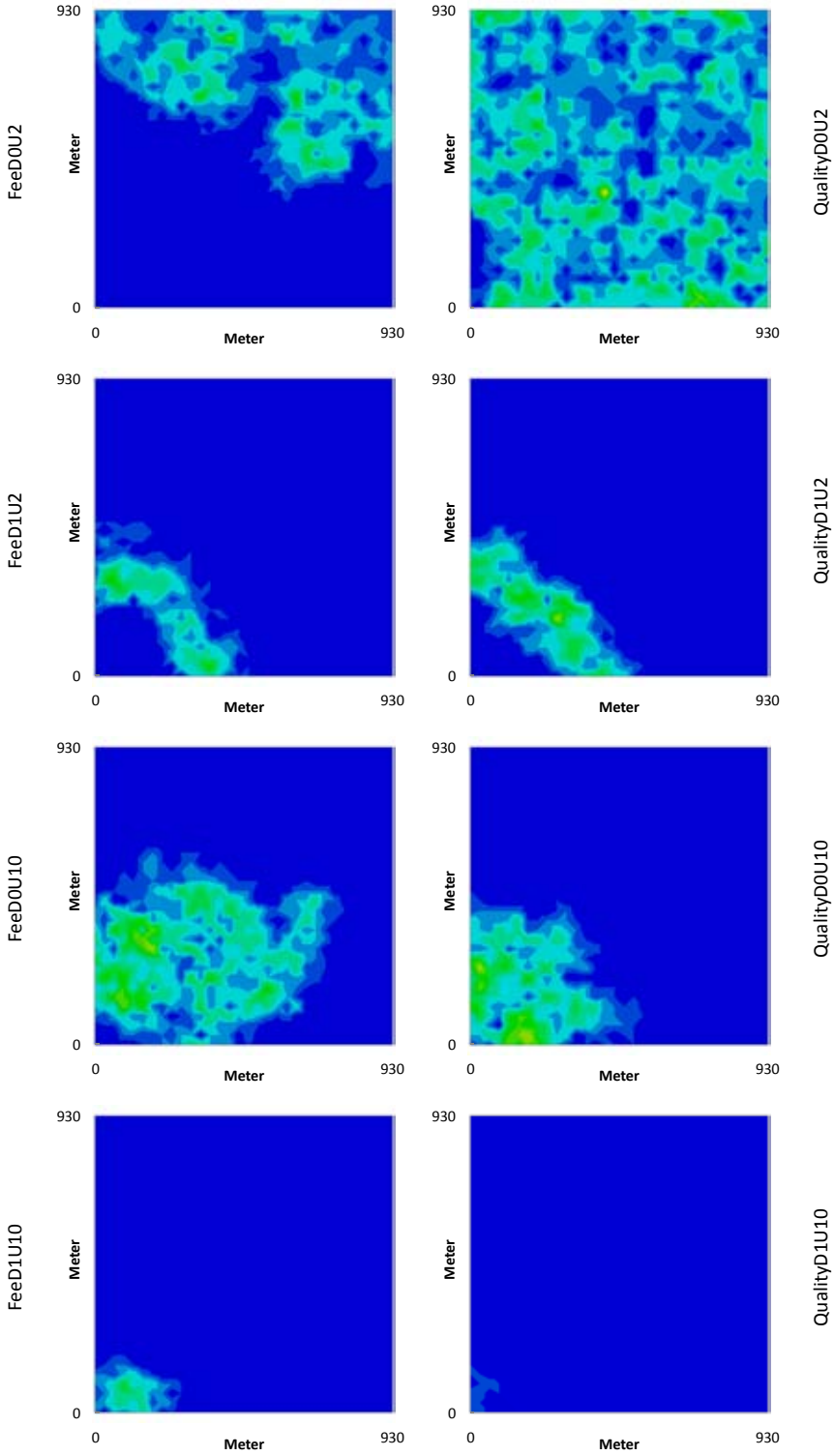


Figure A.7: Network aktivity, time:0.030

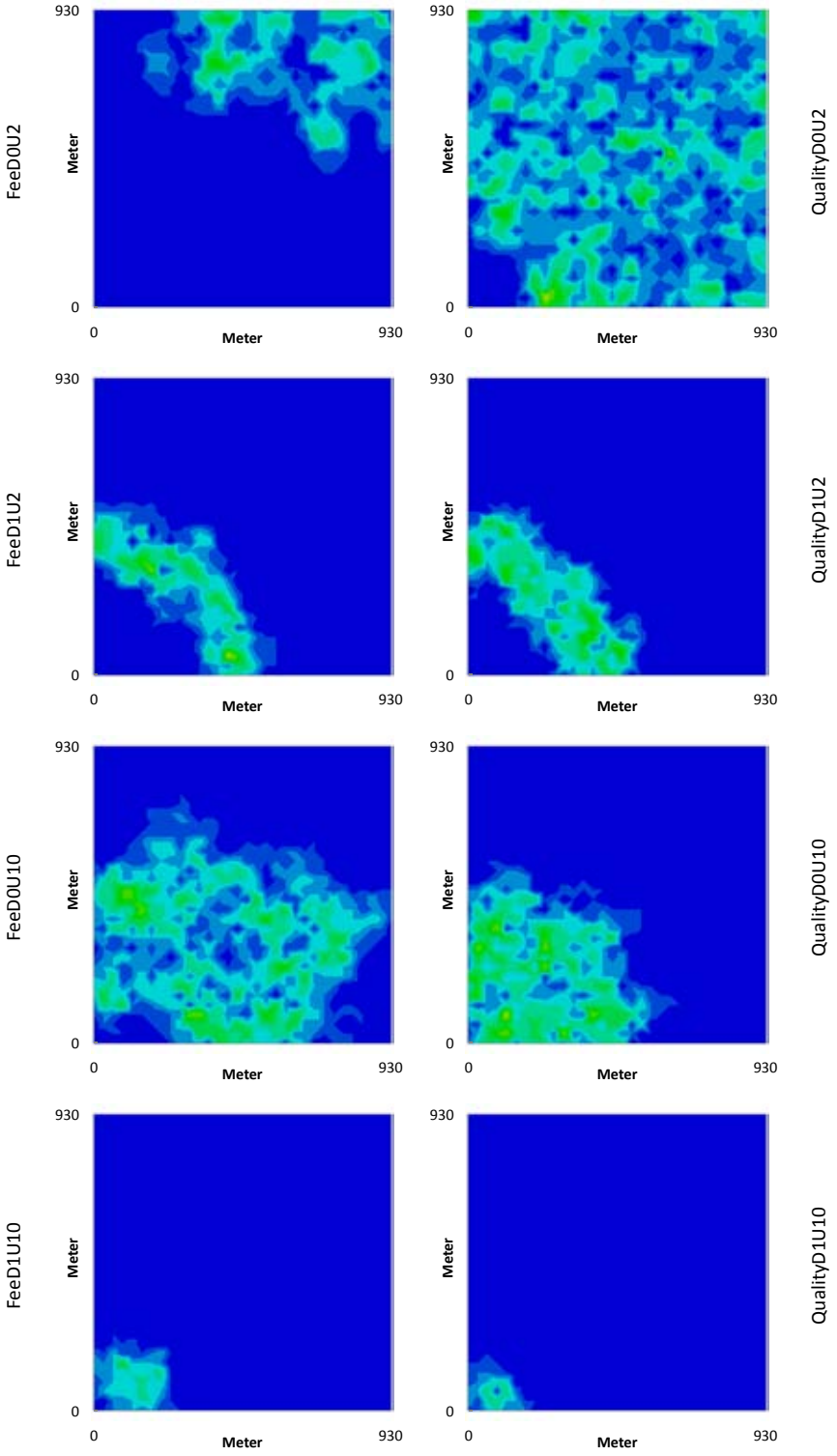


Figure A.8: Network aktivity, time:0.035

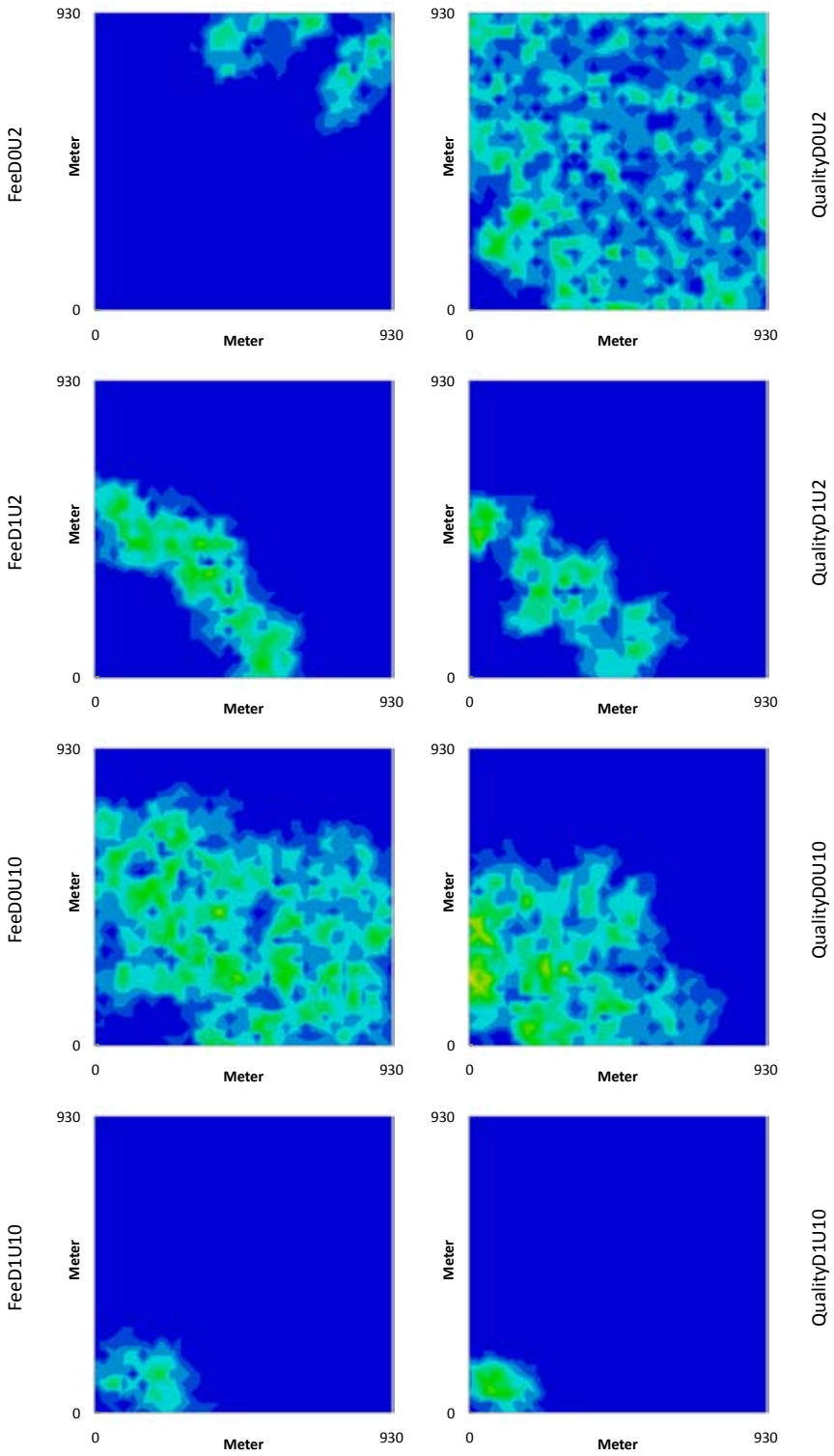


Figure A.9: Network aktivity, time:0.040

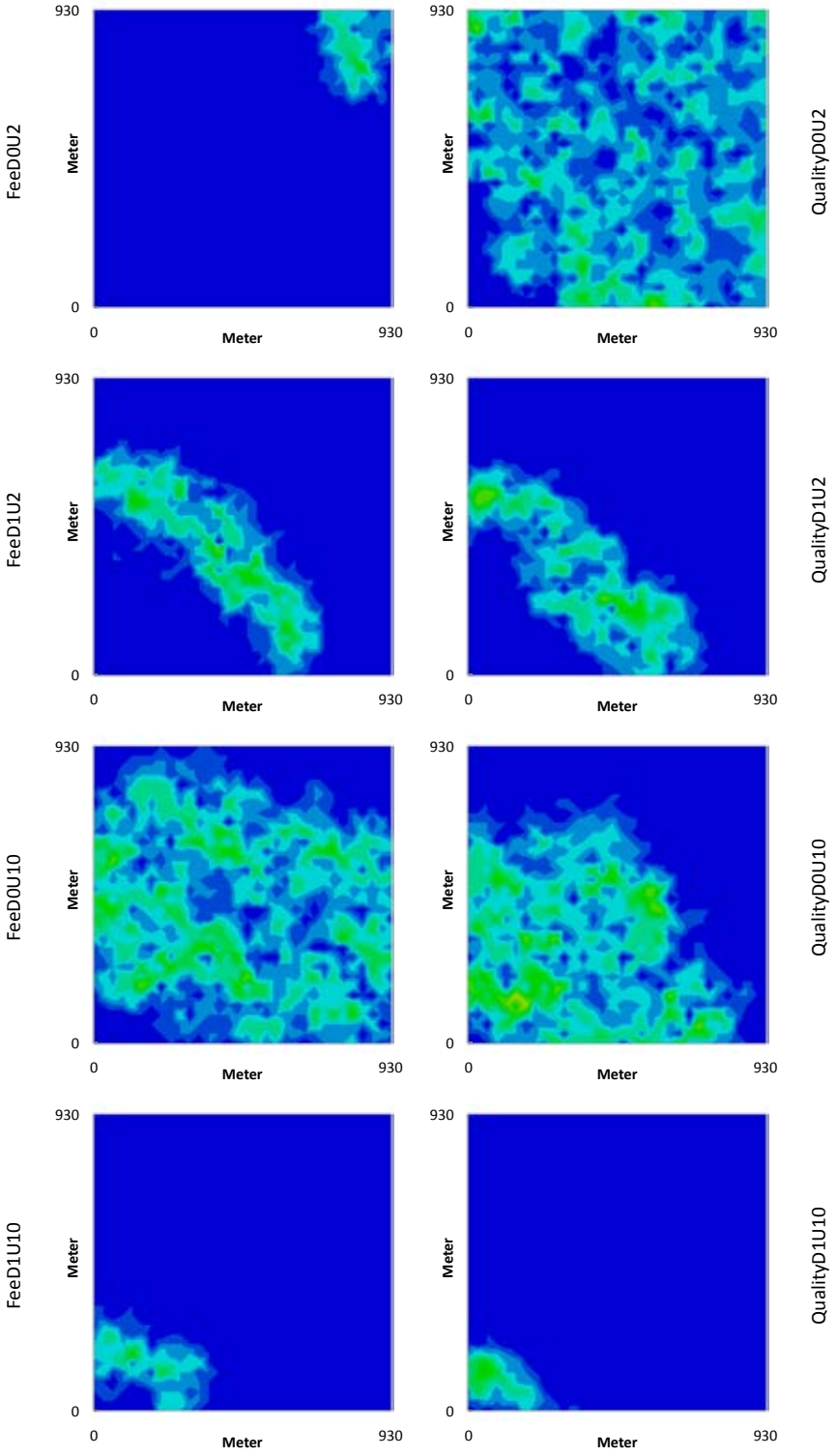


Figure A.10: Network aktivity, time:0.045

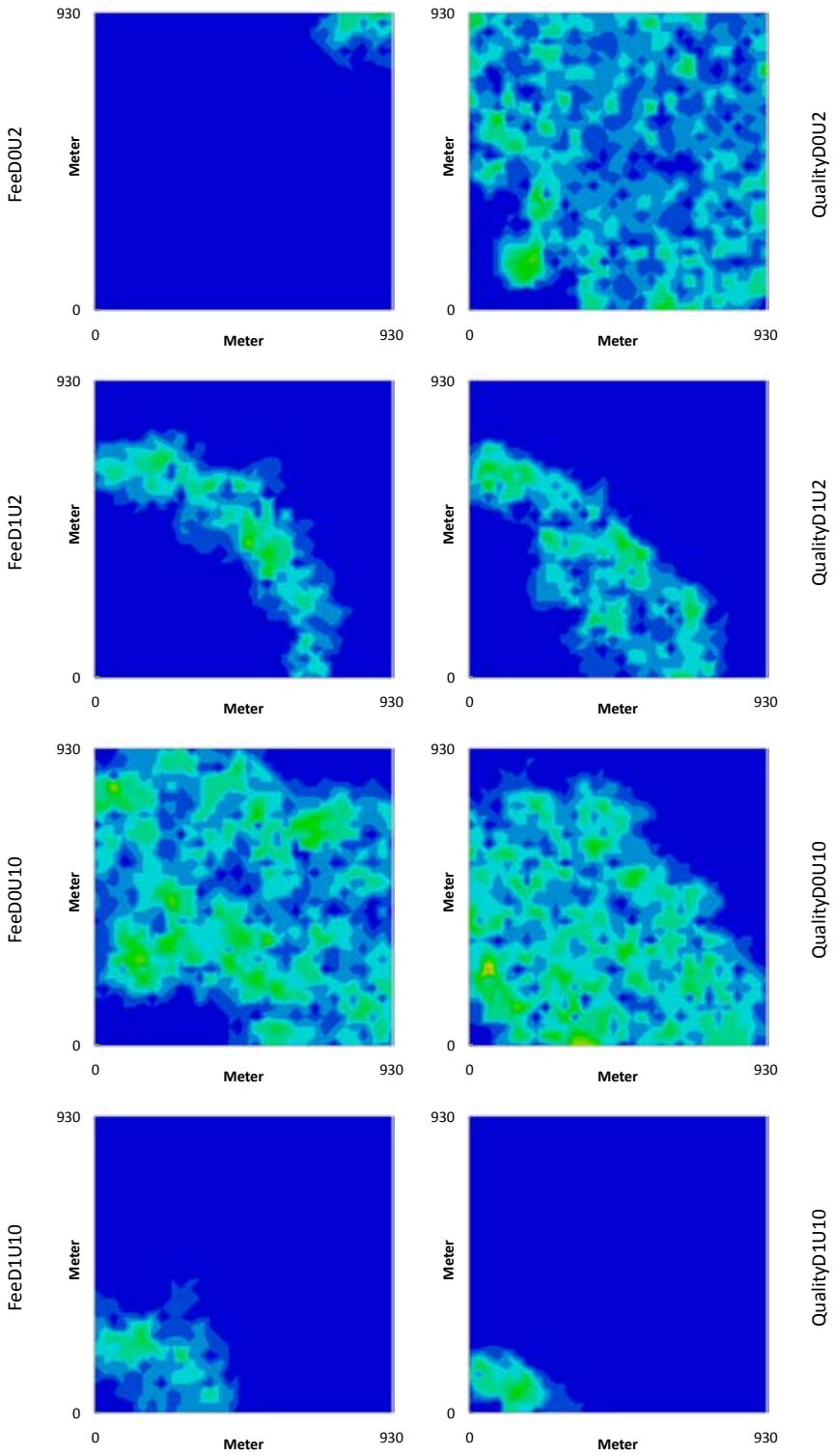


Figure A.11: Network aktivity, time:0.050

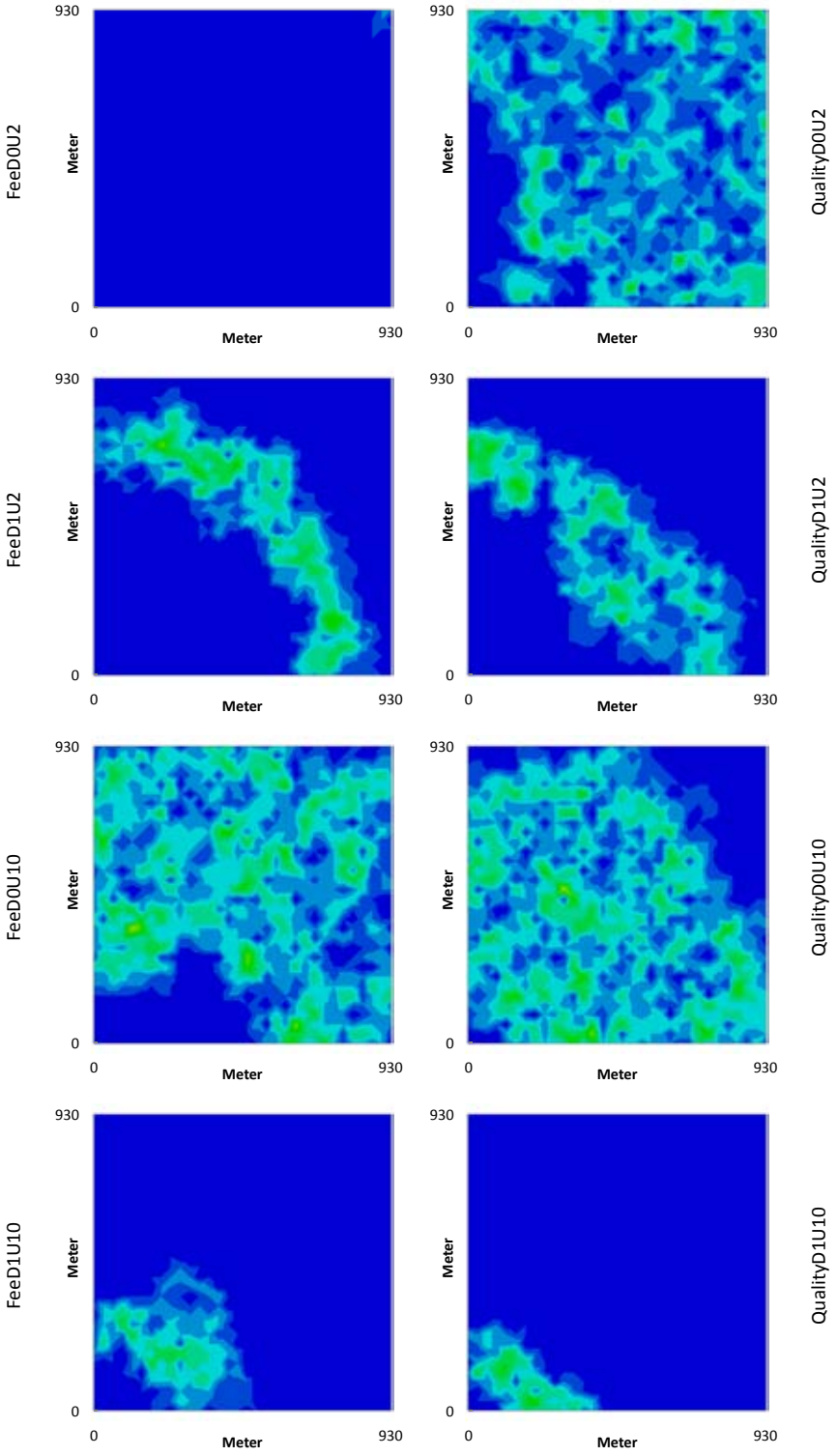


Figure A.12: Network aktivity, time:0.055

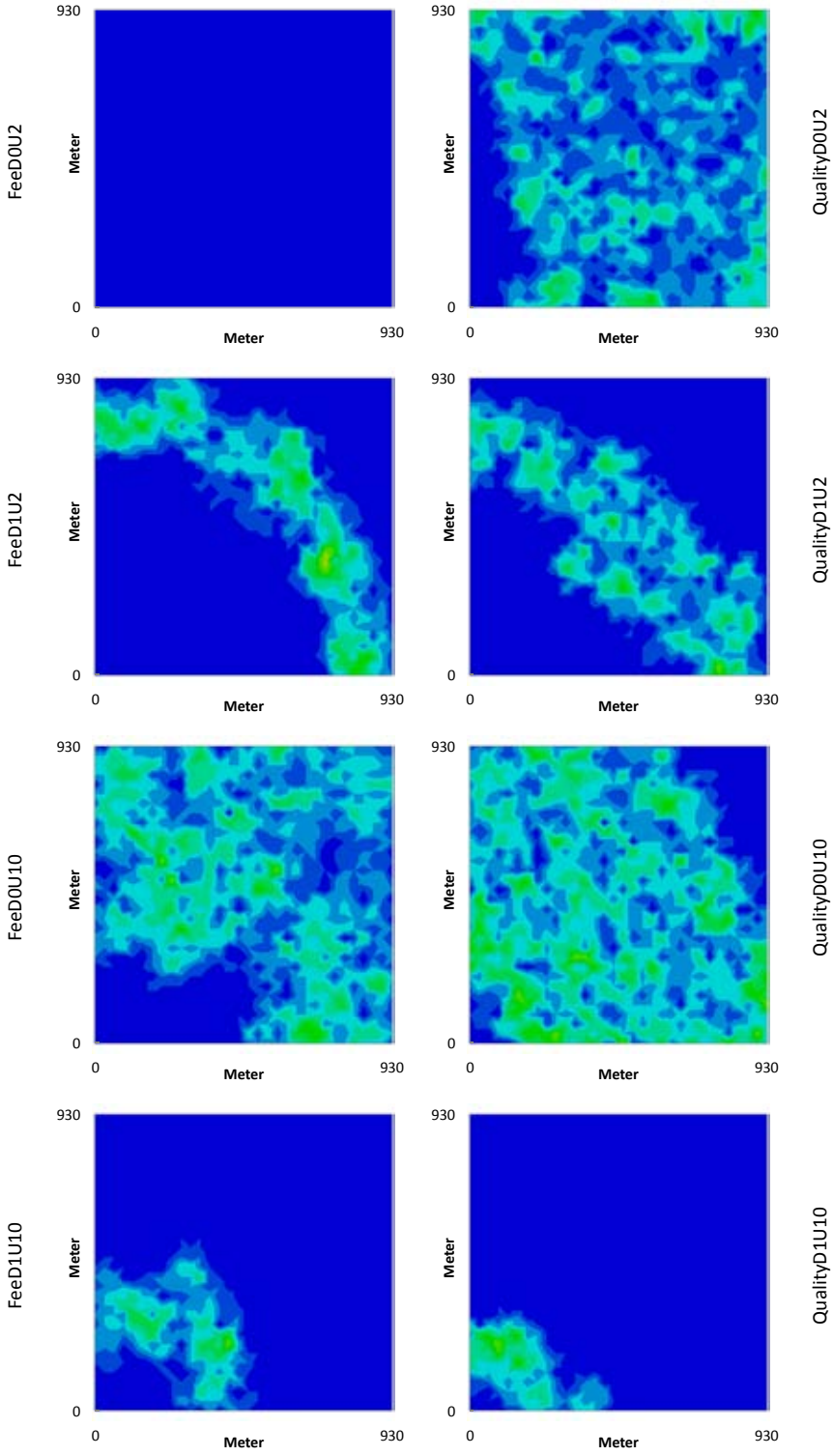


Figure A.13: Network aktivity, time:0.060

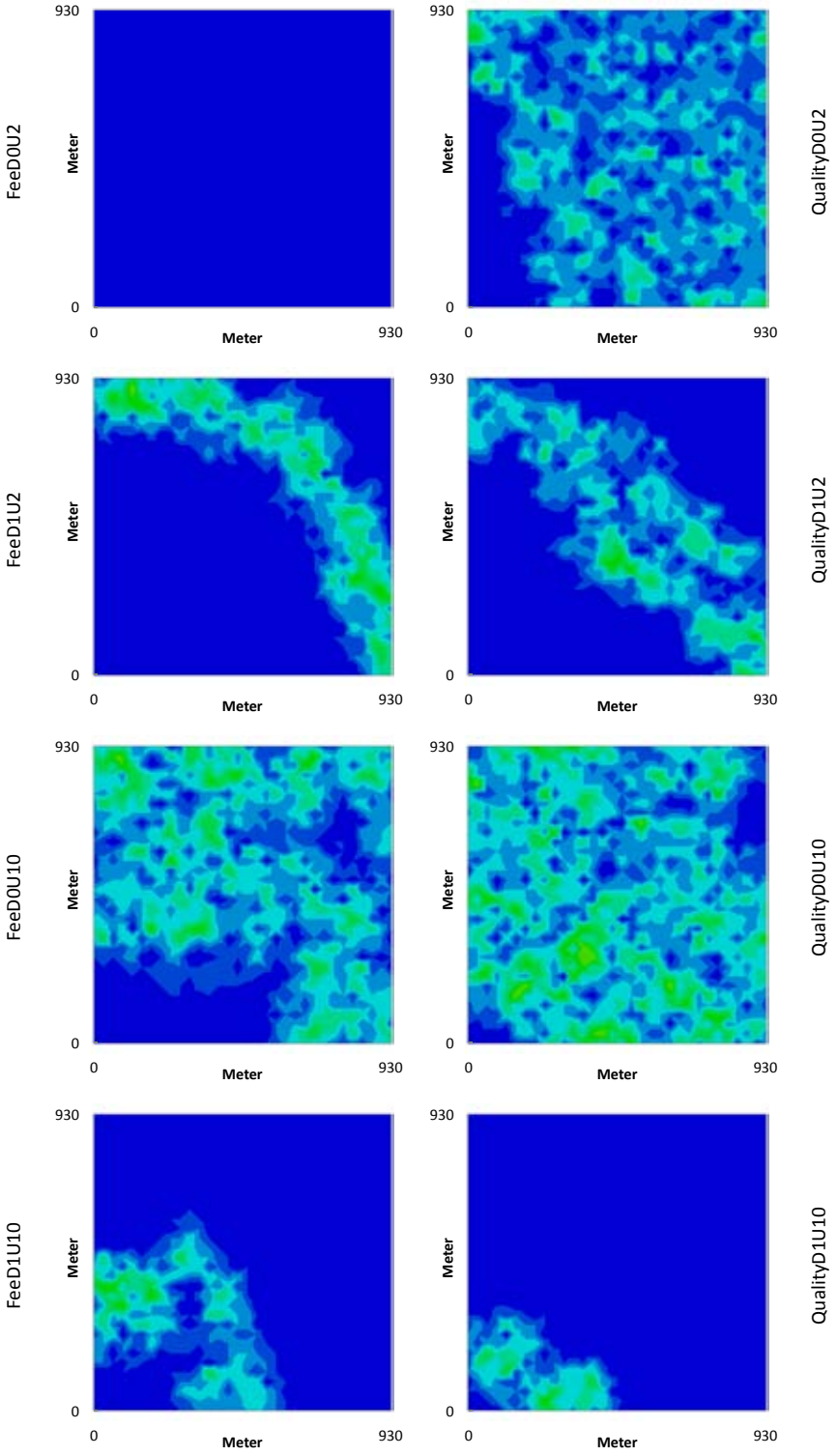


Figure A.14: Network aktivity, time:0.065

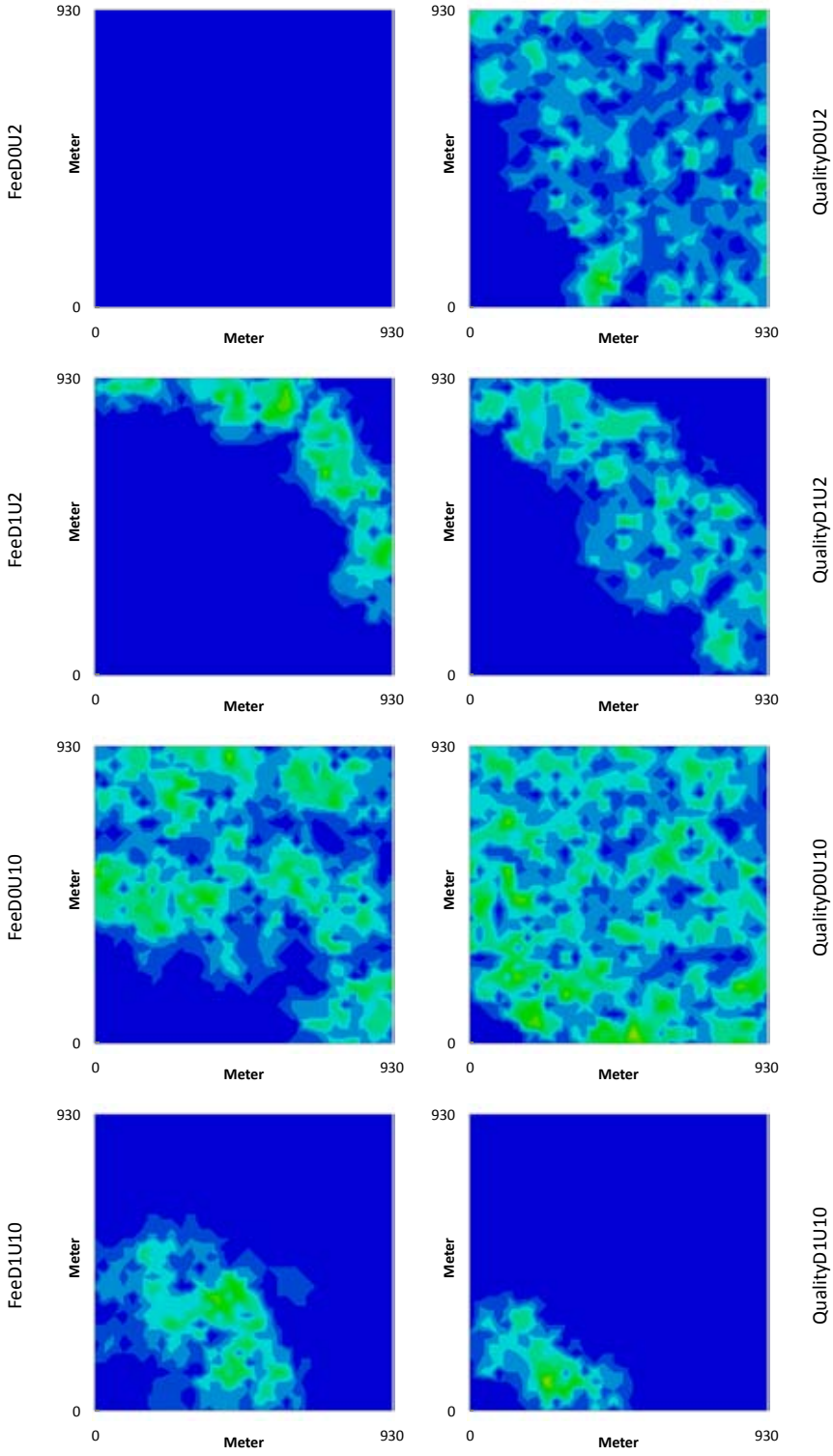


Figure A.15: Network aktivity, time:0.070

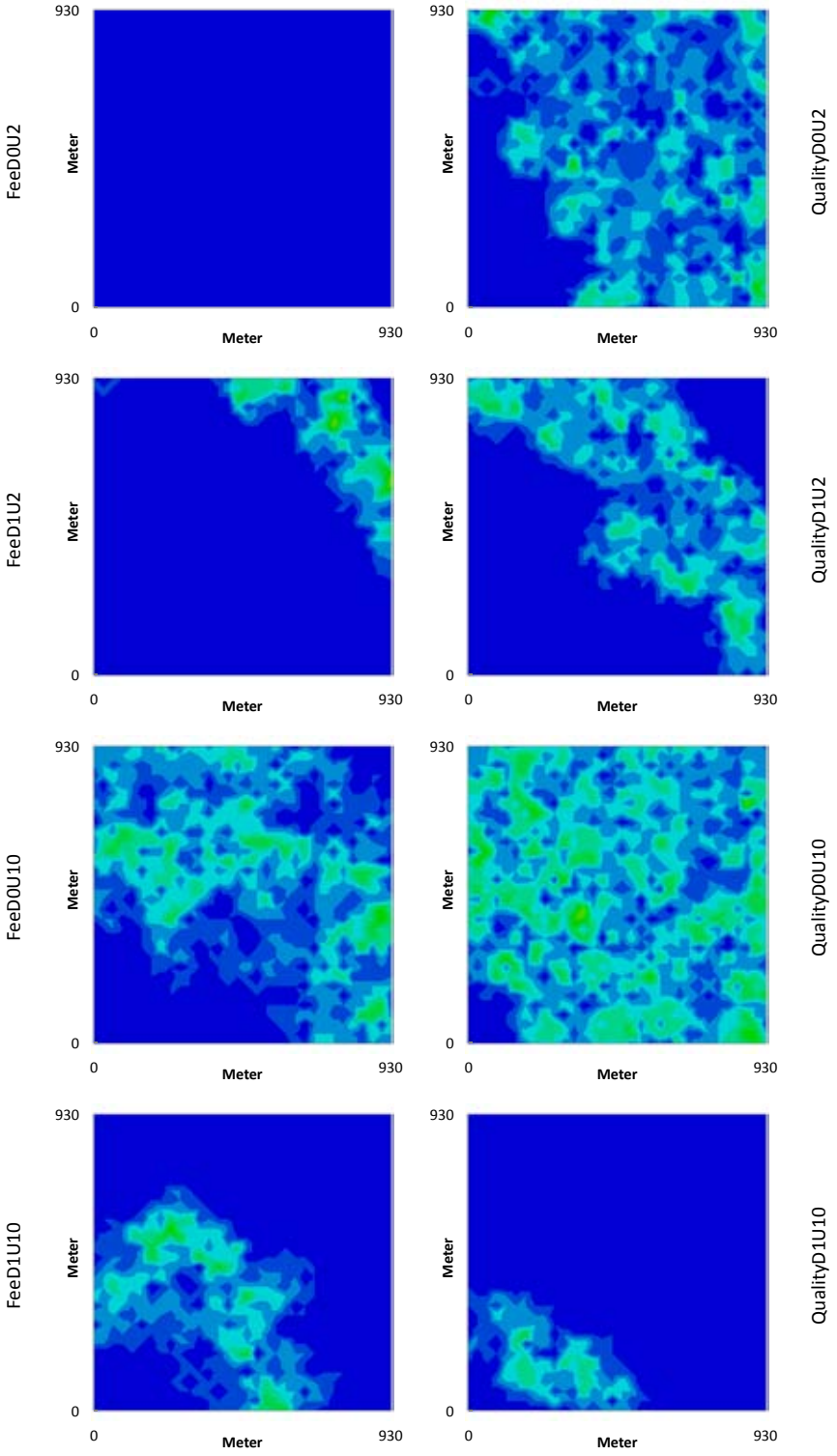


Figure A.16: Network aktivity, time:0.075

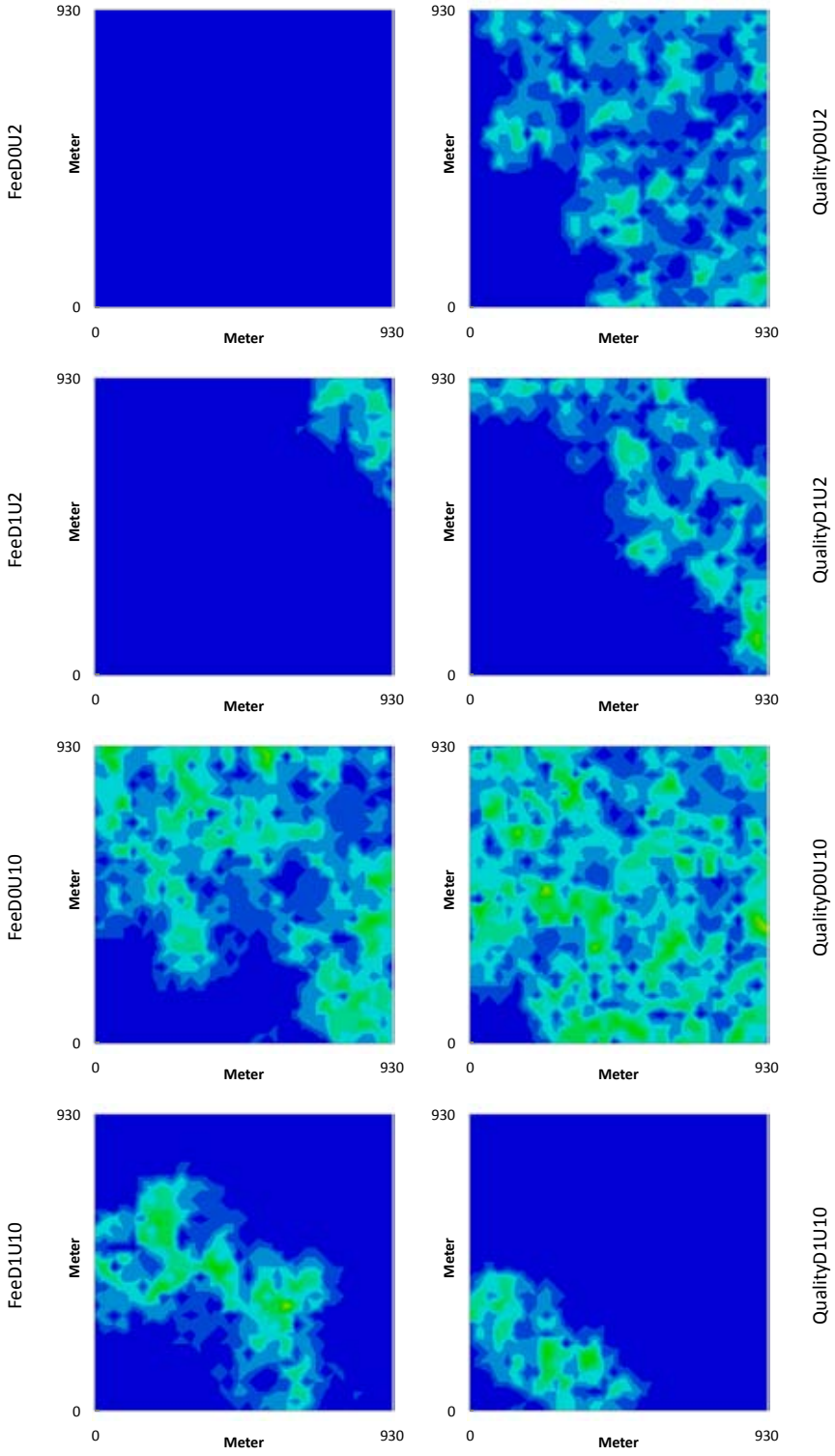


Figure A.17: Network aktivity, time:0.080

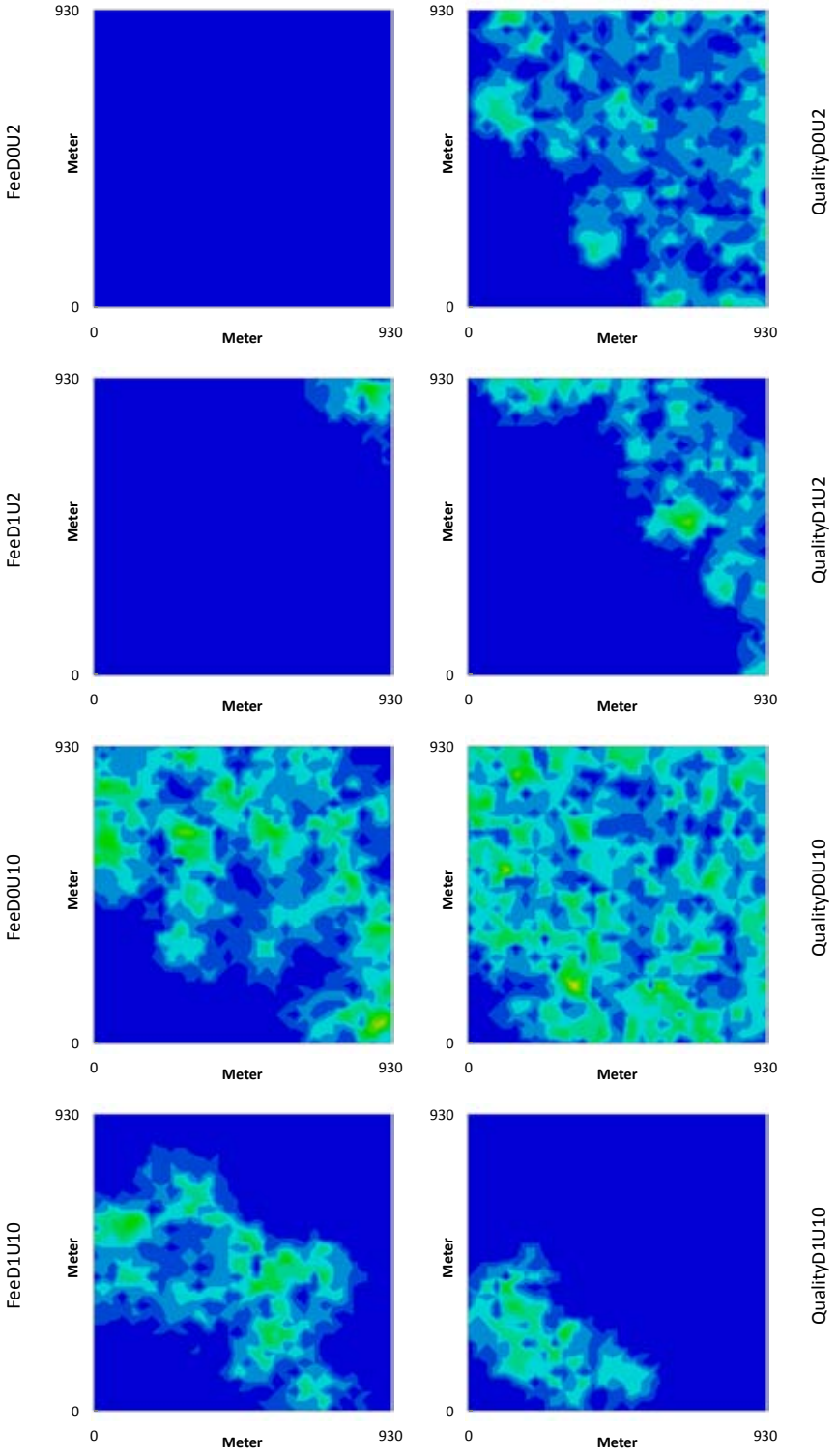


Figure A.18: Network aktivity, time:0.085

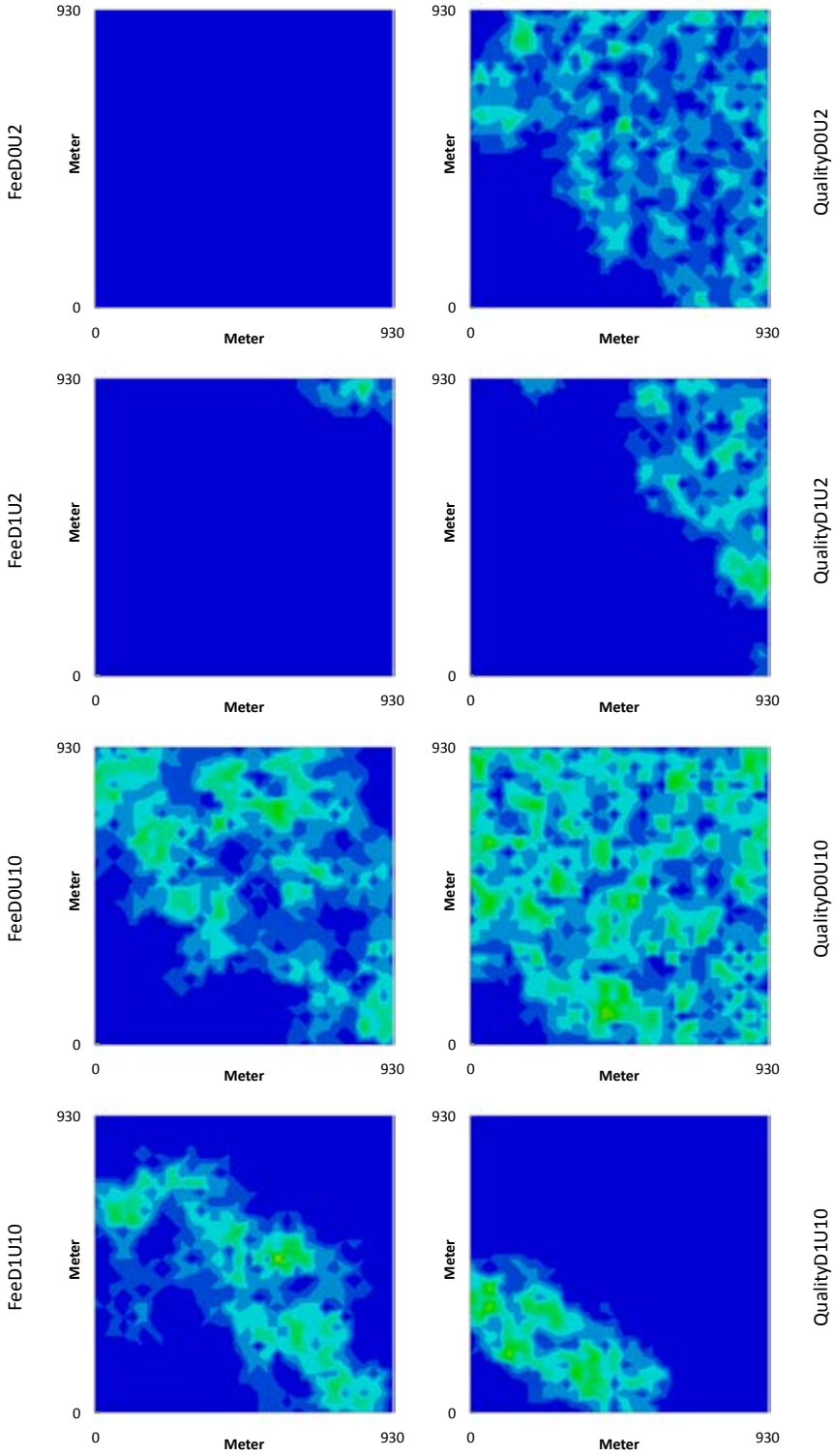


Figure A.19: Network aktivity, time:0.090

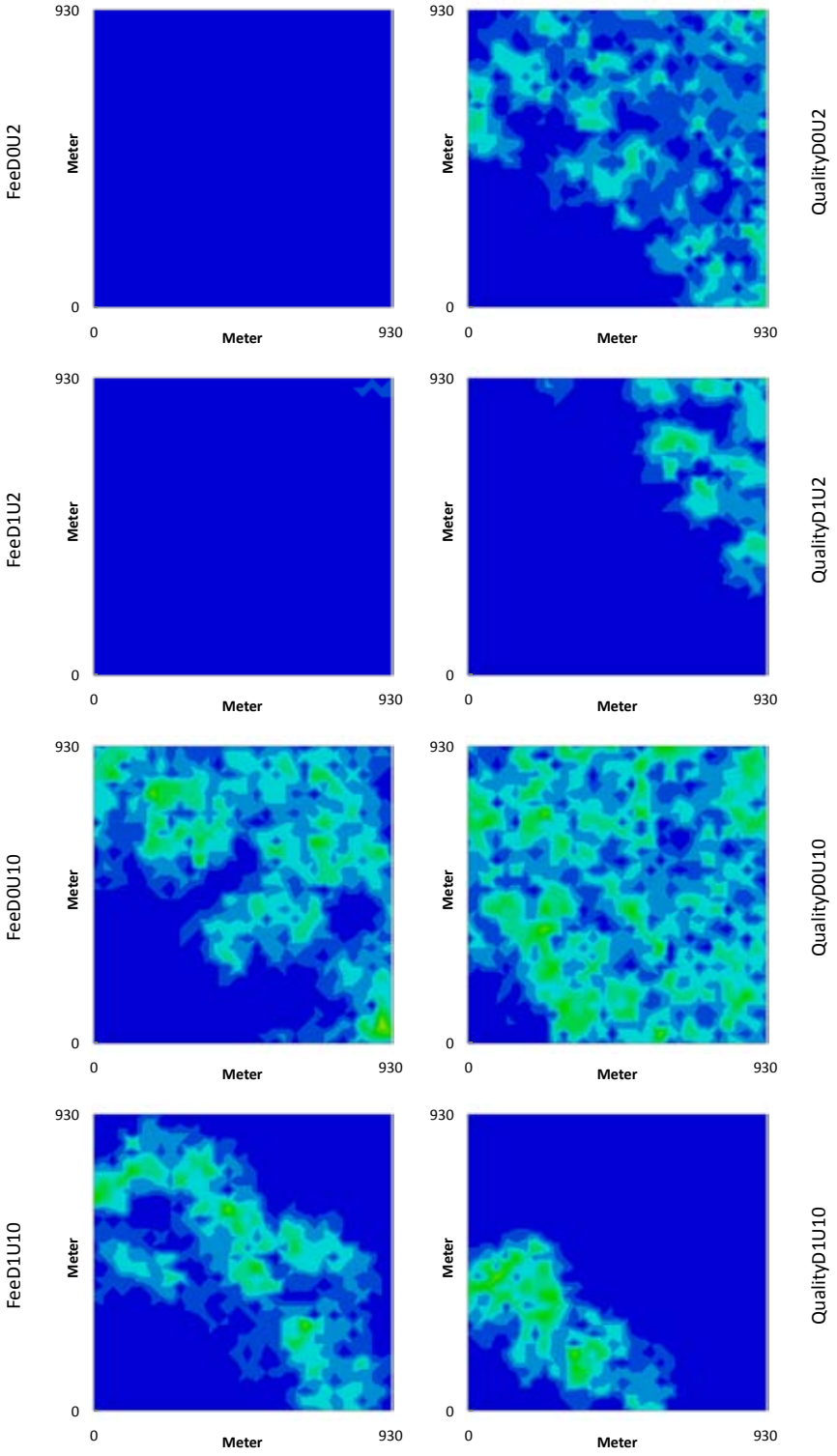


Figure A.20: Network aktivity, time:0.095

Annex B

List of Abbreviations

ACK	Acknowledgment
ACM	Association for Computing Machinery
AODV	Ad-Hoc On-Demand Distance Vector Routing Protocol
CBRP	Cluster Based Routing Protocol
CONFIDENT	Cooperation of Nodes: Fairness in Dynamic Ad-Hoc Networks
CORE	Collaborative Reputation Mechanism
CRC	Cyclic Redundancy Check
CTL	Control
CTS	Clear to Send
dB	decibel
dBm	decibel referenced to one milliwatt
DCF	Distributed Coordination Function
DIFS	DCF Inter Frame Space
DSAP	Destination Service Access Point
DSDV	Destination-Sequenced Distance Vector Routing
DSL	Digital Subscriber Line
DSR	Dynamic Source Routing
FA	Foreign Agent
FSR	Fisheye State Routing
GPRS	General Packet Radio Service
GSR	Global State Routing
GW	Gateway
HSDPA	High-Speed Downlink Packet Access
IBKSim	Institute of Broadband Communications Simulator

ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
LAR	Location Aided Routing
LLC	Logical Link Control
LS	Link State Routing
MAC	Media Access Control
MHz	megahertz
MN	Mobile Node
MPDU	MAC Protocol Data Unit
NAV	Network Allocation Vector
NS2	Network Simulator 2
OCEAN	Observation-based cooperation enforcement in ad hoc networks
OLSR	Optimized Link State Routing
OSI model	Open Systems Interconnection model
PDU	Protocol Data Unit
RFC	Request for Comments
RH	Request Hash
RTS	Request to Send
SHA	Secure Hash Algorithm
SHARP	SHARP Hybrid Ad Hoc Routing Protocol
SIFS	Short inter frame space
SNAP	Subnetwork access protocol
SNIR	Signal to Noise Plus Interference Ratio
SSA	Signal Stability-Based Adaptive Routing Protocol
SSAP	Source service access point
TBRPF	Topology Broadcast based on Reverse-Path Forwarding
VCG	VickreyClarkeGroves auction
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
ZRP	Zone Routing Protocol

List of Figures

List of Tables

Bibliography

[3Com-Corporation06] 3Com-Corporation. *3Com OfficeConnect Wireless 54 Mbps 11g Compact USB Adapter - data sheet*, 2006.

[Anderegg03] L. Anderegg and S. Eidenbenz. *Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents.* In Proceedings of the 9th annual international conference on Mobile computing and networking, pages 245–259. ACM, 2003.

[Bansal03] S. Bansal and M. Baker. *Observation-based cooperation enforcement in ad hoc networks. Arxiv preprint cs/0307012*, , 2003.

[Bellur99] B. Bellur and R.G. Ogier. *A reliable, efficient topology broadcast protocol for dynamic networks.* In Proceedings of the IEEE International Conference on Computer Communications 1999 (INFOCOM 99), volume 1, pages 178–186, 1999.

[Bray00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible markup language (XML) 1.0. W3C recommendation*, vol. 6, , 2000.

[Buchegger02a] S. Buchegger and J.Y. L. Boudec. *Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks.* In Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing, pages 403–410. Citeseer, 2002.

[Buchegger02b] S. Buchegger and J.Y. L. Boudec. *Performance analysis of the CONFIDANT protocol.* In Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, pages 226–236. ACM, 2002.

[Buttyan00] L. Buttyan and J.P. Hubaux. *Enforcing service availability in mobile ad-hoc WANs.* In Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing, page 96, 2000.

[Buttyan01] L. Buttyan and J.P. Hubaux. *Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks*, 2001.

[Cavalcanti05] D. Cavalcanti, D. Agrawal, B. Xie, and A. Kumar. *Issues in Integrating Cellular Networks, WLANs, and MANETs: A Futuristic Heterogeneous Wireless Network. IEEE Wireless Communications*, pp. 31, 2005.

[Chen98] T.W. Chen and M. Gerla. *Global state routing: A new routing scheme for ad-hoc wireless networks.* In Proceedings of the IEEE International Conference on Communications 1998, volume 1, pages 171–175. Citeseer, 1998.

[Chen04a] K. Chen and K. Nahrstedt. *iPass: an incentive compatible auction scheme to enable packet forwarding service in MANET.* In 24th International Conference on Distributed Computing Systems, 2004. Proceedings, pages 534–542, 2004.

[Chen04b] K. Chen and K. Nahrstedt. *iPass: an incentive compatible auction scheme to enable packet forwarding service in MANET. Proceedings of 24th International Conference onDistributed Computing Systems, 2004.*, pp. 534–542, 2004.

[Cheng89] C. Cheng, R. Riley, S.P.R Kumar, and J.J Garcia-Luna-Aceves. *A loop-free extended Bellman-Ford routing protocol without bouncing effect. ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4, pp. 224–236, 1989.

[Chiang97] C.C. Chiang, H.K. Wu, W. Liu, and M. Gerla. *Routing in clustered multihop, mobile wireless networks with fading channel. Proceedings of the Singapore International Conference on Networks 1997*, vol. 97, pp. 197–211, 1997.

[Davis04] M. Davis. *A Wireless Traffic Probe for Radio Resource Management and QoS Provisioning in IEEE 802.11 WLANs.* In ACM Symposium on Modeling, Analysis and Simlulation of Wireless and Mobile Systems (ACM MSWiM 2004), 2004.

[Davis05] M. Davis and T. Raimondi. *A Novel Framework for Radio Resource Management in IEEE 802.11 Wireless LANs.* In Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2005), 2005.

[Dube97] R. Dube, C.D. Rais, K.Y. Wang, and S.K. Tripathi. *Signal stability-based adaptive routing (SSA) for mobile ad hoc networks. IEEE Personal Communications*, vol. 4, no. 1, pp. 36–45, 1997.

[Eidenbenz05] S. Eidenbenz, G. Resta, and P. Santi. *Commit: A sender-centric truthful and energy-efficient routing protocol for ad hoc networks with selfish nodes.* In Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, page 10, 2005.

[Ephremides02] A. Ephremides. *Ad hoc networks: not an ad hoc field anymore. Wireless communications and mobile computing*, vol. 2, no. 5, pp. 441–448, 2002.

[FIP02] *FIPS 180-2: Secure Hash Standard (SHS).* In Federal Information Processing Standard (FIPS), Publication 180-2. National Institute of Standards and Technology, US Department of Commerce, 2002.

[Friedman07] R. Friedman, D. Gavidia, L. Rodrigues, A.C. Viana, and S. Voulgaris. *Gossiping on MANETs: the Beauty and the Beast. ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 74, 2007.

[Gamma95] Erich Gamma and Richard Helm. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Haas99] Z.J. Haas, M.R. Pearlman, and P. Samar. *The zone routing protocol (ZRP) for ad hoc networks. IETF MANET Working Group Internet Draft*, , 1999.

[Hu05] Y.C. Hu, A. Perrig, and D.B. Johnson. *Ariadne: A secure on-demand routing protocol for ad hoc networks. Wireless Networks*, vol. 11, no. 1-2, pp. 38, 2005.

[IEEE99] IEEE. *IEEE Std 802.11-1999, IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless (LAN) Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999. status: Adopted by the ISO/IEC and redesignated as ISO/IEC 8802-11:1999(E).

[IEEE00] IEEE. *Supplement to IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless (LAN) Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, 2000.

[IEEE03] IEEE. *Supplement to IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless (LAN) Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Further Higher Data Rate Extension in the 2.4 GHz Band*, 2003.

[IEEE04] IEEE. *IEEE Standard for Information technology - Telecommunications and Information Exchange*

Between Systems Local and Metropolitan Area Networks Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Medium Access Control (MAC) Security Enhancements, 2004.

[IEEE05] IEEE. *IEEE Standard for Information technology - Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment: Medium Access Method (MAC) Quality of Service Enhancements*, 2005.

[IEEE07] IEEE. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999), IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007.

[IEEE08] IEEE. *Draft Amendment: ESS Mesh Networking, IEEE P802.11s Draft*, 2008.

[IEEE09] IEEE. *IEEE Standard for Information technology - Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, 2009.

[Institute10] Information Sciences Institute. *The Network Simulator - ns-2.* `http://nsnam.isi.edu/nsnam/index.php/Main_Page`.

[Jacquet01] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. *Optimized link state routing protocol for ad hoc networks.* In Proceed-

ings of the IEEE International Multitopic Conference 2001, pages 62–68, 2001.

[Jiang98] M. Jiang, J. Li, and Y.C. Tay. *Cluster based routing protocol (CBRP) functional specification. IETF MANET Working Group Internet Draft*, , 1998.

[Johnson96] D.B. Johnson and D.A. Maltz. *Dynamic source routing in ad hoc wireless networks. Mobile computing*, pp. 153–181, 1996.

[Johnson04] D. Johnson, C. Perkins, and J. Arkko. *RFC3775: Mobility support in IPv6*, 2004.

[Johnson07] D. Johnson, Y. Hu, and D. Maltz. *RFC4728: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*, 2007.

[Johnson08] D. Johnson, N. Ntlatlapa C., and Aichele. *A simple pragmatic approach to mesh routing using BATMAN.* In 2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries, CSIR, Pretoria, South Africa, volume 10, 2008.

[Katz08] J. Katz and Y. Lindell. *Introduction to modern cryptography.* Chapman & Hall/CRC, 2008.

[Ko98] Y.B. Ko and N.H. Vaidya. *Location-Aided Routing (LAR) in Mobile Ad Hoc Networks.* In edings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM98), pages 66–75, 1998.

[Macker98] J.P. Macker and M.S. Corson. *Mobile ad hoc networking and the IETF. ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 1, pp. 14, 1998.

[Macker10] J.P. Macker and I. Chakeres. *IETF Working Group Mobile Ad-hoc Networks.* `http://datatracker.ietf.org/wg/manet/charter/`.

[Michiardi02] P. Michiardi and Molva R. *Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks.* In Proceedings of advanced communications and multimedia security: IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security, September 26-27 2002, page 107. Kluwer Academic Pub, 2002.

[Nahrstedt09] K. Nahrstedt, W. He, and Y. Huang. *Security in Wireless Ad Hoc Networks. Guide to Wireless Ad Hoc Networks*, pp. 391–425, 2009.

[Ni99] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu. *The broadcast storm problem in a mobile ad hoc network.* In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, page 162. ACM, 1999.

[Omn10] *OMNeT++*. `http://www.omnetpp.org/`.

[OSI84] OSI. *Information Technology - Open Systems Interconnection - Basic Reference Model*, 1984. ITU-T Rec. X.200.

[Pei00] G. Pei, M. Gerla, and T.W. Chen. *Fisheye state routing: A routing scheme for ad hoc wireless networks.* In Proceedings of the IEEE International Conference on Communications 2000, volume 1, pages 70–74, 2000.

[Perkins94] C. Perkins and P. Bhagwat. *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers.* In Proceedings of the conference on Communications architectures, protocols and applications, pages 234–244. ACM, 1994.

[Perkins99] C.E. Perkins and E.M. Royer. *Ad-hoc on-demand distance vector routing.* In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100. Published by the IEEE Computer Society, 1999.

[Perkins02] C. Perkins. *RFC3344: IP Mobility Support for IPv4*, 2002.

[Perkins07] C. Perkins, P. Calhoun, and J. Bharatia. *RFC4721: Mobile IPv4 challenge/response extensions (revised)*, 2007.

[Perkins08] C.E. Perkins. *Ad hoc networking*. Addison-Wesley Professional, 2008.

[Ramasubramanian03] V. Ramasubramanian, Z.J. Haas, and E.G. Sirer. *SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks*. In Proceedings of the 4th ACM international symposium on mobile ad hoc networking and computing, page 314. ACM, 2003.

[Rappaport99] T.S. Rappaport. *Wireless Communications: Principles and Practice* . Upper Saddle River, NJ : Prentice Hall PTR, 1999.

[Tewari03a] H. Tewari and D. O'Mahony. *Multiparty micropayments for ad hoc networks*. In Proceedings of IEEE Wireless Communications and Networking - WCNC 2003., pages 2033–2040. IEEE, 2003.

[Tewari03b] H. Tewari and D. O'Mahony. *Real-time payments for mobile IP. IEEE Communications Magazine*, vol. 41, pp. 126–136, February 2003.

[Tobagi75] F. Tobagi and L. Kleinrock. *Packet switching in radio channels: part II–the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 23, no. 12, pp. 1417–1433, 1975.

[Wallentin08] L. Wallentin. *Simulation mobiler Ad-Hoc Netze*. Vienna University of Technology, 2008.

[Wallentin10a] L. Wallentin, J. Fabini, C. Egger, and M. Happenhofer. *A Cross-Layer Route Discovery Strategy for Virtual Currency Systems in Mobile Ad Hoc Networks*. In Proceedings of the Seventh International Conference on Wireless On-demand Network

Systems and Services (WONS 2010), pages 91–98, 2010.

[Wallentin10b] L. Wallentin, J. Fabini, C. Egger, and M. Happenhofer. *Cashflow: A Channel-Oriented, Credit-Based Virtual Currency System to Establish Fairness in Ad-Hoc Networks with Selfish Nodes.* In Proceedings of the Second International Conference on Ad Hoc Networks (ADHOCNETS 2010), pages 1–2, 2010.

[Wallentin10c] L. Wallentin, M. Happenhofer, C. Egger, and J. Fabini. *XML meets Simulation: Concepts and Architecture of the IBKSim Network Simulator. Eurosim - Simulation News Europe*, vol. 20, pp. 16–20, April 2010.

[Wu07] B. Wu, J. Chen, J. Wu, and M. Cardei. *A survey of attacks and countermeasures in mobile ad hoc networks. Wireless Network Security*, pp. 103–135, 2007.

[Yang04] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang. *Security in mobile ad hoc networks: challenges and solutions. IEEE Wireless Communications*, vol. 11, no. 1, pp. 38–47, 2004.

[Zhong03] S. Zhong, J. Chen, and Y.R. Yang. *Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks. INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 3, pp. 1987–1997, 2003.