IMPLEMENTATION OF A MULTICHANNEL MULTIRADIO PROTOTYPE ON EMBEDDED LINUX

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Christine Müller
2010

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

Contents

Contents **iii**

List of Figures **v**

List of Tables **ix**

1 Introduction **1**

2 Related Work on Multichannel Multiradio Communication on WMNs **7**

2.1 Multichannel Protocols 7

2.1.1 Split Phase 8

2.1.2 Common Hopping 9

2.1.3 Dedicated Control Channel 10

2.1.4 Hybrid Protocol 11

2.1.5 Multichannel Protocol Overview 13

2.2 Why a Multichannel Protocol can boost IEEE 802.11 Performance? 13

2.3 Adjacent Channel Interference 14

2.4 Secure Remote Management and Software Distribution for Wireless Mesh Networks 15

3 Net-X Framework **17**

3.1 Channel Abstraction Layer 18

3.2 Interface Device Driver Modifications 19

3.3 Kernel Multichannel Routing Support 21

3.4 Integrated Protocol 22

3.4.1 Multichannel Routing Protocol 22

3.4.2 Interface Management Protocol 23

4 Implementation of the Net-X Framework for Linux Kernels with Version 2.6 25

4.1 Migration of the Channel Abstraction Layer 26

4.2 Migration of the Kernel Multchannel Routing Support Module 26

4.3 Network Interface Device Driver Modifications 27

4.4 Start-up Script 27

4.5 Integration in SRM 30

5 Performance Evaluation 31

5.1 Testbed Environment 31

5.1.1 Hardware and Software 31

5.1.2 Traffic Parameters 31

5.2 Unidirect UDP Traffic Throughput 32

5.2.1 Adjacent Channel Interference in a Static Multichannel Multiradio Network 32

5.2.2 UDP Throughput with Dynamic Multichannel Multiradio Communication 35

5.3 Bidirectional UDP Traffic 44

6 Conclusions and Future Work 47

Bibliography 51

List of Figures

1.1 Wireless network running in infrastructured mode 2
1.2 MANET architecture 2
1.3 WMN running in adhoc mode 4

2.1 Split Phase Protocol [1] 8
2.2 Common Hopping Protocol [1] 10
2.3 Dedicated Control Channel Protocol [1] 11
2.4 Hybrid Protocol 12
2.5 Hidden Node Problem 13
(a) Node C is inside node A's interference range 13
(b) Node A is outside node D's interference range 13

3.1 Multichannel Routing [2] 18
3.2 Position of CAL and device driver in ISO/OSI network stack 19
3.3 The Net-X framework as depicted in [2] 20
3.4 Multichannel Routing Metric (MCR) [3] 22
(a) Channel Divers Routes 22
(b) Reduction of switching delay 22

5.1 Channel interference measurement 32
(a) Test with two nodes. Both are sender and receiver. 32
(b) Test with three nodes. One is sender, one is receiver and one is sender and receiver. 32
5.2 Overall throughput in kbps of mutual transmissions between two nodes 33
5.3 Overall throughput in kbps of two simultaneous one-hop transmissions 34

5.4 Different scenarios of channel interference measurement with four nodes. . . . 35
(a) receiving nodes are close 35
(b) receiving and sending node are close 35
(c) sending nodes are close 35
5.5 Throughput in kbps with parallel transmissions 36
5.6 Measuring throughput after different number of hops 37
5.7 Throughput after a certain hop distance measured in kbps 38
(a) Multichannel network 38
(b) Monochannel network 38
5.8 Impact of the hop distance on the delay (in seconds) in a multichannel network 39
(a) 1500 kbps data stream 39
(b) 3000 kbps data stream 39
5.9 Impact of the hop distance on the delay (in seconds) in a monochannel network 40
(a) 1500 kbps data stream 40
(b) 3000 kbps data stream 40
5.10 Simultaneous Transmission Scenarios 41
(a) Scenario 1 41
(b) Scenario 2 41
(c) Scenario 3 41
5.11 The throughput of two simultaneous transmissions as shown in Figure 5.10(a). . 42
(a) Monochannel network 42
(b) Multichannel network 42
5.12 The throughput of two simultaneous transmissions as shown in Figure 5.10(b). 42
(a) Monochannel network 42
(b) Multichannel network 42
5.13 The throughput of two simultaneous transmissions as shown in Figure 5.10(c). . 43
(a) Monochannel network 43
(b) Multichannel network 43
5.14 The throughput of two simultaneous transmissions in a multichannel network. . 44
5.15 Limited queue length of network devices lowers the TCP network throughput . 45
5.16 Bidirectional UDP traffic 45
5.17 Throughput measurements with bidirectional UDP 46

5.18 Impact of the hop distance on the delay (in seconds) in a multichannel network using bidirectional UDP 46
(a) 1500 kbps data stream 46
(b) 3000 kbps data stream 46

6.1 Proposed solutions to reduce packet loss when switching the channel 47

List of Tables

2.1 Overview of the multichannel protocols 13

2.2 The measured channel power of adjacent channels when an 802.11a radio continuously broadcasts on channel 52 (5260MHz) [4]. 15

5.1 Overall throughput in kbps of mutual transmissions between two nodes 33

5.2 Throughput of two simultaneous one-hop transmissions in kbps 34

5.3 Throughput with parallel transmissions (illustrated in Figure 5.4(a)) 36

5.4 Throughput with parallel transmissions (illustrated in Figure 5.4(b)) 36

5.5 Throughput in kbps with parallel transmissions (illustrated in Figure 5.4(c)) . . 36

5.6 Impact of the hop distance on the throughput (in kbps) in a multichannel network 38

5.7 Impact of the hop distance on the throughput (in kbps) in a monochannel network 38

5.8 Impact of the hop distance on the delay (in seconds) in a multichannel network with a 1500 kbps data stream 39

5.9 Impact of the hop distance on the delay (in seconds) in a multichannel network with a 3000 kbps data stream 39

5.10 Impact of the hop distance on the delay (in seconds) in a monochannel network with a 1500 kbps data stream 39

5.11 Impact of the hop distance on the delay (in seconds) in a monochannel network with a 3000 kbps data stream 40

5.12 Average throughput of two simultaneous transmissions in a monochannel network 41

5.13 Packet loss rate 43

Acknowledgment

I would like to thank very much Peter Dely and Prof. Dr. Andreas Kassler from the University of Karlstad (Sweden) for inviting me to their University and helping me with the measurements for this Bachelor thesis. Further thanks are dedicated to my supervising tutor, Thomas Staub who assisted me with helpful advises and reading through my work and to my colleagues and friends for their mental support. At last, a special thank goes to my parents who afforded my entire education.

Chapter 1

Introduction

In conventional wireless networks running in infrastructured mode (WLANs) the communication between the network clients is handled by a central instance which is responsible for the data transmission between two clients, as shown in Figure 1.1. The central instance is the access point. The access point not only enables connectivity within the wireless network itself, it also interconnects the wireless network to the wired network (Ethernet, DSL, etc.). If too many clients want to communicate with each other or to access another network at the same time the access point becomes a bottleneck. Furthermore, if the access point fails, the whole network is put out of service. But in exchange, routing in such networks is simple, as the clients only communicate via access point with other clients (star topology). In other words the clients simply have to send data to the access point.

Networks running in adhoc mode bypass the need of a centralized access point. Adhoc networks are self-managed since every node in the network provides routing functionality and connects directly or indirectly to all other nodes. Every node maintains its own routing table. Transmission is routed over multiple hops if the source and destination node are not neighbors. The main challenge for adhoc networks is the avoidance of packet collision. Since no centralized instance manages the data traffic of the network, the nodes have to monitor the network on their own by exchanging control packets.
Examples for adhoc networks are mobile adhoc networks (MANETs) [5] (illustrated in Figure 1.2) or wireless mesh networks (WMNs) [6] (illustrated in Figure 1.3), whereas MANETs only consist of mobile clients and WMNs consist of both mobile and stationary instances.

Clients in MANETs join or leave the network or change their position frequently, which constantly leads to changes in the network topology. A link in the path from source to destination may break due to node mobility. In contrast, when a new node joins the network a better path could evolve. Since no centralized routing instance exists and every node has to maintain a routing table, such topology changes complicate routing in MANETs compared to infrastructured WLANs. Further routing difficulties result from the asymmetric network connections. The path from node A to node B can be different than the path from node B to node A due to asymmetric transmission ranges and link qualities. MANET routing protocols should further consider that mobile network devices are often limited regarding to their energy consumption since they rely

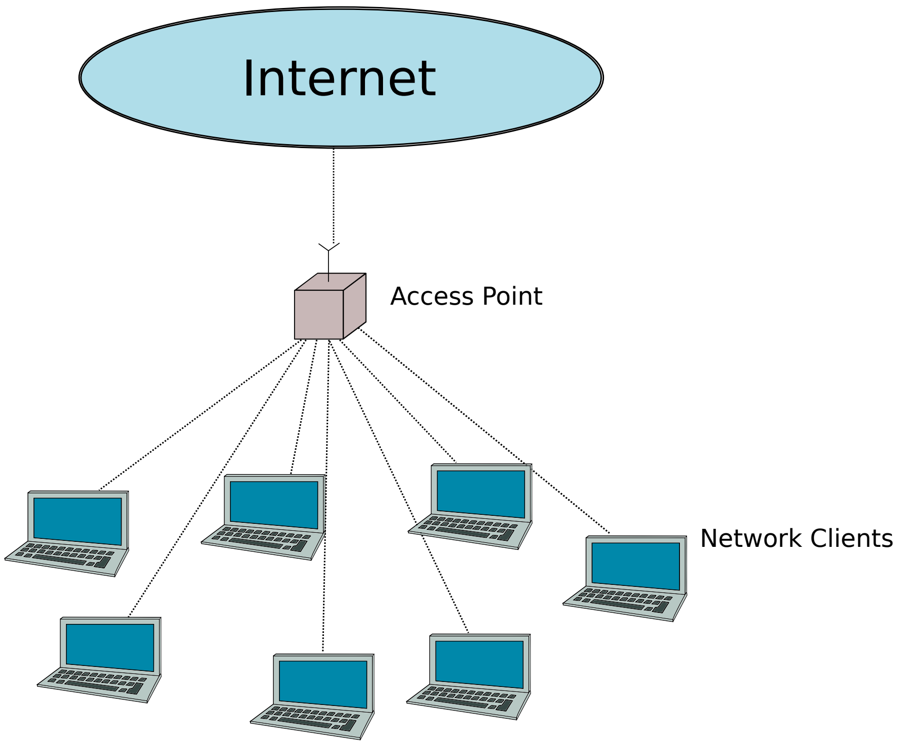


Figure 1.1: Wireless network running in infrastructured mode

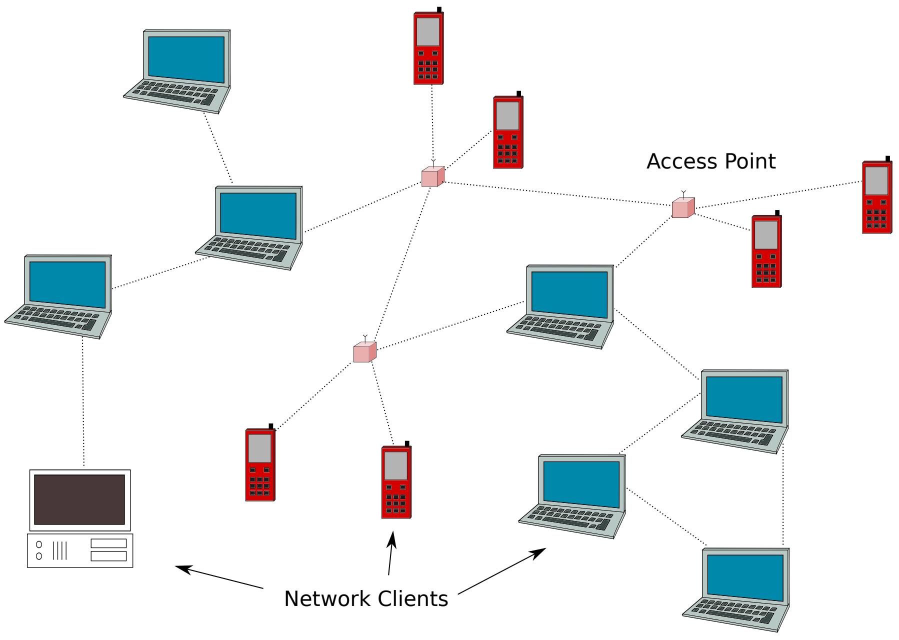


Figure 1.2: MANET architecture

on batteries.

WMNs [6] consist of mesh routers and mesh clients whereas the routers are mostly stationary and build the backbone of the WMN. Some of the routers can act as an access point or they can connect to one, which enables the access to the Internet and hence, to external networks. The mesh clients are mobile and organized like a MANET. Multiple MANETs can join the WMN by connecting to a mesh router as shown in Figure 1.3. The focus of this Bachelor thesis lies on the stationary backbone of the WMN.
WMNs are more robust and therefore more reliable than conventional wireless infrastructured networks. As the access point in infrastructured networks may be a single point of failure, a broken link in adhoc networks does not necessarily yield to a communication interrupt between two nodes because in WMNs several paths from one node to another may exist. The path redundancy in WMNs can further be utilized to distribute the network traffic on several links. This may highly lower data congestion.
Furthermore, WMNs have the ability to integrate different network technologies. For example, mesh nodes can act as a gateway to the Internet. Since the infrastructure and installation costs for WMNs are low, they are used as the last-mile infrastructure between mobile wireless and wired networks.
Moreover, WMNs are easily extensible. If the network coverage should be enlarged, additional mesh routers can be simply installed on the area's border. Due to their straight forward deployment process, WMNs can for example be set up at locations where wired network infrastructure is scarce (e.g. mountainous regions, deserts, developing countries, etc.).

There are two routing concepts which are applicable for routing in MANETs and WMNs - the proactive and the reactive routing. Proactive routing protocols calculate the routing tables even though some routes may never be used. The routing tables have to be calculated periodically as the network topology permanently changes. This generates high control traffic in the network and costs additional energy due to the frequent recalculation of each node's routing table. Therefore, proactive protocols became obsolete for MANETs. Although, for the stationary backbone of WMNs proactive routing is an option, it will not be applied in this thesis. The proactive approach has been implemented in the Destination Sequence Distance Vector (DSDV) [7] routing or in the Optimized Link State Routing (OLSR) [8] protocol.
Reactive, also called on-demand, routing protocols establish routes only if the route is required to transmit data traffic. The route is then kept in the routing table for a certain life-time. The on-demand characteristic reduces significantly the control traffic and the network devices can stay in standby mode as long as they are not involved in any transmission and hence save energy. Examples for on-demand routing protocols are Dynamic Source Routing (DSR) [9], Dynamic MANET On-demand (DYMO) routing [10], Associativity Based Routing (ABR) [11] and Adhoc On-Demand Distance Vector (AODV) routing [12]. Relevant for this thesis is the AODV routing protocol which is the basis of the multichannel routing protocol provided by the Net-X developers [2]. This protocol is therefore explained in detail in Section 3.4.1.

As WMNs are robust, reliable, affordable and everywhere deployable, they have the potential for commercial use. But to establish a network technology for commercial use high throughput is one of the most important requirements because transmission payload is increasing constantly

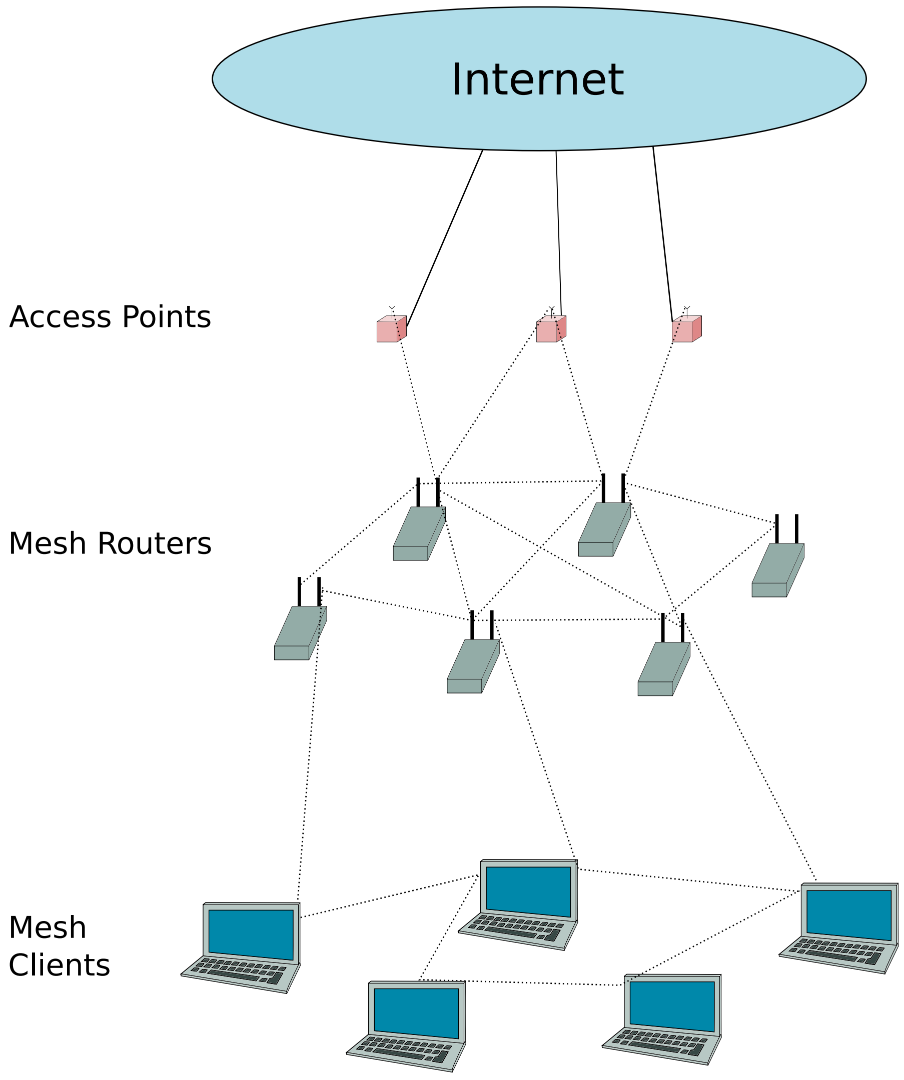


Figure 1.3: WMN running in adhoc mode

with the rise of multimedia computing. Common wireless transmission technologies are still not as efficient as wired ones. The reasons for that are manifold. One example is the medium access control protocol. In contrary to wired networks, wireless networks have no mechanism to detect packet collisions. Therefore, the devices have to use a mechanism to avoid packet collision with the Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) protocol [13]. A node which intends to transmit sends a Request-to-Send (RTS) packet to reserve the network as soon as it assumes that the network is free. If the transmission range of the receiver is free as well, the receiver sends back a Clear-to-Send (CTS) to the sender and the data transmission starts. The exchange of the control packets delays the data transmission. The CSMA/CA protocol helps to avoid packet collisions and enhance the network throughput as less packets have to be retransmitted. However, as shown in Section 2.2 many collisions happen anyway due to the hidden node problem.
Another reason for the lower capacity for wireless data transmission is that it is subject to comparatively high levels of interference. Simultaneous wireless transmissions increase the noise level at the receiver and may even lead to erroneous or lost packets.
Packet loss and packet disturbance lead to lower overall throughput as data packet transmissions have to be repeated. An approach to minimize packet collisions and retransmission is to send data on multiple orthogonal frequencies within a wireless network. This method allows nodes to transmit their data simultaneously without disturbing each others data traffic. Research has shown that this can increase the throughput in the network considerably. Section 2.2 discusses this statement.

The goal of this thesis is to increase the overall throughput in a WMN by implementing a multichannel multiradio prototype running on embedded nodes. The mesh nodes are PCEngines WRAP platforms which are equipped with two MiniPCI 802.11 a/b/g. The operating system, an Embedded Linux (see Section 2.4), and the applications are stored on a CompactFlash card.

The implementation of a multichannel protocol is not trivial. The following aspects have to be taken into account. Connectivity between neighbor nodes is only given if they have at least one of their antennas tuned to the same frequency. This can be easily achieved by using the same channels as there are several interfaces per node. But in most cases, the devices do not have as many radios as there are channels available. Therefore, some channels stay unused. IEEE 802.11a for example provides 12 orthogonal channels which may be used simultaneously. But usual devices only offer one to three radio interfaces and therefore, a simple one-to-one mapping does not achieve an optimal throughput. Furthermore, the static mapping may lead to interference due to common channels on the path.
Another scheme utilizes more channels than interfaces. This makes the system more efficient but also more complex. A mechanism has to be introduced which decides at what time on which frequency an interface should stay in order to guarantee connection to a certain neighbor whenever necessary. At the same time, another tool has to guarantee that the nodes in the network do not switch all their interfaces to the same channel. Otherwise, collisions appear as in monochannel networks and there are no performance benefits. As already shown in literature, despite the higher complexity, the approach where more channels are used than interfaces leads to a clearly higher throughput. Therefore, we will concentrate on this approach.

Furthermore, multichannel routing protocols are poorly supported by common operating systems. A research group at the University of Illinois at Urbana-Champaign has developed a framework, the Net-X framework [2], which provides the lacking support for routing in a multichannel network. The framework was implemented for Linux operating systems with version 2.4, but the mesh nodes that we use in our testbed run a Linux kernel with version 2.6.22.2. Therefore, the main contribution of this Bachelor thesis became the migration of the Net-X framework to the newer kernel version.

Throughout the thesis, IEEE 802.11a is used for setting up the networks, if not explicitly named. IEEE 802.11a operates in the 5GHz frequency band and provides 12 orthogonal channels, whereas the channels 36, 40, 44, 48, 52, 56, 60 and 64 are in the lower band and the channels 100, 104, 108, 112, 116, 120, 124, 128, 132, 136 and 140 are in the upper band.

The outline of the thesis is as follows: Chapter 2 addresses related works which covers the subjects beginning at multichannel routing protocol strategies to the benefit of using these protocols and to the known problem called adjacent channel interference. Furthermore, the embedded operating system running on the mesh nodes is described in this chapter. In Chapter 3, the multichannel framework Net-X is presented. The framework provides solutions for the lacking kernel features in order to support multichannel communication. Furthermore, it features an adapted wireless driver, a multichannel routing protocol and a channel assignment protocol. The Net-X framework was developed for Linux kernels with version 2.4. Then, in Chapter 4, we describe the migration of Net-X to Linux kernels with version 2.6 and the integration into the Embedded Linux which is running on the mesh nodes. In Chapter 5, we present and discuss the results measured on the KauMesh testbed of the University of Karlstad, Sweden. Chapter 6 provides the conclusions and presents some topics for future work.

Chapter 2

Related Work on Multichannel Multiradio Communication on WMNs

Multichannel multiradio communication on WMNs is a popular field of todays research and therefore, lot of related work exist. The most relevant publications are explained in detail in this chapter.
Section 2.1 describes different multichannel protocol approaches which guarantee connectivity between neighbor nodes in a multichannel network. Section 2.2 is about the benefit of using multichannel protocols. Section 2.3 addresses adjacent channel interference which is a reason why multichannel networks do not reach the maximal network throughput, and finally Section 2.4 describes the operating system running on the mesh nodes.

2.1 Multichannel Protocols

The multichannel communication requires a method that assigns channels to interfaces. There are three categories of assignment strategies - a static, a dynamic, and a hybrid channel assignment.

Static assignment strategies bind all interfaces to a fixed channel. Neighboring nodes have to tune at least one of their interfaces to the same channel to hold connectivity. Otherwise, the network splits and connectivity to all nodes in the network cannot be guaranteed. Furthermore, if the network devices have less interfaces than available channels the bandwidth of the network cannot fully be utilized.
For dynamic assignment strategies there is only one interface required per network device. Interfaces are switched to a new channel whenever required in order to set up connectivity to a certain node. This way, all available channels can be fully utilized even if there are less interfaces available than channels. A disadvantage of the dynamic approach is that the devices have to be time synchronized. Otherwise, there is no guarantee that the interfaces of two nodes are tuned to the same channel at the same time and hence, the nodes cannot communicate.
Hybrid assignment strategies combine the static and the dynamic approach. This strategy re-

quires at least two interfaces per network device one of which is assigned to a fixed channel and the other changes the channel dynamically. With this approach both connectivity to all nodes in the transmission range can be guaranteed, all available channels can be exploited (maximum bandwidth) and the devices do not require time synchronization.

The following sections explain different channel assignment schemes. Section 2.1.1 and Section 2.1.2 cover dynamic strategies. The Sections 2.1.3 and 2.1.4 describe hybrid approaches. Static assignment strategies are not covered in this thesis.

2.1.1 Split Phase

The Split Phase Protocol [1] is an example of a dynamic assignment approach. The protocol does only require one interface. The time is split into two phases. During one sequence (control phase) control messages are exchanged and during the other (data phase) data packets are sent. Furthermore, a default channel is determined to which all nodes switch their interface during the control phase. During the control phase a node that intends to send data agrees with the destination node the channel to use in the data phase. The available channel set includes also the default channel.
Some implementations of the Split Phase approach allow more than one node pair to be tuned to the same channel for data exchange. In this case, a scheduling mechanism is necessary for coordination of the data transmissions. If only one node pair is permitted to occupy the same channel during the data phase, additional informations about channel reservations have to be exchanged within the interference range of each node.

Figure 2.1 illustrates the Split Phase Protocol. During the control phase, two nodes inform two other nodes about their intension of transmitting data to them by sending them an RTS (RTS1 and RTS2). The RTS includes a channel number to which the corresponding nodes should tune their interface to as soon as the data phase starts. Like this, the data packets are transmitted on different channels and therefore, they do not interfere each other. As soon as the data phase is over, the nodes change the channel back to the control channel.

An advantage of this protocol is that only one interface per device is required. However, all nodes in the network have to be time synchronized.

Examples for protocols which have been implemented after the Split Phase approach are the Multichannel Access Protocol (MAP) [14] or the Multichannel MAC Protocol (MMAC) [15].

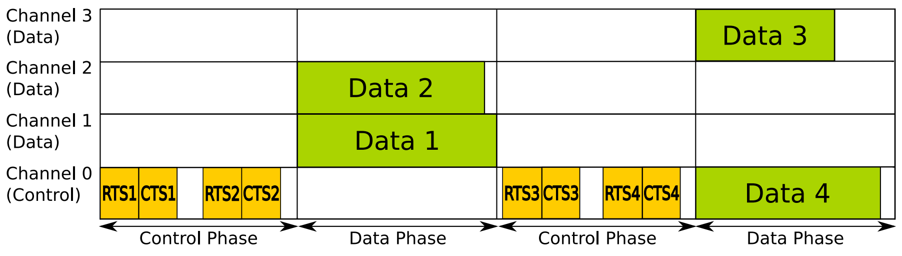


Figure 2.1: Split Phase Protocol [1]

2.1.2 Common Hopping

The Common Hopping Protocol [1] is another dynamic multichannel protocol working with only one interface per network device. In this approach, all network devices follow a common hopping sequence which means that they all switch their interface simultaneously in the same order from one channel to the next. If a node wants to do a transmission to another node, it sends an RTS packet on the common channel to the dedicated receiver and the receiver then sends a CTS packet back. The node pair leaves the hopping pattern and stays on the channel until the data transmission is done. Subsequently, they rejoin the common hopping cycle.

It is possible, that two nodes have not finished their data transmission when the hopping cycle reaches a channel on which a data transmission still takes place. As the idle devices sense the carrier, they are aware of the busy status of channels and devices. So if a node wants to send, it has to follow the hopping pattern until the common channel is free.
Nodes that are about to exchange data and therefore do not follow the common hopping sequence are unaware of the busy status of other devices. Therefore, it happens, that a node sends an RTS to another node that is currently in a data transmission and that is not tuned to the common channel. In this case, the sender of the RTS does not receive the CTS. Hence, instead of starting the data transmission it stays in the common hopping cycle and tries to reach the destination node later.

Figure 2.2 illustrates how the protocol is working. At the moment the first node sends an RTS (RTS1) to another node, every interface is tuned to channel 0. The receiver of RTS1 sends back the CTS1 and both sender and receiver keep their interfaces on channel 0 for data transmission (data1). In the meanwhile the other idle devices hop from one channel to the next following the common hopping sequence. As soon as the data1 exchange ends, the two nodes rejoin the hopping cycle. The same set of actions are executed when other nodes try to communicate. After the first hop in Figure 2.2, no other node has the intension to send. Therefore, the channel stays idle. Only one hop later, an agreement between two nodes is done to stay on channel 2, while the other idle devices - except the two communicating on channel 0 - go on cycling.

The protocol has more or less the same advantages as the Split Phase approach shown in Section 2.1.1. The network nodes require only one interface and - in contrary to the Dedicated Control Channel Protocol (explained in Section 2.1.3) - every available channel can be utilized for data exchange. But the required time synchronization among the nodes is even more strict than in the Split Phase protocol. All devices have to hop exactly at the same time. Therefore, an exact synchronization is required.

Hop-Reservation Multiple Access (HRMA) [16], Channel Hopping Multiple Access (CHMA) [17] and CHMA with Packet Train (CHAT) [18] are predicated on the Common Hopping approach.

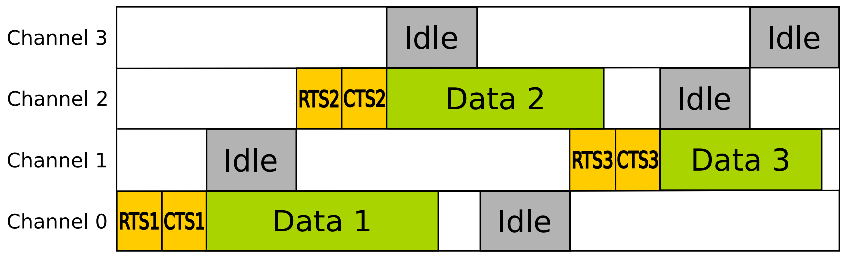


Figure 2.2: Common Hopping Protocol [1]

2.1.3 Dedicated Control Channel

The Dedicated Control Channel Protocol [1] is a hybrid multichannel multiradio protocol. The nodes that are participating in the network require at least two interfaces. The system bandwidth is divided into n channels. One channel is reserved to exchange the network control messages RTS and CTS and is assigned to the control interface. The remaining $n-1$ channels are available to the other interface. This interface can switch the channel whenever necessary. It is exclusively used to transmit and receive data and ACK messages.
When a device wants to send some data to another device the sender transmits an RTS message to the receiver on the common control channel. To the source and destination ID and the packet length, the RTS message additionally includes a list of all available channels, i.e. all channels which are not occupied in the interference range of the sender. This list is called the *Free Channel List* (FCL). When the destination node receives the RTS, it selects the channel which has the lowest number and is free in the receivers environment. The selected channel is inserted into the CTS message that is sent back to the source device. Because all nodes in the network use the same channel for RTS and CTS exchange and the channel information is inside these messages, every neighbor of the communicating nodes knows which channels are currently in use and for how long. The control channel can further be used to spread broadcast packets.

Figure 2.3 shows an example with one control channel (channel 0) and three data channels (channels 1 to 4). All nodes involved in this scenario are in each others interference range. Furthermore, they all have their static interface tuned to the control channel (channel 0). A node decides to send data to another node, and transmits an RTS (RTS1) to the intended receiver on the control channel. As no channel is occupied, the FCL includes all data channels in the RTS1. As soon as the receiver gets RTS1, it chooses the channel with the lowest number out of its FCL (in this case its channel 1), sends this channel number in the CTS1 to the sender and tunes its second interface to the proposed channel. After the sender has received CTS1, it changes its switchable interface to the agreed channel 1 and starts transmitting the data.
In the meantime, the two other nodes have started their data channel selection. This time, the FCL included in RTS2 contains only channel 2 and channel 3, as channel 1 is already reserved. The receiver inserts channel 2 (the lowest number) in its CTS (CTS2) and the two nodes start their data exchange on this channel.
At the time the third node wants to send data, channel 1 and channel 2 are already occupied. Therefore, it can only propose channel 3 to the receiver. Even though channel 1 is free again

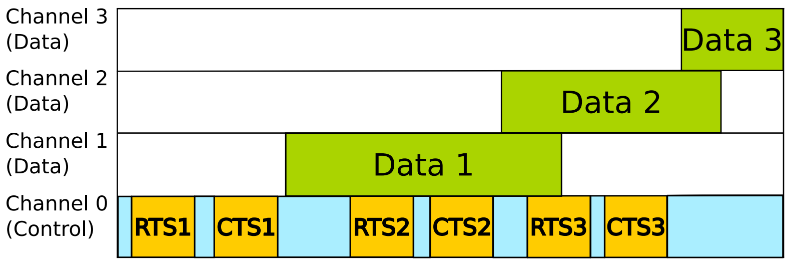


Figure 2.3: Dedicated Control Channel Protocol [1]

when the receiver sends back CTS3, it has to choose channel 3 due to the fact, that channel 3 was the only channel suggested by the sender.

The advantage of this protocol is the simplicity of the implementation and the nature of all hybrid protocols, that they do not require synchronization.
The protocol's drawback lies in the character of the channel distribution. Control messages may only be exchanged on one dedicated channel. If the network contains many nodes and the bandwidth is divided into many narrow channels, the control channel can become a bottleneck. The RTS and CTS messages may collide and reduce the network throughput. The control channel can also become costly if the bandwidth is split into few channels. E.g. IEEE 802.11b only provides three orthogonal channels. So if one channel is reserved as a control channel only 66.6% of the bandwidth remain for data exchange.

Implementations based of the dedicated control channel approach are the protocols Dynamic Channel Allocation (DCA) [19] and Dynamic Channel Allocation with Power Control (DCA-PC) [20].

2.1.4 Hybrid Protocol

The Hybrid Protocol [21] is another hybrid strategy working with at least two interfaces - one of which is assigned to a fixed channel and the other switches channel whenever necessary. The static interface of every device is assigned to a different fixed channel. This interface is for the limited purpose of receiving data. The switchable interface is dedicated only for sending data. Any data which arrives on this sending interface is rejected.
If a device intends to send it switches its sending interface to the fixed channel of the receiver and transmits the data on this channel and interface. For broadcasting the sending device sends a copy of the data packet on every available channel to make sure that every device in the transmission range receives the broadcast message.
Via broadcast packets, a node informs its neighbors about the used receiving channel. The receivers store this information in a list, so that every node knows the receiving channel of its neighbors. Refering this list, a node may change the receiving channel if it notices that other nodes in the interference range are utilizing the same channel as fixed channel.

Figure 2.4 shows a simple example. The receiving interface of node 1 is tuned to channel 0,

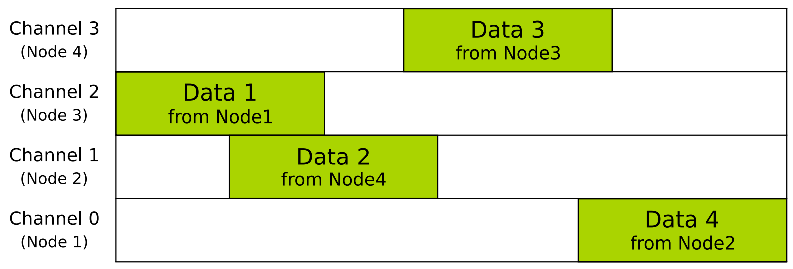


Figure 2.4: Hybrid Protocol

of node 2 to channel 1, of node 3 to channel 2 and of node 4 to channel 3. Node 1 intends to send data (data 1) to node 3. Therefore, it switches the sending interface to channel 2 which is the fixed receiving channel of node 3. On that channel it then starts the transmission. In the meanwhile, node 4 wants to transmit data (data 2) to node 2. As the receiving interface of node 2 is tuned to channel 1 node 4 sends data 2 on this channel.
If the network devices are capable to send and receive simultaneously, node 3 may even start transmitting data to node 4 while node 4 is still sending to node 2 (see data 2 and data 3 in Figure 2.4).

Unlike the Dedicated Control Channel strategy, the Hybrid protocol approach uses all available channels for data transmission. Furthermore, no time synchronization is required. As long as the nodes are aware of the fixed channel of the neighbor devices, connectivity is always guaranteed. Compared to the other channel assignment strategies the Hybrid protocol exploits the available bandwidth the most, since all channels are used for data transmission and control data for time synchronization is omitted.

The Hybrid approach was implemented by the Net-X developers at the University of Illinois [21]. In Section 3.4 the implementation is discribed in more detail.

2.1.5 Multichannel Protocol Overview

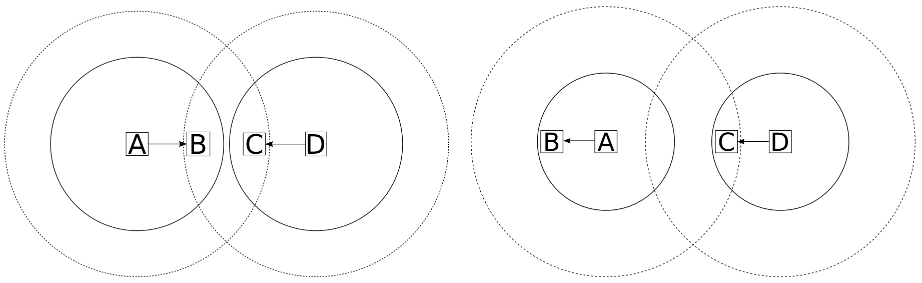
Table 2.1 presents a short overview of the different channel assignment approaches.

Approach	**Type**	**Min. Req. Interfaces**	**Time Synch.**	**Control Channel**	**Protocols**
Split Phase	dynamic	1	yes	yes	MAP, MMAC
Common Hopping	dynamic	1	yes	no	HRMA, CHAT, CHMA
Dedicated Control Channel	hybrid	2	no	yes	DCA, DCA-PC
Hybrid Protocol	hybrid	2	no	no	Hybrid Protocol of Net-X

Table 2.1: Overview of the multichannel protocols

2.2 Why a Multichannel Protocol can boost IEEE 802.11 Performance?

The study [22] investigates the reasons for performance improvements of multichannel mechanisms in an adhoc wireless network. The study assumes that the adhoc wireless network uses one control channel which is reserved for RTS and CTS frames and multiple data channels to transmit data and ACK packets. It identifies that multichannel networks reduce packet collisions and therefore increase the overall throughput due to reduce the amount of packet retransmissions.



(a) Node C is inside node A's interference range (b) Node A is outside node D's interference range

Figure 2.5: Hidden Node Problem

In monochannel wireless networks a common reason for packet collision is the hidden node problem. This problem is illustrated in Figure 2.5. The dotted lines show the interference range, the continuous lines show the transmission range. In Figure 2.5(a) node B is within the transmission range and node C in the interference range of node A. On the other hand, node D does

not overhear transmissions of node A and vice-versa. Hence, the hidden node problem may occur since node A considers the medium as free even though node D is transmitting data to node B. Thus, their packets collide.
The multichannel network determined by [22] avoids the hidden node problem because control and data frames are sent on separate channels. Given the same scenario as above, the RTS message of node A sent to node B does not collide even though node D is transmitting data. Furthermore, since node B is in node D's interference range, it knows about the data transmission although it does not correctly receive the frames. Therefore, node B informs node A to use another data channel than node D in the CTS response. Thus, neither the control nor the data frames sent by node A collide with the packets from node D.

Even in the scenario shown in Figure 2.5(b) where both nodes A and B are outside the interference range of node D a multichannel protocol may increase performance. The RTS frame of node A interferes with the transmission of node D if the bandwidth is not separated in different channels.
Using a separate control channel yields to an error-free control message transmission even though node D is sending data to node C. But as both nodes A and B are outside the interference range of node D none of them knows about the channel node D is using currently. Therefore, node A has to choose a channel randomly. By chance, it picks the same as node D which then would lead to a data packet collision.

The study of [22] concludes that the multichannel approach only boosts performance in presence of hidden nodes. In a full visibility scenario where all nodes know about all transmissions that could interfere with their own transmission, the throughput cannot be increased by using multiple channels. It may rather be low because only a part of the bandwidth can be utilized.
The separation of the control channel from data channels has a bigger impact in terms of performance boost than the separation of data channels. Even when using multiple data channels in a network data packet collisions cannot be avoided for sure.

2.3 Adjacent Channel Interference

The paper of [4] analyses the cause and the effect of adjacent channel interference (ACI) in multiradio multichannel adhoc networks. ACI is the interference of non-overlapping channels. It happens, if a node sends at the same time as it receives data. Even though the sending and receiving antenna are tuned to different channels, the outgoing data disturbs the incoming if the channel distance is too low. ACI affects significantly the network throughput.
The disturbance happens because some of the transmission power is leaking to the adjacent channels. Mainly, the direct neighbor channels are affected. The experiment of [4] resulted in the transmission power leakage for the adjacent channels as shown in Table 2.2.

Table 2.2 points up the strong impact the sending data stream has on the node's receiving data if the channel distance is too low. For example, if the node sends on channel 56 and receives on channel 52, the resulting noise signal on the receiving channel is only 27 dBm lower than

Ch.	36	40	44	48	52	56	60	64
dBm	-59	-59	-52	-27	0	-26	-53	-57

Table 2.2: The measured channel power of adjacent channels when an 802.11a radio continuously broadcasts on channel 52 (5260MHz) [4].

the transmission signal on the sending channel. This high noise averts a proper reception of the incoming data.
In [23], the researchers came to the same conclusion as [4], namely, that a transmission of a node interferes with the reception if the channel distance is less than three channels. ACI can also occur if a neighboring node transmits to another neighboring node on a channel that is fewer than two channels away.
Taking this findings into account, [23] designed a channel allocation algorithm which reduces ACI. The algorithm is based on a local-balancing algorithm called LocBal. LocBal observes which receiving channels are used by the two hop neighbors. This prevents the nodes from allocating a channel which is already frequently used in the neighborhood.
However, LocBal only allows the usage of five channels out of the twelve channels in IEEE 802.11a, if the channel distances mentioned above shall be guaranteed. Therefore, the extended algorithm IntAware was developed. Although this algorithm allows utilizing all of the twelve available channels, it is able to avoid ACI. Instead of having the receiving channel statically assigned, IntAware tunes the receiving interface to another channel as soon as the channel of the sending interface is switched to a channel with a lower distance than three channels away from the current receiving channel.
Furthermore, to avoid neighboring nodes to disturb the node's own communication, a channel selection algorithm assigns channels to the node's receiving interface which is as far away from the channels used by the one and two hop neighbors. Like this, the probability is reduced that neighbors use a channel that is less than two channels away from the own receiving channel. Thereby, the chance of ACI is decreased.
Compared to the channel allocation algorithm LocBal which uses at most five channels, IntAware increases the overall network throughput by providing a strategy which enables the reduction of ACI in a network that uses all of the twelve available channels in IEEE 802.11a.

2.4 Secure Remote Management and Software Distribution for Wireless Mesh Networks

In many cases, wireless mesh networks (WMN) are spread over geographically large areas. The mesh nodes are often located at places which can not easily be reached (e.g. rooftops, towers, masts, ...) due to the terrain or the missing legal access to the spot. Therefore, software updates and reconfigurations are getting very costly if the remote access to a node is disrupted. Erroneous software or incorrect configurations potentially cause such disruption.

The paper [24] presents an architecture which provides a secure way to distribute software updates and reconfigurations in a WMN. It protects each node from loosing connection from the network by providing a fall-back mechanism. If the software upgrade or the configuration changes contain failures which affect the node's connectivity to the network a reboot with the initial configuration or kernel images is activated.

Furthermore, the architecture offers a management console, a development environment and an image generator. These tools are provided by the management nodes. Every network contains at least one management node which is equipped with better hardware than ordinary mesh nodes and therefore also has further features. Thus, a Linux management node can run a LiveCD which provides the above mentioned tools.

The management console supplies making initial network configurations or changing existing ones. When starting, the user is asked to choose an existing network configuration which is stored on an usb device or to define some parameters (domain name, passwords, etc...) to initiate a new network. Then, the user can connect to the LiveCD's SSL web server via his web browser where he finds a setup page. On this page, he can define the network (number of nodes, host names and passwords of the nodes). Optionally, each node's network setup can be configured individually. The LiveCD then generates a configuration image for every node. This images have to be installed on the corresponding nodes.

The development environment facilitates the compilation and installation of new software or kernel upgrades on the mesh nodes. Instead of booting the LiveCD in management mode, it can be started in development mode and operate as a development system. The newly compiled software (binaries, libraries or kernel image) has then to be copied to a specified directory tree from which the node image is generated by an image generation script. The node image is now ready to be flashed on the mesh nodes.

The management console and the development environment provide an easy way to create or change configuration images or to build node images with new software, respectively. Using these tools the secure remote management architecture with its fall-back mechanism can be retained.

Chapter 3

Net-X Framework

Multichannel multiradio communication depends on the use of multiple radios and multiple channels as well as on the ability to change the assigned channel of each interface. Most wireless device interfaces conform to these requirements. IEEE 802.11b/g provide 13 different channels of which three are orthogonal. IEEE 802.11a divides the bandwidth in 19 channels of which 12 are orthogonal. Orthogonal channels do not overlap and therefore do not interfere with each other as long as adjacent channels are not assigned on the sending and receiving interface of the same node. Otherwise, Adjacent Channel Interference (ACI) may occur. The interfaces also provide a mechanism to switch the channel.
But these features are poorly supported by current operating systems. For example, the routing table is only designed for monochannel networks. To route a packet to its destination, current operating systems only keep the interface but not the channel on which the packet has to be sent listed in their routing table. Therefore, the packet cannot be routed correctly in a multichannel network.
Furthermore, since a mesh network runs in adhoc mode, adhoc routing - in our case on-demand-routing (ODR) - is required [25]. Most ODR protocols are designed for routing in monochannel networks. To support multichannel networks, adhoc routing support has to be extended to work with multiple channels.
The Net-X framework [2] offers the required operating system support. The framework consists of two kernel modules. The Channel Abstraction Layer (CAL) module provides the kernel features for utilizing the interface capabilities mentioned above and the Kernel Multichannel Routing Support (KMCR) module facilitates on-demand routing. Moreover, a user space daemon, called Integrated Protocol is provided by Net-X. The Integrated Protocol consists of a routing and a channel assignment protocol.
Section 3.1 describes the CAL module, Section 4.3 goes into the network device driver modifications, Section 3.3 explains the method of operation of the KMCR module and Section 3.4 concentrates on the Integrated Protocol.

3.1 Channel Abstraction Layer

In current operating systems there are several features missing required for multichannel communication. The most important ones are the specification of the channel for sending unicast and broadcast packets. There is no support that specifies the channel to use for the packet to reach the destination. In monochannel networks the knowledge about the interface on which the packet has to be transmitted is enough to send a packet to the destination. In multichannel networks the channel has to be specified additionally (see Figure 3.1). If node A wants to send to node B the interface has to be tuned to channel 1. If it wants to send to node C it has to use channel 2. For broadcast packets it has to send a copy on both channels. A mechanism which provides the required information is necessary.
Furthermore, there is a need for buffering and scheduling support as switching an interface incurs a non-negligible delay and switching too frequently may degrade performance. To minimize the packet and the switching delay, a scheduling mechanism has to decide when the interface may change the channel. Until then, a buffer has to store the packets which are meant to be sent on a different channel.

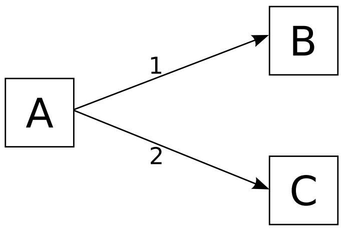


Figure 3.1: Multichannel Routing [2]

To support these features the CAL module was developed as a new layer in the network stack (as shown in Figure 3.2). The CAL operates between the ISO/OSI network layer and the link layer (device driver). It appears as a single virtual interface with a single channel to the network layer. This insulates the network layer and its protocols from the implementation of the multichannel features. In other words, the network layer does not have to care about the fact that multiple channels are used for communication and network layer protocols (e.g. ARP) do not need any modifications. The CAL provides to the device driver the information over which interface and over which channel the packets have to leave the device to reach the destination. The driver itself switches the interface whenever required.

As you can see in Figure 3.3, the CAL module is implemented in three components - unicast table, broadcast table, and a scheduling and buffering component. The unicast table is similar to the normal routing table but it contains the channel on which a packet has to leave the device in addition to the destination address and the interface. The broadcast table defines all the interfaces with their associated channels on which a copy of the packet has to be sent to reach all the neighbors. For every interface there is a separate scheduling and buffering component. The buffering component maintains a queue for every available channel. If a packet cannot be sent because the interface is not yet switched to the accordant channel it is buffered in the queue

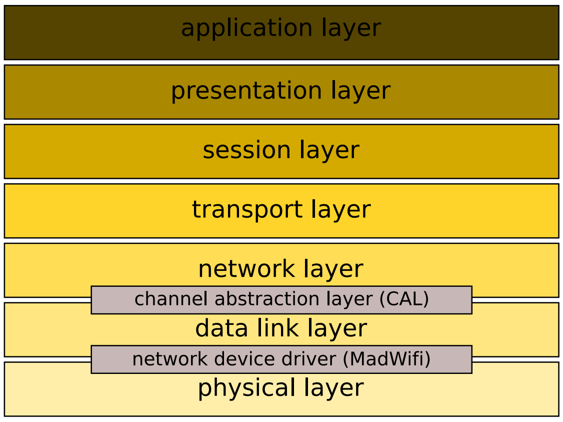


Figure 3.2: Position of CAL and device driver in ISO/OSI network stack

of that channel. The scheduler decides which queue can process the packets and send them to the device driver. The scheduling algorithm can differ between every interface. The Net-X developers use a round robin scheduler.
Moreover, if a new neighbor is detected, a node leaves the network, or changes the channel of its fixed interface, a new entry in the unicast and broadcast table has to be added, deleted or edited, respectively. This is done via *ioctrl*-calls from the user space, see Section 3.4.2.

3.2 Interface Device Driver Modifications

The Net-X project requires some modifications of the network device driver (MadWifi-old) to work optimally. Beaconing has to be disabled and further, there has to be a queue length monitoring at every interface.

Network interfaces running in adhoc mode broadcast beacons periodically. Beacons are broadcast packets which announce to all devices in the transmission range the existence of the sender and the network of which the sender is participating.
In multichannel networks, the sender has to send the beacon on every available channel to make sure that every neighbor receives the message. The frequent interface switching yields to a switching delay. Since every node of the WMN knows to which network it belongs to, time for beaconing is wasted. Therefore, the beaconing of the network device driver of Net-X (MadWifi-old) was disabled. Without beaconing, the throughput of the network could be increased considerably.

Moreover, packets have to be queued if the interface is not yet assigned to the channel on which the packet has to leave the node. As mentioned in Section 3.1 for every channel there is a separate queue in the CAL. When the interface is switched to a certain channel, the CAL sends the packets of this channel down to the device driver. The device driver itself stores the packets in its own queue, until they can be transmitted.
For scheduling channel switching, the length of the packet queue at the device driver has to be

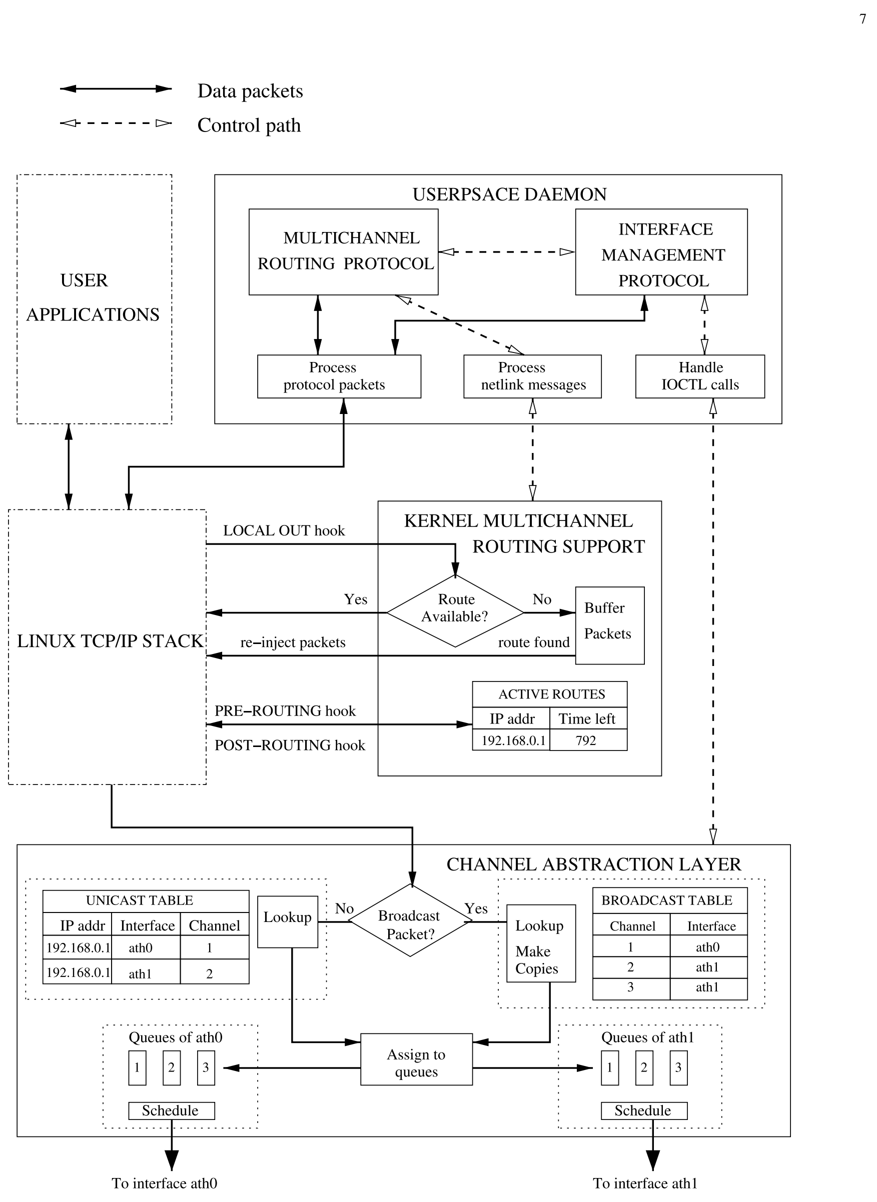


Figure 3.3: The Net-X framework as depicted in [2]

monitored. This prevents packet loss at the interfaces. If the channel is switched, the unsent packets in the interface queue are flushed. Therefore, before switching the interface to the next channel, the channel switching scheduler has to wait until the interface queue is empty. The original Madwifi device driver does not support a functionality which allows the monitoring of the queue length by higher network layers. Hence, the driver was modified to enable higher layers to supervise the queue lengths.
Section 4.3 describes how this functionality is implemented.

3.3 Kernel Multichannel Routing Support

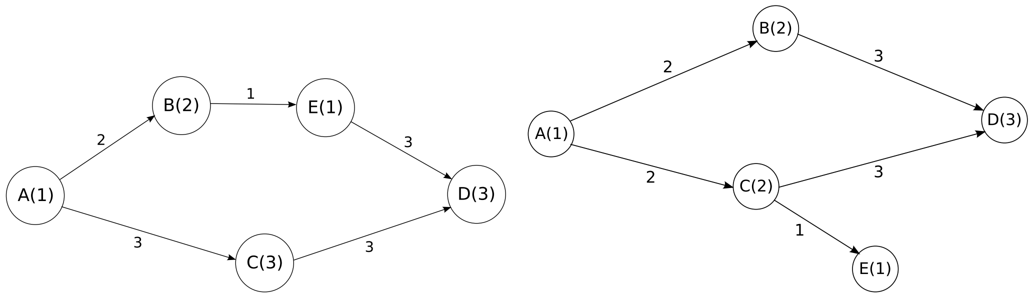
The Kernel Multichannel Routing Support (KMCR) module supports ODR in multichannel networks by keeping a table with all available routes. It is implemented as a Linux kernel module. It is responsible to keep track about the existing routes and informs the user space daemon if a new route has to be discovered.

As shown in Figure 3.3, the module contains a buffer and an $activeroutes$ table. The buffer stores packets which cannot be sent as there is not yet a route to the destination available. The $activeroutes$ table holds the IP-addresses of all nodes to which a route is active and an associated $Time\ left$ field. The $Time\ left$ field represents the time interval during which the route to that address is active. When the time has expired the table entry is deleted.

The KMCR module maintains its tables by filtering all packets that traverse the network stack. The filtering is done by the Netfilter framework [26] which is a framework inside the Linux kernel. Netfilter provides a set of hooks. With these hooks packets can be taken out of the network stack. Kernel modules can register to any of the hooks and when a packet traverses a correspondent hook, Netfilter passes this packet to the registered module. Furthermore, Netlink messages enable the communication between kernel space and user space applications.

By adding itself to some of the Netfilter hooks, the KMCR filters incoming and outgoing packets and processes the packets depending on their source and destination addresses.
On the LOCAL OUT hook the module takes out the packets that come from the node itself. KMCR checks if there is a route available to the destination address, if so, the packet is reinjected into the network stack. Else, the packet will be buffered. Via a netlink message the KMCR module informs the user space routing protocol to look for a route. When the route is found, the packet is put back into the network stack for transmission.
The PRE- and POST ROUTING hooks intercept packets that come from external nodes. If the destination address of the packet is for the node itself, it is filtered by the PRE ROUTING hook. Else, the node has to act as router and forward the packet. In this case, the POST ROUTING hook was intercepting the packet. However, the intercepted packets are used to set the LifeTime of the source of these packets to the maximum in the *active routes*-table because if a packet arrives from a source node, it means that the route is still in use.

The implementation of KMCR is based on the kernel module of AODV-UU [27].



(a) Channel Divers Routes (b) Reduction of switching delay

Figure 3.4: Multichannel Routing Metric (MCR) [3]

3.4 Integrated Protocol

The Integrated Protocol is the user space daemon of the Net-X framework. It is composed of a multichannel routing protocol which is responsible for route discovery in a multichannel network and of an interface management protocol which is concerned with the assignment of the interfaces with the appropriate channel.

3.4.1 Multichannel Routing Protocol

The common routing metric utilized by adhoc multihop wireless networks, like WMNs, is the shortest hop routing metric. This metric evaluates the path with the least hops between sender and receiver. The Net-X multichannel routing protocol utilizes a new multichannel routing metric (MCR) [3, 28]. MCR selects divers channel routes if those routes optimize the throughput. Figure 3.4(a) illustrates the characteristics of that metric. All nodes are labeled with their fixed channel and the links are labeled with the channel assigned to the switchable interface. For exemple, Node A wants to send data to node D. Let us assume that the data is sent on the shortest path, namely over node C. When node C forwards to node D the packets that it receives from node A, these packets would interfere with the subsequent packets that node A sends to node C. Therefore, either link A-C or link C-D can be active which highly decreases the throughput. In this case, selecting the path via node B and E should be preferred, as every link can send simultaneously on different channels. Thus, the end-to-end throughput can be increased, even though the path contains more hops.
Moreover, the multichannel routing protocol considers the non-negligible switching delay. An example is shown in Figure 3.4(b). The route from node A to node C is the same for A-D-C and for A-B-C. But if node B sends data to node E the interface has to be switched between channel 1 and channel 3. This yields to a switching delay which is why the route A-D-C would be preferred.
The best route is chosen by selecting the route with the lowest costs. The costs are assigned to each available route from the sending to the receiving node. Routes which use many different channels are assigned with lower costs than routes which use few channels. In the same way the

interface switching costs are assigned. For further information about the implementation of this protocol, refer to the technical report [3].

3.4.2 Interface Management Protocol

The interface management protocol implemented by [28] is a hybrid assignment approach, (see Section 2.1.4). Every device has one interface assigned to a fixed channel - *the fixed interface* - which stays on the same channel for a long interval. The other interfaces (at least one) are dynamic and switch their channel whenever required. They are called *switchable interfaces*. If node A decides to send data to node B, node A switches its switchable interface to node B's fixed channel and vice-versa. Only the fixed interface is determined to receive data.

The fixed channel of a node is the channel on which the node receives data. Given that two nodes in an interference range have their fixed interface assigned to the same channel. When packets are sent to these nodes at the same time the packets collide what yields to throughput decrease because a retransmission is necessary. Therefore, the available channels should be distributed in a way that the number of same fixed channels in an interference range is balanced.
Moreover, if a node wants to send data it has to send this data on the fixed channel of the receiver node. Therefore, it has to be informed of the fixed channel assignments of its neighbors. Like this, the switchable interface can be assigned to the channel on which it reaches the intended neighbor.

The interface assignment protocol meets these challenges as follows. Every node in the network broadcasts periodically a *Hello* message. The *Hello* message contains the fixed channel of the sender node and its direct neighbors. With this information the receiver maintains the so-called *ChannelUsageList* and the *NeighborTable*. The *ChannelUsageList* quotes to each available channel a counter which states how many nodes in the two-hop range assign the same channel to the fixed interface. The nodes consult this list at regular intervals. If their fixed channel is used by the neighbors with a certain probability they change it to the channel that is utilized the least. In the next *Hello*-message they report the change to their neighborhood.
As mentioned above, the *Hello*-message is also used to maintain the *NeighborTable*. In this table every node stores its neighbor nodes with the corresponding fixed channel. If a node decides to send data, it inquires the *NeighborTable* to know to which channel the switchable interface has to switch to.

The Multichannel Routing Protocol (Section 3.4.1) and the Interface Management Protocol (Section 3.4.2) can be exchanged by any other implementation which satisfy the above described requirements. The kernel modules CAL (Section 3.1) and KMCR (Section 3.3) are implemented to support any routing and channel assignment protocols. In this thesis, the utilized protocols are not exchanged.

Chapter 4

Implementation of the Net-X Framework for Linux Kernels with Version 2.6

This thesis examines the implementation of a multichannel multiradio prototype on a WMN. The mesh nodes used for the prototype are equipped with the secure remote management and software distribution architecture (SRM) mentioned in Section 2.4. The SRM provides an environment to develop and compile all the software which is meant to run on the mesh nodes. Moreover, a management console facilitates the network configuration of the nodes.

The Net-X kernel modules (*CAL* and *KMCR*) have to be compiled into the Linux kernel image running on the mesh nodes. The Net-X framework has been developed for kernels with version 2.4 and the mesh nodes (PCEngines WRAP platforms) are running the Linux kernel with version 2.6.22.2. Since the kernel application programming interface (API) has changed between these two kernel generations the Net-X project has to be ported to the newer version.

Moreover, the network interface driver MadWifi provided by [29] is massively out of date. To work with the multichannel multiradio protocol, the current Madwifi driver (MadWifi v0.9.4) [30] needs the following two modifications. First, the beaconing has to be disabled. If a node sends beacons, it has to switch among all channels. This leads to a switching delay. Since all nodes know to which network they belong to, beaconing is needless and can be disabled without any drawbacks.
Moreover, packets have to be queued if the interface is not yet assigned to the channel on which the packet has to leave the node. As mentioned in Section 3.2 the channel queue length of the device driver has to be monitored to avoid packet loss when the interface is switched.

The porting of the *Channel Abstraction Layer* module and the *Kernel Multichannel Routing Support* module are explained in Section 4.1 and Section 4.2, respectively. Section 4.3 describes the device driver modifications that have been done. Section 4.4 goes into the script, which loads both the *CAL* and *KMCR* modules and the interface device driver *MadWifi*. Furthermore, the start-up script assigns an IP as well as a MAC address to the interfaces and it starts the *channel assignment* (Section 3.4.2) and *routing* protocol (Section 3.4.1).

4.1 Migration of the Channel Abstraction Layer

In conventional Linux operating systems there already exists a module - the bonding driver - which provides an interface between the network layer and the device driver for managing multiple interfaces. The driver bonds interfaces together thus only one interface is visible to the network layer. Like this, network layer protocols do not have to pay attention to the existence of multiple radios and interfaces.

The developers of CAL [29] extend the bonding driver to support the bonding of multiple channels on one physical interface.
For the implementation of CAL on 2.6 series Linux kernels we ported the functionality of the original extended bonding driver for 2.4 kernel to the newer one for 2.6 kernels. We added the multichannel features from [29] to the bonding driver of the current kernel and adapted the code to the new kernel API.

The integration of the extended bonding code into the kernel tree was done by integrating the new code into the Linux network driver directory */linux/drivers/net*. Additionally, the bonding header file */linux/include/linux/if_bonding.h* has been replaced, since the original file has been augmented with multichannel skills. As we just modify the existing bonding driver, the modifications are automatically compiled into the kernel without any *Makefile* changes. The same applies for *Kconfig*-files which make the kernel option visible for the kernel configuration tool *make config*.

4.2 Migration of the Kernel Multchannel Routing Support Module

Every on-demand routing procedure needs a mechanism which invokes the discovery of a new route if an application wants to send a packet to a destination to which no route exists. Several routing implementations implemented this mechanism into the kernel, as the kernel is the only place which has access to all application packets. The Net-X developers [2] did it this way as well.
Furthermore, if there is no route available to a packets destination, this packet has to be buffered until the route is discovered. The network stack has the possibility to send this packet back to the user space for buffering or to provide an own storage. The Net-X developers [2] decided to implement this buffer directly in the network stack, i.e. kernel space.

AODV-UU from Uppsala University [27] provides the above described kernel features for monochannel communication. The developers of the Net-X project [2] adapted this implementation to work with multiple channels. We decided to extend the newest AODV-UU kernel module with the multichannel features in the same way. This way we were able to start with a basic module that is already compilable with our 2.6.22.2 kernel. The basic module has then been extended with the proposed mechanisms of the Net-X project [2].

The integration of KMCR into the Linux kernel tree is more complicated as the module does not correspond to any existing kernel module. We decided to locate it in the network driver directory

/linux/drivers/net. To guarantee the compilation of KMCR with the Linux kernel the *Makefile* of the */linux/drivers/net* directory has been extended with the following line:

```
$(CONFIG_KMCR) += kmcr/
```

Furthermore, the *Makefile* of the KMCR module itself can be simplified because the compilation configurations are already defined in the *Makefile* of the Linux directory (*/linux/*).
Moreover, the *Kconfig*-file of the network driver directory has been extended with the additional line:

```
config KMCR tristate "Kernel MultiChannel Routing Support"
```

This line enables the KMCR module as a kernel option in the kernel configuration tool. This adaptation is not mandatory for the compilation, but makes the integration of the module into the Linux kernel complete.

4.3 Network Interface Device Driver Modifications

The network device driver has to be modified as well. As mentioned above, we use a newer generation of MadWifi than the NetX developers [2]. This simplifies our work concerning the deactivation of the beaconing, but still the mechanism to monitor the queue length at the interfaces has to be implemented to support the scheduling for interface switching.

The current MadWifi driver offers an operational mode called *ahdemo* (adhoc demo). This mode operates in adhoc mode but deactivates the beaconing. Therefore, the changes in the MadWifi code which disable the beaconing mechanism are dispensable.

For the monitoring of the queue length the function *get_wireless_stats()*, provided in Linux wireless device drivers was used. The function returns a data structure with wireless specific informations. Since there is an unused field in the data structure, [2] utilized this field to inform the CAL about how many packets are still in the queue.
Again, we overtook this modification to the Madwifi driver version that we use for our implementation.

4.4 Start-up Script

The start-up script (Listing 4.1) creates the mesh node's interfaces (in our case there are two on each device), sets their IP and MAC addresses, loads the bonding and KMCR module which are necessary for wireless multichannel multiradio communication, and starts the *helloServer*, i.e. the routing and channel assignment protocol. Furthermore, it enables IP forwarding.

The script runs with two parameters. The first parameter is the mesh node's host IP address and the second defines the two last numbers of the MAC address entered in hexadecimal format.

Unlike the MadWifi driver provided by [2] (madwifi-old) the new generation of MadWifi drivers (madwifi-ng) create one or more virtual devices on the real network interface. This behavior leads to a conflict with the functioning of the bonding module and its user level control tool *ifenslave* [31] which attaches and detaches slave devices to the bonding device.

When *ifenslave* bonds interfaces together, one master device is determined and the slave devices can be attached to it. The MAC addresses of the slaves are modified to the MAC address of the master. Like this, all bonded network interfaces appear as one single common interface to incoming network data.
As the MadWifi driver creates virtual interfaces and only they are visible to *ifenslave*, *ifenslave* is incapable to change the MAC address of the physical devices. Therefore, the real MAC addresses cannot be bonded and hence, the interface bonding has no effect.

To solve this problem the MAC addresses of the physical network devices have to be set manually by using the application *macchanger*[32]. The first ten hexadecimal numbers of the address are set statically by the script and the last two are given as a parameter. Thus, maximal 256 different network interfaces can be addressed which is more than enough for our testbed.

Moreover, the script configures the network interfaces to run in *ahdemo* mode. The *ahdemo* mode is similar to *adhoc* mode, but the beaconing is disabled.

As soon as the MadWifi interfaces have been created and configured, the bonding module is loaded and the newly created bonding device (bond0) is furnished with the IP address. The address is composed of the network address determined by the script and the host address which is given as a parameter. Then, the two network interfaces (ath0 and ath1) are attached as slave devices to the master device bond0 via *ifenslave*.

Once the KMCR module is up and running, the user space application *helloServer* is started with the parameters *channelFile*, *myIp*, *broadcastIp* and *bondInterface*. *ChannelFile* takes a file as input which defines all channels that shall be used by the mesh node. *myIp*, *broadcastIp* and *bondInterface* are self-explanatory. The script daemonizes the process to let the routing and channel assignment protocol run in the background.

The mesh node is now ready to communicate in ahdemo mode on multiple channels and multiple radios.

Listing 4.1: start-up script

```
#/bin/sh
NODEID=$1
MACDIGIT=$2

PATH=$PATH:/usr/local/sbin/:/usr/local/bin
export PATH
######################################################################################################################
#                               Default Config                                                                       #
######################################################################################################################

ESSIDBACKBONE="UBernBackbone"
DATARATE="6M"
BBIF="0 1"
RECONFSTAT="100"
MAC="00:0B:6B:57:49:$MACDIGIT"
MACACC="00:0B:6B:57:49:$MACDIGIT"
TXPOWER="16dBm"
MUIC_TIME_MAX=60
MUIC_TIME_MIN=25
MUIC_CHANNEL_MODE=0
AP_MODE="false"

######################################################################################################################
#                               Startup NET-X                                                                        #
```

```
##############################################################################################
ip route del default dev eth0

echo "Starting Net-X Script on node $NODEID. $MAC"
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 0 0 0 > /proc/sys/kernel/printk


createBBIF () {
        echo "Creating backbone interface $1/$2. MAC will be $MACDIGIT"
        wlanconfig $1 destroy
        ip link set dev $2 down
        macchanger --mac=$MAC $2
        ip link set dev $2 up
        wlanconfig $1 create wlandev $2 wlanmode ahdemo
        sleep 1
        macchanger --mac=$MAC $1
        iwpriv $1 mode 1
        echo "Putting interface $1 up"
        ip link set dev $1 up
        iwconfig $1 essid $ESSIDBACKBONE rate $DATARATE txpower $TXPOWER
}



loadBonding () {
        modprobe bonding muic_time_max=$MUIC_TIME_MAX muic_time_min=$MUIC_TIME_MIN
        ip addr add 192.168.30.$NODEID/24 dev bond0
        ip link set dev bond0 up
        ifenslave bond0 ath0 ath1

        echo 1 > /proc/sys/net/ipv4/conf/bond0/accept_redirects
        echo 1 > /proc/sys/net/ipv4/conf/bond0/send_redirects
}

# Loads KMCR Module and copies channel config from nfs
loadKMCR () {
        modprobe kmcr interfaceName="bond0"
        echo "Configuration Done!!!!"
        sleep 5
        echo "Running helloServer..."
        if [ -f  /mnt/conf/channel-$NODEID.txt ]; then
                echo "Using custom channel config...";
                cp  /mnt/conf/channel-$NODEID.txt /root/channel.txt
        fi

        /root/helloServer --channelFile /root/channel.txt --broadcastIp 192.168.30.255 --myIp 192.168.30.$NODEID
             --bondInterface "bond0" --daemonize
        ip route add default dev bond0
}

# Sets some static routes for gateways
setRouting () {
        grep GATEWAY /root/channel.txt
        RETURNCODE=$?
        if [ "${RETURNCODE}" -eq "0" ]; then
                #set routing for gateway
                echo "Setting routes on the gateway"
                ip route del default dev bond0
                ip route add default gw 192.168.30.200
                iptables -t nat -A POSTROUTING -s 192.168.30.0/24 -j MASQUERADE
        fi
}

##############################################################################################
#                            Main Commands                                                   #
##############################################################################################
hostname "meshnode$NODEID"
for NIC in $BBIF; do
        createBBIF ath$NIC wifi$NIC
done

loadBonding
loadKMCR
setRouting
```

4.5 Integration in SRM

To integrate the Net-X framework into the SRM, the modified kernel modules must be compiled in the SRM development environment. This includes the CAL, the KMCR and the MadWifi driver. Furthermore, since the user space multichannel routing and interface management protocol have to be compiled against the Linux kernel this also had to be done in the development environment.

The management console stores the configurations in some special files. (The tutorial [24] describes the procedure in detail.) So, when booting the nodes, the specific configurations are loaded. We entered the start-up script which loads the multichannel specific networking modules and configures the necessary network devices into those files so that it is executed automatically at boot time.

Chapter 5

Performance Evaluation

5.1 Testbed Environment

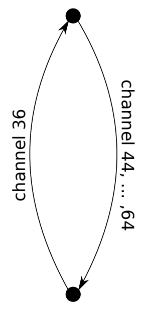
5.1.1 Hardware and Software

The multichannel multiradio network performance was measured on KAUMesh [33], a mesh testbed of the University of Karlstad, Sweden. KAUMesh's mesh nodes consist of a Cambria GW2358-4 Network Computer [34] and are equipped with three Atheros [35] compatible 802.11a/b/g miniPCI WLAN radio cards (Atheros 5212 chipset). One of the cards is running in the 802.11b frequency range to enable communication with clients and the other two cards are operating in the 802.11a frequency band to build the backbone of the WMN. In this thesis we are investigating the backbone that is why we were using only the two 802.11a interfaces.

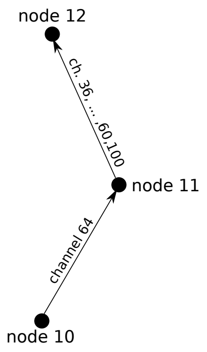
Each node is provided with the following software: The Linux kernel is with version 2.6.22.2 which allows to use our migrated Net-X framework and the integrated multichannel routing and channel assignment protocol (refer to Chapter 4). Further relevant for our purpose are the tools for performance measurements, namely the *Multi-Generator (mgen)* [36] developed by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Networking (PROTEAN) Research Group and *iperf* [37] to generate traffic and *tcpdump* [38] and *trpr* [39] to analyze the measured traffic. For time synchronization on the nodes which is required for delay measurements we used the operating system tools *ntpd* and *ntpdate*.

5.1.2 Traffic Parameters

As mentioned, we generated network traffic with *mgen*. For UDP [40] throughput measurements, we were sending a packet of 1024 KB 650 times per second. This generates traffic of 5.3 mbps, which is high enough to load the network to the maximum and determine the network throughput.
In Section 5.2 we did tests with unidirect UDP. First, we measured ACI by setting up a static multichannel multiradio network and in the second part we did throughput and dely measurements in different dynamic multichannel multiradio network scenarios. We did delay measurements



(a) Test with two nodes. Both are sender and receiver.



(b) Test with three nodes. One is sender, one is receiver and one is sender and receiver.

Figure 5.1: Channel interference measurement

with a data rate of 1.5 mbps and one with 3 mbps. These rates are according to the rate of real-time video streaming where low delay has a high priority.
In Section 5.3 we measured throughput and delay of bidirect UDP traffic. We did this measurements, as the throughput and delay measurements with TCP [41] yield to very bad results. The reasons for that are detailed in Section 5.3.

If not explicitly mentioned, all measurements have been done with a fixed transmission rate of 6 mbps and with the three channels 36, 64 and 140.

5.2 Unidirect UDP Traffic Throughput

With generated UDP traffic we first measured the impact of adjacent channel interference (ACI) on different channel pairs in a static multichannel network (refer to Section 5.2.1). Then, we enabled Net-X (dynamic multichannel network) and measured the impact of the hop distance on the traffic throughput and delay. Last, we observed the overall network throughput when neighboring nodes are sending simultaneously. Both these tests are detailed in Section 5.2.2.

5.2.1 Adjacent Channel Interference in a Static Multichannel Multiradio Network

To measure the channel interference we disabled Net-X [2] and built a static multichannel network manually. For the first test, we took two nodes and assigned one of their radios with one channel and the other with another channel. Then we simultaneously sent traffic from one node to the other and vice-versa like shown in Figure 5.1(a). Like this, both nodes are sender and receiver at the same time. Table 5.1 illustrates the average of the overall throughput we measured while one interface was constantly assigned to channel 36 and the other was tuned from channel 40 to channel 64 (all channels of the lower band) with a distance of four. Figure 5.2 visualizes the behavior of ACI.

A run was a transmission between two nodes during 60 seconds and repeated it three times per channel pair. If the two simultaneous transmissions would be undisturbed, we would expect a throughput of 4500 kbps to 5000 kbps in every direction which leads to an overall throughput of around 9000 kbps. But as you can see in Table 5.1 and Figure 5.2, the throughput we measured indicates that - depending on the used channel pair - the transmissions disturb each other. If the channel distance is less than five the throughput is much lower than what could be possible without interference. The different behaviour leads back to the fact, that the arriving signal is weaker than the sending signal and therefore, the receiving signal is more sensitive to noise.

Channel	**40**	**44**	**48**	**52**	**56**	**60**	**64**
Ch. dist.	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Run 1	365.19	4651.75	5931.41	6868.79	6596.17	9027.03	8948.47
Run 2	2397.8	4691.61	5235.36	5540.48	9674.21	8947.66	9615.92
Run 3	1078.34	5589.09	5145.21	5238.38	9717.73	9612.06	9120.63

Table 5.1: Overall throughput in kbps of mutual transmissions between two nodes

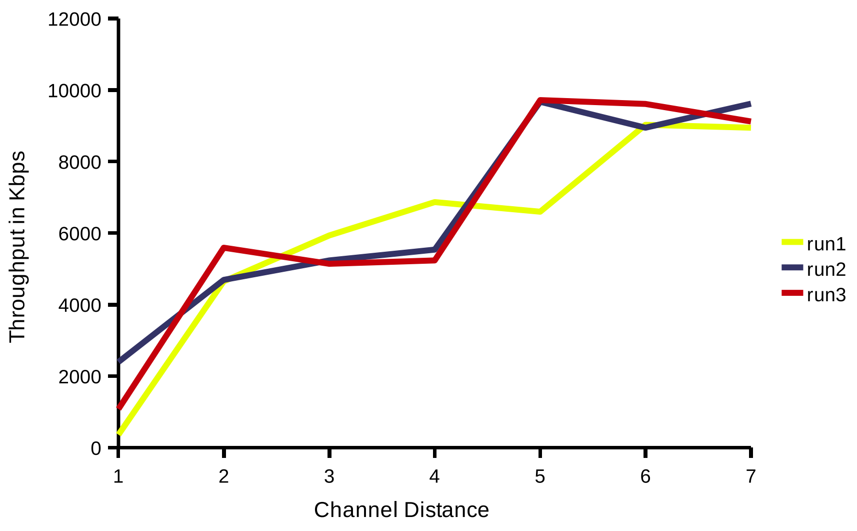


Figure 5.2: Overall throughput in kbps of mutual transmissions between two nodes

The second testing scenario (illustrated in Figure 5.1(b)) consisted of one more node, but still we tested two channels against each other. Node 10 sends to node 11 on one channel and node 11 sends to node 12 on another channel. In this case, the critical point lies at node 11 which is sender and receiver. Node 10 and node 12 are either sender or receiver. The communication of node 10 and node 11 is taking place on channel 64, the transmission from node 11 to node 12 on the channels 36 to 60 (lower band) and channel 100 (lowest channel on upper band). Table 5.2 Figure 5.3 show that at node 11 there is much disturbance if the channel distance is less than three and at node 12 the channel distance must be at least two to avoid interference.

Channel	Channel distance	Throughput from node 10 to node 11	Throughput from node 11 to node 12	Overall throughput
60	1	1011.17	3307.69	4318.85
56	2	1067.65	4921.78	5989.43
52	3	1240.2	4545.89	5786.09
48	4	4919.5	4898.95	9818.45
44	5	4974.69	4781.17	9755.87
40	6	4939.51	4938.84	9878.34
36	7	4943.4	4845.23	9788.63
100	9	4926.77	4901.47	9828.24

Table 5.2: Throughput of two simultaneous one-hop transmissions in kbps

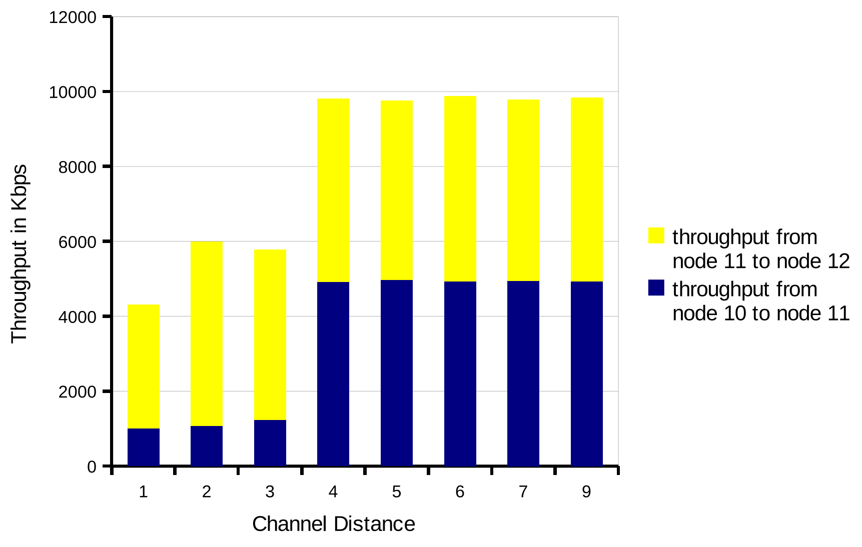
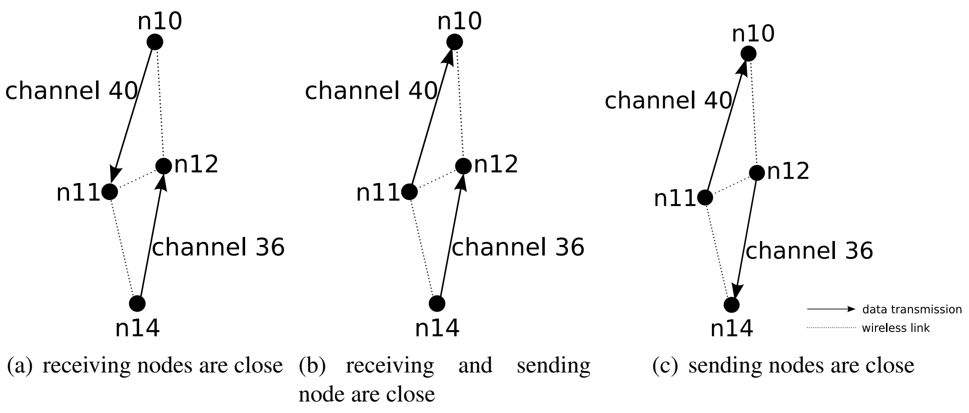


Figure 5.3: Overall throughput in kbps of two simultaneous one-hop transmissions

In the third test we took four nodes to test if adjacent channels also interfere when two simultaneous transmissions are taking place in the same transmission range and the nodes are either sender or receiver but not both. The three scenarios are illustrated in the Figures 5.4(a), 5.4(b), and 5.4(c). First, node 11 and node 12, which are positioned in the middle, acted only as receivers, in the second test one was sender and the other was receiver and in the last test they both were senders. We tested all these three scenarios, because as soon as the signal arrives at a node, the signal strength is much lower than when the signal is produced. Therefore, we would expect the highest interference rate when the senders are closest to each other. What we observed in the three scenarios is presented in the Tables 5.3, 5.4, and 5.5, respectively. Figure 5.5 visualizes the results. In every case the throughput is at the maximum for every channel pair, hence, the data flows do not interfere.



(a) receiving nodes are close (b) receiving and sending node are close (c) sending nodes are close

Figure 5.4: Different scenarios of channel interference measurement with four nodes.

We interprete these results as follows: The observations made in the first two tests lead back to ACI (see Section 2.3), where the transmission on the sending channel produces too much noise on the receiving channel and hence, disturbs the receiving data. The high rate of erroneous packet reception then lowers the throughput significantly.
The third test shows, that ACI only occurs, if sender and receiver are at the same node, or in other words, if the distance of the sending and the receiving antenna is very low (some centimeters).

The results of the measurements indicate that two neighboring nodes should assign their receiving antennas with channels that are at least four channels away from each other to avoid ACI and achieve the maximum possible throughput.

5.2.2 UDP Throughput with Dynamic Multichannel Multiradio Communication

Now, after having measured the mutual disturbance of specific channel pairs, we tested the dynamic multichannel multiradio communication with Net-X. The channels we used were channel 36, channel 64 and channel 140. To be sure, that the fixed channel assignment distribution is perfectly balanced, we assigned the fixed interfaces manually. Furthermore, in all the following tests a run lasts 60 seconds, whereas we measured the throughput every second. The measurements are presented in figures whereas the bars include 50% of all the measured values. Furthermore, the median (black horizontal line) and the maximum and minimum value of the 60 second tests are each identified.

Run	Throughput form node 10 to node 11	Throughput from node 14 to node 12	Overall throughput
Run 1	4839.99	4931.72	9771.71
Run 2	4934	4915.47	9849.47

Table 5.3: Throughput with parallel transmissions (illustrated in Figure 5.4(a))

Run	Throughput form node 11 to node 10	Throughput from node 14 to node 12	Overall throughput
Run 3	4957.91	4960.46	9918.36
Run 4	4932.39	4936.69	9869.08

Table 5.4: Throughput with parallel transmissions (illustrated in Figure 5.4(b))

Run	Throughput form node 11 to node 10	Throughput from node 12 to node 14	Overall throughput
Run 5	4887.94	4918.42	9806.36
Run 6	4918.42	4799.03	9717.46

Table 5.5: Throughput in kbps with parallel transmissions (illustrated in Figure 5.4(c))

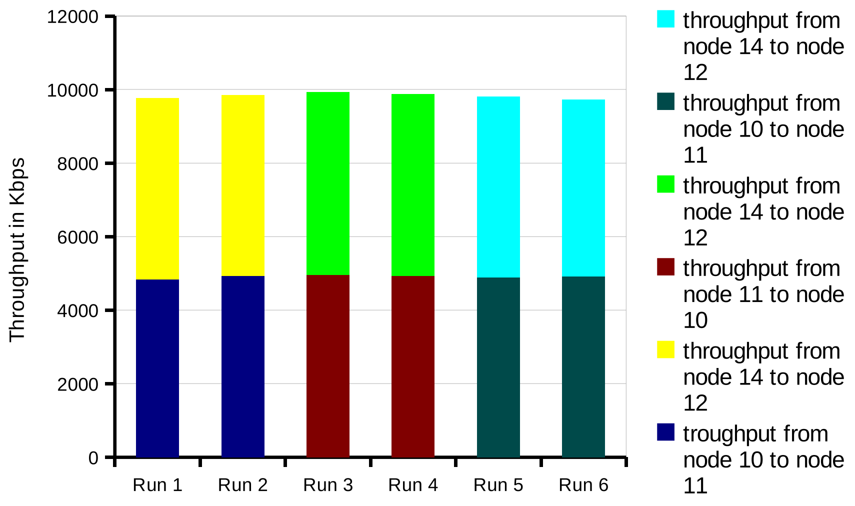


Figure 5.5: Throughput in kbps with parallel transmissions

Impact of the Hop Distance on the Throughput and the Delay

In this scenario we measured how the hop distance impacts the throughput. As you can see in Figure 5.6, we sent data from node 7 via node 10, node 11, node 14 to node 18 and measured the throughput in each case. As shown in Table 5.6 and Figure 5.7(a), we measured an average throughput of 4.9 mbps at node 10, 4.7 mbps at node 11 and 4.1 mbps at node 14. Since the sending and receiving radio of the forwarding links do not disturb each others communication, the data can be forwarded in parallel as new data is received. This yields to the high throughput even at node 14 were two hops are between source and destination. The small throughput decrease with the increase of hops can be explained with the fact, that the node's CPU can not send and receive simultaneously. This weakly slows down the sending process.
At the fourth hop (node 18), the throughput was barely 2.7 mbps in average, which leads back to the weak link and the high packet loss between node 14 and node 18. We measured a packet loss of around 0.5 % between node 10 and node 11 and between node 11 and node 14, whereas the loss between node 14 and node 18 was around 16 % in average.
Table 5.7 and Figure 5.7(b) show the throughput we measured in the same scenario but with monochannel communication. The Table and the Figure clearly illustrate that the throughput decreases rapidly when the hop distance increases since two neighboring nodes can not send simultaneously. With the same scenario we have measured the delay in a multichannel and in

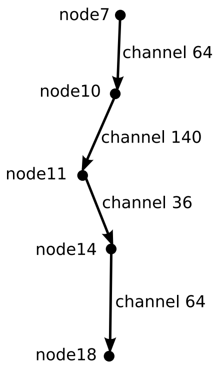


Figure 5.6: Measuring throughput after different number of hops

a monochannel network. As mentioned before, the delay measurements were performed with a data stream of 1.5 mbps and of 3 mbps. The measurements are illustrated in Table 5.8 and Figure 5.8(a), and in Table 5.9 and Figure 5.8(b) respectively. The results show a correlation between the throughput and the delay. When we compare the throughput and the delay on a certain route we notice that the routes where the throughput measurements have shown a capacity higher than the data stream we were sending for the delay tests (1.5 mbps and 3 mbps), the delay was tiny. For exemple, the throughput at node 18 was around 2.7 mbps. For the 1.5 mbps data stream even at the forth hop the delay is only 1 millisecond. But if we stream 3 mbps the data arrives at node 18 200 milliseconds delayed.
The measurements done in the monochannel network are recorded in Table 5.10 and Figure 5.9(a) for the 1.5 mbps data stream, and in Table 5.11 and Figure 5.9(b) for the 3 mbps

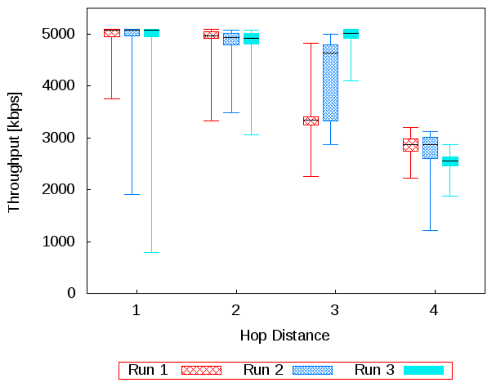
data stream. As the results show, the delay increases rapidly even for the low data stream and the network throughput decreases immensely with every further hop.

Hop distance	Run 1	Run 2	Run 3	Average throughput
1	4888.475	4815.016	4934.892	4879.461
2	4847.112	4792.051	4785.202	4808.122
3	3613.343	3957.541	4772.176	4144.353
4	2771.984	2731.293	2649.239	2717.505

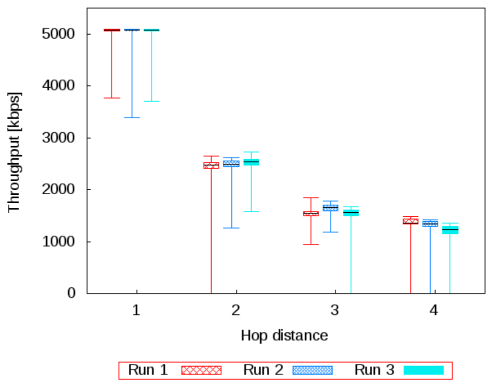
Table 5.6: Impact of the hop distance on the throughput (in kbps) in a multichannel network

Hop distance	Run 1	Run 2	Run 3	Average throughput
1	4962.875	4965.695	4968.649	4965.739
2	2287.582	2452.899	2507.021	2415.834
3	1523.041	1633.621	1519.817	1558.826
4	1370.707	1322.083	1241.220	1311.337

Table 5.7: Impact of the hop distance on the throughput (in kbps) in a monochannel network



(a) Multichannel network



(b) Monochannel network

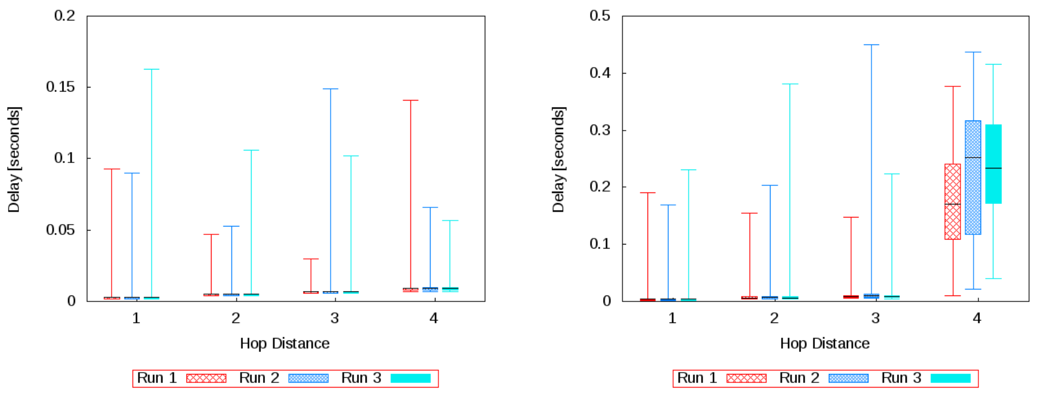
Figure 5.7: Throughput after a certain hop distance measured in kbps

Hop distance	**Run 1**	**Run 2**	**Run 3**	**Average delay**
1	0.00331	0.00329	0.00362	0.00340
2	0.00564	0.00541	0.00550	0.00517
3	0.00743	0.00782	0.00756	0.00760
4	0.01022	0.01045	0.00996	0.01021

Table 5.8: Impact of the hop distance on the delay (in seconds) in a multichannel network with a 1500 kbps data stream

Hop distance	**Run 1**	**Run 2**	**Run 3**	**Average delay**
1	0.00484	0.00482	0.00499	0.00488
2	0.00711	0.00767	0.00821	0.00766
3	0.01011	0.01209	0.00948	0.01056
4	0.18028	0.20980	0.23587	0.20865

Table 5.9: Impact of the hop distance on the delay (in seconds) in a multichannel network with a 3000 kbps data stream



(a) 1500 kbps data stream

(b) 3000 kbps data stream

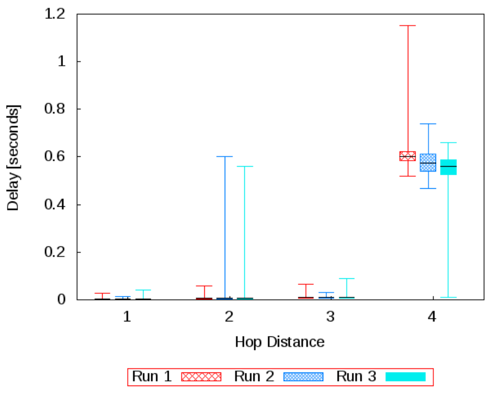
Figure 5.8: Impact of the hop distance on the delay (in seconds) in a multichannel network

Hop distance	**Run 1**	**Run 2**	**Run 3**	**Average delay**
1	0.00264	0.00397	0.00266	0.00309
2	0.00584	0.00854	0.00720	0.00719
3	0.00937	0.00919	0.00932	0.00930
4	0.56691	0.52833	0.52353	0.53959

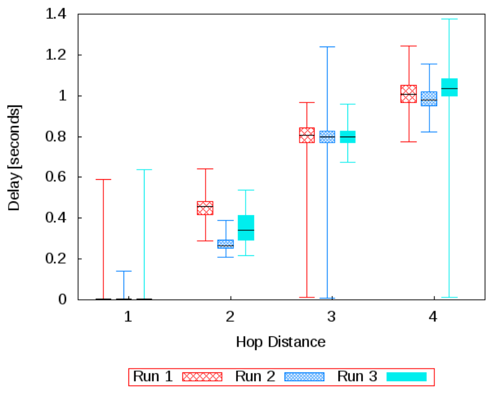
Table 5.10: Impact of the hop distance on the delay (in seconds) in a monochannel network with a 1500 kbps data stream

Hop distance	Run 1	Run 2	Run 3	Average delay
1	0.00465	0.00399	0.00472	0.00445
2	0.41071	0.28674	0.37070	0.35605
3	0.78643	0.78589	0.76491	0.77908
4	0.97042	0.93678	0.99289	0.96670

Table 5.11: Impact of the hop distance on the delay (in seconds) in a monochannel network with a 3000 kbps data stream



(a) 1500 kbps data stream

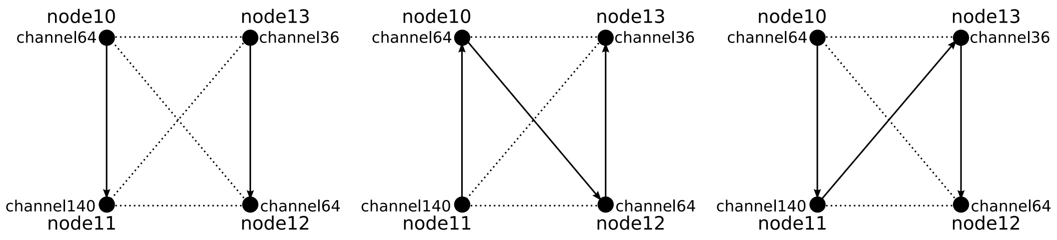


(b) 3000 kbps data stream

Figure 5.9: Impact of the hop distance on the delay (in seconds) in a monochannel network

Simultaneous Transmissions

For these tests, we used four nodes being all direct neighbors to each other. The fixed interfaces of the nodes are assigned as follows: Node 10 with channel 64, node 11 with channel 140, node 13 with channel 36 and node 12 with channel 64.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

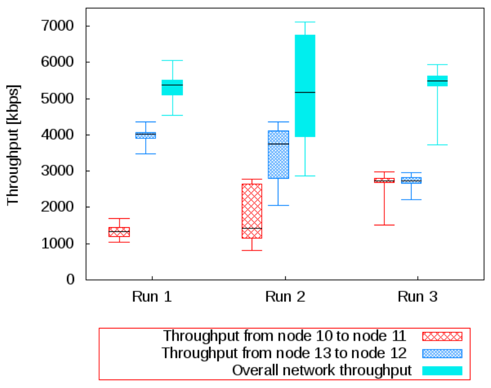
Figure 5.10: Simultaneous Transmission Scenarios

Scenario 1 is illustrated in Figure 5.10(a). This test measured the throughput of two simultaneous transmissions, whereas two nodes only sent and the two other nodes only received data. Node 10 sent data to node 11 on channel 140 and node 13 to node 12 on channel 64. In monochannel networks we measured low performance because the two data flows interfered and a high amount of retransmissions was necessary. The average throughput of three runs in a monochannel network are shown in Table 5.12 and in Figure 5.11(a).
The multichannel multiradio WMN testbed furnished the performance shown in Figure 5.11(b). As the Figure shows, the throughput from node 10 to node 11 and from node 13 to node 12 is 5 mbps in average, which makes a total network throughput of 10 mbps. Considering these results, we can conclude that the throughput is almost doubled by using different channels for sending two streams inside one interference range.

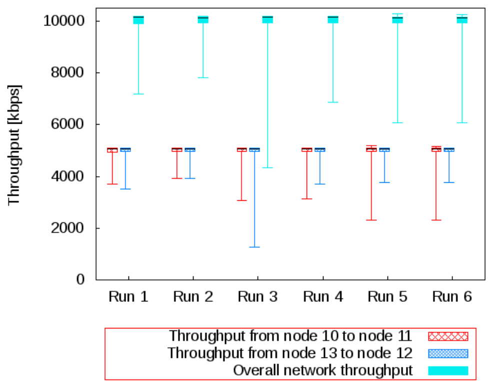
Run	Node 10 to node 11	Node 13 to node 12	Overall throughput
1	1400.4291	3968.9568	5369.3859
2	1927.1344	3435.9396	5363.0740
3	2708.7318	2757.2123	5465.9441

Table 5.12: Average throughput of two simultaneous transmissions in a monochannel network

In scenario 2, node 11 transmits to node 10, node 10 transmits to node 12 and node 12 transmits to node 13. The scenario is illustrated in Figure 5.10(b). Here, node 10 and node 12 act both as sender and receiver whereas node 11 only sends and node 13 only receives.
The results presented in Figure 5.12(b) show that the throughput from node 11 to node 10 and from node 12 to node 13 is more or less at the maximum. But from node 10 to node 12 it is very poor.
The explanation for the bad throughput performance is the fact, that node 10 sends and receives on the same channel. The channel assignment protocol uses the receiving interface for sending if the fixed interface of the receiver is assigned with the same channel. Therefore, node 10 only



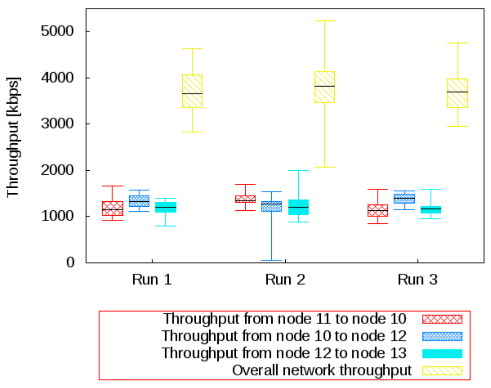
(a) Monochannel network



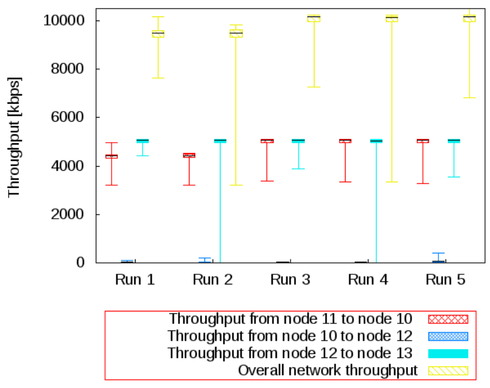
(b) Multichannel network

Figure 5.11: The throughput of two simultaneous transmissions as shown in Figure 5.10(a).

sends data to node 12 if the receiving interface is idle. In our tests this is rarely the case and causes the poor throughput. Even though channel 64 is used twice, the traffic between node 11 and node 10 is very high. This yields back to the fact that almost no traffic is sent from node 10 to node 12 and hence, the disturbance of the transmission is rather low. Furthermore, the amount of data arriving at node 13 is extremely high because the CPU at node 12 has only few receiving data to process and therefore can focus on handling the sending data.
Even though the throughput from node 10 to node 12 is poor, the overall network throughput still is higher than what we observed in a monochannel WMN. The measurements in a monochannel WMN are presented in Figure 5.12(a).



(a) Monochannel network

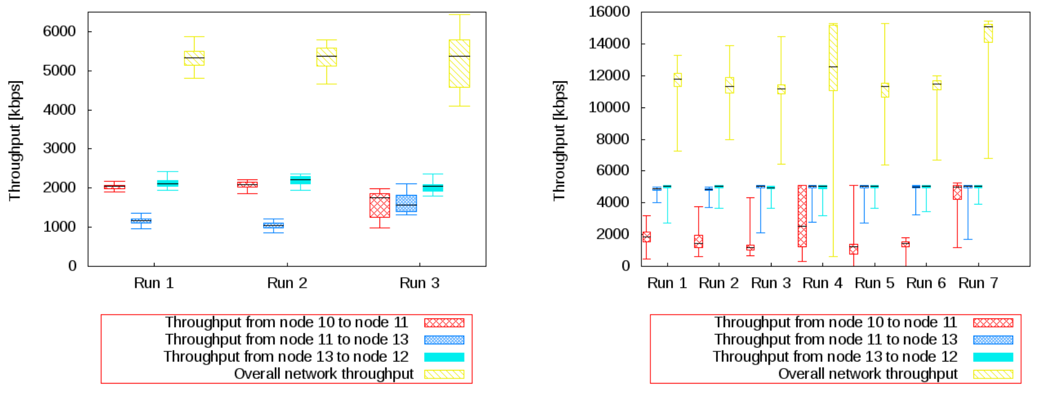


(b) Multichannel network

Figure 5.12: The throughput of two simultaneous transmissions as shown in Figure 5.10(b).

In scenario 3 (Figure 5.10(c)) every receiving interface which was involved was tuned to another channel. Node 10 transmits to node 11 on channel 140, node 11 transmits to node 13 on

channel 36, and node 13 transmits to node 12 on channel 64. The measured throughput through every link is illustrated in Figure 5.13(a) for monochannel transmissions and in Figure 5.13(b) for multichannel multiradio communication.
As expected, the monochannel network shows a rather low performance since the nodes cannot transmit simultaneously. The links have an average capacity of 1000 kbps to 2000 kbps which yields to an overall network throughput of 5000 kbps to 6000 kbps.
The tests using multiple channels and multiple radios for the transmissions resulted in an average overall network throughput of 11000 kbps to 15000 kbps which is twice to three times the capacity of what we measured in the monochannel network.



(a) Monochannel network (b) Multichannel network

Figure 5.13: The throughput of two simultaneous transmissions as shown in Figure 5.10(c).

Run	Minimum	Maximum	Average
1	0.371467	0.894118	0.647547
2	0.230769	0.893506	0.694007
3	0.086626	0.919854	0.754167
4	0	0.908023	0.420562
5	0	0.935950	0.720660
6	0.636651	0.909050	0.742478
7	0	0.735572	0.151972

Table 5.13: Packet loss rate

But Figure 5.13(b) exposes that the throughput through the link from node 10 to node 11 varies from 1000 kbps to 5000 kbps. To find the reason for the low capacity, we first analyzed the packet loss on every run as presented in Table 5.13. The extremely high rate of packet loss infers to data traffic disturbance on this link. Since in scenario 1 we already measured the network capacity when node 10 and node 13 send to node 11 and node 12, respectively, we know that these two links do not interfere with each other. The throughput that we measured was as high as we expected it to be.

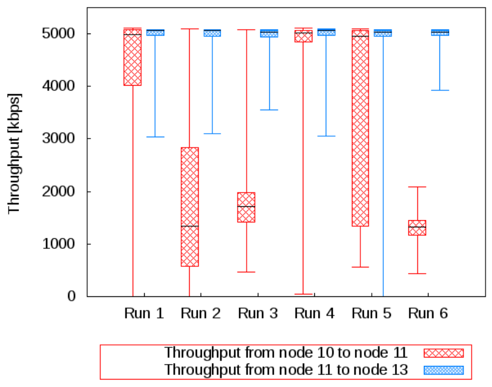


Figure 5.14: The throughput of two simultaneous transmissions in a multichannel network.

Therefore, we studied how the throughput from node 10 to node 11 is affected, when node 11 sends to node 13 in parallel. Figure 5.14 presents the results and it shows that the throughput is decreased. Accordingly, the link from node 11 to node 13 appears to be the reason for the low throughput and the high packet loss rate at the other link. There is evidence to suggest that this performance is a matter of ACI because only the receiving data is decreased. As explained earlier the receiving signal is much weaker than the sending signal. But as one transmission takes place on channel 36 and the other on channel 140 the channel distance is too high for ACI. Therefore, the reason for the weak capacity must be caused by something else like e.g. the performance of the CPU of node 11, incorrect channel switching at node 10, or signal interference at the network card (cross-boarding talking). Whatever, the overall network capacity in the multichannel multiradio network is still at a much higher level than in the monochannel network.

5.3 Bidirectional UDP Traffic

Measuring the network throughput with TCP traffic [41] in our multichannel testbed yields to strictly low results. The reason for this behavior lies in the congestion control mechanism of TCP [42]. The TCP congestion control reacts sensitively to packet loss. As soon as a packet gets lost, the congestion window is set to the minimum size. Step by step the window increases until the next transmission failure occurs. As shown in Figure 5.15, the bonding driver holds queues with unlimited length for every channel while the queue length of the network devices is limited. The network device drops incoming packets, as soon as the queue is full.
Given that the queues of the network interfaces can hold at most 50 packets, that ath0 is set to channel 1 and that the CAL queue of channel 2 stores 80 packets. As soon as ath0 switches from channel 1 to channel 2, CAL flushes the queue for channel 2 to ath0. But the queue of the network driver is only able to store 50 packets. Therefore, some of the flushed packets are dropped by the driver. However, TCP reacts on the packet loss and decreases the congestion window even though the network would have had the capacity to handle the current window

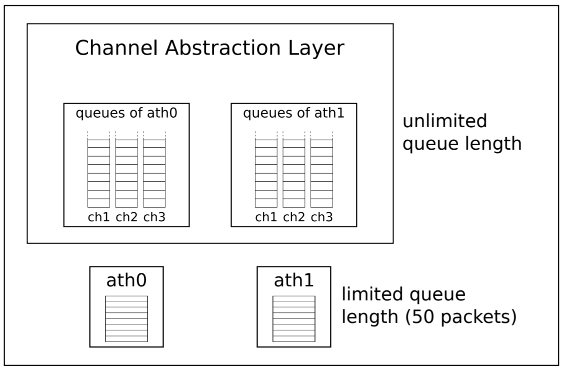


Figure 5.15: Limited queue length of network devices lowers the TCP network throughput

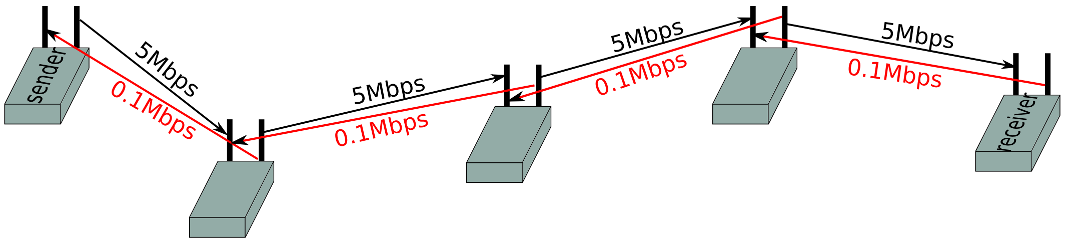


Figure 5.16: Bidirectional UDP traffic

size. In Chapter 6 a possible solution to this problem is presented.
Packets are also dropped by the network interface, when the channel is switched and there are still packets in the buffer. Net-X has implemented a mechanism to defer this kind of packet flush by monitoring the device driver's queue length (refer to Section 3.2 and Section 4.3. But the deferral often does not suffice to save all the packets. This scenario again yields to packet loss and hence to the decrease of the TCP congestion window.

These issues do not allow a correct throughput measurements of the network itself. The network capacity does not get fully exploited because the TCP congestion window is lower than the network would allow. However, we measured bidirect UDP traffic where the sending node sends a high amount of data to the destination node and the receiving node sends back low traffic to feign the control traffic (Figure 5.16).
This scheme resembles TCP traffic as in a TCP connection the receiver periodically sends control messages to keep up the connection and to signal to the sender that packets have arrived correctly. This yields to a bidirectional transmission where the control messages produce low traffic in one direction and the data transmission produces a high amount of traffic in the other direction. But the scheme neither covers the behaviour of TCP when retransmissions occur nor TCP's congestion control.

With the appliance of this method we measured the similar throughput and delay as we did in the measurements without simulating control messages.
Figure 5.17 presents the throughput that we measured with the bidirectional UDP traffic. It

shows that although there is traffic in both directions (data and control traffic) the data throughput is as high as what we measured with unidirectional UDP where we have much less control messages. This is possible because the control messages which are sent from the receiver to the sender are transmitted on another channel than the data and hence the data and control flow do not interfere.
Figure 5.18 shows the delay that we measured over multiple hops bidirectional UDP traffic. Figure 5.18(a) presents the delay that results from a 1.5 mbps data stream and Figure 5.18(b) presents the delay that results from a 3 mbps data stream. Again, the weak link between node 14 and node 18 decreases the throughput and consequently also increases the delay. So the delay is approximately the same as with unidirectional UDP traffic.
But

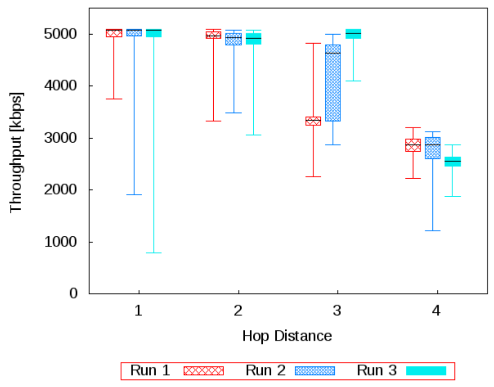
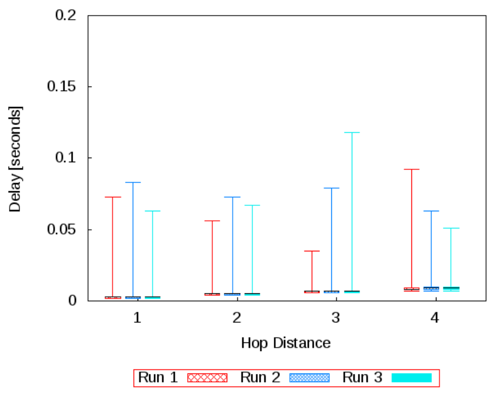
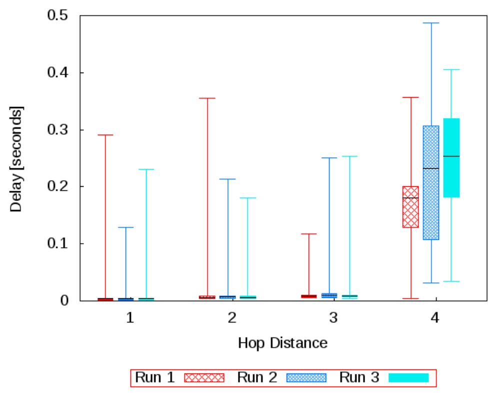


Figure 5.17: Throughput measurements with bidirectional UDP



(a) 1500 kbps data stream



(b) 3000 kbps data stream

Figure 5.18: Impact of the hop distance on the delay (in seconds) in a multichannel network using bidirectional UDP

Chapter 6

Conclusions and Future Work

In this thesis we could demonstrate how multichannel multiradio communication can lower the interference rate in a wireless mesh network and increase the overall network throughput compared to the throughput in a common monochannel WMN. However, to enable multichannel routing protocols we had to migrate the Net-X framework from Linux kernel with versions 2.4 to our Linux kernel with version 2.6.22.2. The Net-X framework provides the necessary operation system support for multichannel routing.
The UDP transmission measurements have shown that ACI may occur if neighboring nodes have their fixed channels assigned with a low distance. But even if simultaneous transitions disturb each other due to ACI, the overall network throughput is higher than in monochannel networks. If the channel distance is high enough the network throughput can be increased significantly by using multiple channels.

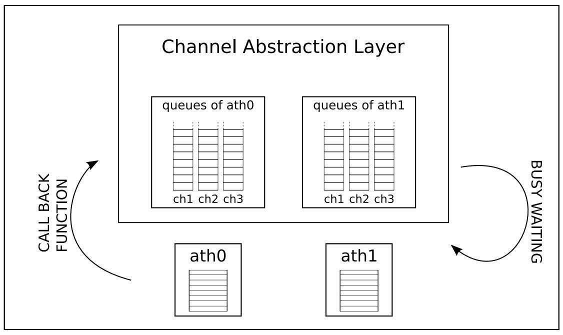


Figure 6.1: Proposed solutions to reduce packet loss when switching the channel

However, our multichannel multiradio implementation could still be optimized. As explained in Section 5.3, TCP traffic does not reach the maximum possible throughput. Packets are getting lost when the channel is switched. One reason for that is that CAL flushes the packets which are in its queue to the relatively small buffer of the network interface controller (NIC) as soon as the channel is switched. If the queue is too big for the buffer of the NIC packets get lost which induces TCP to reduce the transmission rate. For future work this problem could be solved by

introducing a callback function in the NIC which informs the CAL how many packets it may send.
The queue control could also be implemented as a busy waiting loop in the CAL. The busy waiting loop periodically checks if there is free space in the buffer of the NIC. Like this, CAL only sends packets if the buffer of the NIC can hold them. But a busy waiting loop utilizes a lot of CPU capacity. Figure 6 illustrates the two mentioned approaches.

Furthermore, as WMNs are well suitable for connecting mobile wireless devices like mobile phones, personal digital assistants, digital radios, etc. which mainly are applied for real time applications like IP telephony, video streaming, or radio broadcasts, WMNs should support these applications. Real time applications require low delay transmissions else, the received audio or video signal is stuttering and that makes the application unusable. Therefore, WMNs should privilege these flows over other flows to minimize the delay.

Glossary

ABR	Associativity Based Routing
ACI	Adjacent Channel Interference
AODV	Adhoc On-demand Distance Vector Routing
AODV-UU	AODV implementation from Uppsala University
API	Application Programming Interface
CAL	Channel Abstraction Layer
CHAT	CHAM with Packet Train
CHMA	Common Hopping Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Network Control Message (Clear-to-Send)
DCA	Dynamic Channel Allocation
DSDV	Destination Sequence Distance Vector
DSR	Dynamic Source Routing
DYMO	Dynamic MANET On-demand Routing
FCL	Free Channel List
HRMA	Hop Reservation Multiple Access
KAU	Karlstads Universitet
Kbps	Kilo bits per second
KMCR	Kernel Multichannel Routing Support
MAC	Media Access Control
MANET	Mobile Adhoc Network
MAP	Multichannel Access Protocol
Mbps	Mega bits per second
MCR	Multichannel Routing Metric
MMAC	Multichannel MAC Protocol
NIC	Network Interface Controller
ODR	On-Demand Routing
OLSR	Optimized Link State Routing
RTS	Network Control Message (Request-to-Send)
SNR	Signal to Noise Ratio
SRM	Secure Remote Management and Software Distribution for WMN

TCP	Transmission Control Protocol
VoIP	Voice over IP
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network

Bibliography

[1] J. Mo, H.-S. Wilson So, and J. Walrand, "Comparison of multichannel mac protocols," *IEEE Transactions on Mobile Computing*, vol. 7, no. 1, pp. 50–65, 2008.

[2] P. Kyasanur, C. Chereddi, and N. H. Vaidya, "Net-x: System extensions for supporting multiple channels, multiple interfaces, and other interface capabilities," Wireless Networking Group, University of Illinois at Urbana-Champaign," Technical Report, 2006. [Online]. Available: https://www-2.crhc.uiuc.edu:443/wireless/netxform.html

[3] P. Kyasanur and N. H. Vaidya, "Routing in multi-channel multi-interface ad hoc wireless networks," University of Illinois at Urbana-Champaign, Tech. Rep., 2004.

[4] C.-M. Cheng, P.-H. Hsiao, H. T. Kung, and D. Vlah, "Wsn07-1: Adjacent channel interference in dual-radio 802.11a nodes and its impact on multi-hop networking," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, 2006, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/GLOCOM.2006.500

[5] S. Corson and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," RFC 2501 (Informational), Internet Engineering Task Force, Jan. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2501.txt

[6] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445–487, March 2005.

[7] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *ACM SIGCOMM*, 1994, pp. 234–244.

[8] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[9] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," RFC 4728 (Experimental), Internet Engineering Task Force, Feb. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4728.txt

[10] I. Chakeres and C. Perkins, "Dynamic manet on-demand (dymo) routing." IETF Draft, March 2009, work in progress.

[11] E. M. Royer and C.-K. Toh, “A review of current routing protocols for ad hoc mobile wireless networks.” IEEE Personal Communications, April 1999.

[12] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” RFC 3561 (Experimental), Internet Engineering Task Force, July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[13] A. Colvin, “Csma with collision avoidance,” *Computer Communications*, vol. 6, no. 5, pp. 227 – 235, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYP-48TD57H-NS/2/60a05963e2834a5d0e8f7e6d5fb8ca84

[14] J. Chen, S. Sheu, and C. Yang, “A new multichannel access protocol for ieee 802.11 ad hoc wireless lans,” *In Proc. of PIMRC*, vol. 3, pp. 2291–2296, 2003.

[15] J. So and N. H. Vaidya, “Multi-channel mac for ad hoc networks: handling multi-channel hidden terminals using a single transceiver,” in *MobiHoc 2004: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*. New York, NY, USA: ACM, 2004, pp. 222–233.

[16] Z. Tang and J. J. Garcia-Luna-Aceves, “Hop-reservation multiple access (hrma) for ad-hoc networks,” in *In IEEE Infocom*, 1999, pp. 194–201.

[17] A. Tzamaloukas and J. J. Garcia-Luna-Aceves, “Channel-hopping multiple access,” in *Proc. IEEE ICC 2000*, 2000, pp. 415–419.

[18] A. Tzamaloukas and J. Garcia-Luna-Aceves, “Channel hopping multiple access with packet trains for ad hoc networks,” in *Proc. IEEE Mobile Multimedia Communications (MoMuC 2000)*, Tokyo, 2000.

[19] S.-L. Wu, C.-Y. Lin, Y.-C. Tseng, and J.-P. Sheu, “A new multi-channel mac protocol with on-demand channel assignment for multi-hop mobile ad hoc networks,” *Parallel Architectures, Algorithms, and Networks, International Symposium on*, vol. 0, p. 232, 2000.

[20] S.-L. Wu, Y.-C. Tseng, C.-Y. Lin, and J.-P. Sheu, “A multi-channel mac protocol with power control for multi-hop mobile ad hoc networks,” *The Computer Journal*, vol. 45, no. 1, pp. 101–110, January 2002. [Online]. Available: http://dx.doi.org/10.1093/comjnl/45.1.101

[21] C. Chereddi, P. Kyasanur, and N. H. Vaidya, “Net-x: a multichannel multi-interface wireless mesh implementation,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 11, no. 3, pp. 84–95, 2007.

[22] A. Baiocchi, A. Todini, and A. Valletta, “Why a multichannel protocol can boost ieee 802.11 performance,” in *MSWiM ’04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM Press, 2004, pp. 143–148. [Online]. Available: http://dx.doi.org/10.1145/1023663.1023689

[23] V. Raman and N. Vaidya, "Adjacent channel interference reduction in multichannel wireless networks using intelligent channel allocation," University of Illinois at Urbana-Champaign, Tech. Rep., March 2009.

[24] T. Staub, D. Balsiger, M. Lustenberger, and T. Braun, "Secure remote management and software distribution for wireless mesh networks," in *7th International Workshop on Applications and Services in Wireless Networks*. Santander, Spain: ASWN 2007, May 24–26 2007, pp. 47–54.

[25] M. Satyanarayanan and D. A. Maltz, "On-demand routing in multi-hop wireless mobile ad hoc networks," Tech. Rep., 2001.

[26] "Netfilter: firewalling, nat, and packet mangling for linux." [Online]. Available: http://www.netfilter.org/

[27] E. Nordström, "Ad-hoc on-demand distance vector routing, from uppsala university."

[28] P. Kyasanur and N. H. Vaidya, "Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 10, no. 1, pp. 31–43, 2006.

[29] C. Chereddi, P. Kyasanur, and N. H. Vaidya, "Design and implementation of a multi-channel multi-interface network," in *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*. New York, NY, USA: ACM, 2006, pp. 23–30.

[30] "Multiband atheros driver for wireless fidelity (madwifi)." [Online]. Available: http://madwifi-project.org/

[31] D. Becker and et al., "ifenslave," linux-2.6.22.2/Documentation/Networking/ifenslave.c. [Online]. Available: www.kernel.org

[32] A. L. Ortega, "Gnu macchanger." [Online]. Available: http://www.alobbs.com/macchanger

[33] P. Dely and A. Kassler, "Kaumesh demo." Proceedings of 9th Scandinavian Workshop on Wireless Ad-hoc and Sensor Networks, 2009.

[34] "Cambria gw2358-4 network computer." [Online]. Available: http://www.gateworks.com/products/cambria/gw2358-4.php

[35] "Atheros." [Online]. Available: http://www.atheros.com/

[36] P. E. A. N. P. R. Group, "Multi-generator." [Online]. Available: http://cs.itd.nrl.navy.mil/work/mgen/index.php

[37] "Iperf." [Online]. Available: http://sourceforge.net/projects/iperf/

[38] "Tcpdump." [Online]. Available: http://www.tcpdump.org/

[39] "Trpr." [Online]. Available: http://www.trpr.org/

[40] J. Postel, "User Datagram Protocol," RFC 768 (Standard), Internet Engineering Task Force, Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[41] ——, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sept. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[42] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr. 1999, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt