TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

Design and Verification of an IEEE 802.11e EDCF Simulation Model in ns-2.26

Sven Wiethölter, Christian Hoene

{wiethoel,hoene}@tkn.tu-berlin.de

Berlin, November 2003

TKN Technical Report TKN-03-19

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Contents

List of Figures

List of Tables

Abstract

In my work, I have extended the Network Simulator ns-2 (ns-2.26) to support the Enhanced Distributed Coordination Function (EDCF) combined with Contention Free Bursting as it is described in the drafts of IEEE 802.11e. The Network Simulator ns-2 is frequently used for simulations of the wireless LAN IEEE 802.11 protocol. About half of the publications are based on results which ns-2 has produced. Nevertheless, I identified and removed a couple of bugs in the ns-2 802.11 DCF model. Furthermore, I verified my EDCF extension with the IEEE 802.11e simulation model of Mangold *et al.* [6]. Mangold used a WARP simulator which differs fundamentally from ns-2. However, both simulation models yield similar results for the same simulation scenarios.

My EDCF extension is open source and has been published on the ns-2 mailing list so that other researchers benefit from this modular simulation model.

Chapter 1

Basic Overview

1.1 Introduction

1.1.1 Motivation

Mobile communication devices like Laptops and PDAs become more and more popular. For easy communication between these devices as well as the connection to the Internet, Wireless LAN (IEEE 802.11) is used in a lot of scenarios today. Especially the number of WLANs in public facilities like railway stations, official buildings and airports increases rapidly, not taking into account all the small private "home" WLANs.

The increase in popularity of Wireless LANs led to more closely considerations with respect to multimedia traffic over WLANs in the past. The most sensitive case of multimedia traffic is the Internet telephony (Voice-over-IP). In particular the delay is most critical in Voice-over-IP (VoIP) applications. For a sufficient speech quality the mouth-to-ear delay must be kept small and should be released of jitter at the receiver. While the absence of a small jitter can be realized by introducing a playout buffer at the receiver, the avoidance of a high jitter / delay is much more complex: for achieving a small and constant delay the lag of time in the layers must be kept small. Especially retransmissions and contention-based medium access schemes are accountable for high delays and jitters.

In order to enable several types of flows over one wireless medium, several MAC-layer solutions have been developed:

- legacy IEEE 802.11
- IEEE 802.11e (the enhanced version of 802.11)
- Packet Frame Grouping (PFG)

1.1.2 Task

The task of this work was the modeling of the IEEE 802.11e EDCF draft combined with PFG in Network Simulator version 2.26. To ensure a draft-conform behavior, the sim-

ulation model should be tested and validated by comparing my results with a reference paper [6].

1.1.3 Overview

In the rest of chapter 1, I explain the different MAC-layer solutions IEEE 802.11 DCF and PCF, the extensions RTS/CTS and fragmentation, the IEEE 802.11e EDCF and HCF as well as PFG. In the 802.11e section, I give an overview of the differences with respect to the EDCF between several drafts [9, 11, 10].

In chapter 2, I first point out the important inaccuracies of the existing ns-2.26 802.11 DCF model. Secondly, I explain the important steps of my EDCF model by considering my backoff timer solution.

The verification of my 802.11e EDCF model is described in chapter 3. There I compare the behavior of my MAC with the reference results [6].

1.2 MAC-Layer Protocols

1.2.1 legacy 802.11

In 802.11 two basic functions exist for the medium access: the Point Coordination Function (PCF) and the Distributed Coordination Function (DCF). While the DCF is responsible for asynchronous data services, the PCF was developed for time-bounded services. The PCF is used in the contention-free period (CFP), while the DCF handles the contention period (CP). One CFP and one CP are combined to a superframe. Superframes are separated by periodic management frames, the so-called Beacon frames.

802.11 uses three different inter-packet gaps, denoted as interframe spaces, to control the medium access, i.e. to give stations in specific cases a higher or lower priority (see figure 1.1):

- short interframe space (SIFS)
- PCF interframe space (PIFS)
- DCF interframe space (DIFS)

SIFS is the shortest interframe space and is used for acknowledgments (ACKs), CTS (clear-to-send) frames and several following MPDUs of a fragment burst as well as for the response of a polled station in the PCF. SIFS personates the highest priority and assures that a station is able to finish a frame-exchange sequence before other stations are in the position to gain access to the medium.

In the CFP the point coordinator (PC) polls stations and must have therefore a prioritized access to the medium. This is realized by PIFS which is longer than SIFS but smaller than DIFS.

The DIFS is used in the CP and describes the duration of time in which the medium has to be idle before a station is allowed to send or decrement its backoff. DIFS is the longest interframe space and consequently has the lowest priority.

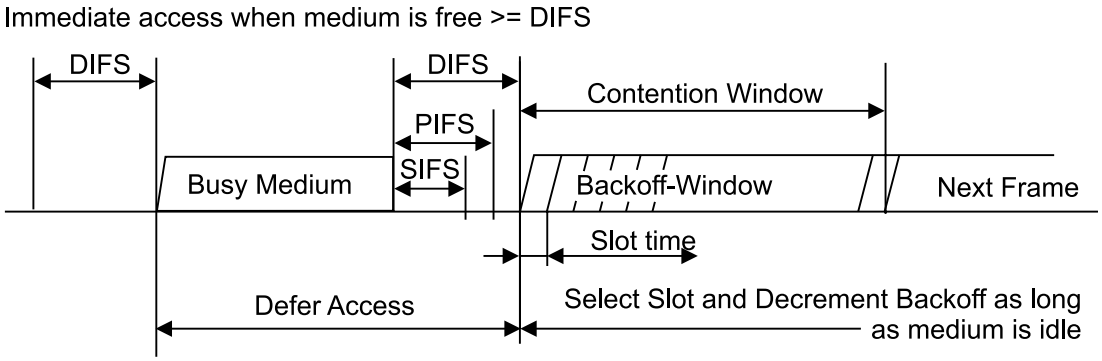

Figure 1.1: IFS relationships in IEEE802.11 (source: [3])

Distributed Coordination Function (DCF)

In the DCF, sending stations stand in contention to each other. They use the medium access scheme carrier sense multiple access with collision avoidance (CSMA/CA). Before sending a frame the station has to sense the channel. If the medium is at least idle for DIFS the station is allowed send. In the case of a busy medium the station starts the random backoff procedure. It determines a slotted random *backoff time=slotTime*Random*. *Random* is a pseudo-random integer value out of the uniformly distributed contention window [0, CW] with $CW_{min} \leq$CW≤ 255. Initially CW=CW_{min} is set to 7 in 802.11. If the medium is idle again at least for DIFS, it decrements the random backoff time as long as the medium is idle. The station is allowed to send immediately if the random backoff time equals zero.

The random backoff procedure has to be started after every transmission. In the case of a successful acknowledged transmission the procedure will be started after the received ACK. Otherwise the procedure will be started after the expiration of the ACK timeout interval.

A collision occurs if two (or more) stations have detected the medium as idle for DIFS, both are allowed to send and both start their transmissions immediately. This affects the random backoff procedure after the expiration of the ACK-timeout interval, too. To resolve repeating collisions increasing $CW_{min}s$ have to be chosen. For the first up to the fourth retransmission the CW_{min}-value has to be set to $CW_{min,new}$=2*$CW_{min,old}+1$ ($\Rightarrow$ 15, 31, 63, 127). For five or more retransmissions the CW_{min}-value has to be set to CW_{max}=255.

Point Coordination Function (PCF)

The PCF can only be used in an infrastructure-based network because it requires an access point (AP). Usually the Point Coordinator (PC) is installed on this AP. The PC manages the access to the medium in the CFP by polling stations. The CFP, as the first part of a superframe, is being started periodically by a beacon frame sent by the PC after a PIFS-idle medium. Therefore the CFP may be delayed due to a long frame sent in the end of the CP. Because the PCF was developed on top of the DCF all stations have to

set their NAV in the beginning of the CFP to the CFPMaxDuration value. After sending a beacon frame the PC has to wait at least SIFS before it can send a poll or a data frame (or both piggybacked).

The PC polls all stations sequentially in the CFP. A polled station is allowed to answer with a data / ACK frame after SIFS to the PC or to any other station in the network. If a polled station does not answer, the PC polls the next one after PIFS. If neither the PC nor the stations have frames to send, the CFP ends with a CFP-end frame which is sent by the PC. All receiving stations reset their NAV and the CP begins.

If the PCF is used for the transmission of time-bounded data the PC should support a polling list. Every station listed there should be polled at least once per CFP. Station are able to request for a place in the polling list with association management frames. Due to its complexity as well as its overhead the PCF is implemented but mostly not used in current installed WLANs.

extension 1: RTS/CTS

A crucial problem in the CP is the hidden terminal problem which appears when a station is not able to listen to the full communication sequence of a pair of communicating stations and thus believes the channel as idle. A collision occurs if this station attempts to send data. To prevent collisions as a result of the hidden terminal problem additional control packets were introduced to the DCF: The request to send (RTS) packet is sent in the beginning of the transmission (after the channel was idle for at least DIFS) by the sender and contains a duration field. This states the duration for CTS, Data and ACK packet (inclusive all inter framespaces). All receiving stations set their net allocation

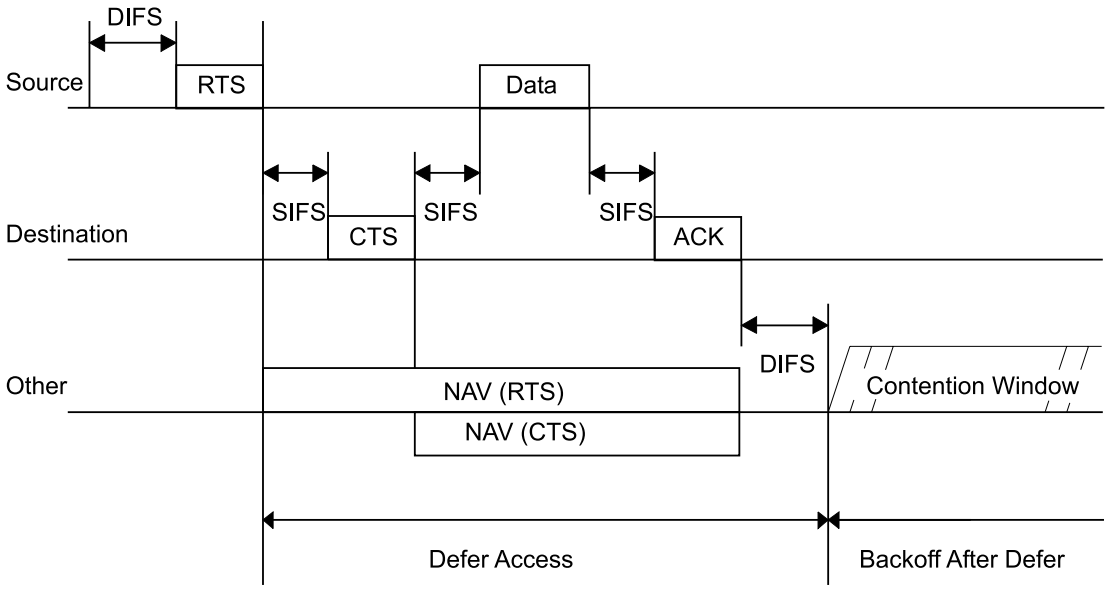

Figure 1.2: RTS/CTS in 802.11-DCF (source:[3])

vector (NAV) to this value (figure 1.2). The NAV describes the point the of time until

the medium will be busy. The receiver answers the RTS with the clear to send (CTS) packet after SIFS containing a duration field too with the time interval for data, ACK and interfame spaces (IFS). Every station which receives the CTS sets its NAV to the duration field. With this RTS/CTS mechanism all stations in the coverage of sender and receiver are informed about the duration of this transmission and should not disturb. As the RTS packet can be sent at least after a DIFS-idle medium the RTS/CTS mechanism does not prioritize stations. As collisions can only occur with a RTS-packet the mechanism is an excellent collision protection for big frames. The main disadvantage is the overhead of RTS/CTS which leads to a waste of bandwidth and a higher delay.

extension 2: fragmentation mode

Because of the high bit error rates (BER) in Wireless LAN due to interference, smaller frames have a higher probability of being transmitted without any errors than bigger frames. Utilizing this, the fragmentation mode was added to the 802.11 MAC mechanism (figure 1.3). Big frames are divided into smaller fragments when they cut across the fragmentation threshold. Each fragment is transmitted and acknowledged separately. After a station has the right to access the medium, it is allowed to send several DATA/ACK sequences separated by SIFS. Therefore, no other station is able to disturb the communication sequence and a contention for each fragment is prevented.

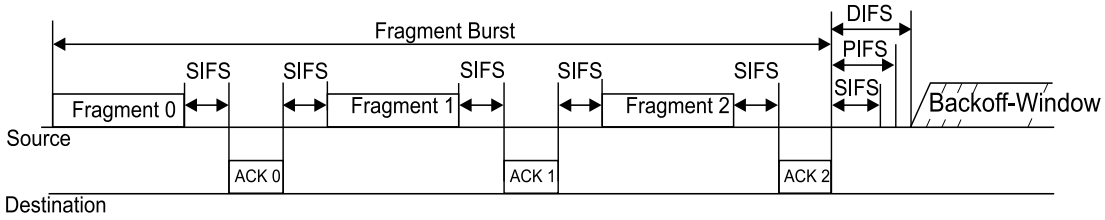

Figure 1.3: fragmentation in 802.11 (source:[3])

1.2.2 802.11e

The 802.11 extension was initiated due to the problems of the PCF with time-bounded traffic. The PCF is still susceptible to unpredictable delays because

- a wireless station may transmit a frame of arbitrary size (up to 2312 bytes) in the CFP,
- the beginning of the next CFP may be delayed due to a busy medium at TBTT (the point of time, at which the PC starts a new CFP by sending a Beacon frame). This occurs if a transmission of a long frame was started at the end of the preceding CP,
- VoIP and all other traffic is stored in one queue and is handled therefore in the same manner according to the medium access.

To solve these problems, the medium access schemes of legacy 802.11 are enhanced to the upcoming future standard 802.11e. For achieving QoS, 802.11e uses multiple queues for the prioritized and separate handling of different traffic categories (TCs). In addition, 802.11e introduces the Enhanced Distributed Coordination Function (EDCF) and the Hybrid Coordination Function (HCF). The EDCF manages the medium access in the CP while the HCF is responsible for the CFP and the CP. Both functions are described below. In the beginning I had to base my studies over 802.11e on the paper Mangold *et. al.* [6] because current 802.11e drafts can only be looked in by members of this group. In the course of my work I have found the November 2001 draft [9] on the Internet. In addition, I got a copy of the May 2003 version [11] and the current 2003 draft version 5 [10]. [10] introduces several changes in the nomenclature (8 TCs in EDCF $\Rightarrow$ 4 access categories (AC), EDCF $\Rightarrow$ enhanced distributed channel access (EDCA), HCF $\Rightarrow$ HCF controlled channel access (HCCA)). [9] uses eight traffic priorities (with own queues) which are specified in IEEE 802.1D [2]. In [11, 10], only four queues are used due to the mapping of the eight priorities to four ACs. I decided to keep the old notation in order to avoid unnecessary confusion. Who can be sure that the next draft does not come up with new notation again?

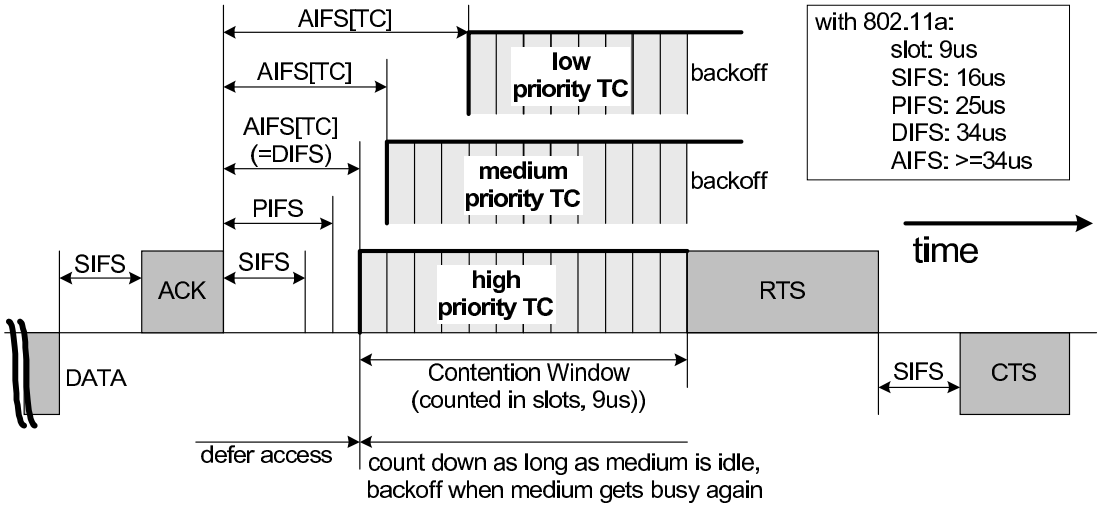

Figure 1.4: IFS relationships in 802.11e (source [6])

EDCF

The EDCF enhances the 802.11 DCF by introducing an own backoff instance with a separate backoff parameter set for each priority queue. Each TC on a station contends for a transmission opportunity (TXOP). A TXOP is defined in [6] as *"an interval of time when a station has the right to initiate transmissions, defined by a starting time and the maximum duration"*. Other parameters for prioritization of TCs are the arbitration interframe space (AIFS), which is at least DIFS long, and the minimum size of the CW

($CW_{min}[TC]$) (see figure 1.4). With $CW_{min}[TC] < 7$ enhanced stations have a higher priority than legacy 802.11 stations (with CW_{min}=7).

The determination scheme of the backoff time for each TC is similar to the legacy DCF backoff procedure. The CW is enlarged after an unsuccessful transmission to

$$newCW[TC] = [(oldCW[TC] + 1) * PF] - 1. \tag{1.1}$$

The persistence factor (PF) ϵ [1,16] is another prioritization parameter, which was included only in [9]. The CW will be doubled as in the DCF with a PF of two (binary exponential backoff). [11, 10] use only the binary exponential backoff an therefore do not need a PF anymore.

As in DCF, the backoff counter is stopped if the medium is sensed to be busy and is decremented if the medium is idle for at least AIFS.

Different TCs on one station with their own set of parameters (AIFS, CW, PF) and their own backoff instance are shown in figure 1.5. Up to eight queues [9], resp. four queues [11, 10] can be set up per station. This may lead to a collision if two TCs are allowed to send at the same time. This collision is solved by a virtual scheduler which grants access to the TC with the highest priority.

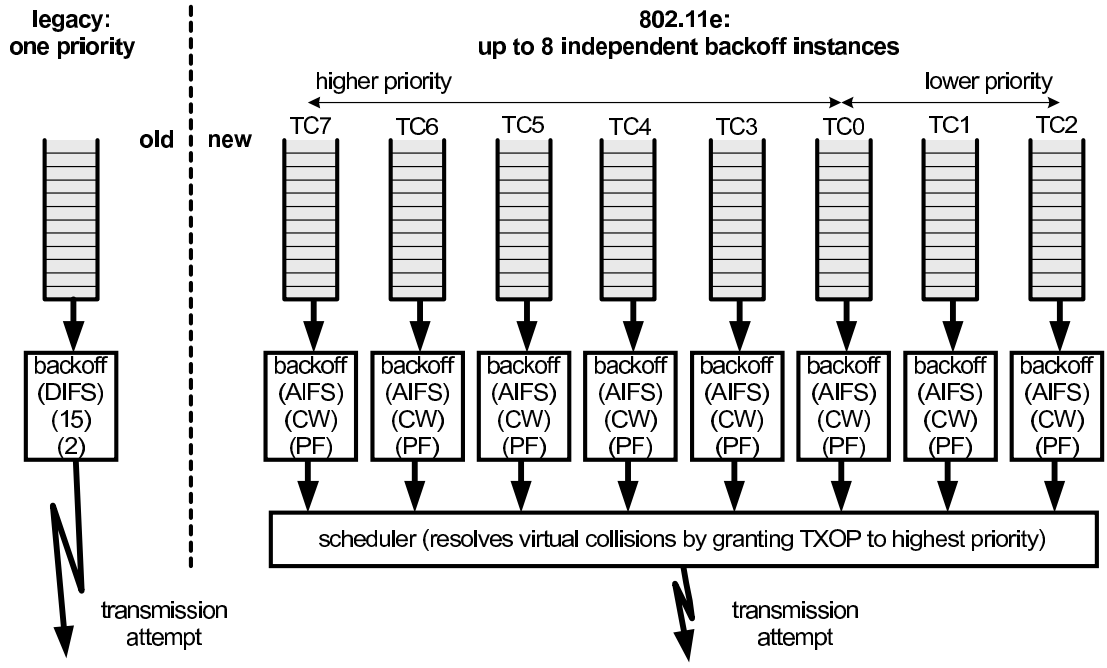

Figure 1.5: IEEE802.11e MAC structure (source [6])

HCF

The HCF is the basis for the EDCF and controls both the CFP and CP. Therefore one enhanced station must be responsible for the management of the medium access. This

station is denoted as Hybrid Coordinator (HC). The HC is normally colocated to the AP. It is able to allocate TXOPs to itself after an idle medium of PIFS. This leads to priority over other enhanced stations which can access the medium not until AIFS$\geq$DIFS$>$PIFS.

Two TXOPs exist in the HCF: the EDCF-TXOP for contention and the polled-TXOP. EDCF-TXOPs can be obtained only in the CP while polled-TXOPs can be allocated by the HC in the whole superframe. Medium access can be distributed both in the CP according to the EDCF rules (EDCF-TXOPs) or by polling of the HC. Therefore the HC can take control by sending a QoS CF-poll frame after a PIFS-idle medium (without any backoff).

The TXOPs in the CFP are allocated by the HC with CF-poll frames which include the starting point of time and the maximum duration. A CF-end frame of the HC or the point of time announced in the beacon frame leads to the end of CFP.

1.2.3 Packet Frame Grouping (PFG)

PFG was introduced in [8] to improve the performance for small packets (of time-bounded services) in Wireless LANs by decreasing overhead and delay and by increasing the throughput. PFG basically uses the idea of a fragment burst (of 802.11 DCF) where a station sends small fragments of a large MSDU as a burst if it gains access to the medium (see figure 1.6). With PFG, not fragments of one large MSDU but a series of small MSDUs are transmitted in a burst. It is possible to send packets to different destinations in one burst frame. Between an ACK and the following packet only a time interval of SIFS is required. Therefore the station keeps control over the medium for the whole burst. Sending multiple small packets in a burst avoids contention for each single packet. This results in a higher efficiency and lower delay according to [8].

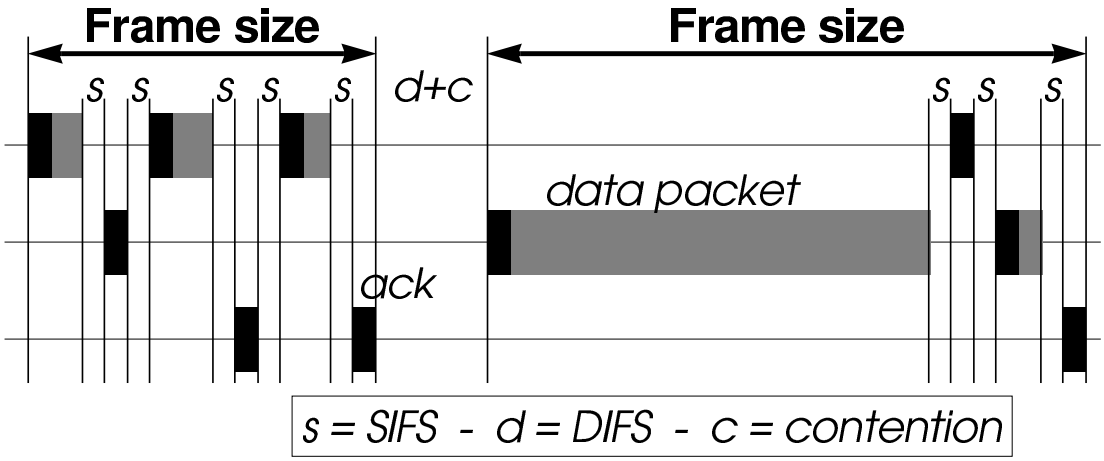

Figure 1.6: principle of PFG (source: [8])

As described in [8] WLAN is not fair to small packets in terms of their size. Fairness in 802.11 DCF is achieved by contention with respect to packets, i.e. each station has

the same probability to send a packet. Stations which send only relative small packets are treated unfair in contrary to stations which send mostly large frames.

By introducing a maximum frame size in bytes for a fragment burst not only the risk of high delay but also the unfairness is reduced according to Tourrilhes [8]. He sets this frame size to 2000 Bytes because he wants to be able to transmit a larger packet in combination with a small one in one burst frame to decrease the delay.

From my point of view, it is not very useful to define the maximum duration as a number of bytes: How to calculate SIFS in a number of bytes? Should we take therefore the lower PLCPRate or the higher data rate into account? If we send several small packets, we get an ACK for each. These ACKs are sent with the PLCPRate. But if we are transmitting only one big packet without PFG we are using the higher data rate. So the number of bytes does not give us the amount of time in which the medium is being captured by one station. To solve this, I am going to implement a time-dependent counter and transmission limit.

In the 802.11e drafts [9] and [11] PFG is denoted as Contention Free Burst (CFB). Today, CFB is also referred to as TXOP Bursting. The latest draft [10] contains the CFB mechanism in "Obtaining a Continuation of EDCA TXOP". It proposes fragmentation/defragmentation only for single MSDUs.

Chapter 2

Simulation Model

2.1 Network Simulator ns-2.26

Ns-2 is a discrete event simulator which supports wired and wireless networking protocols. The simulator is an open source project, so the code as well as some patches are available on the Internet at [1].

The basic structure of ns-2 and the networking protocols are realized in the programming language C++. The script language Tcl is used for easy control and assembly of new simulations.

The idea of a discrete event simulator is that actions may only be started as a result of further events or inputs. Therefore ns-2 consists of a scheduler and a scheduling list. Each event has to be inserted into the scheduling list together with its expiration date. The scheduler goes through the scheduling list at runtime and starts the actions which are associated with the expired date.

2.2 Existing IEEE 802.11 MAC in ns-2.26

The ns-2 MAC simulation model can be found in the directory *../ns-2.26/mac/*.

The LLC hands packets to the MAC through a priority interface queue. It is an advanced drop-tail queue which facilitates the insertion of routing packets at its head. The MAC itself consists of the IEEE 802.11 DCF but not of the PCF. There is a PCF-patch available on the Internet, but I have never tested it.

Beacons and superframe structures are not really included due to the lack of the PCF. In ns-2 beacons are only realized by routing update messages which have intervals in a range of several seconds, so the notation "beacon" may be a little far-fetched.

The DCF MAC protocol is RTS/CTS/DATA/ACK and broadcast capable. It is able to scan the medium by virtual and physical carrier sense. The MAC provides the interframe spaces SIFS, PIFS, DIFS as well as EIFS. The EIFS is applied after every detected unsuccessful transmission attempt. It assures that a station may be able to answer with an ACK.

The ns MAC includes several timers:

- defer timer: is used when the MAC has to sense the medium being idle for the period of DIFS or if the MAC has to wait a period of SIFS
- backoff timer: counts down the residual time of a backoff
- interface timer: indicates, how long the interface will be in transmit mode when sending a packet
- send timer: is used for the indication of the time up to which an ACK should be received after a transmission attempt
- nav timer: is started
  - for EIFS if a collision has been detected
  - for the period contained in the duration field of a successful received data frame
  - for the duration of a RTS/CTS/DATA/ACK exchange

These timers have start, stop, pause, resume and handle methods which are implemented in *../ns-2.26/mac/mac-timers.cc/h*. The event for the corresponding point of time is added to the scheduling list in the start method (resp. in *resume()* after a timer was paused and should be restarted again). Descheduling of events is realized in *pause()* and *stop()*. If a timer expires, its *handle()* is called where all following actions are defined.

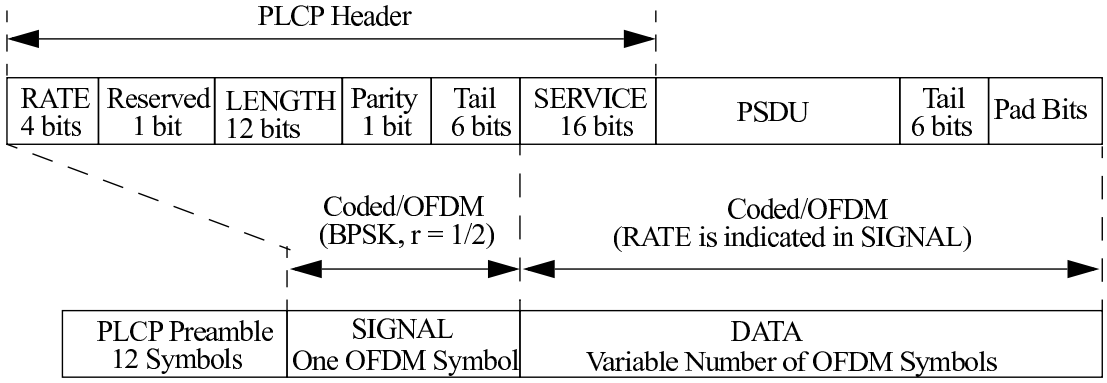

Figure 2.1: Physical Layer frame of 802.11a (source [4])

Several inaccuracies, which result in non-standard-conform behavior, were found in the 802.11 implementation:

- deferrals: please assume a station A that wants to transmit a packet. Station A has to sense the medium at first. The defer timer will be started for a period of DIFS if the medium is determined to be idle. Let us assume that the medium is idle and station A starts the defer timer. If another station B afterwards begins

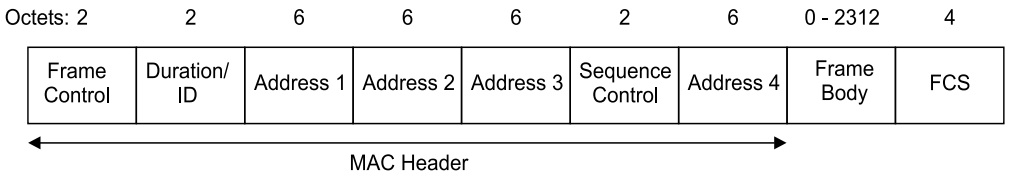

Figure 2.2: MAC Sublayer frame of 802.11 (source [3])

a transmission, station A should stop its defer timer and should start a backoff due to the busy medium. In ns-2, the defer timer won't be stopped. It will expire and a backoff would only be started, when the medium is still busy. This is the case for a low PLCP rate of 1 Mbps. If one uses a different PHY (e.g. IEEE 802.11a) with a basic rate of 6Mbps and a data rate of up to 54 Mbps, the defer timer may expire when the medium is physically and virtually idle again because DIFS is longer than the frame. This usually is the case if station B sent a very short broadcast packet which would not take an ACK after SIFS. In the following a short example with a payload of 5Bytes and a data-rate of 24Mbps is given. The PHY-802.11a frame is shown in figure 2.1 . The PLCP preamble (with a duration of 16μs) and reserved, length, parity and tail bits are transmitted at 6Mbps, while service, PSDU, tail and pad bits are transmitted with 24Mbps. The 802.11 MAC frame is pictured in 2.2. With these values the following generation time can be computed:

$$duration = 16\mu s + \frac{24bit}{6Mbps} + \frac{16bit + 34bit + 40bit + 6bit}{24Mbps} = 24\mu s \quad (2.1)$$

$$802.11aDIFS = SIFS + 2 * slotTime = 16\mu s + 2 * 9\mu s = 34\mu s > duration \quad (2.2)$$

- backoff: if a backoff is started and the medium is idle (usually this is the case after a successful DATA/ACK exchange, denoted as post-backoff), the backoff is scheduled only with its residual time not taking the period of DIFS into account. From my point of view, the medium must be idle for DIFS also for a post-backoff before the residual time can be decremented.

- calculation of a backoff: the backoff time is calculated in the ns-802.11 model by *(Random::random() % cw) * SlotTime*. Due to the modulo operation the backoff time will always be smaller than *cw * SlotTime*. Therefore I suggest to replace *cw* by *cw + 1*.

- NAV/collision handling:

  – after a collision was detected, the NAV should be set to *EIFS = SIFS + ACKSize + PreambleLength + PLCPHeaderLength + DIFS*. In this time the medium is "virtually" busy. If the EIFS is over, a station may start a transmission or the decrease of a backoff immediately (according to part

9.2.5.1 Basic access in [3]). In ns-2 stations wait an additional period of DIFS. Therefore I suggest to exclude DIFS from the ns-2 EIFS computation.

– please assume that the NAV was set to EIFS due to a collision on the medium. According to part *9.2.3.4 Extended IFS* of [3] the NAV should be reset if an error-free frame will be received in this EIFS. Resetting the NAV is not included in ns-2 802.11.

- no separate SIFS timer: as explained in the first item, DIFS may be longer than the transmission duration of one frame for high transmission rates. If a station started a defer timer before, but currently has to send an ACK after SIFS due to the reception of a data frame, an error occurs in the scheduler because the same timer must not be scheduled twice in ns-2.

2.3 802.11e EDCF Model

For ns-2 there was an 802.11e EDCF + HCF model together with the November-2001 802.11e draft available on the Internet [9], when I started my work. This solution was directly built into a daily snapshot of ns-2.1b6a, which is not available anymore. Since I was asked for a modular and easy adaptable EDCF model, I could not use this solution due to significant changes between the ns versions. In addition, there was nearly no documentation included and I was afraid that it would take more time to understand this model than to write a new one by myself. Therefore I decided to implement my own 802.11e solution based on the ns-802.11 MAC. Some ideas, especially the realization of the queues, was taken from the model on [9], because it provides a nice and elegant way for the queueing.

Since the proceedings of [6], significant changes have been made to the IEEE 802.11e draft. We had the possibility to get a copy of the current draft [10]. As described in 1.2.2, only four priority queues are used today. The backoff parameters for each instance have been changed, too. The persistence factor PF is not included anymore. Because I started my programming part a little earlier I still have the PFs. They can simply be neutralized by setting them to a value of two, what results in a binary exponential backoff. The recommended AIFS values have been changed significantly between the November-2001 [9] and the May-2003 draft [11]. While in [9] the lowest AIFS is equal to DIFS, in [11] it can take a value equal to SIFS plus one slottime. This was changed to give QoS stations prioritized access to the medium prior to legacy 802.11 stations. In [10], it was changed back to a lowest AIFS equal to DIFS.

The simulation model described below can be found in the directory *../ns-2.26/mac/mac-802.11e* (after you have installed the patch) Please take a look at the *README* file or at the appendix A on page 33 for all necessary changes to a legacy ns version.

To enable 802.11e simulations the following parameters must be inserted in your Tcl-simulation script:

```
set opt(mac)            Mac/802_11e
set opt(ifq)            Queue/DTail/PriQ
```

2.3.1 Queueing

As indicated in figure 1.5 on page 7, an own queue for each TC is needed. The priority of each flow is given by the *prio_ field* of the *hdr_ip* structure in each packet header. An instance above the queues must decide which packet has to be inserted into which queue dependent on the *prio_ field*. This instance is derived from the ns class *Priqueue* and is denote as *priq*. Basically *priq* extends the functionality of a drop-tail queue. I introduced not only one but four drop-tail queues beneath *priq*, one for each priority. Therefore I had to derive my own drop-tails, too. The source code for the queueing can be found in the files *priq.cc/h, d-tail.cc/h*.

The parameters for each backoff instance are bounded to the corresponding drop-tail queue. These backoff parameters are specified in the file priority.tcl as shown below:

```
proc priority { ifq_name } {
   upvar $ifq_name ifq

    #parameters for Queue 0
          $ifq Prio 0 PF 2
          $ifq Prio 0 AIFS 2
      $ifq Prio 0 CW_MIN 7
      $ifq Prio 0 CW_MAX 7
      $ifq TXOPLimit 0 0.003

    #parameters for Queue 1
      $ifq Prio 1 PF 2
      $ifq Prio 1 AIFS 4
      $ifq Prio 1 CW_MIN 10
      $ifq Prio 1 CW_MAX 31
      $ifq Prio 1 TXOPLimit 0.006
        ...
}
```

The Tcl procedure *priority{}* must be called when the interface queues are created. In legacy ns, the creation of queues is done in *../ns-2.26/tcl/lib/ns-mobilenode.tcl*. For creating a modular solution, I had to derive my own *ns-mobilenode_802_11e.tcl*, which is included in *../ns-2.26/mac/802.11e/*. The PF, AIFS, CW_MIN and CW_MAX TXOPLimit values are assigned to each queue via the methods *setPF() / setAIFS() / setCW_MIN() / setCW_MAX() / setTXOPLimit()*. In the beginning of a simulation the MAC will request these parameters for each priority, therefore the get-methods were created in d-tail.cc.

In my model, queue 0 and its parameters represents the highest priority.

2.3.2 MAC

As mentioned before, I used the ns 802.11 model as a basis for my work. I had to change it into a multidimensional MAC, i.e. the handling of all four priorities must be possible at the same time. Therefore I transformed all necessary transmission states, packet buffers, parameters and timers into arrays (or arrays of pointers). I changed the MAC with respect to several non-standard-conform inaccuracies, which I have described in section 2.2. In my solution the defer (DIFS) timer is stopped when the medium becomes busy. For SIFS I have introduced a separate timer, the NAV/EIFS is reset if a correct frame will be received after a collision and the backoff is calculated from the interval [0,CW].

2.3.3 MAC Timers

The coding of the timers was a little complicated due to the concurrent handling of different priority flows. The question is: how to handle several calls of the same timer object, while in ns-2 only one event can be added to the scheduling list? Of course it is possible to create a timer object for each backoff instance, but this isn't an elegant way and it would introduce a lot of more complexity to the virtual scheduler. ns itself does not know concurrency because the ns scheduler runs through the scheduling list and works off all events in sequence, i.e. a first come first served behavior. To overcome this, a virtual scheduler below the MAC must wait until one point of time ends. This requires a rescheduling of the virtual scheduler itself with zero delay for being the last one in the scheduling list. The disadvantage of this solution is, that the MAC and / or the medium may be in a different state when the virtual scheduler is called up again at the same time. Therefore one should change the whole ns 802.11 MAC model.

From my point of view, it is a little easier to handle the concurrency in the methods of one timer object. One can simply save the start, pause, resume and end points as well as the residual time to decide for the higher priority in a concurrent case.

I want to consider the backoff timer in the following as an example for my timer implementation: When a backoff is started, the start method gets the priority, the contention window cw and the indication if the medium is idle or not. The current starting time is saved in *stime_[pri]*, while the *rtime_[pri]* indicates the residual backoff time. Let us assume that the medium is idle and there is no other active backoff, then the backoff will be scheduled with the residual time plus AIFS as shown below in the second half of the *start()*:

```
void
BackoffTimer_802_11e::start(int pri, int cw, int idle)
{
  Scheduler &s = Scheduler::instance();
  if(busy_) { //already a backoff running!
      offset_[pri] = 0;
      stime_[pri] = s.clock(); //Start-Time
      rtime_[pri] = (Random::random() % (cw + 1))
                              * mac->phymib_->SlotTime;
      backoff_[pri] = 1;
      AIFSwait_[pri] = 0.0;

      if(idle == 0){
          if(!paused_){
               pause();
          }
      } else {
          if(!paused_) {
               pause();
               AIFSwait_[pri] = mac->getAIFS(pri);
               restart();
          }
      }
  }
  else { //no other backoff
    backoff_[pri] = 1; // active flag for the
                       // corresponding priority
    busy_ = 1;         // active timer flag
    offset_[pri] = 0;
    stime_[pri] = s.clock(); //Start-Time
    rtime_[pri] = (Random::random() % (cw + 1))
                   * mac->phymib_->SlotTime;
    AIFSwait_[pri] = mac->getAIFS(pri);
    if(idle == 0){
        paused_ = 1;
    }
    else {
      s.schedule(this, &intr, rtime_[pri] + AIFSwait_[pri]);
    }
  }
}
```

If there have been one or more backoffs active before, *pause()* must be called, the new backoff must be inserted and all backoffs together are restarted again. This is done in the *if(busy){...}* loop.

As the name implies, *pause()* is used to stop the backoff timer temporarily. It is called when either the medium changes from the idle to the busy state or when a backoff is started (resp. handled) while there are other active backoffs. In *pause()* the elapsed time must be computed and subtracted from the residual time:

```
void
BackoffTimer_802_11e::pause()
{
  Scheduler &s = Scheduler::instance();
  for(int pri = 0; pri < MAX_PRI; pri++) {
     if(backoff_[pri]) {
         double st = s.clock();        // now
         double rt = stime_[pri]
                     + AIFSwait_[pri]; // start time
                                       // + waiting time
         double sr = st - rt;          // elapsed time
         double mst = (mac->phymib_->SlotTime);
         int slots = int (sr/mst);     // whole number of
                                       // bygone slots
         if(slots < 0)
             slots = 0;
         if(sr >= 0) offset_[pri] = sr - (slots * mst);
         if(rtime_[pri]
            - (slots * mac->phymib_->SlotTime) >= 0.0) {
           rtime_[pri] -= (slots * mac->phymib_->SlotTime);
         }
         else{
           if(rtime_[pri] + round
               - (slots * mac->phymib_->SlotTime) >= 0.0) {
                 rtime_[pri] = 0;
          } else {
             cout<<"ERROR in BackoffTimer::pause() \n";
             exit(0);
          }
          if(st - stime_[pri] >= mac->getAIFS(pri))
             AIFSwait_[pri] = 0.0;
          else{
            AIFSwait_[pri] -= st - stime_[pri];
            if(AIFSwait_[pri] < 0) AIFSwait_[pri] = 0;
          }
     }
  }
  s.cancel(&intr);
  paused_ = 1;
}
```

To avoid unexpected behavior due to rounding errors it needs not only be checked if *rtime - elapsed time* $\geq$ *0* but also if *rtime + round - elapsed time* $\geq$ *0*. For *round* I chose a value of $1*10^{-12}$.

The *offset* array is used to save the amount of the bygone time which is not a multiple of a slot time. This is necessary when a backoff is paused and restarted immediately, i.e. when a backoff is started while others are active, too. In *restart()* this offset will be subtracted from the residual time to keep it synchronous to the slot time.

Due to only one timer object per MAC, in *restart()* and *resume()* the smallest residual time has to be determined and scheduled, because only one event can be inserted into the scheduling list. I want to consider *resume()* in the following:

```
void
BackoffTimer_802_11e::resume()
{
  double delay = inf;
  int prio = MAX_PRI + 1;
  Scheduler &s = Scheduler::instance();
  for(int pri = 0; pri < MAX_PRI; pri++){
     if(backoff_[pri]){
         offset_[pri] = 0;
         round_time(pri);
         AIFSwait_[pri] = mac->getAIFS(pri);
         stime_[pri] = s.clock();
         double delay_ = rtime_[pri] + AIFSwait_[pri];
         // find smallest rtime + AIFS
         if((delay_ < delay) && backoff_[pri]) {
              delay = delay_;
              prio = pri;
         }
      }
  }
  if(prio < MAX_PRI + 1) {
     s.schedule(this, &intr, rtime_[prio] + AIFSwait_[prio]);
     paused_ = 0;
  } else {
     cout<<"ERROR: ";
     cout<<"wrong priority in BackoffTimer::resume() \n";
     exit(0);
  }
}
```

It is important to assure a residual time equal to a whole number of slots for the synchronization. This is realized in *round_time()* which rounds the rtime to a multiple of a slot time. Afterwards the duration of rtime + AIFS has to be calculated for all priorities and the smallest value is scheduled.

*AIFSwait_* plays an important role in the backoff timer. It indicates whether a timer was paused and resumed or paused and restarted due to the start or handle of another backoff while others are active. This is necessary because an event in *restart()* is scheduled with the *rtime* or *rtime + residual AIFS* (if the older backoff has been active not long enough for decrementing the rtime) only and not with *rtime* + whole AIFS as in *resume()*.

Due to the fact that only one event is scheduled, it is not possible to derive the corresponding priority from this event. So also in the *handle()* method, which is called when a timer expires, the decision must be made, which priority invoked the timer expiration. This is realized by the smallest delay determination, too:

```
void
BackoffTimer_802_11e::handle(Event *)
{
  Scheduler &s = Scheduler::instance();
  paused_ = 0;
  double delay = inf;
  int prio = MAX_PRI + 1;

  //determine smallest delay
  for(int pri = 0; pri < MAX_PRI; pri++){
       double delay_ = rtime_[pri] + AIFSwait_[pri];
       if((delay_ < delay) && backoff_[pri]) {
            delay = delay_;
            prio = pri;
        }
  }
  if(prio < MAX_PRI + 1) {
      busy_ = 0;
      stime_[prio] = 0;
      rtime_[prio] = 0;
      backoff_[prio] = 0;
      AIFSwait_[prio] = 0.0;
   }
   else {
      cout<<"ERROR: ";
      cout<<"wrong priority in BackoffTimer::handler()";
      exit(0);
   }
```

```
    // check for other active backoffs
    busy_ = 0;
    for(int pri = 0; pri < MAX_PRI; pri++){
        if(backoff_[pri]) { // check if there is another
            busy_ = 1;      // backoff process
            if(rtime_[pri] + AIFSwait_[pri] == delay) {
                mac->inc_cw(pri);
                AIFSwait_[pri] = 0;
                stime_[pri] = s.clock(); //Start-Time
                rtime_[pri] =(Random::random()
                              % (mac->getCW(pri) + 1))
                              * mac->phymib_->SlotTime;
                backoff_[pri] = 1;
            }
        }
    }
    if(busy_ && !paused_){
        pause(); // pause + restart because the
                 // backoff could be a post-backoff!
        restart();
    }
    mac->backoffHandler(prio);
}
```

After one knows which priority invoked the *handle()*, it has to be checked if there are still other active backoffs. Of course, it is possible that another active backoff with a lower priority has the same smallest delay as the handled priority. This would lead to a virtual collision. Therefore the lower priority has to be backed off again with an increased contention window but without an increase of the retry counter. With this solution the virtual scheduler is simply integrated into the backoff timer.

2.3.4 Contention Free Burst (CFB)

For enabling or disabling the CFB I introduced the *cfb_* flag which is set in *../ns-2.26/tcl/lan/ns-mac.tcl*. The May-2003 802.11e draft [11] defines a maximum duration of a transmission opportunity (*TXOPLimit*) for each priority. *TXOPLimit* is given in milliseconds and should take different values dependent on the Physical Layer (table 2.1).

In my model the *TXOPLimits* are defined in *../ns-2.26/mac/802_11e/priority.tcl* for each priority as described in 2.3.1.

I inserted the first part of the CFB code at the end of the method *recvACK()* in *../ns-2.26/mac/802_11e/mac-802_11e.cc*. *recvACK()* is called when a station receives an ACK on time for its last transmitted packet. If this ACK is error free, it has to be checked if this station should have the chance to send the next packet without contention,

Table 2.1: TXOP-Limits for different 802.11-PHYs

priority	802.11b	802.11a/g
0	3.0ms	1.5ms
1	6.0ms	3.0ms
2	3.0ms	1.5ms
3	0	0

i.e. if *cfb_* is equal to one. If the previous transmitted packet was the first one in the burst, its transmission duration must be taken into account by adding PKT_txtime + SIFS + ACK_txtime to *cfb_dur*, which is the duration counter:

```
void
Mac802_11e::recvACK(Packet *p)
{
 int pri = LEVEL(p);
 struct hdr_cmn *ch = HDR_CMN(p);
 ...
 // successful reception of this ACK packet
 // => data transmission was successful
 if(!cfb_ || ch->size() > MAC_RTSTHRESHOLD) {
 // CFB disabled => start post-backoff
     assert(mhBackoff_.backoff(pri) == 0);
     rst_cw(pri);
     mhBackoff_.start(pri, getCW(pri), is_idle());
     assert(pktTx_[pri]);
     Packet::free(pktTx_[pri]); pktTx_[pri] = 0;
     tx_resume();
 }
 else{
   // if this is the first packet in cfb, we must take its
   // duration into account, too.
   if(cfb_dur == 0) {
       cfb_dur = txtime(pktTx_[pri])
                 + sifs_
                 + txtime(ETHER_ACK_LEN, basicRate_);
   }
   pktRx_ = 0;
   rx_resume();
   assert(pktTx_[pri]);
   Packet::free(pktTx_[pri]); pktTx_[pri] = 0;
   cfb(pri); // next steps for CFB
 }
}
```

Afterwards the next packet should be taken out of the queue and its duration must be calculated. Therefore it is necessary to divide between unicast and multicast packets because the last one does not take an ACK what results in a smaller duration. This computation is done in *cfb()*:

```
void Mac802_11e::cfb(int pri)
{
  double timeout;
  struct hdr_mac802_11 *mh;
  // take next packet out of queue
  if(queue_->pri_[pri].getLen() > 0) {
      Packet* p = queue_->pri_[pri].deque();
      sendDATA(pri, p); // framing
      hdr_cmn *ch = HDR_CMN(pktTx_[pri]);
      mh = HDR_MAC802_11(pktTx_[pri]);
      if((u_int32_t)ETHER_ADDR(mh->dh_da) != MAC_BROADCAST)
          cfb_dur +=  sifs_
                      + txtime(pktTx_[pri])
                      + sifs_
                      + txtime(ETHER_ACK_LEN, basicRate_);
      else cfb_dur += sifs_
                      + txtime(pktTx_[pri]);
  } else cfb_dur = txop_limit_[pri] + 1; //nothing to send

  if(cfb_dur <= txop_limit_[pri]) {
    if((u_int32_t)ETHER_ADDR(mh->dh_da) != MAC_BROADCAST)
      timeout = txtime(pktTx_[pri])         // timeout for
                + DSSS_MaxPropagationDelay // send-timer
                + sifs_
                + txtime(ETHER_ACK_LEN, basicRate_)
                + DSSS_MaxPropagationDelay;
    else
      timeout = txtime(pktTx_[pri]);
    cfb_active = 1;
    mhSifs_.start(pri, sifs_); // next packet after SIFS
  }
  else { // nothing to send or
         // TXOPLimit is reached => start backoff
    cfb_dur = 0;
    rst_cw(pri);
    mhBackoff_.start(pri, getCW(pri), is_idle());
    tx_resume();
  }
}
```

The next packet in the burst should be sent if *cfb_dur* is smaller than the TXOPLimit

of the corresponding priority. Otherwise a backoff should be invoked. The sending procedure is realized in *transmit()*.

If the first packet in the CFB is a multicast packet it would not be answered with an ACK. So *recvACK()* will not be entered. The send timer, which determines the amount of time in which an ACK should arrive for unicast packet, is started for a multicast packet only for the transmission duration (and not for the whole DATA + SIFS + ACK sequence). The method *retransmitDATA()* is called when the send timer expires. Therefore the cfb-code in *recvACK* has to be inserted into *retransmitDATA()* to take a first multicast packet into account, too.

Chapter 3

Verification

3.1 Model Verification

In Jain [5] on page 413 *et seqq.*, model verification is referred to as procedure to “ensure that the model is correctly implemented” and is denoted as debugging, too.

I used the model verification techniques “Antibugging”, “Structured Walk-Through” and “Run Simplified Cases” [5] at first. Antibugging introduces “additional checks and outputs in programs that will point out the bugs, if any.” [5]. The detection of bugs with the Structured Walk-Through method is realized by “explaining the code to another person or group“ [5]. Run Simplified Cases checks the behavior of a module for e.g. “only one packet, or only one source”. Because these simple cases do not assure the correct work for more complex cases, I verify the implementation of my simulation model in the following by comparing it with the simulation results of Mangold *et al.* [6].

3.2 Akaroa 2.7.4

Akaroa [7] was used in the following simulations for the stochastic analysis. It has been developed at the University of Canterbury in Christchurch, New Zealand. Akoroa speeds up stochastic simulations and enables MRIP (Multiple Replications In Parallel) with different random number generator seeds. The use of Akaroa avoids the analysis of trace files due to "real-time simulation data output analysis" and "automatic run length control" [7].

The interface between Akaroa and ns-2.26, which I have used, has been developed at the TU Berlin [12].

All my simulations below have been controlled and terminated by Akaroa. In the maximum-throughput simulation in section 3.4 I used a confidence level of 95% and a mean-value precision of 5%. For the test of the EDCF in section 3.5 I chose a confidence level of 90% and a precision of 10%.

3.3 Simulation Model

3.3.1 PHY parameters

Mangold utilized the IEEE 802.11a-PHY with a data rate of 24Mbps, therefore I had to adopt the PHY parameters of my model, too. I set the PHY-802.11a parameters in *../ns-2.26/mac/802_11e/mac-802_11e.h* to:

- SlotTime: 9μs
- CCATime: 3μs
- RxTxTurnaroundTime: 2μs
- SIFSTime: 16μs
- PreambleLength: 96bits = 16μs
- PLCPHeaderLength: 40bits
- PLCPDataRate: $6*10^6$Mbps

The data rate of 24Mbps was set in *../ns-2.26/tcl/lan/ns-mac.tcl*.

Small inaccuracies due to the 802.11a-framing may occur because the last 16bits of the PLCP header (i.e. the service bits) in an 802.11a PHY frame (figure 2.1 on page 11) spill into the payload, which is transmitted with a data rate of 24Mbps. This spilling over as well as the six tail and the padding bits at the end of an 802.11a PHY frame are not realized in ns-2.26.

3.3.2 Metric and Traffic

The throughput of each traffic category is the metric in all simulations.

Mangold used 3 different TCs in his simulations: a high-priority isochronous flow with 128kbps and 80byte MSDUs, a medium- and a low-priority Poisson flow with each 160kbps and 200byte MSDUs.

I decided to use isochronous flows for all three TCs due to interface queue lengths of 50 packets as well as a medium interarrival time of 0.01ms for the 200byte packets. Therefore Poisson arrivals will be averaged out in overload scenarios, which I am interested in for testing my EDCF solution. The advantage of this simplification is an earlier termination of the simulations.

3.3.3 Error and Propagation Model

In my work I am only interested in the behavior of the EDCF. Therefore I did not use an error model for the wireless medium but ns' TwoRayGround Propagation model which assumes specular reflection of a flat ground plane. It uses an attenuation $\sim \frac{1}{d^4}$ where d is the distance between two nodes.

In Mangold's simulations an error model was included.

3.3.4 MAC / priority parameters

The backoff parameters for Mangold's simulations are displayed in table 3.1. AIFS

Table 3.1: Mangold's backoff parameters

	High	Medium	Low
AIFS	2	4	7
CWmin	7	10	15
CWmax	7	31	255
PF	2	2	2

given in a whole number *n* means that it is calculated by AIFS = SIFS + *n* * SlotTime. CWmin = CWmax turns off the binary exponential backoff for the high priority. PF equal to two implies a binary exponential backoff for all other priorities.

3.4 Maximum Throughput in 802.11a

At first I want to consider and compare the maximum achievable throughput in ns-2.26 with Mangold's simulations. His first scenario is a BSS consisting of an AP and only one wireless station. On this station one flow is sent to the AP. He performed one separate simulation for each TC with an increased generation rate resp. a larger MSDU. The throughputs are listed in table 3.2.

Table 3.2: *My* / Mangold's maximum 802.11a throughput [Mbit/s]

Data frame size	80bytes	200bytes	2304bytes
High	*3.52* / 3.5	-	*19.98* / 19.81
Medium	-	*6.32* / 6.22	*19.32* / 19.16
Low	-	*5.29* / 5.21	*18.37* / 18.22

The throughputs of my simulations are displayed on the left side of the slash, while Mangold's results are presented on the right side. I reached very similar results. Because Mangold does not give confidence intervals in his paper, I am not able to point out whether or not the results differ significantly.

3.5 Test of EDCF

For the test of my EDCF I chose a simulation of Mangold in which each station carries all three priority flows. Mangold increases the number of stations in the BSS continuously from 1 to 15 as shown in figure 3.1. All stations are in the range of each other and the stations are not moving.

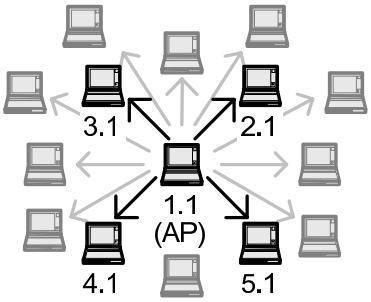

Figure 3.1: AP with variable number of stations (source: [6])

Mangold's results are shown in figure 3.2. The throughput of all TCs increases up to a number of 9 stations. Behind this point, the throughput of the low-priority flows decreases (in terms of throughput per station and flow). For more than 12 stations, the throughput decreases for the medium priority, too, while the high priority flows can carry their traffic for up to 15 stations.

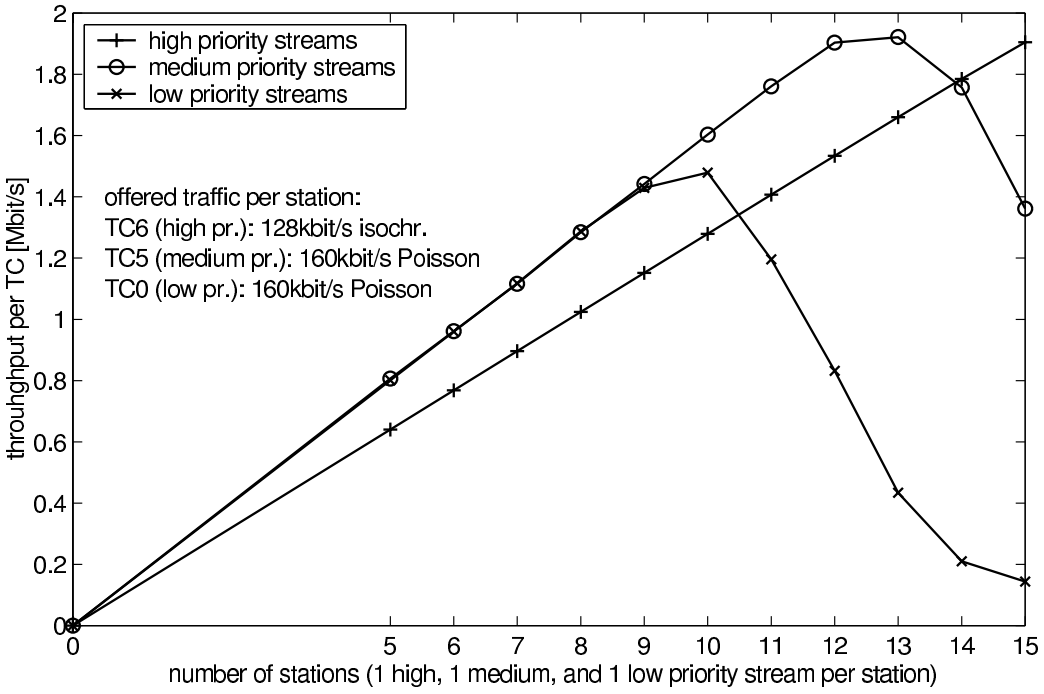

Figure 3.2: Mangold's results for increasing number stations vs. throughput [6]

The first results of my simulations are shown in figure 3.3. The low flow can carry its traffic up to 9 and the medium flow only up to 11 stations. It attracts attention that the throughput of low and medium flows decreases faster than in Mangold's graph.

The throughput of the high priority flows increases continuously only up to a number

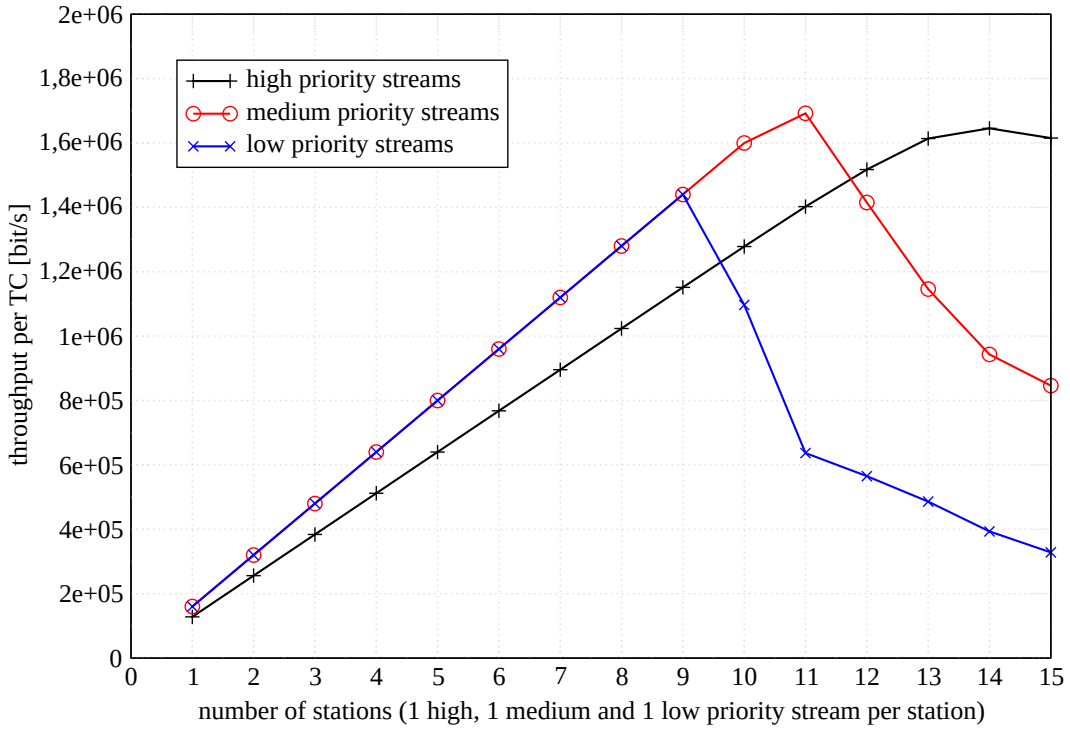

Figure 3.3: Results of my 802.11e EDCF model

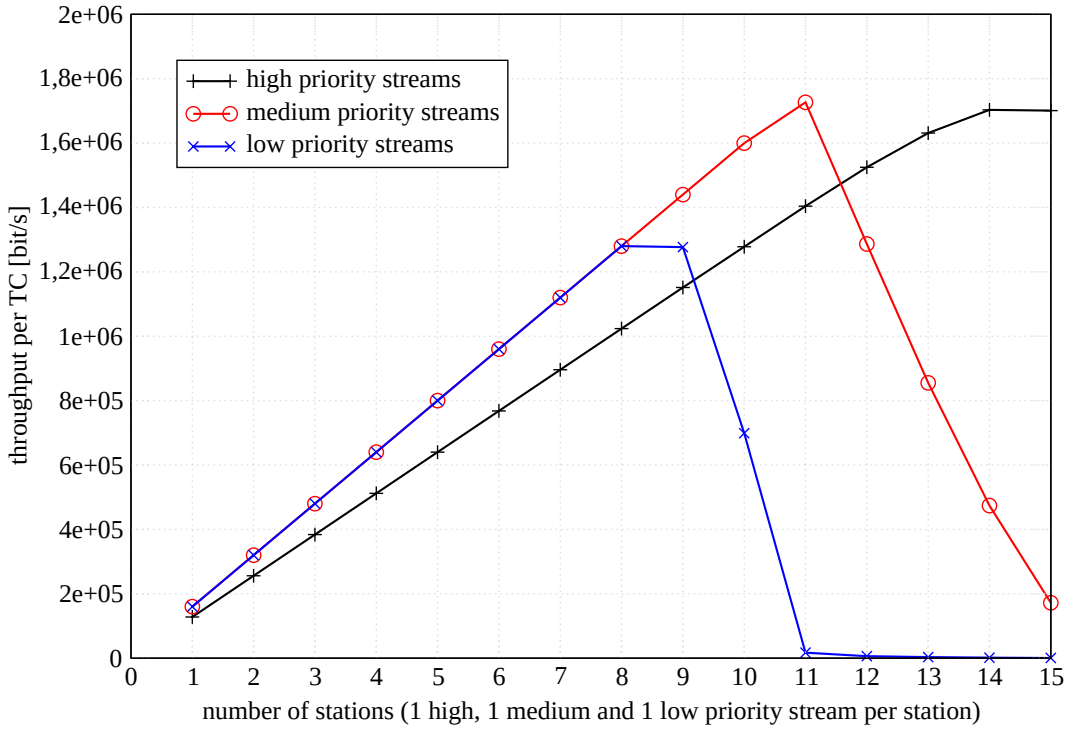

Figure 3.4: Results of my improved 802.11e EDCF model

of 13 stations, afterwards it decreases slightly (in terms of throughput per station). From my point of view, this makes sense due to the chosen backoff parameters CWmin = CWmax = 7 which switch the binary exponential backoff off and increase the probability of a collision on the medium. Actually the short retry counter (SRC), which counts the retransmission attempts of a packet, often expires due to a lot of repeating collisions. For the throughput calculation I do not take packets into account which are discarded due to the SRC expiration after the fourth unsuccessful transmission.

In the course of my work, I reviewed my code together with my advisor several times. Thereby I found a small error in the backoff timers: If there is already an active backoff and another backoff is started when the medium is idle, all backoffs must be paused and restarted. I implicitly assumed that the older backoff has been longer active than a period of AIFS. Therefore I set AIFSwait in *pause()* to 0 . In *restart()* therefore the backoff with the smallest delay will be scheduled only with the residual backoff time. In a case where the medium has been idle and the backoff has been active for an amount of time smaller than AIFS I falsely forgot the residual amount of AIFS.

To avoid this error, it has to be checked in *pause()* if the past time since the last busy medium is longer or shorter than AIFS. If it is shorter, AIFSwait must be set to the residual AIFS. The code in chapter 2.3.3 already includes the correction.

After the removal of this small bug the results look slightly different (figure 3.4). Now, the low flow can carry its traffic only up to 8 eight stations, while the medium flow has a maximum throughput at 11 stations. The throughput of both flows decreases afterwards very fast in favor of the high priority flows with its maximum at 14 stations. With this example, I want to point out that one may reach great differences only due to minor changes in the backoff timers. Although I compared two totally different simulation models, I gained similar results. Mangold's as well as my model are able to regulate the medium access in order to serve data flows with high priority on the cost of medium and low priority flows in overload scenarios.

Chapter 4

Conclusions

In this work, I have given an overview over the existing wireless LAN MAC IEEE 802.11 and its problems of achieving a sufficient QoS with respect to time bounded data like VoIP or Video. I have explained the proposed enhancement IEEE 802.11e with its medium access functions EDCF and HCF by comparing three different drafts. Especially the changes in the contention based solution EDCF have been considered. In addition, I have described the principle of PFG which basically descends from the fragmentation / defragmentation mode of IEEE 802.11.

The design of my simulation model has been based on the ns-2.26-802.11 model which includes several inaccuracies and partial erroneous behavior. I have explained these bugs and have proposed solutions. To delineate the design of my 802.11e EDCF model, I have basically described the structure as well as the backoff timer solution, which was the most important part of my coding work.

For debugging and verification, I have compared my simulation model with results of Mangold *et. al.*'s model [6]. I have divided my verification into two parts: In the first part, I have compared the maximum achievable throughput of the 802.11a PHYs to estimate non-MAC differences. The results of both models are similar. For the second and most important part, I have used a simulation scenario out of which not only the prioritization of timebounded data but also the maximum number of high priority stations arises. Basically, I have reached a similar behavior that has nevertheless different results: Due to a lot of retransmissions and an expiration of the short retry counter, the maximum number of concurrent sending stations with high priority traffic is 14 in my simulations, while Mangold was able to meet the requirements for 15 stations. In my model, also the throughput of medium and low priority traffic decreases faster and drops much more down.

Despite some varieties as well as two total different simulation languages and models, I am able to present a modular, functioning and verified IEEE 802.11e EDCF simulation model for ns-2.26, in which all the bugs of the ns-802.11 model have been replaced.

Bibliography

[1] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns.

[2] ANSI/IEEE. 802.1D: Media Access Control (MAC) Bridges. IEEE, 1998.

[3] ANSI/IEEE. 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE, 1999.

[4] IEEE. Supplement to IEEE Standard 802.11-1999: High-speed Physical Layer in the 5 GHz Band. IEEE, 1999.

[5] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. May 1991.

[6] S. Mangold, S. Choi, P. May, O. Klein, G. Hiertz, and L. Stibor. IEEE 802.11e Wireless LAN for Quality of Service (invited paper). In *Proceedings of the European Wireless*, volume 1, pages 32–39, Florence, Italy, February 2002.

[7] K. Pawlikowski, G. Ewing, and D. McNickle. Project akaroa. http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/.

[8] J. Tourrilhes. Packet Frame Grouping: Improving IP Multimedia Performance over CSMA/CA. Hewlett Packard Laboratories, Bristol, UK, 1997.

[9] IEEE 802.11 WG. Draft Supplement to Standard 802.11-1999: Medium Access Control (MAC) Enhancements for Quality of Service (QoS). IEEE 802.11e/D2.0a, November 2001.

[10] IEEE 802.11 WG. Draft Supplement to IEEE Standard 802.11-1999: Medium Access Control (MAC) Enhancements for Quality of Service (QoS). IEEE 802.11e/D5.0, 2003.

[11] IEEE 802.11 WG. Draft Supplement to IEEE Standard802.11-1999: Medium Access Control (MAC) Enhancements for Quality of Service (QoS). IEEE 802.11e/D4.3, May 2003.

[12] Hagen Woesner, Andreas Köpke, and Eugen Gillich. The ns-2/akaroa-2 project. http://www-tkn.ee.tu-berlin.de/research/ns-2_akaroa-2/ns.html.

Appendix A

Hints for the Installation of the EDCF Model

A.1 Changes to legacy ns-2

1. please change into the directory *ns-allinone-version/ns-x.y/mac/* and unpack the file *mac80211e.tgz,*

2. changes to your Makefile.in in *ns-allinone-version/ns-x.y/*:

   - add to INCLUDES:
     -I./mac/802_11e
   - add to OBJ_CC:
     mac/802_11e/mac-802_11e.o mac/802_11e/priq.o
     mac/802_11e/d-tail.o mac/802_11e/mac-timers_802_11e.o
   - exclude in NS_TCL_LIB:
     tcl/lib/ns-mobilenode.tcl \
   - add to NS_TCL_LIB:
     mac/802_11e/ns-mobilenode_802_11e.tcl \
     mac/802_11e/priority.tcl \

3. changes to your *ns-allinone-version/ns-x.y/tcl/lib/ns-lib.tcl*:

   - exclude from the source list:
     source ns-mobilenode.tcl
   - add to the source list:
     source /ns-allinone-2.26/ns-2.26/mac/802_11e/ns-mobilenode_802_11e.tcl
     source /ns-allinone-2.26/ns-2.26/mac/802_11e/priority.tcl

4. add to your ns-allinone-version/ns-x.y/tcl/lib/ns-default.tcl:
   Queue/DTail set drop_front_ false
   Queue/DTail set summarystats_ false
   Queue/DTail set queue_in_bytes_ false

Queue/DTail set mean_pktsize_ 500
Queue/DTail/PriQ set Prefer_Routing_Protocols 1
Queue/DTail/PriQ set Max_Levels 4
Queue/DTail/PriQ set Levels 4

5. add to tcl/lan/ns-mac.tcl:
   if[TclObject is-class Mac/802_11e]{
   ...
   copy settings of *Mac/802.11* (which are contained
   in this file) into this section and change them into *Mac/802.11e*
   ...
   Mac/802_11e cfb_ 0 ;# disables CFB
   }

6. run ./configure; make depend; make in your ns directory.

A.2 Tcl-Settings for Simulations with EDCF

1. changes to your simulation script:

   - add MAC and queue type
     set opt(mac) Mac/802_11e
     set opt(ifq) Queue/DTail/PriQ
   - after defining your transport_agent
     % set transport_agent [new Agent/UDP]
     , just add
     % $your_transport_agent prio_ x
     to give a certain flow a specific priority (x between 0 and 3, 0 being the highest, 3 being the lowest priority),

2. changes to 802.11e parameters (CW_MIN, CW_MAX, AIFS, PF, TXOPLimit) can be made in *802_11e/priority.tcl* for each queue. Please rerun make afterwards.