Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwurf und Implementierung eines Stream-basierten, dynamischen und fairen Scheduling-Verfahrens für WiFi-Netzwerkverkehr

Jona Herrmann

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Christian Becker
Betreuer/in:	Robin Laidig, M.Sc.
Beginn am:	24. Mai 2023
Beendet am:	24. November 2023

Kurzfassung

Heutzutage sind Wireless Local Area Network (WLAN)-Netzwerke, basierend auf dem IEEE 802.11 Standard, für mobile Endgeräte besonders wichtig, da diese nur so Zugang zum Internet bekommen können. Dementsprechend gibt es solche Netzwerke fast überall wie beispielsweise in öffentlichen Verkehrsmitteln, in Hotels oder auch bei der Arbeit. Doch die Qualität und Geschwindigkeit einer Verbindung kann dabei stark variieren. Um dies zu verbessern, wird zuerst eine Analyse dieser Netzwerke vorgenommen, um die Stelle des Bottlenecks zu identifizieren. Dabei konnte gezeigt werden, dass der Bottleneck an zwei Stellen vorliegen könnte. Das wäre einmal der Uplink zum Internet und andererseits der entsprechende Downlink, wobei sich der Bottleneck am Downlink letztendlich beim Internet-Provider befindet. Mit einem neu entwickelten Stream-basierten, dynamischen und fairen Scheduling-Verfahren soll die Qualität und Geschwindigkeit im WLAN-Netzwerk verbessert werden. Dafür wird eine neue Art von Fairness definiert, sodass die Pakete von Endgeräten mit einem geringen Datenverbrauch eine höhere Priorität erhalten. Dadurch bekommen letztendlich Endgeräte, welche gutmütig sind, eine bessere Antwortzeit als diese, die eine große Datenmenge übertragen. Zum Schluss wird die Linux-Implementierung des Scheduling-Verfahrens noch unter verschiedenen Metriken evaluiert. Dabei konnte gezeigt werden, dass die gewünschte Art von Fairness damit realisiert werden kann. Dies wurde sowohl unter Laborbedingungen als auch mit realen Anwendungen des Internets erfolgreich evaluiert. Außerdem wurde durch Messungen gezeigt, dass es im Punkt Performanz keinen signifikanten Unterschied zu dem Standard Scheduling-Verfahren in Linux gibt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

AC Access Category. 19

ACK Acknowledgement. 19

AIFS Arbitration Interframe Space. 20

AIFSN Arbitration Interframe Space Number. 20

AP Access Point. 17

BSS Basic Service Set. 17

BSSID Basic Service Set Identifier. 17

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. 18

CW Contention Window. 18

DCF Distributed Coordination Function. 18

DIFS Distributed Coordination Function Interframe Spacing. 18

DS Distribution System. 17

DSCP Differentiated Services Code Point. 20

EDCA Enhanced Distributed Channel Access. 19

ESS Extended Service Set. 17

FIFO First In – First Out. 22

HCF Hybrid Coordination Function. 19

HTML Hypertext Markup Language. 26

HTTP Hypertext Transfer Protocol. 26

IEEE Institute of Electrical and Electronics Engineers. 17

LAN Local Area Network. 17

MAC Media Access Control. 18

MU-MIMO Multi-User Multiple Input Multiple Output. 21

OFDMA Orthogonal Frequency-Division Multiple Access. 21

PHY Physical Layer. 20

Qdisc Queueing-Discipline. 22

QoS Quality of Service. 19

QUIC Quick UDP Internet Connections. 32

SIFS Short Interframe Space. 19

SKB Socket-Buffer. 40

SSID Service Set Identifier. 17

STA Station. 17

tc Traffic-Control. 21

TCP Transmission Control Protocol. 24

ToS Type of Service. 20

TXOP Transmission Opportunity. 19

UDP User Datagram Protocol. 24

UP User Priority. 20

WLAN Wireless LAN. 17

WMM WiFi Multimedia. 19

1 Einleitung

Heutzutage sind WiFi-Netzwerke, basierend auf dem IEEE 802.11 Standard, nicht mehr wegzudenken. Denn solche sind besonders wichtig für mobile Endgeräte, welche dadurch einfach und schnell Zugang zum Netzwerk und damit letztendlich zum Internet bekommen können. Ein wesentlicher Vorteil von WiFi sind die drahtlosen Verbindungen und die damit einhergehende einfache Installation eines WiFi-Netzwerkes. Dies führt dazu, dass es solche Netzwerke fast überall gibt. Beispielsweise bei der Arbeit, in öffentlichen Verkehrsmitteln oder Hotels, aber auch Zuhause. Somit kann überall leicht ein Zugang zum Internet ermöglicht werden.

1.1 Problemstellung

In drahtlosen Netzwerken teilen sich alle Endgeräte das gemeinsame Medium, was dazu führt, dass die Qualität (Quality of Service, QoS) und Geschwindigkeit der Verbindungen stark variieren kann. Deshalb wird in dieser Arbeit zuerst analysiert, an welcher Stelle im WiFi-Netzwerk der Bottleneck vorliegt. Denn an dieser Stelle kann es zu einer Überlast kommen, wodurch einzelne Teilnehmer möglicherweise eine sehr schlechte Antwortzeit bekommen oder sogar aushungern können. In diesem Zusammenhang wird auch untersucht, ob und inwieweit die Anzahl an aktiven Endgeräten einen Einfluss auf die Verbindungsqualität hat. Außerdem wird analysiert, wie sich in Überlastsituationen die Fairness zwischen den Transportprotokollen verhält.

An dem identifizierten Bottleneck soll mit einem neuen Stream-basiertem Scheduling-Verfahren Fairness realisiert werden. Dabei ist die in der Netzwerktechnik meist genutzte Art von Fairness nicht sinnvoll. Bei dieser wird die verfügbare Datenrate zwischen allen aktiven Endgeräten im WiFi-Netzwerk aufgeteilt. Doch in diesem befinden sich oft viele Geräte zur gleichen Zeit und somit bekommt jedes nur eine sehr geringe Datenrate. Diese fällt sowieso schon gering aus, da die maximal verfügbare Datenrate des Internetzuganges meist nicht sehr hoch ist. Ein Beispiel dafür sind öffentliche Verkehrsmittel. Deshalb wird in dieser Arbeit eine neue Art von Fairness definiert. Dazu betrachten wir die Situation, bei dem sich ein oder mehrere aktive Geräte in einem öffentlichen WiFi-Netzwerk befinden, welche zum Beispiel im Internet surfen oder ein Video anschauen. Wenn jetzt ein neues Gerät hinzukommt, welches zum Beispiel nur kurz eine Nachricht oder E-Mail senden will, wäre es wünschenswert, dass dieses Gerät den Vorzug beim Senden bekommt. Dadurch wäre sichergestellt, dass das Senden der Nachricht oder E-Mail schnell erfolgreich ist. Um dies zu erreichen, wird hier Fairness im Bezug auf die verbrauchte Datenmenge verwendet. Dies wird mithilfe dynamischer Prioritäten, wie es zum Beispiel im Dynamic Priority Token Bucket Verfahren [LDR+23] genutzt wird, umgesetzt. Dabei hängt die Priorität eines Endgerätes von dessen verbrauchter Datenmenge ab. Somit erhalten Geräte, welche einen geringen Datenverbrauch haben, eine bessere Priorität als solche, die eine große Datenmenge verbrauchen. Dementsprechend ist das Ziel dieser Masterarbeit das Entwerfen und Implementieren eines Stream-basierten, dynamischen und fairen Scheduling-Verfahrens für WiFi-Netzwerkverkehr.

1.2 Struktur der Arbeit

Die Arbeit ist in folgender Weise strukturiert: In Kapitel 2 werden zuerst die Grundlagen, welche für das Verständnis dieser Arbeit wichtig sind, beschrieben. Außerdem wird darin auf verwandte Arbeiten eingegangen. Anschließend wird in Kapitel 3 das WiFi-Netzwerk unter realistischen Annahmen analysiert, an welcher Stelle es zu einer Überlast kommen kann. In Kapitel 4 wird der Entwurf des neuen Scheduling-Verfahrens beschrieben, welches die neue Art von Fairness realisiert. Die Implementierung davon wird anschließend in Kapitel 5 erläutert. Schließlich wird in Kapitel 6 das entwickelte Verfahren mit geeigneten Metriken evaluiert. Zum Schluss erfolgt in Kapitel 7 eine Zusammenfassung der Ergebnisse und es wird ein Ausblick auf anschließende Arbeiten gegeben.

2 Hintergrund und verwandte Arbeiten

In diesem Kapitel werden zuerst die grundlegenden Technologien eingeführt, welche für das Verständnis dieser Arbeit notwendig sind. Anschließend wird auf verwandte Arbeiten in diesem Themenfeld eingegangen.

2.1 Ethernet

Ethernet ist eine weitverbreitete Technologie für die Kommunikation in Local Area Networks (LANs). Das Institute of Electrical and Electronics Engineers (IEEE) hat diese im Standard IEEE 802.3 [IEE22] definiert. Über die Jahre wurde Ethernet ständig weiterentwickelt. Heutzutage ist die Weiterentwicklung IEEE 802.3ab am verbreitetsten. Diese ist auch unter dem Namen Gigabit Ethernet bekannt und erreicht eine Datenrate von bis zu 1 GBit/s. Als Verbindungen werden dabei Full-Duplex Twisted-Pair-Kabel verwendet. Um ein großes LAN mit vielen angeschlossenen Geräten aufbauen zu können, werden Switches als Kopplungselemente benötigt. Von diesen gibt es zwei verschiedene Arten: Hardware-Switches und Software-Switches. Wie der Name schon sagt, ist die Funktionalität einmal in Hardware und einmal in Software realisiert. Dementsprechend wird bei einem Software-Switch ein Rechner benötigt, auf dem dieser ausgeführt wird.

2.2 WLAN

Wireless LAN (WLAN) bezeichnet ein lokales Funknetz, welches im IEEE 802.11 Standard [IEE21a] definiert ist. Dieser spezifiziert verschiedene Betriebsmodi für WLANs, wobei in dieser Arbeit der Infrastruktur-Modus verwendet wird. In Abbildung 2.1 ist ein Beispiel Netzwerk in diesem Modus mit zwei Access Points (APs) und mehreren drahtlosen Geräten, welche allgemein als Stationen (STAs) bezeichnet werden, dargestellt. Der AP fungiert als zentrale Komponente und die STAs müssen sich zuerst durch den Association-Prozess bei einem AP registrieren. Ein AP und alle damit verbundenen STAs werden als Basic Service Set (BSS) bezeichnet. In Abbildung 2.1 sind diese als dunkelblaue Mengen dargestellt. Diese werden durch eine eindeutige Nummer, der Basic Service Set Identifier (BSSID), identifiziert. Typischerweise wird als Nummer die MAC-Adresse des zugehörigen AP verwendet. Es können mehrere BSSs durch ein Distribution System (DS) verbunden werden, diese bilden dann ein Extended Service Set (ESS), welches über den Service Set Identifier (SSID) identifiziert wird. Das DS kann zum Beispiel mit Ethernet realisiert werden und verbindet, wie in Abbildung 2.1 zu sehen, die verschiedenen APs untereinander.

Die WiFi Alliance ist eine Organisation, welche aus Unternehmen im Bereich der Netzwerktechnik besteht. Das Ziel dieser Alliance ist es, Interoperabilität zu erreichen, indem WLAN-Geräte nach dem IEEE 802.11 Standard zertifiziert werden.

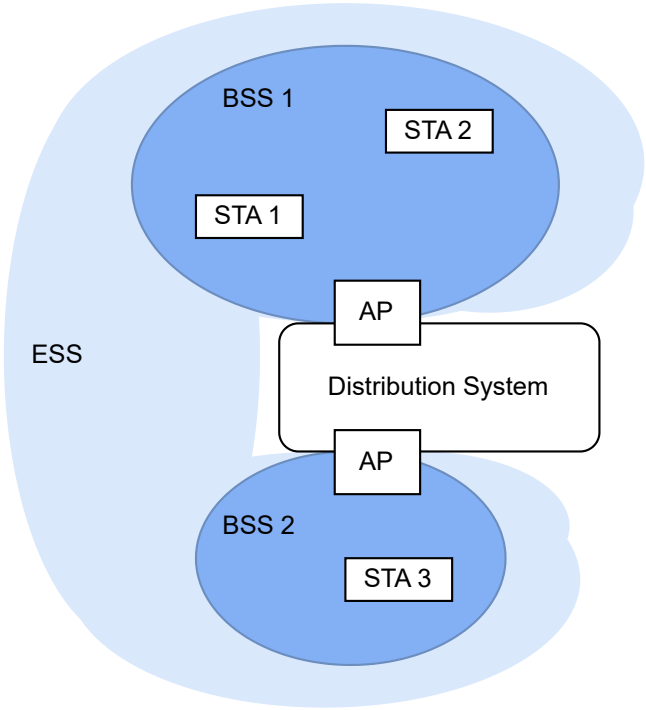


Abbildung 2.1: Beispiel WLAN-Netzwerk im Infrastruktur-Modus.

2.2.1 Media Access Control

In einem WLAN teilen sich alle STAs und der AP das kabellose Übertragungsmedium. Falls mehrere Teilnehmer gleichzeitig senden, entsteht eine Kollision und die gesendeten Frames sind dann unbrauchbar. Um das gleichzeitige Senden möglichst zu vermeiden, wird ein Media Access Control (MAC) Protokoll benötigt.

In IEEE 802.11 wird die Distributed Coordination Function (DCF) als Standard MAC Protokoll beschrieben, welches auf dem Prinzip Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) basiert. In Abbildung 2.2 ist eine Beispielübertragung unter Verwendung dieses Protokolls dargestellt. DCF regelt den Zugriff auf das gemeinsame Medium wie folgt:

1. Wenn sich ein Frame in der Sende-Queue befindet, beginnt die STA das Medium abzuhören (in Abbildung 2.2 markierte der pinke Pfeil, das sich das Frame in der Queue befindet).
2. Falls das Medium für die Dauer von Distributed Coordination Function Interframe Spacing (DIFS) frei ist, wird das Frame sofort gesendet.
3. Wenn das Medium nicht für DIFS frei war, also davor belegt wurde, wird gewartet, bis das Medium wieder für DIFS frei ist. Danach wird eine zufällige Backoffzeit (Vielfaches der Slot-Zeit, siehe in Abbildung 2.2 roter Bereich) aus dem Contention Window (CW) bestimmt. Diese Zeit wird anschließend, solange das Medium frei ist, verringert.
4. Falls das Medium belegt wird, die Zeit aber noch nicht abgelaufen ist, wird mit dem Verringern gestoppt, bis das Medium anschließend wieder für DIFS frei ist.
5. Wenn die Zeit null erreicht, wird das Frame gesendet.

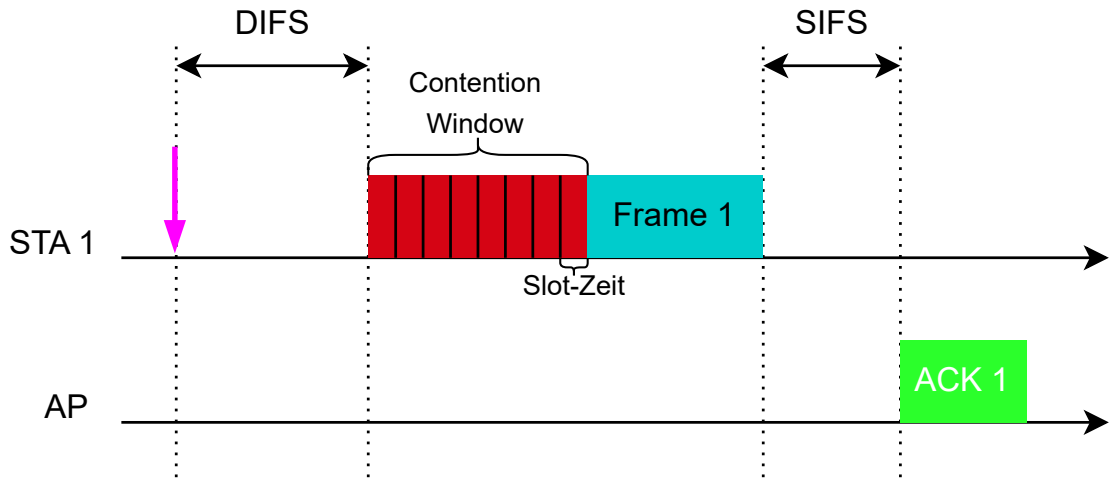


Abbildung 2.2: Beispielhafte Übertragung auf einem kabellosem Medium.

Der Empfang eines Frames muss durch den Empfänger mit einem Acknowledgement (ACK) bestätigt werden. In Abbildung 2.2 ist dieser Ablauf entsprechend dargestellt. Dabei sendet der AP nach dem korrekten Empfang von *Frame 1*, nachdem er Short Interframe Space (SIFS) gewartet hat, das ACK Frame. Da SIFS kleiner ist als DIFS haben ACKs eine höhere Priorität als Daten Frames und dadurch wird das ACK immer vor weiteren Daten Frames zuerst übertragen. Nachdem *STA 1* in Abbildung 2.2 den Sendevorgang von *Frame 1* beendet hat, erwartet diese nach SIFS das ACK. Wenn dieses nicht empfangen wird, wird eine Kollision angenommen. In diesem Fall startet *STA 1* den Binary Exponential Backoff Algorithmus. Bei diesem wird das CW jedes Mal verdoppelt, wenn eine Kollision aufgetreten ist, bis ein Maximalwert erreicht ist. Anschließend wird mit den Schritten 3-5 von oben fortgefahren. Wenn das Senden erfolgreich war, wird das CW anschließend wieder auf einen definierten minimalen Wert zurückgesetzt.

2.2.2 IEEE 802.11e

Der Standard IEEE 802.11e [IEE05] stellt eine Erweiterung zu IEEE 802.11 dar. Dieser definiert ein Konzept um Quality of Service (QoS) im WLAN zu ermöglichen. Die WiFi Alliance zertifiziert eine Teilmenge dieses Standards unter dem Begriff WiFi Multimedia (WMM). Diese besteht aus Enhanced Distributed Channel Access (EDCA) und Transmission Opportunity (TXOP), welche entsprechend bei zertifizierten Geräte vorhanden sein müssen. Die Erläuterung zu den beide erfolgt im restlichen Abschnitt.

Umgesetzt wird die Erreichung von QoS, indem mit der Hybrid Coordination Function (HCF) ein neues MAC Protokoll spezifiziert wird. HCF definiert zwei verschieden Mechanismen, wobei aber hauptsächlich nur EDCA verwendet wird. EDCA funktioniert dabei grundsätzlich wie DCF, außer das die Frames in vier verschiedene Access Categorys (ACs) eingeteilt werden: AC_VO, AC_VI, AC_BE und AC_BK welche für Voice, Video, Best Effort und Background Traffic stehen. Die ACs haben dann je nachdem wie kritisch diese sind, unterschiedlich hohe Prioritäten, welche der Tabelle 2.1 entnommen werden können. Eine AC wird durch verschiedene Parameter beschrieben. Durch entsprechend andere Werte werden die unterschiedlichen Prioritäten erreicht.

Priorität	AC	AIFSN	CWmin	CWmax
niedrig	AC_BK	7	aCWmin	aCWmax
↓	AC_BE	3	aCWmin	aCWmax
↓	AC_VI	2	(aCWmin+1)/2 – 1	aCWmin
hoch	AC_VO	2	(aCWmin+1)/4 – 1	(aCWmin+1)/2 – 1

Tabelle 2.1: Default Parameter der ACs in EDCA [IEE05].

Priorität	UP	AC	DSCP
niedrig	1	AC_BK	8
	2	AC_BK	16
↓	0	AC_BE	0
↓	3	AC_BE	24
↓	4	AC_VI	32
↓	5	AC_VI	40
	6	AC_VO	48
hoch	7	AC_VO	56

Tabelle 2.2: Mapping zwischen UP und AC und Default Mapping zwischen DSCP und UP [IEE05][SHB18].

Ein wichtiger Parameter ist der Arbitration Interframe Space (AIFS), welcher den DIFS im CSMA/CA Ablauf ersetzt. Für jede AC ist eine Arbitration Interframe Space Number (AIFSN) definiert, siehe Tabelle 2.1. Mit dieser wird der AIFS mit folgender Formel berechnet: *SIFS + AIFSN * Slot-Zeit*. Somit sind die verschiedenen AIFSs alle größer als SIFS, damit weiterhin ACKs die höchste Priorität haben. Letztendlich haben ACs mit höherer Priorität ein kleineres AIFS und müssen dadurch bei CSMA/CA kürzer warten. Dies führt dann zu einem schneller Zugriff auf das Medium.

Für jede AC wird außerdem ein minimaler und maximaler Wert für das CW spezifiziert. In Tabelle 2.1 sind diese in Abhängigkeit von *aCWmin* und *aCWmax* dargestellt, wobei diese Werte von der Implementierung des Physical Layer (PHY) abhängen. Diese Werte für das CW werden dann bei einer Kollision im Binary Exponential Backoff Algorithmus verwendet. Dadurch haben ACs mit höherer Priorität ein kleineres CW und somit ist statistisch gesehen die zufällige gewählte Backoffzeit kleiner und dies führt zu einem schnelleren Zugriff auf das Medium.

Ein weiterer Parameter ist die TXOP, welche ein Zeitintervall beschreibt. In diesem kann eine STA, nachdem diese den Zugriff auf das Medium bekommen hat, so viele Frames wie möglich senden. Dabei haben ACs mit höherer Priorität ein größeres Zeitintervall. Die konkreten Werte der TXOP hängen von der Implementierung des PHY ab.

Um die Frames in die vier ACs einzuteilen, werden acht User Prioritys (UPs) definiert. In Tabelle 2.2 ist die Zuordnung dieser acht UPs auf die ACs dargestellt. Für jedes Frame muss dann eine entsprechend UP angegeben werden. Dies kann zum Beispiel über das Differentiated Services Code Point (DSCP) Feld erfolgen, welches sich im Type of Service (ToS) Feld des IPv4-Headers befindet. Da das DSCP Feld 6 Bit groß ist, die UP aber nur 3 Bit, muss ein Mapping definiert werden. Beim Default Mapping spezifizieren die 3 MSB des DSCP Feldes die UP [SHB18]. In Abschnitt 6.1.1 wird dieses Mapping durch Messungen verifiziert.

2.2.3 IEEE 802.11ax

Die aktuell neueste Weiterentwicklung des IEEE 802.11 Standards ist IEEE 802.11ax [IEE21b]. Dieser bietet eine Reihe von Verbesserungen gegenüber seinen Vorgängern. Von der WiFi Allianz wird dieser Standard unter der Bezeichnung WiFi 6 zertifiziert. Die maximal zu erreichende Datenrate hängt dabei von der verwendeten Frequenz und Bandbreite sowie der Anzahl an Antennen/Streams ab. Mit WiFi 6 kann das 2,4-GHz-Band sowie das 5-GHz-Band verwendet werden. Im Folgenden werden die grundlegenden neuen Konzepte dieses Standards erläutert.

Mit Orthogonal Frequency-Division Multiple Access (OFDMA) wird der verwendete Kanal in kleinere Unterabschnitte, den Resource Units (RUs) aufgeteilt. Diese werden dynamisch den Geräten zugewiesen und somit können mehrere Geräte gleichzeitig senden. Wodurch dann eine geringere Latenz erreicht wird.

Beim Vorgänger Standard, WiFi 5, wurde schon Downlink Multi-User Multiple Input Multiple Output (MU-MIMO) verwendet. Bei WiFi 6 ist dies jetzt auch in Uplink Richtung möglich. Dabei können gleichzeitig verschiedene Daten an mehrere Geräte gesendet werden. Indem diese Daten jeweils über unterschiedliche Antennen geschickt werden.

Mit dem Konzept BSS Coloring soll die Performanz erhöht werden. Denn in Bereichen, in den sich viele BSSs befinden, ist die Wahrscheinlichkeit, das diese sich überlappen, hoch. Im CSMA/CA Ablauf wird vor dem Senden das Medium abgehört und dabei können Signale aus anderen BSSs das Senden entsprechend verhindern. Dieses Problem wird durch BSS Coloring gelöst, indem jedes BSS eine eigene Farbe, genauer eine Nummer bekommt. Diese wird in die gesendeten WLAN-Frames hinzugefügt. Beim Abhören kann dann nur ein Frame mit der gleichen Farbe das Medium belegen. Eine andere Farbe belegt das Medium nicht, da das dazugehörige Frame nicht zum eigenen BSS gehört. Dadurch kann ein Frame möglicherweise schneller gesendet werden und dies erhöht letztendlich die Performanz.

2.3 Traffic-Control

Traffic-Control (tc) [23b] ermöglicht das Beeinflussen des Netzwerkverkehrs im Linux-Kernel. Dazu gehört einmal Scheduling, also die Auswahl des nächsten zu sendenden Paketes, sowie Traffic-Shaping zum Kontrollieren des ausgehenden Verkehrs. Dies kann verwendet werden, um zum Beispiel die Datenrate auf einen gewünschten Wert zu begrenzen. Außerdem gibt es Traffic-Policing, um zum Beispiel bestimmte eingehende Pakete zu verwerfen.

Um tc konfigurieren zu können, gibt es im Package iproute2 ein tc-Command-Line-Programm. Dieses Package ist eine Sammlung mehrerer Programme, um eine Konfiguration der verschiedenen Netzwerkkomponenten im Kernel vornehmen zu können. In dieser Arbeit wird Version 6.5.0 von iproute2 verwendet, welche aus dem offiziellen GitHub-Repository [23a] heruntergeladen werden kann. Da alle diese Programme eine einfache und schnelle Konfiguration ermöglichen sollen, werden diese über das Terminal bedient. Dementsprechend laufen diese und damit auch tc im User-Space. Da die Konfigurationen aber letztendlich im Kernel erfolgen muss, wird Kommunikation zwischen User-Space und Kernel-Space benötigt. Dazu wird Netlink verwendet, welches Inter-process Communication (IPC) zwischen Kernel- und User-Space in Linux ermöglicht. Dabei werden zuerst

die auszuführenden tc-Befehl eingelesen, um daraus eine tc-Nachricht zu generieren. Diese wird anschließend eingebettet in einer Netlink-Nachricht über einen Socket in den Kernel gesendet. Dort wird dann die entsprechende Konfiguration vorgenommen. [Her21]

2.3.1 Queueing Discipline

Das Scheduling wird durch die Queueing-Discipline (Qdisc) umgesetzt. Diese befindet sich zwischen dem Network-Layer und Interface im Linux-Network-Stack. Somit müssen alle Pakete, welche gesendet werden, die Qdisc passieren. Dementsprechend kann eine Qdisc nur an egress spezifiziert werden, da nur bei ausgehendem Verkehr Scheduling notwendig ist. Dabei hat jedes Interface eine separate Qdisc. Je nachdem, welche Art von Qdisc gesetzt ist, wird eine bestimmte Scheduling-Funktionalität erreicht. Zum Beispiel gibt es die Token-Bucket-Filter (tbf) Qdisc, welche Traffic-Shaping umsetzt. Die unterschiedliche Funktionalität wird hauptsächlich in den Methoden *enqueue()* und *dequeue()* implementiert. Dabei wird mit *enqueue()* ein neues Paket hinzugefügt und *dequeue()* entnimmt das nächste zu sendende Paket entsprechend der realisierten Funktionalität. Im restlichen Abschnitt werden die in dieser Arbeit verwendeten Qdiscs kurz beschrieben.

pfifo/bfifo Qdisc

Die pfifo Qdisc und bfifo Qdisc sind die simpelsten Qdiscs, welche jeweils nur eine First In – First Out (FIFO) Reihenfolge umsetzen. Der einzige Unterschied zwischen diesen beiden liegt beim Spezifizieren der Buffergröße beziehungsweise des Limits. Genauer liegt der Unterschied in der verwendeten Einheit, wobei das „p“ in pfifo für Paket und das „b“ in bfifo für Byte steht. [Her21]

PRIO Qdisc

Mit der PRIO Qdisc kann eine bestimmte Anzahl an Klassen (hier Bänder genannt) angelegt werden. Damit kann eine Priorisierung des Netzwerkverkehrs vorgenommen werden. Dabei hat jede Klasse eine bestimmt Priorität. Beim Senden wird schließlich die höchste Priorität bevorzugt, indem dazugehörige Pakete zuerst gesendet werden, solange sich in dieser Klasse Pakete befinden. Erst wenn diese keine Pakete mehr beinhaltet, wird mit der nächst niedrigeren Priorität weitergemacht und so weiter. Die Bestimmung, in welche Klasse ein Paket gehört, kann zum Beispiel mithilfe von Filtern, siehe dazu Abschnitt 2.3.2 oder über das ToS Feld des IPv4-Headers erfolgen.

fq_codel Qdisc

Die fq_codel Qdisc realisiert ein Fair Queuing (FQ) mit Controlled Delay (CoDel). Dazu werden die eingehenden Pakete mithilfe eines Hashwertes in verschiedene Flows eingeteilt. Dies ist ein stochastisches Verfahren, da bei der Hashfunktion möglicherweise Kollisionen auftreten können und dies führt dann dazu, das unterschiedliche Pakete in den gleichen Flow kommen können. Um letztendlich Fairness zwischen den Flows zu erreichen, wird die verfügbare Datenrate auf alle fair aufgeteilt. Jeder Flow wird außerdem durch ein CoDel Algorithmus gemanagte. Diese Qdisc ist die Standard Qdisc unter Linux und ist dementsprechend meist automatisch gesetzt.

netem Qdisc

Mit der Network Emulator (netem) Qdisc können reale Netzwerke emuliert werden, womit dann zum Beispiel neu entwickelte Verfahren oder Protokolle getestet werden können. Erreicht wird dies, indem die Eigenschaften von Netzwerken wie beispielsweise Verzögerungszeit, prozentualer Verlust oder die Reihenfolgeänderung von Paketen emuliert werden. Durch das Spezifizieren entsprechender Werte kann das Verhalten des gewünschten Netzwerks erreicht werden.

2.3.2 Filter

Mit Filtern kann der Netzwerkverkehr an einem Interface gefiltert werden. Damit kann zum Beispiel Traffic-Policing des ein- und ausgehenden Verkehrs realisiert werden. Es gibt verschiedene Typen von Filtern, wobei hier nur auf zwei eingegangen wird. Das ist einmal der Typ *basic*, welcher aus einem Match und einer Aktion besteht. Der Match gibt an, auf welche Pakete der Filter angewendet wird. Dieser kann aus Teilen des gesamten Paket-Headers bestehen, zum Beispiel aus der Quell-IP-Adresse oder dem Ziel-Port. Die Aktion gibt an, was mit Paketen gemacht werden soll, welche den Match erfüllen. Mögliche Aktionen sind zum Beispiel die Modifikation von Feldern des Paket-Headers, das Verwerfen von Paketen oder das Duplizieren von Paketen.

Außerdem gibt es den Filter Typ *matchall*. Bei diesem muss kein Match spezifiziert werden, da wie der Name schon sagt, der Filter auf alle Pakete angewendet wird. Somit muss bei diesem Typ nur die Aktion angegeben werden. Dieser Typ ist zum Beispiel sinnvoll, wenn alle empfangenen Pakete über ein Interface auf ein anderes Interface dupliziert werden sollen.

Um die Filter setzen zu können, wird eine spezielle Qdisc, die clsact Qdisc benötigt. Diese ist nur für das Hinzufügen von Filtern und realisiert somit kein Scheduling. Die clsact Qdisc erlaubt das setzten von Filter, welche entweder den eingehenden (ingress) oder ausgehenden (egress) Verkehr filtern. Somit muss dies beim Erzeugen der Filter noch spezifiziert werden, indem entsprechend ingress oder egress angegeben wird.

2.4 Verwandte Arbeiten

In den letzten Jahren gab es viele Veröffentlichungen in dem Themenbereich faire Koexistenz von WiFi und Long Term Evolution (LTE) im unlizenzierten Frequenzbereich [CL15; CLGS17; KLP16; NMK+23]. Diese beschreiben verschiedene faire Mechanismen, welche es ermöglichen das WiFi und LTE ohne Probleme den gleichen Frequenzbereich verwenden können. Da in dieser Arbeit nur WLAN allein betrachtet wird, ist dieser Themenbereich nicht von Bedeutung.

Es gibt eine Reihe von Arbeiten, die versuchen, die verfügbare Bandbreite bei dem IEEE 802.11e Standard fair aufzuteilen. Riza et al. [RLPP10] entwickelte ein Dynamic Weighted Fair Scheduling Scheme, welches den vier ACs dynamische Gewichte zuweist, um dadurch die Bandbreite dynamisch anzupassen. Eine andere Richtung sind dynamische Prioritäten, welche erreicht werden, indem die Parameter der ACs entsprechend angepasst werden. He und Fang [HF09] ändern die Parameter entsprechend einer definierten Prioritätsfunktion. Diese berücksichtigt die Wartezeit der Pakete in der jeweiligen Queue und die Belegungszeit des Mediums der AC. Im Vergleich dazu wird in [GF07] nur der AIFS dynamisch angepasst. Dazu wird die erwartete und tatsächliche Übertragungszeit

verwendet. Da diese Verfahren für eine dynamische Priorität auf dem IEEE 802.11e Standard aufbauen, sind diese klassenbasiert und dementsprechend auf die vier vorhandenen ACs beschränkt. Das in dieser Arbeit entwickelte Scheduling-Verfahren ist Stream-basiert, welches den Vorteil bietet, das jeder Stream beziehungsweise STA entsprechend separat behandelt werden kann. Außerdem wird in Kapitel 3 erläutert, dass nicht die drahtlose Übertragung den Bottleneck darstellt. Dies bedeutet, dass mit diesen Verbesserungen das hier untersuchte Problem nicht gelöst werden kann.

Valenzuela et al. [VMS+04] entwickelten einen Hierarchical Token Bucket Algorithmus, um QoS im WLAN zu ermöglichen. Dabei wurde davon ausgegangen, dass die Funkverbindung der Bottleneck im WLAN darstellt. Wie in Kapitel 3 erläutert wird, befindet sich der Bottleneck heutzutage aber am Uplink zum Internet. Da die verwendet Hierarchical Token Bucket (HTB) Qdisc entsprechend vorher konfiguriert werden muss, ist dies kein dynamisches Scheduling.

In [KMG19] wird eine faire Aufteilung der Datenrate mithilfe eines Time Token Bucket beschrieben. Dabei wird eine minimale und bevorzugte Datenrate für jede STA definiert. Die minimale muss immer gegeben sein und die restliche Datenrate wird dann entsprechend fair zugeteilt. Dies Zuteilung erfolgt dynamisch, um auf neue STA oder Topologie Änderungen reagieren zu können. Somit wird hier die Datenrate fair aufgeteilt. Im Gegensatz dazu wird in dieser Arbeit eine andere Art von Fairness definiert, siehe dazu Kapitel 4.

In [Kol01] wird die Fairness zwischen Transmission Control Protocol (TCP) und User Datagram Protocol (UDP) betrachtet. Da UDP im Vergleich zu TCP nicht auf eine Überlast reagiert, entsteht dadurch eine unfaire Aufteilung der Datenrate. Gelöst wird dieses Problem dadurch, das TCP und UDP getrennt werden, indem eine Einteilung in unterschiedliche Klassen erfolgt. Mithilfe eines 3 Color Markers wird dann die faire Aufteilung der Datenrate realisiert. Im Gegensatz zu dieser Arbeit ist diese Umsetzung nicht dynamisch, dementsprechend erfolgt die Aufteilung immer gleich.

3 Überlastsituation

In einem WLAN-Netzwerk im Infrastruktur-Modus gibt es vier Stellen, an den eine Überlastsituation entstehen könnte. Diese Stellen sind in Abbildung 3.1 entsprechend markiert, wobei beispielhaft nur ein AP dargestellt ist. Zum Analysieren und um eine Aussage treffen zu können, ob dort ein Bottleneck vorliegt, werden diese vier Szenarien im Folgenden genauer untersucht. Am Ende des Kapitels wird noch die Fairness zwischen den Transportprotokollen untereinander untersucht.

3.1 Technische Annahmen

Für die Analyse der Szenarien werden folgende Annahmen verwendet. Als Downlink und Uplink zum Internet wird eine Datenrate von jeweils 100 MBit/s angenommen. Das DS ist als Gigabit-Ethernet implementiert. Als WLAN-Router beziehungsweise AP wird der *RT-AX53U AX1800*[1] von Asus verwendet. Dieser ist ein WiFi 6 Dual Band Router mit jeweils zwei Antennen für das 2,4-GHz- und 5-GHz-Band. Bei der Bezeichnung *AX1800* beschreibt das *AX* den IEEE 802.11ax Standard und die *1800* steht für die maximal zu erreichende Datenrate von circa 1,8 GBit/s. Diese setzt sich aus 574 MBit/s über das 2,4-GHz-Band und 1201 MBit/s über das 5-GHz-Band zusammen. Außerdem unterstützt dieser Router die MU-MIMO und OFDMA Technologie. Die Ethernetports des Routers sind Gigabit-Ports.

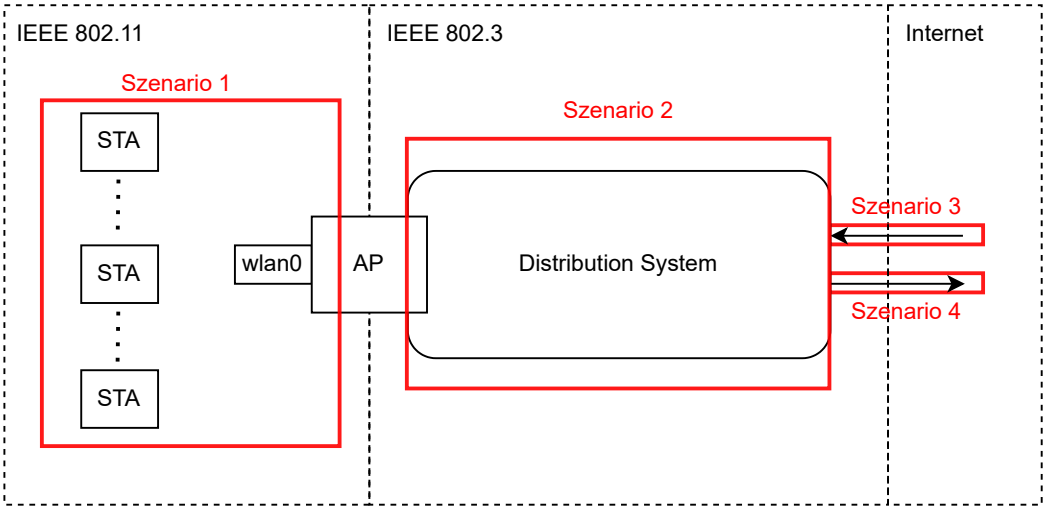


Abbildung 3.1: Mögliche Stellen in einem WLAN-Netzwerk im Infrastruktur-Modus, an welchen eine Überlast entstehen kann.

[1] https://www.asus.com/de/networking-iot-servers/wifi-routers/asus-wifi-routers/rt-ax53u/

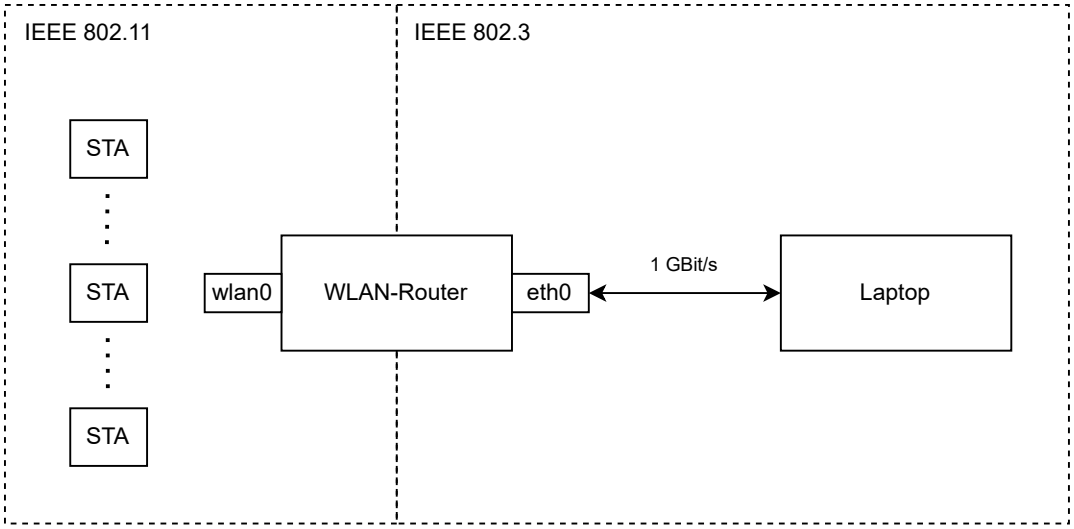


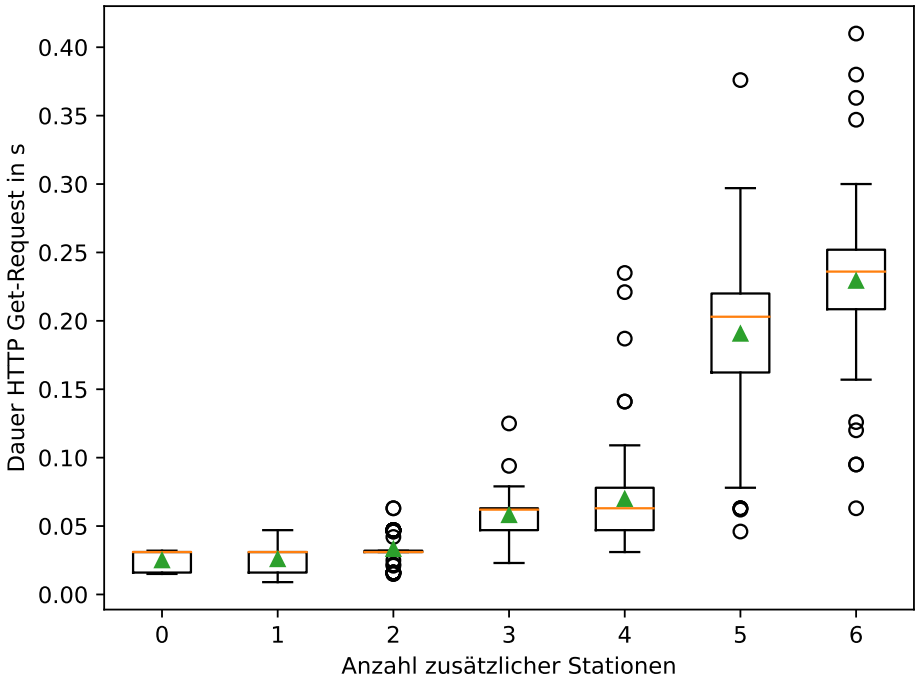
Abbildung 3.2: Setup für die Messung über die Auswirkungen der Anzahl an STAs auf die Funkverbindung.

3.2 Szenario 1: Funkverbindung ist Bottleneck

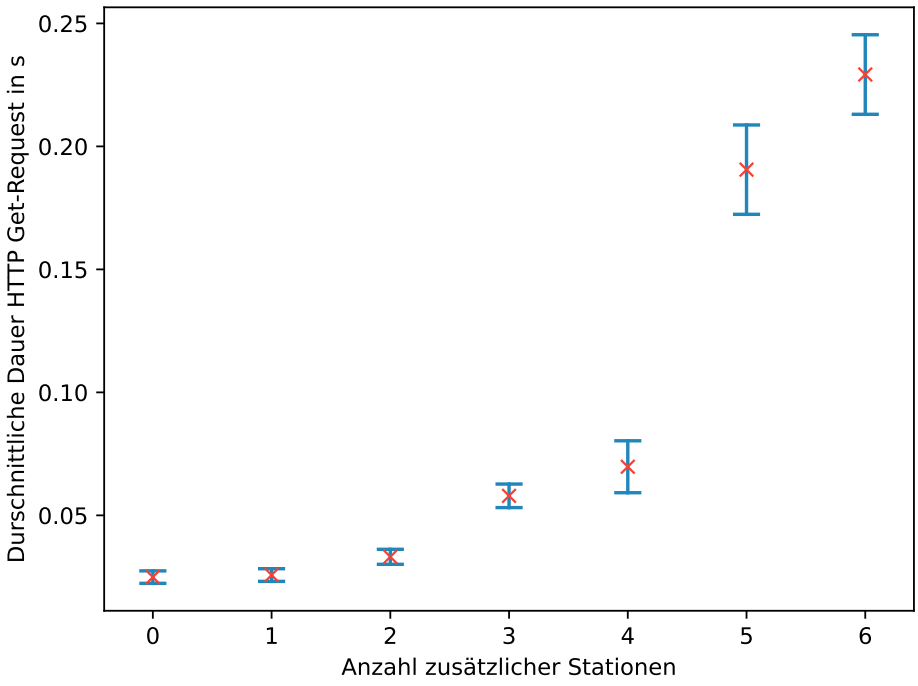
Die Funkverbindung, also zwischen AP und STAs, stellt nicht den Bottleneck dar, denn der Downlink hat eine Datenrate von 100 MBit/s und diese wird durch den AP unterstützt. In Uplink Richtung steht dann die maximale Datenrate des APs abzüglich der 100 MBit/s zur Verfügung. Dies ist um ein Vielfaches größer als die Datenrate des Uplinks, somit stellt dies auch kein Bottleneck dar.

Da sich alle aktiven STAs das Medium teilen, könnte durch eine hohe Anzahl an STAs ein Bottleneck entstehen. Um dies zu untersuchen, werden Messungen mit dem in Abbildung 3.2 dargestellten Setup durchgeführt. Als WLAN-Router wird der oben beschriebene genutzt, wobei alle STAs das 5-GHz-Band verwenden. Es werden mehrere Messungen mit unterschiedlicher Anzahl zusätzlicher STAs durchgeführt. Diese senden so schnell es geht UDP Pakete mit 50 Byte Daten zum Laptop. Die kleine Datenmenge wurde gewählt, damit die STAs öfter auf das Medium wie in Abschnitt 2.2.1 beschrieben, zugreifen müssen. Außerdem wird dadurch mit geringer Datenrate gesendet. Dies ist wichtig, damit die Summe der Datenraten aller STAs kein Problem für Router oder DS darstellt. Auf dem Laptop wird ein Hypertext Transfer Protocol (HTTP)-Server gehostet. Von einer STA aus wird die Zeit gemessen, wie lange ein erfolgreicher Get-Request zu diesem Server benötigt. Dies ist die Zeit zwischen dem Aufruf des Get-Requests bis zum vollständigen Erhalt der circa 100 kByte großen Hypertext Markup Language (HTML)-Datei. Für die Auswertung wird der Get-Request 100-mal ausgeführt und gemessen.

In Abbildung 3.3a sind die gemessenen Zeiten unter verschiedener Anzahl STAs aufgetragen. Darin kann gesehen werden, je mehr zusätzliche STAs es sind, desto länger dauern die einzelnen Get-Requests. Außerdem kann an der Größe der Boxplots gesehen werden, dass die Schwankungen zwischen den einzelnen Messungen bei größerer Anzahl STAs stärker werden. Der Grund hierfür ist das zufallsbasierte MAC Protokoll. In Abbildung 3.3b sind noch die Mittelwerte mit den 99,9 % Konfidenzintervallen dargestellt. Darin kann relative gut das exponentielle Verhalten gesehen werden, welches durch den Binary Exponential Backoff Algorithmus im MAC Protokoll entsteht.



(a) Boxplots.



(b) Mittelwerte mit 99,9 % Konfidenzintervall.

Abbildung 3.3: Ergebnis der Messungen über die Anzahl STAs.

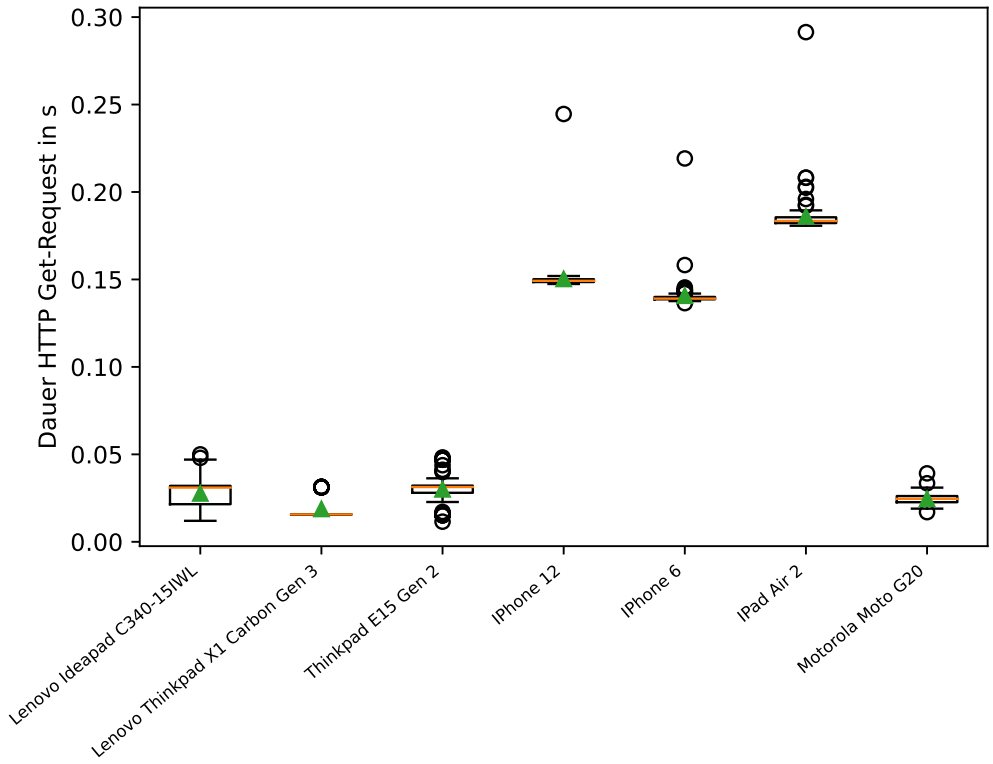


Abbildung 3.4: Performanz der verschiedenen verwendeten STAs.

Durch die Messungen wurde gezeigt, je mehr aktive STAs sich im WLAN befinden, desto länger dauert der Zugriff auf das Medium und damit auch die Antwortzeit des Servers. Letztendlich hängt dies auch stark von dem verwendeten AP ab. Dabei spielen Eigenschaften wie Bandbreite, Frequenz und Anzahl Antennen eine entscheidende Rolle. Doch diese Aspekte können nicht kontrolliert werden und falls der AP die Schwachstelle ist, sollte ein leistungsstärkerer AP angeschafft werden oder weitere APs hinzugefügt werden, um damit die Anzahl an STAs pro AP reduzieren zu können. Mit dem in [GK08] entwickeltem Algorithmus kann der Association-Prozess zwischen STA und APs dynamisch erfolgen, um so die Last möglichst auf die APs zu verteilen.

Außerdem unterscheidet sich die Performanz der verschiedenen STAs. Um dies zu verifizieren, wurde mit dem Setup in Abbildung 3.2 Messungen durchgeführt. Dabei wurde von jeder STA aus 100-mal der Get-Request zum Server gemessen, wobei sich nur die entsprechende STA im WLAN befand. In Abbildung 3.4 sind die Ergebnisse der einzelnen verwendeten STAs dargestellt. Darin kann der Unterschiede in der Performanz zwischen den STAs eindeutig gesehen werden. Somit hat die STA, welche die Get-Requests sendet, auch einen Einfluss auf die Antwortzeit des Servers.

Um an der Funkverbindung etwas zu verbessern, müsste das entsprechende MAC Protokoll angepasst werden. Doch dies wären Änderungen am IEEE 802.11 Standard, welche sich in der Praxis nicht durchsetzen würden. Außerdem müssten diese auch abwärtskompatibel sein, um die Verwendung alter STAs weiterhin zu unterstützen und dies stellt ein Problem für Verbesserungen dar. Deshalb wird dies in dieser Arbeit nicht weiter verfolgt.

3.3 Szenario 2: Distribution System ist Bottleneck

Das DS stellt nicht den Bottleneck dar, denn der Downlink von 100 MBit/s kann dieses mit einer maximalen Datenrate von 1 GBit/s ohne Probleme verarbeiten. Die 100 MBit/s des Uplinks kann auch problemlos durch das DS geleistet werden. Falls das DS falsch konfiguriert oder aufgebaut ist, könnte dieses trotzdem der Bottleneck sein. Doch diesen Fall schließen wir in dieser Arbeit aus und somit stellt das DS nicht den Bottleneck dar.

3.4 Szenario 3: Downlink ist Bottleneck

Die Verarbeitung der eingehenden Daten beim DS stellt kein Bottelneck dar, denn diese kommen mit einer maximale Datenrate von 100 MBit/s und das DS unterstützt bis zu 1 GBit/s. Somit können die empfangenen Daten durch das DS ohne Probleme verarbeitet und weitergeleitet werden. Der Internet-Provider drosselt aber letztendlich die Datenrate entsprechend des gebuchten Tarifs (hier 100 MBit/s). An dieser Stelle beim Internet-Provider kann eine Überlast entstehen. Das bedeutet, dass es am Downlink ein Bottleneck gibt, doch dieser befindet sich nicht im lokalen WLAN. Somit kann an dieser Stelle nicht direkt irgendetwas umgesetzt werden. Doch mit dem in Abschnitt 4.4 beschriebenen Feature kann dennoch Einfluss auf diesen Bottleneck genommen werden.

3.5 Szenario 4: Uplink ist Bottleneck

Der Uplink stellt ein Bottleneck dar, denn über das DS kann eine Datenrate von bis zu 1 GBit/s erreicht werden, der Uplink aber nur maximal 100 MBit/s hat. Somit kann es an dieser Stelle zu einer Überlast kommen.

Um dies zu zeigen, wird eine Messung mit dem in Abbildung 3.5 dargestellten Setup durchgeführt. Dabei wird ein 100 MBit/s Hardware-Switch verwendet, um die begrenzte Datenrate zum Internet zu simulieren. Als Software-Switch wird die im Linux-Kernel vorhandene Linux-Bridge verwendet, welche im Setup das DS darstellt. Dabei ist am Interface *eth1* die pfifo Qdisc mit einem Limit von 1000 Paketen gesetzt, welche eine simples FIFO Verhalten umsetzt. Gemessen wird die Dauer eines HTTP Get-Requests zum Server, wobei die angefragte HTML-Datei circa 100 kByte groß ist. Für den Get-Request wird eine Timeout von 10 Sekunden definiert. Es werden mehrere Messungen unter verschiedener Last durchgeführt. Diese Last wird erzeugt, indem von einer STA aus UDP Pakete mit der entsprechenden Datenrate gesendet werden. Die Messungen werden im 5-GHz-Band durchgeführt.

Für die Messungen soll der Get-Request jeweils 400-mal gemessen werden. Da die Verbindung aber manchmal nicht aufgebaut werden kann oder es zu einem Timeout kommt, werden mehr Requests benötigt. Der prozentuale Anteil an zusätzlichen Requests ist in Abbildung 3.6b dargestellt. Darin ist zu erkennen, dass bei höherer Last zusätzliche Requests benötigt werden, denn dabei ist die Wahrscheinlichkeit, das Pakete verworfen werden, entsprechend größer.

In Abbildung 3.6a sind die Mittelwerte inklusive den 99,9 % Konfidenzintervalle der jeweils exakt 400 gemessenen Get-Requests unter verschiedener Last aufgetragen. Darin kann gesehen werden, solange die zusätzliche Last kleiner oder gleich 90 MBit/s ist, bleibt die mittlere Antwortzeit

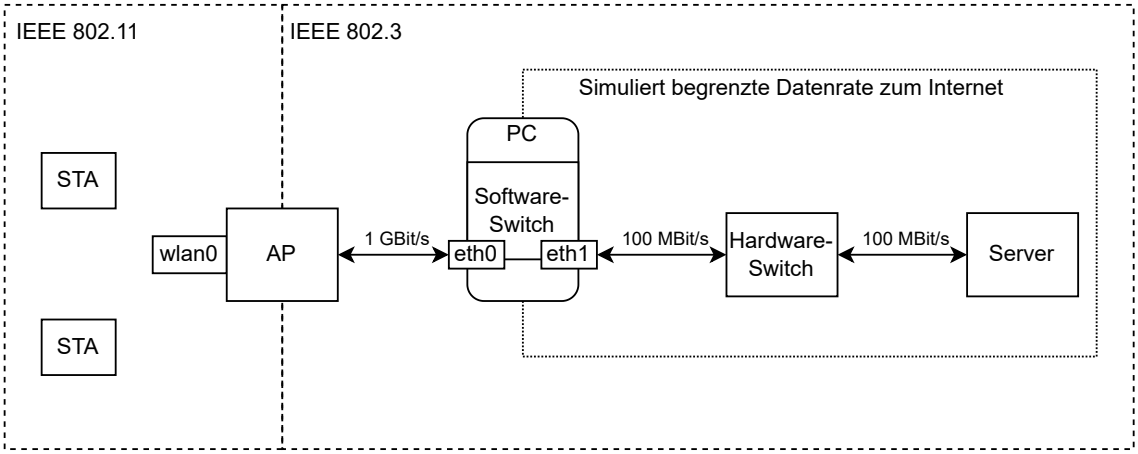
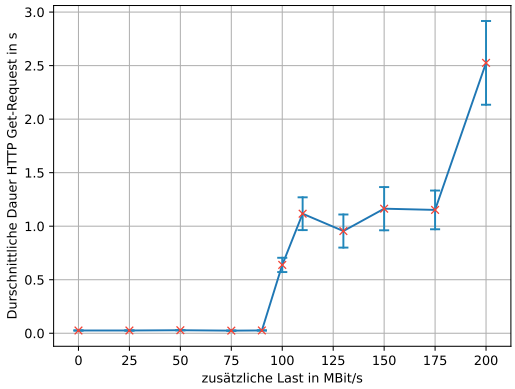
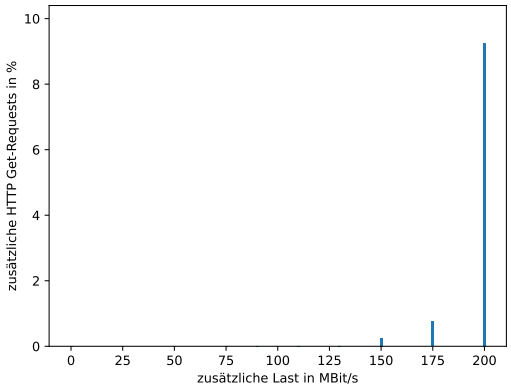


Abbildung 3.5: Setup zum Erzeugen einer Überlastsituation am Uplink.



(a) Mittelwerte mit 99,9 % Konfidenzintervall.



(b) Zusätzlich benötigte Get-Requests im Bezug zur Gesamtzahl an Messungen.

Abbildung 3.6: Ergebnis der Messungen der Überlast am Uplink.

identisch. Dies ist zu erwarten, da der Hardware-Switch diese Last ohne Probleme verarbeiten kann. Wenn die zusätzliche Last 100 MBit/s erreicht, also gerade die maximale Datenrate des Hardware-Switchs, wird die mittlere Antwortzeit des Servers um circa das 24-fache schlechter. Diese Verschlechterung ist das erwartete Verhalten, da es nun zu einer Überlast kommt und dementsprechend Pakete am Interface *eth1* verworfen werden. Steigt die zusätzliche Last weiter, wird die Antwortzeit entsprechend schlechter, wobei dies nicht monoton ist. Der Grund hierfür ist, dass die Pakete komplett zufällig verworfen werden. Entscheidend dafür ist ob die Queue zum Empfangszeitpunkt gerade voll ist oder nicht. Bei zusätzlicher Last von 200 MBit/s, also dem doppelten der maximalen Datenrate, wird die durchschnittliche Antwortzeit noch mal deutlich schlechter.

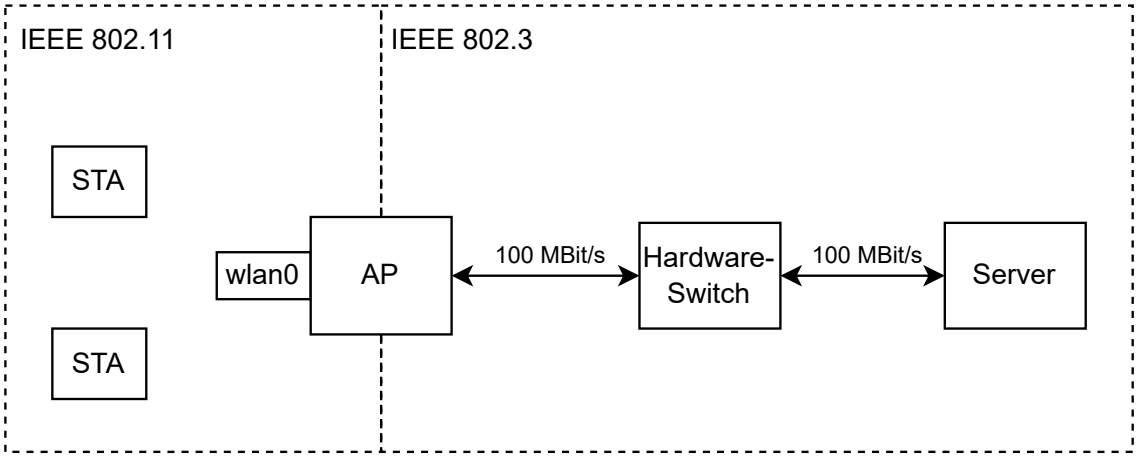


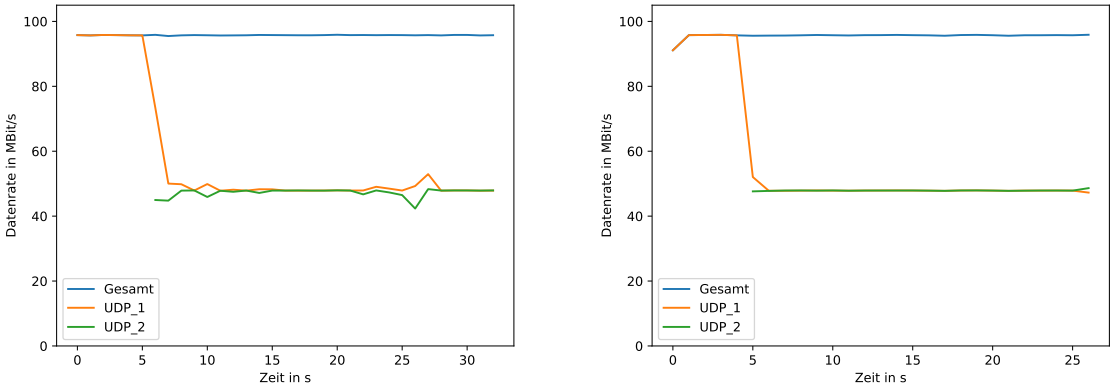
Abbildung 3.7: Setup für die Untersuchung von Fairness zwischen Transportprotokollen.

3.6 Fairness von Transportprotokollen

In diesem Abschnitt wird der Einfluss der Transportprotokolle auf die Fairness in Überlastsituationen untersucht. Betrachten wir zunächst, wie sich gleiche Transportprotokolle zueinander verhalten. Bei TCP Verbindungen teilt sich die verfügbare Datenrate immer entsprechend fair auf alle auf. Dies ist zu erwarten, da TCP ein Mechanismus zur Überlastkontrolle (engl. congestion control) hat. Dieser reagiert entsprechend, wenn zum Beispiel Timeouts auftreten. Dies bedeutet, dass Pakete nicht angekommen sind und deutet letztendlich auf eine Überlast hin.

UDP hingegen hat keine Überlastkontrolle und deswegen wird mit Messungen untersucht, wie diese sich verhalten. Die Messungen werden mit dem in Abbildung 3.7 dargestellten Setup durchgeführt. Dabei wird mithilfe des Hardware-Switches die Datenrate auf 100 MBit/s begrenzt. Die beide STAs senden jeweils UDP Pakete mit 500 Byte Daten zum Server. In Abbildung 3.8a ist das Ergebnis der Messung zusehen, wenn beide STAs unbegrenzt schnell senden. In diesem Fall bekommt jede STA circa 50 MBit/s, was genau der Hälfte der maximalen Datenrate entspricht. Somit wird hierbei die Datenrate fair aufgeteilt. In Abbildung 3.8b ist das Ergebnis einer weiteren Messung zusehen, wobei mit 70 MBit/s und 100 MBit/s gesendet wurde. Dabei wird die verfügbare Datenrate wieder gleich aufgeteilt. Die jeweiligen STAs bekommen aber prozentual gesehen unterschiedlich viel von der gewünschten Senderate, nämlich 70 % beziehungsweise 50 %. Außerdem ist der Anteil an verworfenen Paketen bei der einen STA mit der höheren Senderate größer. Dennoch kann gesagt werden, dass sich UDP Ströme zueinander fair verhalten.

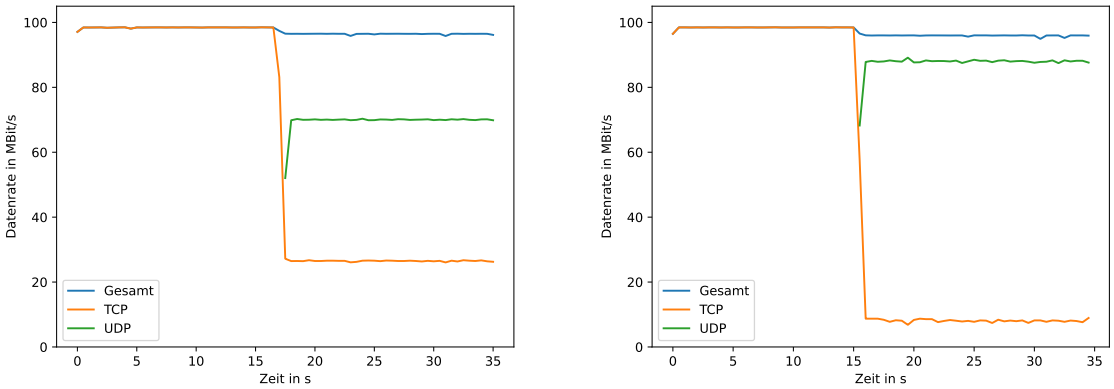
Betrachten wir jetzt die Fairness zwischen unterschiedlichen Transportprotokollen. Dazu werden Messungen mit dem in Abbildung 3.7 dargestellten Setup durchgeführt, wobei eine STA TCP Pakete und die andere UDP Pakete mit 500 Byte Daten sendet. Dabei wird zuerst die TCP Verbindung zum Server aufgebaut und mit dem Senden begonnen. In Abbildung 3.9a ist das Ergebnis zusehen, wobei der UDP Sender mit 70 MBit/s erst nach circa 15 Sekunden mit Senden beginnt. Darin kann gesehen werden, dass UDP seine komplette Datenrate bekommt und TCP nur die restlich zu Verfügung stehende Datenrate. Somit wird TCP benachteiligt und es gibt keine Fairness zwischen TCP und UDP. In Abbildung 3.9b ist das Ergebnis einer weiteren Messung dargestellt, wobei



(a) Senden jeweils so schnell wie möglich.

(b) Senden mit 100 MBit/s und 70 MBit/s.

Abbildung 3.8: Ergebnis der Messung von Fairness zwischen UDP und UDP.



(a) UDP sendet mit 70 MBit/s.

(b) UDP sendet so schnell wie möglich.

Abbildung 3.9: Ergebnis der Messung von Fairness zwischen TCP und UDP.

aber die STA mit UDP so schnell wie möglich sendet. Hierbei bekommt UDP eine Datenrate von ungefähr 90 MBit/s und TCP ungefähr nur 10 MBit/s. Das bedeutet letztendlich, dass TCP durch UDP größtenteils verdrängt wird und dies ist nicht fair.

Mit Quick UDP Internet Connections (QUIC) wurde ein neues Transportprotokoll entwickelt, welches auf UDP aufbaut, aber zuverlässig und verbindungsorientiert ist. Dementsprechend ist zum Beispiel eine Überlastkontrolle vorhanden. Somit müsste zwischen TCP und QUIC eine faire Aufteilung erfolgen, da beide eine Überlastkontrolle verwenden. Dies muss aber nicht in allen Fällen sein, wie in [CTG+19] gezeigt wurde. Wenn bei einer QUIC Verbindung mehrere Streams genutzt werden, bekommen diese insgesamt im Vergleich zur TCP Verbindung eine höhere Datenrate. Somit ist auch für die Kombination TCP mit QUIC Fairness nicht gegeben, da TCP benachteiligt wird. Zusammenfassend kann gesagt werden, dass generell keine Fairness zwischen den Transportprotokollen erwartet werden kann.

4 Entwurf

In diesem Kapitel wird der Entwurf des Stream-basierten, dynamischen und fairen Scheduling-Verfahrens beschrieben, um die Qualität und Geschwindigkeit im WLAN-Netzwerk zu verbessern, indem Fairness umgesetzt wird. Eine Möglichkeit für Fairness ist, die insgesamt verfügbare Datenrate dynamisch und fair auf alle aktiven Teilnehmer aufzuteilen. Doch diese Art von Fairness ist, um das Ziel dieser Arbeit zu erreichen, nicht anwendbar. Deshalb definieren wir zunächst die in dieser Arbeit benötigte Art von Fairness.

Definition 4.0.1 (Datenverbrauch-Fairness)
Datenverbrauch-Fairness ist gegeben, wenn alle aktiven Teilnehmer mit identischer Priorität in einem definierten vergangenen Zeitintervall annähernd die gleiche Datenmenge verbraucht haben. Dabei bekommen Teilnehmer mit hohem Datenverbrauch eine niedrigere Priorität.

Aus dieser Definition folgt, dass STAs, die in letzter Zeit wenig Daten verbraucht haben, eine höhere Priorität bekommen. Höhere Priorität bedeutet dabei, dass die Pakete dieser STA beim Senden bevorzugt werden. Dadurch werden immer STA mit einem geringen Datenverbrauch bevorzugt. Dies führt bei diesen STAs dann zu einer schnelleren Antwortzeit. Im Rest dieses Kapitels wird zuerst die grundlegende Architektur des Scheduling-Verfahrens erläutert. Anschließend werden verschiedene Ansätze beschrieben, wie der aktuelle Datenverbrauch bestimmt werden kann. Danach werden zwei grundlegend verschiedene Varianten des Scheduling-Verfahrens beschrieben. Zum Schluss wird noch ein Feature am Downlink beschrieben, mit dem Datenverbrauch-Fairness auch erreicht werden kann, falls sich der Bottleneck am Downlink befindet.

4.1 Architektur

In Abbildung 4.1 ist die grundlegende Architektur des Scheduling-Verfahrens dargestellt, welche aus drei Komponenten besteht. Das ist einmal eine Komponente für das Monitoring, welche als Input alle Pakete in Uplink und Downlink Richtung bekommt. Der Grund, warum auch die Pakete des Downlinks betrachtet werden, ist der, dass die größere Datenmenge in dieser Richtung auftritt, welche deshalb auch berücksichtigt werden soll. Aus diesen Paketen wird dann der aktuelle Datenverbrauch bestimmt, genauere Erklärung siehe Abschnitt 4.2. Eine weitere Komponente realisiert das Scheduling, indem das nächste zu sendende Paket in Uplink Richtung ausgewählt wird. Diese stellt dafür eine Schnittstelle mit zwei Methoden zur Verfügung. Die erste Methode dient zum Einfügen eines zusendenden Paketes und die andere entnimmt ein Paket, welches als nächstes gesendet wird, wobei dieses die aktuell höchste Priorität hat. Ausführlicher wird die Funktionalität der Scheduling-Komponente in Abschnitt 4.3 beschrieben. Die dritte Komponente realisiert ein Feature am Downlink, um Datenverbrauch-Fairness auch sicher zustellen, falls sich der Bottleneck am Downlink befindet. Genaueres zu diesem Feature wird in Abschnitt 4.4 erläutert.

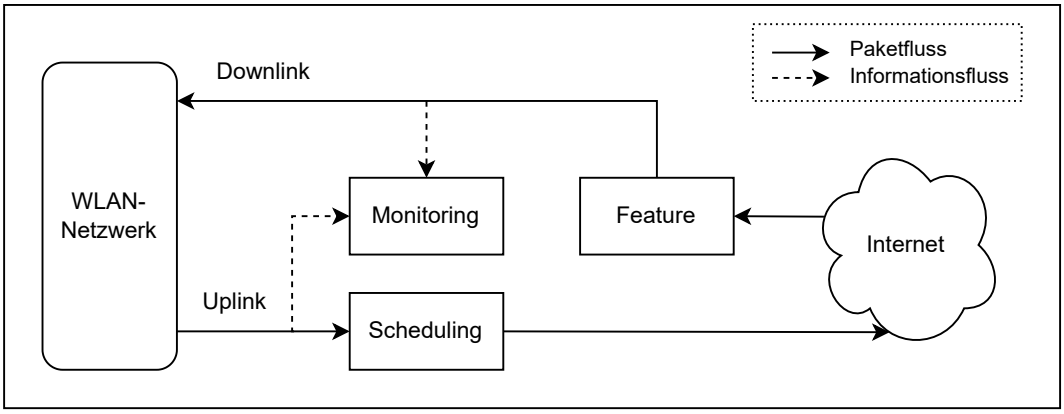


Abbildung 4.1: Grundlegende Architektur des Scheduling-Verfahrens.

4.2 Bestimmung Datenverbrauch

Da das Scheduling Stream-basiert sein soll, muss der Datenverbrauch pro Endgerät bestimmt werden. Zur Zuordnung der Pakete auf die verschiedenen Endgeräte wird die jeweilige IP-Adresse verwendet. Der Datenverbrauch jedes Gerätes wird zusammen mit der dazugehörigen IP-Adresse in einer Datenstruktur gespeichert. Damit diese immer einen aktuellen Datenverbrauch beinhaltet, muss dieser entsprechend periodisch aktualisiert werden. Dies ist wichtig, damit daraus die korrekten Prioritäten bestimmt werden können. Im Folgenden werden vier unterschiedliche Ansätze zur Bestimmung des Datenverbrauchs beschrieben.

4.2.1 Ansatz 1: Unbegrenzt-exakt

Bei diesem Ansatz wird durch das Monitoring für jedes Paket die Größe und ein Zeitstempel gespeichert. Aus diesen gespeicherten Informationen wird durch summieren aller Paketgrößen, welche im betrachteten vergangenen Zeitintervall liegen, der Datenverbrauch berechnet. Diese Berechnung erfolgt dann entsprechend periodisch, damit immer ein aktueller Datenverbrauch zur Verfügung steht. Der Vorteil von diesem Ansatz ist, dass somit immer der exakte Verbrauch bestimmt wird. Das Problem dabei ist aber, das dafür alle Pakete im betrachteten Zeitintervall gespeichert werden müssen. Dies bedeutet, dass beim Monitoring ein entsprechend großer Speicherplatz benötigt wird. Da das Speichern für jedes Endgerät separat erfolgen muss, wäre der Speicherbedarf dafür sehr groß und somit ist dieser Ansatz praktisch nicht realisierbar.

4.2.2 Ansatz 2: Begrenzt-approximiert

Dieser Ansatz beschreibt eine Verbesserung von Ansatz 1 im Bezug auf den benötigten Speicherplatz. Umgesetzt wird dies, indem die Größe der Datenstruktur des Monitorings begrenzt wird. Dies hat dann zur Folge, dass nur eine maximale Anzahl von Paketen gespeichert werden kann. Die Berechnung des Datenverbrauchs erfolgt durch summieren der Paketgrößen, wobei aber möglicherweise die Pakte für das zu betrachtende Zeitintervall nicht ausreichen. Wenn dies der Fall ist, wird der Verbrauch der fehlenden Zeit approximiert. Dazu wird aus den vorhandenen Paketen

ein mittlerer Datenverbrauch pro Zeiteinheit bestimmt. Dieser wird anschließend so oft wie nötig zur bisherigen Summe addiert, sodass das betrachtete Intervall komplett berücksichtigt ist. Somit ist dies eine Worst Case Approximation, da davon ausgegangen wird, dass im gesamten Intervall Daten verbraucht wurden. Der Vorteil bei diesem Ansatz ist, dass der Speicherbedarf begrenzt ist. Da der Datenverbrauch aber möglicherweise approximiert ist, ist dies nicht immer der exakte Verbrauch. Dies stellt aber kein großes Problem dar, denn ein ungefährer Verbrauch ist hierfür ausreichend.

4.2.3 Ansatz 3: Gewichtete Formel

Bei diesem Ansatz muss durch das Monitoring nur die letzten i Paketgrößen gespeichert werden. Daraus wird dann mit $verbrauch = verbrauch \cdot w + \sum_i paket_i \cdot w_i$ der neue Verbrauch berechnet, wobei die $w's$ unterschiedliche Gewichte darstellen. Beim periodischen Update wird diese Formel dann immer entsprechend neu ausgewertet. Wenn keine neuen Pakete hinzukommen, muss der Datenverbrauch kleiner werden. Doch dies ist mit der Formel schwierig umzusetzen. Eine Lösung dafür wäre das jedes Paket nur einmal bei der Bestimmung berücksichtige wird. Dadurch und das bei diesem Ansatz das betrachtete Zeitintervall nicht direkt abgebildet werden kann, führt zu einem ungenauen Verbrauch. Außerdem ist die Schwierigkeit bei diesem Ansatz, die unterschiedlichen Gewichte sinnvoll zu bestimmen und deshalb wird dieser hier nicht implementiert.

4.2.4 Ansatz 4: Additive Erhöhung-prozentuale Verringerung

Bei diesem Ansatz addiert das Monitoring die Paketgröße direkt zum vorhandenen Datenverbrauch in der entsprechenden Datenstruktur. Das bedeutet, dass keine separate Datenstruktur für das Monitoring benötigt wird. Damit der Verbrauch, wenn keine neuen Pakete kommen, kleiner wird, wird dieser periodisch um einen definierten Prozentsatz reduziert. Der Nachteil daran ist, dass der bestimmte Verbrauch sehr stark von der Periode und dem Prozentsatz abhängt. Dementsprechend ist es schwierig, dafür gute Werte zu finden, welche für jegliches Sendeverhalten einen akzeptablen Verbrauch liefern. Somit kann der berechnete Datenverbrauch möglicherweise stark vom tatsächlichen abweichen. Der entscheidende Grund hierfür ist, dass auch bei diesem Ansatz das betrachtete Zeitintervall nicht abgebildet werden kann. Dies führt somit oft zu einem sehr ungenau bestimmten Verbrauch. Demgegenüber steht aber der Vorteil, dass dieser Ansatz sehr simpel und einfach zu implementieren ist und dadurch das keine Monitoring Datenstruktur benötigt wird, ist auch der Speicherbedarf sehr gering.

4.3 Scheduling

Das Scheduling muss wie oben erwähnt zwei Methoden als Schnittstelle zur Verfügung stellen. Das ist *enqueue()* zum Einfügen eines Paketes und *dequeue()* zum Entnehmen des nächsten zu sendenden Paketes. Im Vergleich zu der grundlegenden Architektur in Abschnitt 4.1 wird hier das Monitoring in Uplink und Downlink aufgeteilt. Der Grund hierfür ist das über *dequeue()* alle Pakete zum Senden entnommen werden und dementsprechend kann auch direkt hier das Uplink Monitoring erfolgen. Das bedeutet dann, dass nur für das Downlink Monitoring eine separate Komponente benötigt wird. Außerdem bietet dies den Vorteil, falls das Downlink Monitoring nicht gewünscht wird, kann diese Komponente einfach weggelassen werden.

Im Folgenden werden zwei grundlegend verschiedene Varianten des Scheduling beschrieben, welche Datenverbrauch-Fairness realisieren. Dabei wird der Aufbau jeweils grafisch dargestellt, wobei darin auch die Komponenten des Monitorings und für das periodische Update des Datenverbrauchs mit enthalten sind. Die entsprechende Datenstruktur für das Monitoring ist nur auf einer allgemeinen Ebene vorhanden. Der Grund hierfür ist, dass die folgenden Beschreibungen unabhängig von einem konkreten Ansatz sein sollen. Dementsprechend wird davon ausgegangen, dass der Datenverbrauch mit einem in Abschnitt 4.2 beschriebenen Ansatz bestimmt wird und somit in der zugehörigen Datenstruktur immer ein aktueller Wert für jede IP-Adresse vorhanden ist. In den Grafiken ist mit gestrichelten Pfeilen dargestellt, welche Komponente welche Datenstrukturen verwendet. Genauer gesagt beschreibt ein Pfeil von einer Komponente zu einer Datenstruktur nur Schreibzugriffe. Entsprechend beschreibt ein Pfeil in die andere Richtung reine Lesezugriffe.

4.3.1 Feingranulare Variante

Der gesamte Aufbau dieser Variante ist in Abbildung 4.2 grafisch dargestellt. Die zusendenden Pakete werden intern jeweils pro Quell-IP-Adresse in FIFO Ordnung gespeichert, wobei jeweils nur eine maximale Anzahl gespeichert werden können. Das bedeutet, in *enqueue()* wird ein neues Paket in diese Datenstruktur entsprechend hinzugefügt. Falls es kein Platz mehr hat, wird das Paket verworfen. Da Pakete der Protokolle auf der Sicherungsschicht keine IP-Pakete sind, müssen solche Pakete separat gespeichert werden. Diese Pakete bekommen die höchste Priorität und werden dadurch bevorzugt von *dequeue()* ausgewählt. Wenn es keine solchen Pakete gibt, wird die IP-Adresse mit aktuell kleinstem Verbrauch bestimmt. Dabei werden nur die IP-Adressen berücksichtigt, bei denen zurzeit Pakete zum Senden vorhanden sind. Dafür muss in der Datenstruktur mit dem Datenverbrauch das Minimum bestimmt werden. Falls mehrere Minima vorhanden sind, muss eine entsprechende Regel spezifiziert werden, sodass beispielsweise immer das erste gefundene Minimum gewählt wird. Mit der zugehörigen IP-Adresse des gefundenen Datenverbrauchs kann anschließend das nächste zu sendende Paket aus der Datenstruktur entnommen werden.

Ein Vorteil dieser Variante ist, dass die Prioritäten mithilfe des aktuellen Verbrauchs erste in *dequeue()*, also beim Senden, bestimmt werden. Das bedeutet dann, dass tatsächlich immer ein Paket des Endgerätes mit aktuell geringstem Verbrauch gesendet wird. Somit sind die Prioritäten komplett dynamisch und werden feingranular bestimmt. Aber diese Variante hat im Bezug auf den benötigten Speicher einen Nachteil, denn für jede IP-Adresse muss eine entsprechend große Datenstruktur zum Speichern der Pakete zur Verfügung stehen. Außerdem erfolgt die Bestimmung des Minimums sehr oft, nämlich für jedes Paket und muss deshalb effizient möglich sein.

4.3.2 Klassenbasierte Variante

Wie im dargestellten Aufbau in Abbildung 4.3 gesehen werden kann, ist diese Variante klassenbasiert, wie beispielsweise das in [LDR+23] entwickelte Verfahren. Dabei beschreibt jede der n Klassen eine Verbrauchsklasse mit einer maximalen Größe. In diesen werden die Pakete jeweils in FIFO Ordnung gespeichert. Außerdem wird jede Klasse durch einen dazugehörigen Threshold beschrieben. Dieser gibt an, bis zu welchem Verbrauch die Pakete noch zur jeweiligen Klasse gehören. Das bedeutet somit, dass in *enqueue()* ein neues Paket anhand des dazugehörigen Datenverbrauchs in die entsprechende Klasse eingefügt wird. Falls die entsprechende Klasse voll ist, wird das Paket verworfen. Wie bei der anderen Variante sollen Pakete, die keine IP-Pakete sind, die höchste Priorität

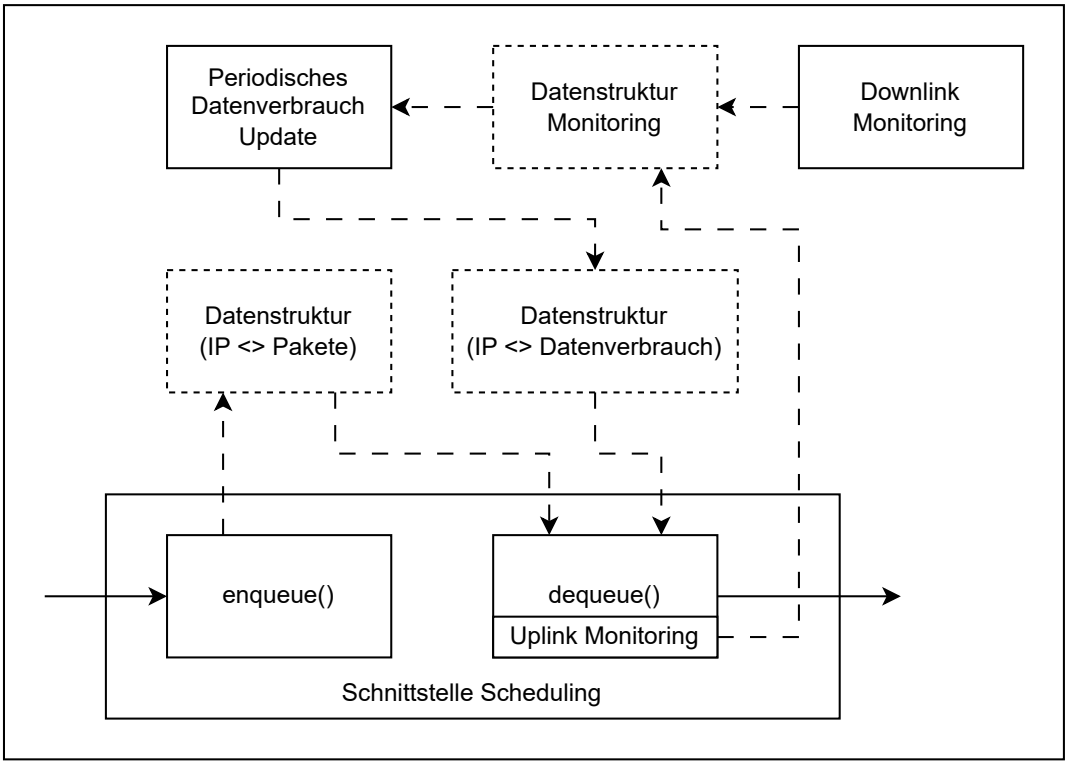


Abbildung 4.2: Aufbau der feingranularen Variante des Scheduling-Verfahrens.

bekommen und werden deshalb in Klasse C_0 hinzugefügt. In *dequeue()* wird zuerst versucht, ein Paket aus Klasse C_0 zu entnehmen, um dieses zu senden. Wenn darin keine Pakete sind, wird entsprechend in der nächsten Klasse weiter gesucht und so weiter.

Der Vorteil bei dieser Variante ist, dass der benötigte Speicherplatz durch die Anzahl an Klassen beschränkt ist. Außerdem ist die Implementierung dieser Variante im Vergleich zur anderen einfacher, da hier kein Minimum bestimmt werden muss. Ein Nachteil durch den klassenbasierten Ansatz ist jedoch, dass die Prioritäten an die Klassen gebunden sind und nicht wie bei der feingranularen Variante direkt an den Datenverbrauch. Außerdem muss für jede Klasse ein sinnvoller Threshold bestimmt werden.

4.4 Feature am Downlink

Das bisher beschrieben Scheduling-Verfahren arbeitet ausschließlich am Uplink, wo sich der Bottleneck im eigenen WLAN-Netzwerk befindet. An diesem wird der Netzwerkverkehr entsprechend dem Datenverbrauch priorisiert, um Datenverbrauch-Fairness zu erreichen. Doch wie in Kapitel 3 ausgeführt wurde, existiert am Downlink auch ein Bottleneck, wobei an diesem mit dem jetzigen Verfahren Datenverbrauch-Fairness nicht realisiert werden kann. Da sich dieser Bottleneck außerdem beim Internet-Provider befindet, kann dementsprechend an diesem nichts direkt realisiert werden. Deshalb soll mit einem Feature am Downlink versucht werden Einfluss auf diesen zunehmen. Dazu soll bei Endgeräten, welche einen hohen Datenverbrauch haben, ein gewisser Anteil an Paketen

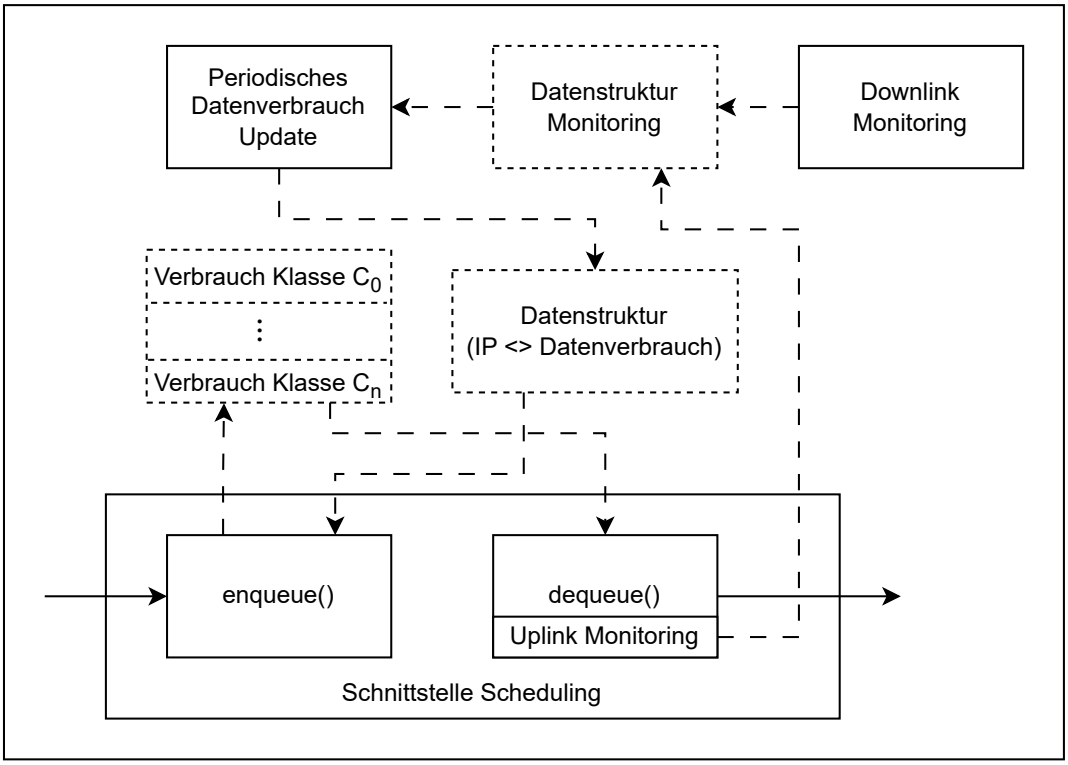


Abbildung 4.3: Aufbau der klassenbasierten Variante des Scheduling-Verfahrens.

verworfen werden. Dafür wird bei der feingranularen Variante der Wertebereich des Datenverbrauchs in Abschnitte aufgeteilt und für jeden dieser Abschnitte wird ein Prozentwert spezifiziert. Dieser gibt an, wie viel Prozent der ankommenden Pakete verworfen werden sollen. Für Abschnitte, welche einen geringen Datenverbrauch beschreiben, muss dieser Wert logischerweise null sein. Bei der klassenbasierten Variante werden die Prozentwerte entsprechend für jede Klasse spezifiziert. Durch das Verwerfen von Paketen soll erreicht werden, dass die Überlastkontrolle beim Sender auf die Verluste reagiert und die Datenrate reduziert. Dabei gilt, je größer der Prozentwert ist, desto kleiner wird letztendlich die Datenrate. Somit werden die verschiedenen Prioritäten am Downlink dadurch abgebildet, dass die Downlink-Datenrate entsprechend reduziert wird. Mit diesem Feature kann somit auch Datenverbrauch-Fairness erreicht werden, falls sich der Bottleneck hauptsächlich am Downlink befindet, indem Endgeräte mit hohem Datenverbrauch eine geringere Downlink-Datenrate bekommen.

5 Implementierung

In diesem Kapitel wird die Implementierungen des in Kapitel 4 beschriebenen Scheduling-Verfahrens erläutert, wobei hier nur die klassenbasierte Variante implementiert wird. Der Grund dafür ist, da die feingranulare Variante für jede vorhandene IP-Adresse jeweils eine geeignete Datenstruktur zur Speicherung der Pakete benötigt und somit nicht skaliert. Das Verfahren wird mithilfe zweier verschiedener Komponenten von tc realisiert, siehe rote Markierungen in Abbildung 5.1. Das ist zum einen eine Qdisc, mit der das eigentliche Scheduling umgesetzt wird und zum anderen ein Filter mit dem das Downlink Monitoring realisiert wird. Die Funktionalität des in Abschnitt 4.4 beschriebene Features wird dabei in das Downlink Monitoring integriert. Im Folgenden wird die Implementierung dieser beiden Komponenten detaillierter beschrieben. Dabei wird die Version 6.2 des Linux-Kernels verwendet. Außerdem wird für die Implementierung zuerst einmal angenommen, dass es nur IPv4-Adressen gibt. Am Ende dieses Kapitels werden anschließend noch die Änderungen beschrieben, welche vorgenommen werden müssen, um nur IPv6-Adressen zu unterstützen. Dadurch kann dann gezeigt werden, dass das Scheduling generell auch mit IPv6-Adressen funktioniert.

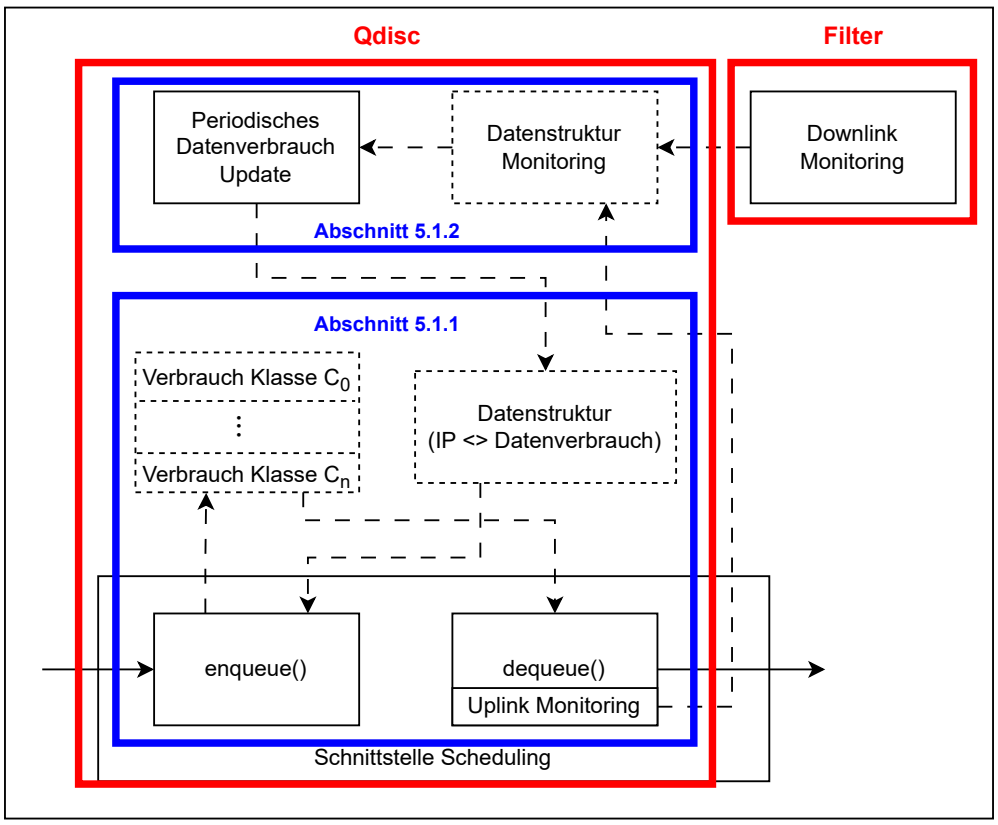


Abbildung 5.1: Implementierung der klassenbasierten Variante des Scheduling-Verfahrens.

5.1 Qdisc

In diesem Abschnitt wird die Implementierung der neuen Qdisc erläutert. Dafür wird als Basis die PRIO Qdisc verwendet und die vorhandene Funktionalität entsprechend angepasst. Der Abschnitt ist in zwei Teile aufgeteilt, wobei zuerst auf die Implementierung des Schedulings und anschließend auf die Bestimmung des Datenverbrauchs eingegangen wird. In Abbildung 5.1 ist durch die blauen Markierungen dargestellt, welche Komponenten beziehungsweise Datenstrukturen zu welchem Teil gehören.

5.1.1 Scheduling

Zuerst wird die Realisierung der beiden benötigten Datenstrukturen, also einmal für die Klassen und einmal für den Datenverbrauch beschrieben. Anschließend wird die Implementierung der Scheduling-Funktionalität erläutert.

Datenstrukturen

Die Realisierung der Klassen ist einfach, da dafür die vorhandene Implementierung der PRIO Qdisc direkt verwendet werden kann. Damit wird für jede Klasse eine pfifo Qdisc angelegt. Diese werden als Array gespeichert, wobei an Position null sich die Qdisc beziehungsweise Klasse mit der höchsten Priorität befindet. Für den Zugriff darauf als zum Beispiel das Einfügen oder Entnehmen eines Paketes sind entsprechende Funktionen vorhanden.

Für die Datenstruktur, welche den Datenverbrauch in Byte jeder IP-Adresse beinhaltet, wird zuerst ein Struct definiert. Die Definition davon ist in Listing 5.1 zusehen. Damit kann der Datenverbrauch einer IP-Adresse beschrieben werden. Als Datentyp für den Verbrauch wird ein `unsigned int` gewählt, da der mögliche Wertebereich davon unter der Annahme einer maximalen Datenrate von 100 MBit/s für ein betrachtetes Zeitintervall von mindestens 300 Sekunden ausreichend ist. Falls aber ein größeres Intervall betrachtet werden soll oder die maximale Datenrate größer ist, muss möglicherweise ein größerer Datentyp verwendet werden. Mit einem Array aus diesem Struct wird die benötigte Datenstruktur erstellt. Eine Variable spezifiziert, wie viele verschiedene IP-Adressen unterstützt werden und diese gibt die Länge des Arrays an. Für den benötigten Zugriff auf diese Datenstruktur werden zwei Funktionen implementiert. Das ist einmal eine Funktion, über die der aktuelle Verbrauch einer gewissen IP-Adresse abgefragt werden kann. Außerdem wird eine Funktion zum Aktualisieren des Verbrauchs einer bestimmten IP-Adresse benötigt. Falls beim Aktualisieren die IP-Adresse noch nicht existiert, wird ein entsprechend neuer Eintrag im Array angelegt. In beiden Funktionen wird zum Finden der benötigten IP-Adresse über das Array iteriert und jeweils die IP-Adressen verglichen.

Funktionalität

Um die erforderliche Funktionalität zu implementieren, müssen nur die Funktionen *enqueue()* und *dequeue()* entsprechend angepasst werden. In *enqueue()* muss zuerst die Klasse für das Paket bestimmt werden. Dazu wird der aktuelle Datenverbrauch über die IP-Adresse bestimmt. Wie die IP-Adresse bekommen wird, wird hier jetzt einmal genauer beschrieben. Dazu wird der Socket-Buffer

Listing 5.1 Struct für den Datenverbrauch

```
struct consumption_node {
      unsigned int ip;
      unsigned int consumption;
};
```

(SKB) verwendet. Dies ist eine Datenstruktur des Linux-Kernels, womit ein Paket im Network-Stack repräsentiert wird. Der SKB ermöglicht es, auf die Header der verschiedenen Layer des Paketes mithilfe entsprechender Structs zuzugreifen. Über den `struct iphdr` für den IPv4-Header kann die IP-Adresse dann ausgelesen werden. Falls es sich nicht um ein IP-Paket handelt, kann dies über das Protokoll Feld des Ethernet-Headers erkannt werden und es wird Klasse 0 gewählt. Ansonsten wird mit dem Verbrauch und den Thresholds die richtige Klasse gefunden. Die Thresholds befinden sich dabei in einem Array und sind in der gleichen Ordnung wie die Klassen gespeichert. In die dazugehörige Qdisc der Klasse wird das Paket dann eingefügt.

Die Funktionalität von *dequeue()* muss nicht geändert werden, da die Entnahmereihenfolge die gleiche ist wie in der PRIO Qdisc. Aber wie in Abbildung 5.1 zusehen ist, muss hier noch das Uplink Monitoring erfolgen. Dafür wird vor dem Zurückgeben des Paketes noch die Funktion *monitoring_pkt()* aufgerufen, aber nur falls es sich um ein IP-Paket handelt. Als Parameter für die IP-Adresse wird an dieser Stelle die Quell-IP-Adresse des Paketes verwendet. Eine genauere Erklärung dieser Funktion erfolgt in Abschnitt 5.1.2.

5.1.2 Bestimmung Datenverbrauch

Von den vier in Abschnitt 4.2 beschriebenen Ansätzen werden zwei implementiert. Dies sind die grundlegend verschiedenen Ansätze 2 (Begrenzt-approximiert) und 4 (Additive Erhöhung-prozentuale Verringerung). Dabei wird für das Monitoring die Funktion *monitoring_pkt()* implementiert, welche eine IP-Adresse und die Paketgröße als Parameter bekommt. Die Paketgröße beinhaltet dabei auch die Größe jeglicher Header. Die exakte Funktionalität hängt von dem verwendeten Ansatz ab und wird deshalb erst jeweils unten erläutert. Aber für beide Ansätze gilt das die Implementierung gegen Nebenläufigkeit gesichert werden muss, denn das Uplink und Downlink Monitoring erfolgt parallel. Deshalb muss diese Funktion mit einem Spinlock entsprechend abgesichert werden. Im Folgenden die Implementierung der beiden Ansätze beschrieben, wobei aber zuerst auf die allgemeine Umsetzung des periodischen Updates eingegangen wird.

Periodisches Update

Wie in Abschnitt 4.2 beschrieben wird ein periodisches Update des Datenverbrauchs benötigt. Dies wird mithilfe eines Timers umgesetzt, wobei der im Linux-Kernel vorhandenen `hrtimer` verwendet wird. Die entsprechende Periode wird in einer Variablen definiert, welche letztendlich an gibt, nach welcher Zeit der Timer auslöst. Die dann auszuführende Funktionalität wird in einer Funktion implementiert, welche dem Timer hinzugefügt wird. Am Ende dieser Funktion wird der Timer wieder mit der definierten Periode neu gestartet. Die genaue Implementierung dieser Funktion wird jeweils unten erläutert.

Listing 5.2 Structs für das Monitoring von Ansatz 2 (Begrenzt-approximiert)

```
struct packet {
      unsigned int size;
      u64 timestamp;
};

struct packet_buffer {
      struct packet * buffer;
      int head;
      int maxlen;  // enstpricht der Monitoring Größe
};

struct monitoring {
      unsigned int ip;
      struct packet_buffer packet_buf;
};
```

Ansatz 2: Begrenzt-approximiert

Für die Implementierung von Ansatz 2 wird wie in Abschnitt 4.2.2 beschrieben eine Datenstruktur für das Monitoring benötigt. Dafür werden verschiedene Structs definiert, welche in Listing 5.2 zusehen sind. Dies ist einmal `struct packet` womit ein Paket beschrieben wird. Mit diesem wird ein *packet_buffer* definiert, welcher ein Ringpuffer darstellt, wobei intern ein Array verwendet wird. Ein Ringpuffer wurde gewählt, da dieser eine feste Größe hat und wenn dieser voll ist, wird beim Einfügen automatisch immer das älteste Paket überschrieben. Für das Einfügen eines neuen Paketes wird die Funktion *add_packet()* implementiert. Die Größe des Ringpuffers, ab jetzt mit Monitoring Größe bezeichnet, wird beim Initialisieren über eine entsprechende Variable spezifiziert. Mit dem `struct monitoring` wird der Ringpuffer noch mit der zugehörigen IP-Adresse verbunden. Ein Array davon stellt dann die Monitoring Datenstruktur dar, welche separat für jede IP-Adresse die Pakete beinhaltet. Die Länge des Arrays wird von der Variable spezifiziert, welche die maximale Anzahl an unterstützen IP-Adressen beschreibt.

Die Funktion *monitoring_pkt()* fügt bei diesem Ansatz die Paketgröße dem entsprechenden Ringpuffer hinzu, wobei als Zeitstempel die aktuelle Zeit verwendet wird. Der Timer ruft bei jedem Auslösen die Funktion *generate_consumption()* auf. Diese bestimmt für jede IP-Adresse den aktuellen Datenverbrauch und schreibt diesen in die Datenstruktur für den Datenverbrauch. Die Berechnung des Verbrauchs erfolgt, indem über die Pakete des Monitorings iteriert wird, beginnend mit dem neusten. Dabei werden die Paketgrößen aufsummiert, solange der Zeitstempel des Paketes sich noch im betrachteten Intervall befindet, wobei der Wert des Intervalls in einer Variable gespeichert ist. Falls das letzte vorhandene Paket noch im betrachteten Intervall liegt, sind nicht genug Pakete gespeichert, um den Datenverbrauch exakt zu bestimmen. Dann wird wie in Abschnitt 4.2.2 beschrieben, die fehlende Zeit approximiert.

Die Berechnung der Approximation erfolgt auf Millisekunden Genauigkeit, da dies dafür ausreichend ist. Als erstes wird berechnet, welches Zeitintervall mit den vorhandenen Paketen bestimmt wurde. Falls diese Zeit kleiner als 1 Millisekunde ist, wird diese auf 1 Millisekunde gesetzt. Dieser Check

ist wichtig, da es sonst zum Teilen durch null kommen kann. Mit dieser Zeit und der Summe der Paketgrößen kann der durchschnittliche Verbrauch pro Millisekunde bestimmt werden. Dieser durchschnittliche Verbrauch muss anschließend noch entsprechend oft zum berechneten Verbrauch hinzugefügt werden, wobei die Differenz von betrachtetem und vorhandenem Zeitintervall angibt, wie oft.

Ansatz 4: Additive Erhöhung-prozentuale Verringerung

Für die Umsetzung dieses Ansatzes wird eine Multiplikation mit einem Faktor kleiner eins benötigt. Da aber im Linux-Kernel die Benutzung von Gleitkommazahlen aus Effizienzgründen nicht möglich ist, müssen dafür Festkommazahlen verwendet werden. Es werden hier zwei Nachkommastellen genutzt, da diese Genauigkeit für den Faktor ausreichend ist. Da die Multiplikation beim Datenverbrauch nötig ist, müssen alle entsprechenden Variablen als solche Zahlen implementiert werden. Dabei muss beachtet werden, dass dadurch der mögliche Wertebereich entsprechend kleiner wird. Konkret bedeutet dies, dass sich der Wertebereich bei der Benutzung von 2 Nachkommastellen um den Faktor 10^{-2} reduziert.

Bei der Verwendung von Festkommazahlen müssen auch die benötigten Berechnungen angepasst werden. Das bedeutet, bei einer Addition muss der Summand zuerst mit 100 multipliziert werden. Bei der Multiplikation mit dem Faktor muss zusätzlich mit 100 multipliziert werden, wobei das Ergebnis anschließend wieder durch 100 dividiert werden muss. Dies Berechnung führt dann aber zu großen Zwischenergebnissen, welche im jetzigen Wertebereich nicht mehr darstellbar sind. Deshalb muss bei diesem Ansatz der Datentyp des Datenverbrauchs mit `unsigned long` vergrößert werden, damit diese Berechnung korrekte Ergebnisse liefert. Bei der restlichen Implementierung dieses Ansatzes wird von der korrekten Verwendung der Festkommazahlen ausgegangen und dementsprechend nicht mehr drauf eingegangen.

In der Monitoring Funktion *monitoring_pkt()* wird einfach die Paketgröße zum aktuellen Datenverbrauch der entsprechenden IP-Adresse addiert. Die vom Timer ausgeführte Funktion aktualisiert den Datenverbrauch jeder IP-Adresse, indem dieser mit dem Faktor multipliziert wird. Der Wert des Faktors ist dabei in einer Variable definiert.

5.2 Downlink Monitoring

Das Downlink Monitoring wird wie oben erwähnt über ein tc Filter realisiert. Dabei wird der *matchall* Typ verwendet, da alle Pakete berücksichtigt werden sollen. Da der Downlink die eingehenden Pakete darstellt, wird der Filter entsprechend an egress gesetzt, um nur diese Pakete zu bekommen. Für die Umsetzung des Monitorings über den Filter wird eine neue Monitoring Action implementiert.

Als Basis für die Monitoring Action wird die vorhandene Simple Action verwendet. Wie der Name schon sagt, ist dies eine simple Action, denn diese gibt für jedes Paket nur etwas aus. Diese Ausgabe besteht aus einem beim Anlegen definierten String und einer Zahl, welche die Anzahl an bisher verarbeiteten Paketen beschreibt. Um damit das Monitoring umzusetzen, muss diese Ausgabe nur durch den Aufruf der Funktion *monitoring_pkt()*, welche in der neuen Qdisc implementiert ist, ersetzt werden. Dabei wird in diesem Fall als Parameter für die IP-Adresse die Ziel-IP-Adresse des Paketes verwendet.

Da diese Funktion in der Qdisc und somit in einem anderen Kernel-Modul implementiert ist, kann diese nicht einfach aufgerufen werden. Dafür muss in der Qdisc die Funktion zuerst mithilfe von *EXPORT_SYMBOL(monitoring_pkt)* exportiert werden. Danach kann diese Funktion dann in anderen Kernel-Modulen verwendet werden. Das bedeutet, dass dadurch aber die Monitoring Action abhängig von der Qdisc ist, da eine Funktion aus dieser genutzt wird. Diese Abhängigkeit muss beim Laden der beiden Module beachtet werden. Konkret bedeutet dies, dass zuerst das Qdisc Modul und anschließend erst das Modul der Monitoring Action geladen werden muss.

5.3 Feature am Downlink

In diesem Abschnitt wir die Implementierung des in Abschnitt 4.4 beschriebenen Features erläutert. Dabei wird die entsprechende Funktionalität in das Downlink Monitoring integriert. Dementsprechend muss eine Anpassungen an der Monitoring Action vorgenommen werden. Außerdem muss die neue Qdisc um die benötigte Funktionalität ergänzt werden. Für die Umsetzung des anteiligen Verwerfens von Paketen wird die vorhandene Implementierung der netem Qdisc verwendet. Diese realisiert den prozentualen Anteil, indem der gewünschte Prozentwert auf den Wertebereich eines `unsigned int` skaliert wird. Das bedeutet, dass 0 % dem Wert 0 und 100 % dem maximal darstellbaren Wert entspricht. Für jedes zu sendende Paket wird zuerst eine Zufallszahl aus diesem Wertebereich bestimmt. Das Paket wird anschließend verworfen, falls dieser zufällige Wert kleiner oder gleich dem skalierten gewünschten Prozentwert ist. Ansonsten wird das Paket gesendet. Somit wird mithilfe dieser Bedingung der gewünschte Anteil an Paketen verworfen.

Betrachten wir zuerst die Anpassungen, die im Kernel-Modul der Qdisc gemacht werden müssen. Als erstes muss ein Array angelegt werden, welches den Anteil der zu verwerfenden Pakete für jede Klasse enthält. Diese Werte müssen dabei wie oben beschrieben als skalierte Prozentwerte angegeben werden. Außerdem müssen diese in der gleichen Reihenfolge wie die Klassen gespeichert werden. Damit von der Monitoring Action aus der entsprechende Wert bekommen werden kann, wird die Funktion *get_loss()* implementiert. Diese hat als Parameter eine IP-Adresse, für die der dazugehörige Prozentwert benötigt wird. Über den Datenverbrauch der IP-Adresse kann die aktuelle Klasse gefunden werden und der dazugehörige skalierte Prozentwert wird zurückgegeben. Diese Funktion muss wieder mit *EXPORT_SYMBOL(get_loss)* exportiert werden, damit diese in der Monitoring Action verwendet werden kann.

Anschließend muss noch die Monitoring Action so angepasst werden, dass der entsprechende Anteil an Paketen verworfen wird. Dazu muss vor dem Funktionsaufruf des Monitorings die dafür benötigte Funktionalität ergänzt werden. Als erstes muss der Aufruf von *get_loss()* erfolgen, um den Prozentwert zu bekommen. Als Parameter wird dabei die Ziel-IP-Adresse des Paketes verwendet. Außerdem muss noch eine if-Abfrage ergänzt werden, wobei die Bedingung angibt, ob ein Paket verworfen wird oder nicht. Dazu wird die Bedingung von der netem Qdisc genutzt, welche oben beschrieben ist. Die benötigten Zufallszahlen werden wie bei der Implementierung der netem Qdisc mithilfe der Funktion *get_random_u32()* generiert. Somit ist diese Bedingung erfüllt, falls das aktuelle Paket verworfen werden muss. Dies wird erreicht, indem im if-Block die aktuelle Funktion durch das Zurückgeben von *TC_ACT_SHOT* beendet wird. Dieser Rückgabewert spezifiziert dann im Folgenden, dass das dieses Paket verworfen werden soll. Beachtet werden muss, dass davor noch der Spinlock freigegeben wird. Für den Fall, dass die Bedingung falsch ist und das Paket somit nicht verworfen wird, erfolgt der bisherige Ablauf inklusive Monitoring.

5.4 IPv6 Variante

Die bisherige Implementierung unterstützt nur IPv4-Adressen. Doch es gibt auch IPv6-Adressen, welche mit der jetzigen Implementierung nicht funktionieren. Um zu zeigen, dass das Scheduling mit diesen aber generell auch möglich ist, werden nachfolgend die Änderungen beschrieben, um IPv6-Adressen verwenden zu können. Dabei wird davon ausgegangen, dass nur IPv6-Adressen existieren.

Die erste Änderung betrifft den Datentyp der IP-Adressen. Bisher ist dieser `unsigned int`, doch für IPv6-Adressen muss dieser überall zu `struct in6_addr` geändert werden, denn so werden im Linux-Kernel diese Adressen gespeichert. Außerdem können die IPv6-Adressen nicht wie bisher mit einem einfachen Zahlenvergleich verglichen werden. Dafür muss jetzt die im Kernel vorhandene Funktion *ipv6_addr_equal()* verwendet werden. Diese nimmt als Parameter zwei IPv6-Adressen und gibt eins zurück, wenn diese identisch sind, ansonsten null. Außerdem muss beim Bestimmen der IP-Adresse eines Paketes jetzt `struct ipv6hdr` verwendet werden, da es sich logischerweise nun um einen IPv6-Header handelt. Ansonsten muss nichts mehr angepasst werden und das Scheduling funktioniert jetzt mit IPv6-Adressen.

6 Evaluierung

In diesem Kapitel wird die Implementierung des entwickelten Scheduling-Verfahrens evaluiert. Dabei wird zuerst das entwickelte Verfahren mit WMM verglichen, ob insbesondere mit WMM Fairness in einer Überlastsituation am Up- oder Downlink erreicht werden kann. Anschließend wird mit verschiedenen Metriken die neue Implementierung evaluiert.

6.1 WMM

In diesem Abschnitt wird das neue entwickelte Scheduling-Verfahren mit WMM verglichen. Zuerst wird jedoch das in Tabelle 2.2 vorhandene Default Mapping zwischen DSCP und UP durch entsprechende Messungen verifiziert.

6.1.1 Default DSCP-UP Mapping

Die Verifizierung des in Tabelle 2.2 aufgeführten Default Mappings zwischen DSCP und UP erfolgt dabei nicht direkt mit der UP, sondern über die AC. Dies ist problemlos möglich, denn in IEEE 802.11e ist das Mapping von AC und UP eindeutig definiert. Um das Mapping zu verifizieren, werden Messungen mit dem in Abbildung 6.1 dargestellten Setup durchgeführt. Dabei senden die beide Sender jeweils UDP Pakete mit einer Datenrate von 250 MBit/s zur STA. Die Datenrate wurde so gewählt, dass die Summe dieser beiden größer ist als die effektiv erreichbare Datenrate in diesem Setup. Dies ist wichtig, um das Verhalten der verschiedenen ACs im Blick auf die jeweils unterschiedlichen Prioritäten sehen zu können. Bei allen Messungen startet jeweils ein Sender mit dem Senden und der andere beginnt damit erst nach circa 5 Sekunden.

In Abbildung 6.2 sind die Ergebnisse von zwei verschiedene Messungen zu sehen. Der Grund, warum nicht die Ergebnisse aller durchgeführten Messungen dargestellt sind, liegt daran, dass viele jeweils identisch sind. Deswegen wird exemplarisch nur jeweils ein Ergebnis der zwei auftretenden Fälle gezeigt. Das ist einmal der Fall, wenn die Sender jeweils mit verschiedenen DSCP Wert senden, welche aber über das Mapping letztendlich in die gleiche AC kommen. Das zugehörige Ergebnis solch einer Messung kann in Abbildung 6.2a gesehen werden. Da die Pakete beider Sender in die gleiche AC kommen, haben diese dementsprechend auch dieselbe Priorität und deshalb wird die verfügbare Datenrate auf beide Sender gleichmäßig aufgeteilt.

Der andere Fall ist der, dass die Sender jeweils mit verschiedenen DSCP Werten senden, welche aber durch das Mapping unterschiedlichen ACs zugeordnet werden. Das Ergebnis einer solchen Messung ist in Abbildung 6.2b dargestellt. Dabei beginnt der Sender mit dem DSCP Wert, welcher letztendlich in die AC mit der höheren Priorität kommt. Dass dies korrekt ist, kann daran gesehen werden, dass die Datenrate dieser STA gleich bleibt, auch nachdem die andere STA mit senden

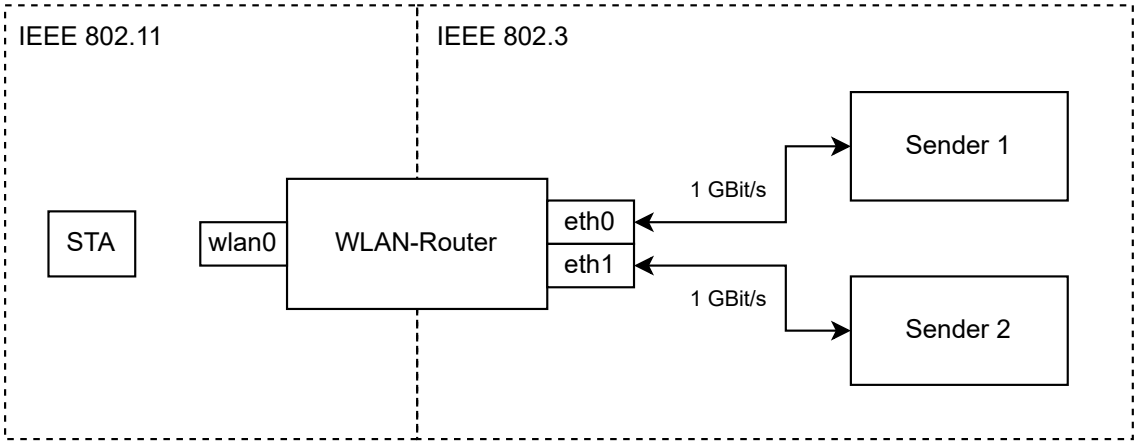
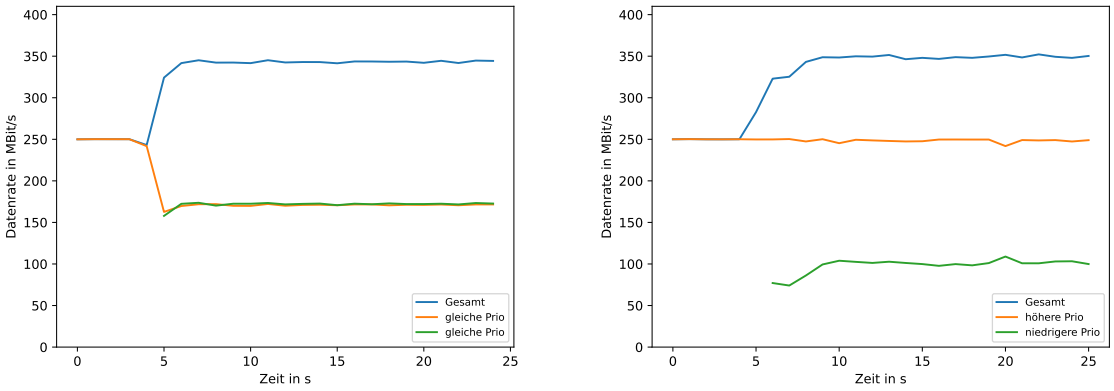


Abbildung 6.1: Setup für die Verifizierung des Default DSCP-UP Mappings.



(a) DSCP Werte aus der gleichen AC.

(b) DSCP Werte aus unterschiedlichen ACs.

Abbildung 6.2: Verifizierung des Default Mappings zwischen DSCP und UP.

beginnt. Dementsprechend hat der Sender, der später beginnt, eine geringere Priorität und bekommt nur die restliche zu Verfügung stehende Datenrate. Somit wurde das in Tabelle 2.2 dargestellte Default Mapping von UP und DSCP verifiziert.

6.1.2 In Überlastsituation

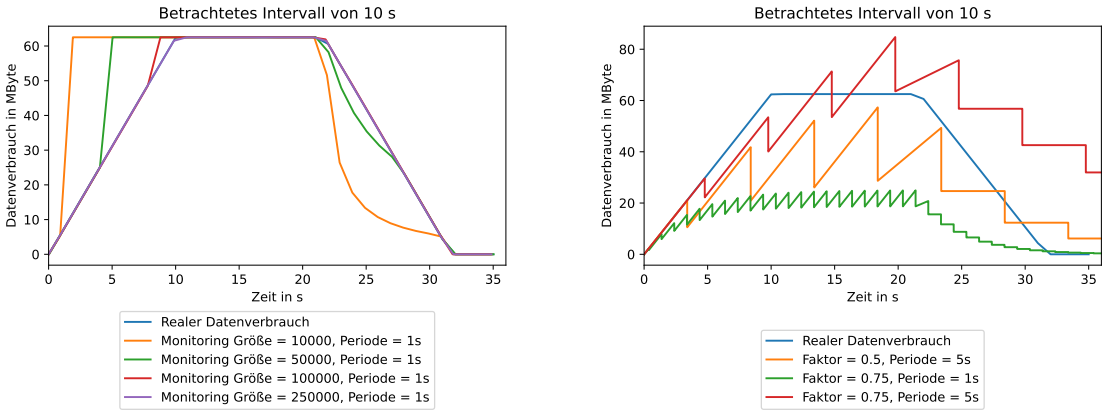
Wie in Kapitel 3 beschrieben ist, liegt der Bottleneck bei einem WLAN-Netzwerk am Up- oder Downlink. Der von WMM ermöglichte QoS betrifft ausschließlich die Funkverbindung und hat dem zufolge keinen Einfluss auf eine Überlast an diesen Stellen. Somit kann mit WMM dort generell keine Fairness erreicht werden. Außerdem hat WMM den großen Nachteil, dass es nur die vier festen ACs gibt und somit ist letztendlich auch die Anzahl an Prioritäten auf vier begrenzt. Durch WMM will allgemein erreicht werden, dass beispielsweise Voice over IP dauerhaft eine hohe Priorität hat, um damit eine entsprechende Qualität des Anrufs sicherzustellen. Dementsprechend sind die Prioritäten intern bei WMM statisch, da diese durch die Pakete spezifiziert werden. Demzufolge

kann WMM diese nicht dynamisch anpassen. Auf der Anwendungsebene könnten die Prioritäten aber dynamisch geändert werden indem die Pakete mit entsprechenden DSCP Werten gesendet werden. Doch dies entspricht nicht dem gewünschten dynamischen Verhalten, denn dabei sollten die Prioritäten am Bottleneck entsprechend automatische verändert werden, um damit Fairness zu erreichen. Zusammenfassend kann gesagt werden, dass WMM keine Auswirkung auf den Bottleneck an Up- oder Downlink hat und das damit außerdem nicht das erforderliche dynamische Verhalten realisiert werden kann.

6.2 Datenverbrauch

In diesem Abschnitt wird untersucht, wie exakt der bestimmte Datenverbrauch dem tatsächlichen Verbrauch entspricht. Dazu werden die beiden implementierten Ansätze, also Ansatz 2 (Begrenzt-approximiert) und Ansatz 4 (Additive Erhöhung-prozentuale Verringerung) jeweils mit unterschiedlichen Konfigurationen getestet. Die Messungen werden mit dem Setup, welches in Abbildung 3.5 dargestellt ist, durchgeführt. Bei diesen wird beispielhaft der Datenverbrauch in einem Intervall von 10 Sekunden betrachtet. Um die Bestimmung des Datenverbrauchs in verschiedenen Szenarien bewerten zu können, werden Messungen mit zwei unterschiedlichen Sendeverhalten durchgeführt. Das wäre einmal das Senden für eine gewisse Zeit, sodass genau ein Burst entsteht. Dies entspricht dann zum Beispiel einem Nutzer, der etwas up- oder downloaded. Bei dem anderen Sendeverhalten werden mehrere kleinere Bursts gesendet, wobei diese Poisson-verteilt sind mit einem Mittelwert von 5 Bursts/Minute. Dadurch kann ein Nutzer simuliert werden, der im Internet surft, wobei jeder Burst dem Aufruf einer neuen Webseite entspricht.

In Abbildung 6.3 sind die Ergebnisse mit dem Senden des einen Burst zusehen. Dabei sind in Abbildung 6.3a die Ergebnisse von Ansatz 2 (Begrenzt-approximiert) mit unterschiedlichen Monitoring Größen dargestellt. Als Periode wurde jeweils immer 1 Sekunde gewählt, da dies eine ausreichende Genauigkeit liefert. In der Abbildung kann gesehen werden, dass eine Konfiguration exakt dem realen Verbrauch entspricht. Dies ist zu erwarten, da bei dieser alle Pakete in der Monitoring Datenstruktur gespeichert werden können und dementsprechend der Datenverbrauch exakt bestimmt werden kann. Bei den anderen Konfigurationen können nicht alle Pakete gespeichert werden und somit erfolgt bei diesen jeweils die Approximation. Wie in Abschnitt 4.2.2 beschrieben ist dies eine Worst Case Approximation und dies kann in den Ergebnissen gut erkannt werden. Denn sobald die Monitoring Datenstruktur voll ist, wird davon ausgegangen, das über die komplette fehlende Zeit die gleiche Datenmenge verbraucht wurde und dadurch wird der bestimmte Verbrauch schlagartig entsprechend größer. Der Punkt, an dem dieser Sprung passiert, hängt von der Monitoring Größe ab und somit ist dieser Punkt bei größeren Werten erst später. Am Ende fällt der bestimmte Verbrauch bei kleinen Monitoring Größen, also bei der orangenen und grünen Konfigurationen, schneller ab als der reale. Der Grund dafür ist, dass bei der Approximation der durchschnittliche Verbrauch verwendet wird und dieser halbiert sich nach jedem Update, wenn keine neuen Pakete mehr gesendet werden. Bei den anderen Konfigurationen entspricht gegen Ende die Bestimmung nahezu dem realen Verbrauch. Allgemein kann gesagt werden, wenn der reale Verbrauch null ist, dass dies auch auf alle möglichen Konfigurationen zutrifft. Dies liegt daran, dass bei diesem Ansatz das betrachtete Zeitintervall direkten Einfluss auf die Berechnung hat.



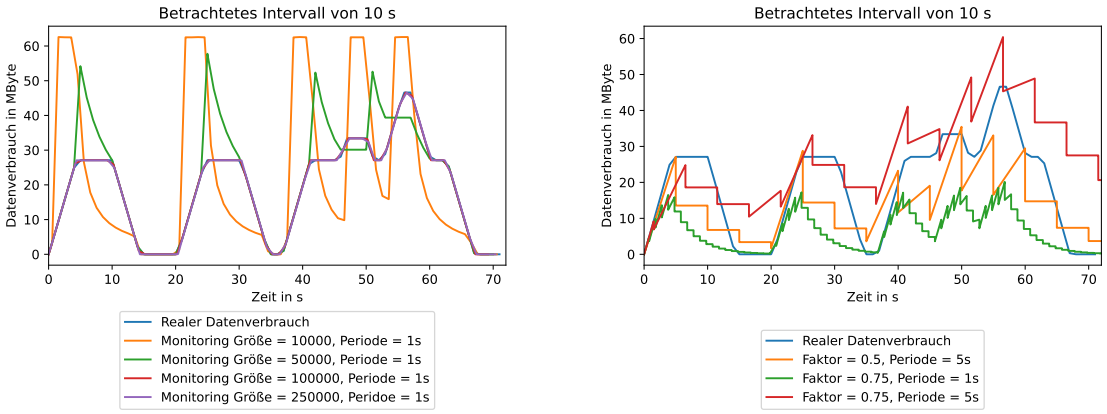
(a) Mit Ansatz 2: Begrenzt-approximiert.

(b) Mit Ansatz 4: Additive Erhöhung-prozentuale Verringerung.

Abbildung 6.3: Evaluierung des Datenverbrauchs bei einem Burst.

In Abbildung 6.3b sind die Ergebnisse von verschiedenen Konfigurationen mit Ansatz 4 (Additive Erhöhung-prozentuale Verringerung) zusehen. Darin kann gesehen werden, dass diese generell im Vergleich zu Ansatz 2 wegen den charakteristischen entstehenden Zacken ungenauer sind. Je nach Konfiguration können die Abweichungen zum realen Verbrauch sehr groß sein, wie beispielsweise mit der grünen, welche die meiste Zeit um circa 66 % vom realen Datenverbrauch abweicht. Somit spielt die Wahl der Parameter eine entscheidende Rolle für die Genauigkeit. Bei diesem Sendeverhalten würde die rote Konfiguration am Anfang gute Ergebnisse liefern. Doch diese hat dann gegen Ende, wenn keine neuen Pakete mehr gesendet werden, große Abweichungen, denn dabei nähert sich der berechnete Verbrauch nur sehr langsam dem realen, also in diesem Fall null an. Dieses Verhalten ist bei diesem Ansatz generell ein Problem, denn wenn keine neuen Pakete kommen, fällt der Datenverbrauch exponentiell mit dem Faktor gegen null ab, wobei dies in Zeitschritten der Periode erfolgt. Je nach Konfiguration kann dieses Verhalten dazu führen, das es sehr lange dauern könnte, bis annähernd null erreicht wird.

In Abbildung 6.4 sind die Messungen mit dem Senden der Poisson-verteilten Bursts dargestellt. Wenn bei Ansatz 2 (Begrenzt-approximiert) die Monitoring Größe ausreicht, um alle Pakete speichern zu können, wird der Verbrauch wie in Abbildung 6.4a gesehen werden kann, wieder exakt bestimmt. Bei diesem Szenario ist dies bei zwei Konfigurationen der Fall, welche dementsprechend genau dem realen Verbrauch entsprechen. Bei den anderen Konfigurationen ist die Monitoring Größe nicht ausreichend und es muss approximiert werden. Dabei kann wie oben auch am Beginn eines Burst die Worst Case Annahme gut erkannt werden. Bei diesem Sendeverhalten ist deswegen die orangene Konfiguration nicht sehr genau, denn es können wenige Pakete gespeichert werden, womit eine große Zeitspanne approximiert werden muss und dies liefert starke Abweichungen. Im Vergleich dazu liefert die grüne Konfiguration eine exaktere Bestimmung des Datenverbrauchs, wobei starke Abweichungen nur für eine kurze Zeit vorhanden sind ansonsten entspricht diese nahezu dem realen Verbrauch. In Abbildung 6.4b sind die Ergebnisse verschiedener Konfigurationen mit Ansatz 4 (Additive Erhöhung-prozentuale Verringerung) dargestellt. Dabei kann wie oben auch gesehen werden, dass die Konfiguration eine entscheidende Auswirkung auf die Genauigkeit der Bestimmung hat. In diesem Fall ist die grüne Konfiguration sehr ungenau, da diese die meiste Zeit



(a) Mit Ansatz 2: Begrenzt-approximiert.

(b) Mit Ansatz 4: Additive Erhöhung-prozentuale Verringerung.

Abbildung 6.4: Evaluierung des Datenverbrauchs bei Poisson-verteilten Bursts.

eine Abweichung von mehr als 50 % zum realen Verbrauch hat. Die anderen beiden Konfigurationen liefern akzeptable Resultate, wobei die rote noch ein bisschen genauer ist. Doch diese ist, wenn keine neuen Pakete mehr gesendet werden, sehr ungenau, da es lange dauert, bis der bestimmte Verbrauch null erreicht.

Zusammenfassend kann gesagt werden, dass mit Ansatz 2 (Begrenzt-approximiert), falls eine Monitoring Größe verwendet wird, die entsprechend groß ist, der Datenverbrauch im allgemeinen genauer bestimmt werden kann als mit Ansatz 4 (Additive Erhöhung-prozentuale Verringerung). Der entscheidende Grund dafür liegt darin, dass die Werte der Parameter bei Ansatz 4 dabei eine entscheidende Rolle spielen, um akzeptable Ergebnisse zu erreichen. Diese Wahl der Werte hängt stark vom Sendeverhalten und der Senderate ab, sodass es keine Konfiguration gibt, welche für alle Szenarien sinnvoll ist. Bei Ansatz 2 ist dies nicht der Fall, da mit entsprechender Monitoring Größe in beliebigen Szenarien gute Resultate erreicht werden. Deshalb wird in der weiteren Evaluierung Ansatz 2 verwendet, da dieser genauer und einfacher zu konfigurieren ist.

Dennoch hat Ansatz 4 (Additive Erhöhung-prozentuale Verringerung) Vorteile im Vergleich zu Ansatz 2 (Begrenzt-approximiert). Dies wäre beispielsweise, dass kein zusätzlicher Speicherplatz benötigt wird und dass die erforderliche Berechnung sehr simpel ist. Besonders da dabei keine Approximation und Iteration über die gespeicherten Pakete notwendig ist.

6.3 Fairness

In diesem Abschnitt wird durch Messungen untersucht, ob die Implementierung des neu entworfenen Scheduling-Verfahrens Datenverbrauch-Fairness am Uplink korrekt umsetzt. Für diese Untersuchung werden zwei verschiedene Setups verwendet. Das ist einmal das Setup, mit dem in Kapitel 3 gezeigt wurde, dass am Uplink eine Überlast entstehen kann. Außerdem wird ein neues Setup verwendet, welches eine reale Situation darstellt. Bei den Messungen wird die Monitoring Größe so gewählt, dass alle Pakete, welche zur Bestimmung des Datenverbrauchs benötigt werden, gespeichert werden können. Dementsprechend kann der Verbrauch immer exakt bestimmt werden.

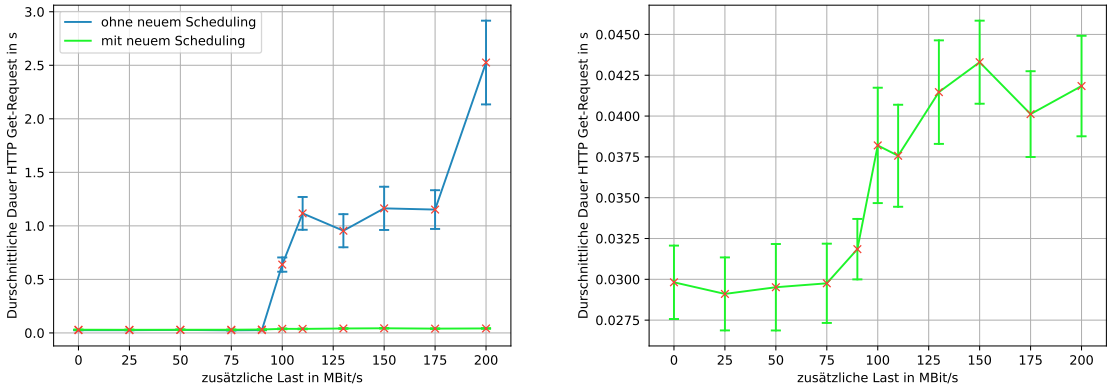
6.3.1 Setup aus Überlastkapitel

Hier wird das gleiche Setup verwendet, mit welchem in Kapitel 3 die Überlast am Uplink gezeigt wurde. Der einzige Unterschied zum Setup in Abbildung 3.5 ist, dass am Interface *eth1* nun das neue Scheduling-Verfahren verwendet wird. Die Messungen erfolgen analog wie in Abschnitt 3.5, außer das hier nur 100 Get-Requests gemessen werden. Das neue Scheduling-Verfahren wird dabei mit zwei Klassen konfiguriert. Als Threshold von Klasse 0, also der höheren Priorität, wird 11.000.000 verwendet. Dieser ist so gewählt, dass die STA welche die Get-Requests sendet immer in Klasse 0, der Verkehr zur Überlast Generierung aber in Klasse 1 zugeordnet wird. Der Wert muss deshalb mindestens $100 \cdot 100$ kByte sein, da dies der angefragten Datenmenge der 100 Get-Requests entspricht. Da außerdem bei den Paketen noch mehrere Header vorhanden sind und Handshake-Nachrichten benötigt werden, wird ein entsprechend größerer Wert verwendet. Das betrachtete Intervall für den Datenverbrauch sollte groß genug sein, damit der Verkehr der Überlast in Klasse 1 zugeordnet wird. Um dies sicher zu stellen, wird ein Intervall von 10 Sekunden gewählt. Für jede Messung wird das Setup neu konfiguriert, damit jeder Datenverbrauch wieder auf null zurückgesetzt ist. Somit erfolgt jede Messung unter den gleichen Bedingungen.

In Abbildung 6.5 sind die Mittelwerte der einzelnen Messungen jeweils mit den 99,9 % Konfidenzintervallen zu sehen. Dabei ist in Abbildung 6.5a zusätzlich zum Ergebnis der Messungen mit dem neuen Scheduling auch das Resultat ohne dieses aus Abschnitt 3.5 dargestellt, um den Unterschied zu verdeutlichen. Darin kann gesehen werden, dass durch das neue Verfahren die durchschnittliche Antwortzeit nahezu gleich bleibt unabhängig von der zusätzlichen Last. Somit wird Datenverbrauch-Fairness erreicht, denn die STA, welche die Get-Requests sendet, hat einen geringeren Datenverbrauch als diese, welche die zusätzliche Last generiert. Dementsprechend werden nur die Pakete der Get-Requests der Klasse 0 zugeordnet und bekommen damit eine höhere Priorität. Dadurch bleibt die Antwortzeiten immer identisch, egal wie viel zusätzlich Last vorhanden ist, denn diese kommt in Klasse 1. In Abbildung 6.5b ist noch mal das Ergebnis mit dem neuen Scheduling einzeln aufgetragen. Darin kann gesehen werden, dass die durchschnittliche Zeit bei höherer Last um circa 10 Millisekunden schlechter wird, wobei dies für einen Nutzer nicht wahrnehmbar ist. Diese Verschlechterung liegt letztendlich daran, dass je mehr zusätzliche Last vorhanden ist, desto länger dauert das Senden der Get-Requests im WLAN.

6.3.2 Reale Situation

Zum Betrachten der Fairness in einer realen Situation wird das Setup in Abbildung 6.6 verwendet. Dabei wird das neue Scheduling-Verfahren an Interface *eth1* gesetzt, da dort der Bottleneck vorliegt. Wie darin dargestellt ist, werden für die Messungen drei STAs benötigt. Eine davon sendet entsprechend viele UDP Paket über den Software-Switch an das Interface *eth2*, wodurch die zusätzliche Last simuliert wird. Die anderen beiden STAs wollen eine Webseite im Internet aufrufen, wobei eine davon zusätzlich noch ein Video aus der ZDF Mediathek anschaut. Als Webseite wurde FAZ.net verwendet. Von diesen beiden STAs aus wird jeweils die Zeit von 100 erfolgreichen Get-Requests zu dieser Webseite gemessen. Dabei erfolgt dies von beiden STA gleichzeitig um vergleichbare Ergebnisse zu bekommen. Für die Bestimmung des Datenverbrauchs wird ein Zeitintervall von 60 Sekunden betrachtet.



(a) Vergleich mit und ohne neuem Scheduling.

(b) Nur das neue Scheduling.

Abbildung 6.5: Evaluierung der Fairness des neuen Scheduling-Verfahrens.

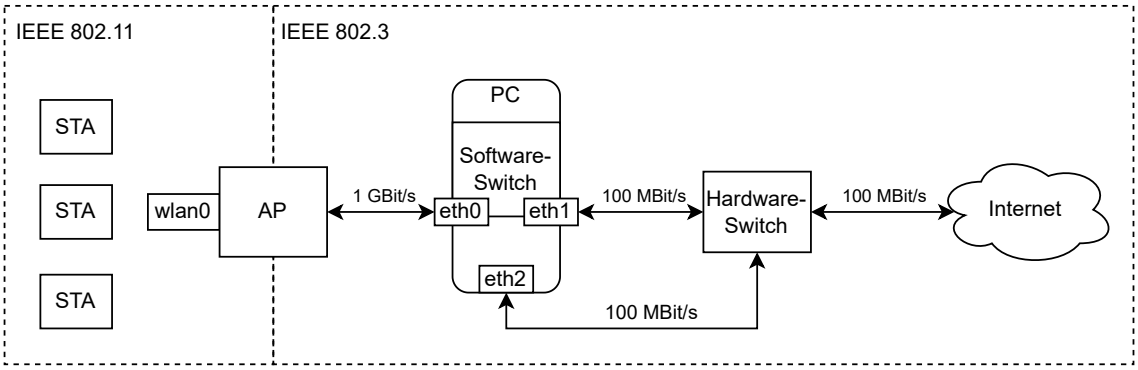
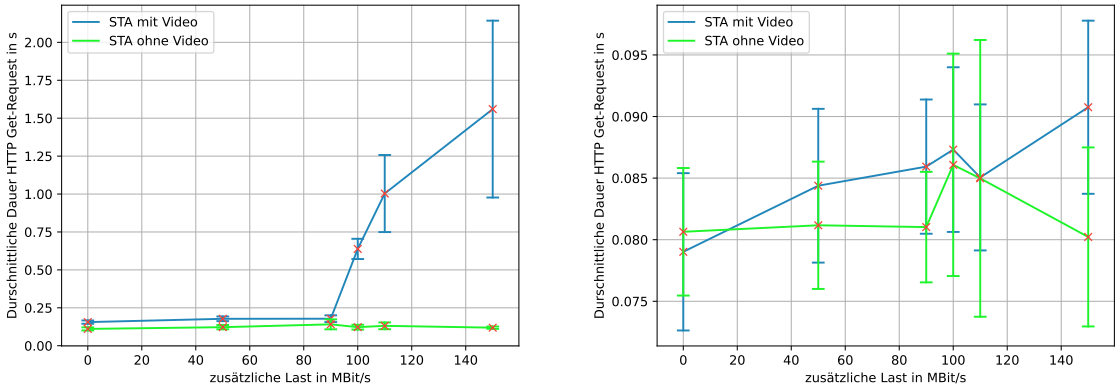


Abbildung 6.6: Setup zum Testen der Fairness in realer Situation.

In Abbildung 6.7a sind jeweils die Mittelwerte von den gemessenen Zeiten der Get-Requests inklusive der 99,9 % Konfidenzintervallen dargestellt. Dabei wurde das Scheduling-Verfahren mit zwei Klassen konfiguriert. Um zu erreichen, dass der UDP-Verkehr und die STA mit Video Klasse 1 zugeordnet werden, wurde der Threshold von Klasse 0 auf 20.000.000 gesetzt, wobei dieser Wert durch Messungen des Datenverbrauchs gefunden wurde. Dementsprechend ist die STA ohne Video in Klasse 0 und sollte demzufolge bevorzugt werden. Dies kann in den Ergebnissen auch gesehen werden, da die durchschnittliche Zeit zum Laden der Webseite von der STA ohne Video identisch ist unabhängig von der zusätzlichen Last. Im Vergleich dazu wird die Ladezeit bei der STA mit Video bei höherer Last schlechter, da sich die Klasse mit dem UDP-Verkehr geteilt werden muss und somit Pakete verworfen werden. Diese müssen dann erneut gesendet werden, wodurch die durchschnittliche Ladezeit entsprechend länger wird.

Es wurde noch eine weitere Messung durchgeführt, wobei in diesem Fall das Scheduling mit drei Klassen konfiguriert wurden. Dabei wurden die Thresholds so gewählt, dass jeder der drei STAs einer anderen Klasse zugeordnet wird. Dazu wurde der Threshold von Klasse 0 auf 20.000.000 und der von Klasse 1 auf 200.000.000 gesetzt, wobei diese Werte durch Messungen bestimmt wurden. Dies führt dazu, dass der UDP-Verkehr, mit der die zusätzliche Last generiert wird, der Klasse 2



(a) Konfiguration mit zwei Klassen.

(b) Konfiguration mit drei Klassen.

Abbildung 6.7: Evaluierung der Fairness in realer Situation.

zugeordnet wird und somit die niedrigste Priorität hat. Die höchste Priorität bekommt damit die STA ohne Video, da diese in Klasse 0 kommt. Die andere STA wird in Klasse 1 eingeordnet und bekommt somit die mittlere Priorität. In Abbildung 6.7b sind jeweils die Mittelwerte der gemessenen Zeiten inklusive der 99,9 % Konfidenzintervall dargestellt. Darin kann gesehen werden, dass beide STAs ungefähr die gleiche Antwortzeit haben, wobei die STA ohne Video meist eine minimal bessere durchschnittliche Zeit hat. Die vorhandenen Schwankungen kommen durch die Verwendung unterschiedlicher Typen von STAs und Schwankungen im Internet zustande. Somit kann erkannt werden, dass es in diesem Fall nahezu kein Unterschied macht, ob die STA in Klasse 0 oder 1 ist. Dementsprechend hat es auch keine Auswirkung, dass die eine STA noch zusätzlich ein Video anschaut. Der Grund hierfür ist, dass die verfügbare Datenrate groß genug ist, um alle Pakete aus diesen beiden Klassen problemlos zu senden und dementsprechend sind die gemessenen Zeiten nahezu identisch.

Zusammenfassend kann gesagt werden, dass in der realen Situation Datenverbrauch-Fairness erreicht wurde, falls sich der Bottleneck am Uplink befindet. Mit der Anzahl an Klassen kann noch entsprechend verfeinert werden. Außerdem wurde mit diesen Messungen auch gezeigt, dass das Downlink Monitoring funktioniert, denn die Datenmenge des Videos wurde korrekt berücksichtigt.

6.4 Feature am Downlink

Um zu zeigen, dass mit dem entwickelten Feature Datenverbrauch-Fairness in einer Überlastsituation am Downlink erreicht werden kann, wurden entsprechende Messungen durchgeführt. Dazu wird das in Abbildung 6.8 dargestellte Setup verwendet, wobei an Interface *eth1* das Scheduling-Verfahren gesetzt ist. Mit dem Hardware-Switch wird erreicht, dass die maximal mögliche Datenrate auf 100 MBit/s begrenzt wird. Bei der Bestimmung des Datenverbrauchs werden die letzten 100 Sekunden berücksichtigt. Als Monitoring Größe wurde 250.000 verwendet, da damit der Datenverbrauch bei den Messungen mit ausreichender Genauigkeit bestimmt werden kann.

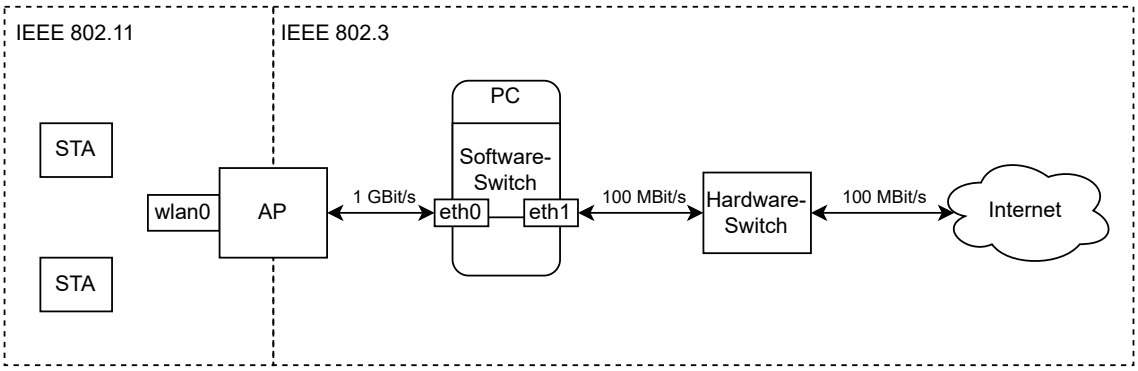


Abbildung 6.8: Setup zum Analysieren des Features am Downlink.

Eine Messung wurde durchgeführt, um zu verifizieren, dass durch das Verwerfen von Paketen am Downlink generell erreicht werden kann, dass der dazugehörige Sender seine Datenrate reduziert. Dazu wird von einer STA aus ein Download über das Internet gestartet und die entsprechende Datenrate betrachtet. Das Scheduling-Verfahren wurde dabei mit drei Klassen verwendet. Um das erwartete Verhalten der Datenrate gut erkennen zu können, wurde als Threshold von Klasse 0 der Wert 200.000.000 und bei Klasse 1 der Wert 400.000.000 gewählt. Der prozentuale Anteil an zu verwerfenden Pakete wurde bei Klasse 0 auf 0 %, bei Klasse 1 auf 0.1 % und bei Klasse 2 auf 0.6 % gesetzt. Diese Prozente entsprechend den skalierten Werten 0, 4.294.967 und 25.769.804 in der gleichen Reihenfolge. Die Werte wurden durch entsprechendes Testen gefunden, sodass die Unterschiede in der Datenrate gut sichtbar sind.

In Abbildung 6.9 ist die gemessene Downlink-Datenrate an dieser STA aufgetragen. Darin kann gesehen werden, dass zu Beginn, wenn noch keine Pakete verworfen werden, die maximale Datenrate erreicht wird. Nach circa 20 Sekunden fällt die Datenrate ab, da die STA nun wegen des Datenverabrauches in Klasse 1 zugeordnet wird und dementsprechend werden nun Pakete verworfen. Somit wird durch das anteilige Verwerfen von Paketen erreicht, dass der Sender die Datenrate entsprechend reduziert. Die großen Schwankungen kommen zustande, da die Pakete letztendlich zufällig verworfen werden und dies kann dazu führen, dass für eine längere Zeit kein Paket verworfen wird. Dies hat dann zur Folge, dass die Datenrate wieder kurzzeitig steigt, wie in Abbildung 6.9 gesehen werden kann. Nach circa 50 Sekunden reduziert sich die Datenrate ein weiteres Mal, da sich ab diesem Zeitpunkt die STA in Klasse 2 befindet. Dabei werden nun durch den größeren Prozentwert mehr Pakete verworfen und dies führt dazu, dass der Sender die Senderate weiter reduziert.

Mit einer weiteren Messung wurde gezeigt, dass sich mit dem Feature die Antwortzeiten von Teilnehmern verbessern lassen, falls sich der Bottleneck am Downlink befindet. Dabei wurde das Scheduling mit zwei Klassen verwendet, wobei der Threshold von Klasse 0 auf 370.000.000 gesetzt wurde. In Klasse 0 sollen keine Pakete verworfen werden und somit hat diese den skalierten Prozentwert 0 spezifiziert. Bei Klasse 1 sollen 0.5 % der Pakete verworfen werden und dies wird mit dem skalierten Wert 21.474.836 erreicht. Bei der Messung downloaded eine STA etwas über das Internet und von einer anderen STA aus wurde parallel die Dauer von 100 erfolgreichen Get-Requests zur FAZ.net Webseite gemessen. Um vergleichen zu können, ob mit dem Feature das gewünschte Verhalten erreicht werden kann, wurde diese Messung einmal mit und einmal ohne dem Feature durchgeführt.

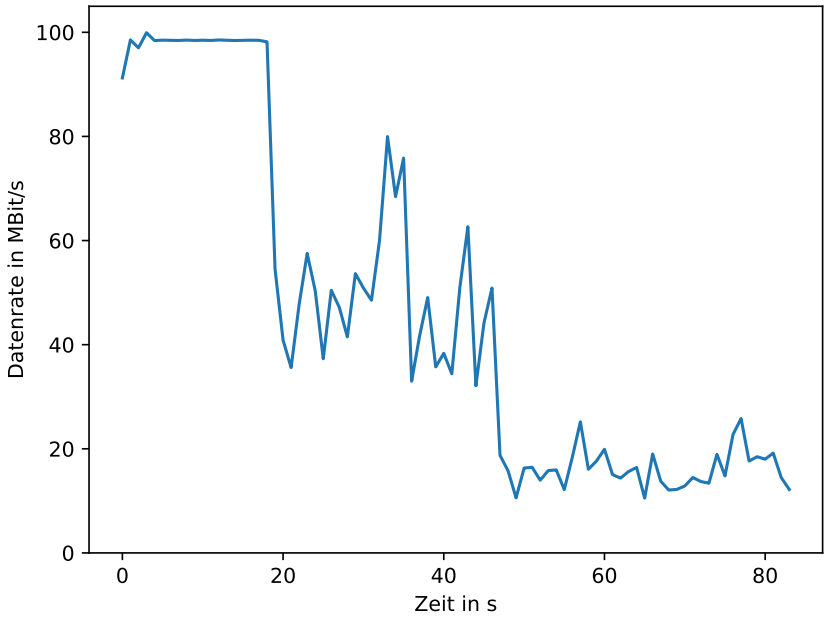


Abbildung 6.9: Ergebnis der gemessenen Downlink-Datenrate mit dem Feature am Downlink.

In Abbildung 6.10 sind jeweils die gemessenen 100 Zeiten aufgetragen. Darin kann anhand des Medians gesehen werden, dass durch das Feature die Antwortzeiten signifikant besser werden. Solange der Download eine Datenrate von circa 98 MBit/s hat, also die maximal zu erreichende, brauchen die Get-Requests durch die vorhandene Überlast entsprechend lange. Dies ist bei der Messung ohne Feature immer der Fall. Bei der Messung mit Feature betrifft dies nur die Ausreißer, da diese Requests zu der Zeit gemacht wurden, wo die Datenrate des Downloads noch nicht reduziert wurde und somit maximal ist. Doch nachdem die STA mit dem Download in Klasse 1 zugeordnet wird, wird die Senderate durch das Verwerfen von Paketen reduziert und die Überlast somit aufgelöst. Anschließend ist wieder ein entsprechender Teil der Datenrate verfügbar, welche nun von den Paketen der Get-Requests verwendet werden kann. Dies führt dann dazu, dass die Antwortzeiten der restlichen Requests besser werden. Da dies letztendlich auf ein Großteil der 100 Get-Request zutrifft, sind die meisten Antwortzeiten bei der Messung mit Feature entsprechend kürzer. Damit wurde gezeigt, dass mit diesem Feature auch Datenverbrauch-Fairness erreicht werden kann, falls sich der Bottleneck am Downlink befindet.

6.5 Performanz

In diesem Abschnitt wird die Performanz des neu entwickelten Scheduling-Verfahrens evaluiert. Dazu werden die beiden vorhandenen Komponenten, also Scheduling und Downlink Monitoring, jeweils separat betrachtet. Alle Messungen werden mit dem in Abbildung 6.11 dargestellten einfachen Setup durchgeführt. Der Grund, warum dabei insbesondere auf das WLAN verzichtet wurde, ist dieser, dass dadurch die Messungen ohne die dort möglichen Schwankungen durchgeführt

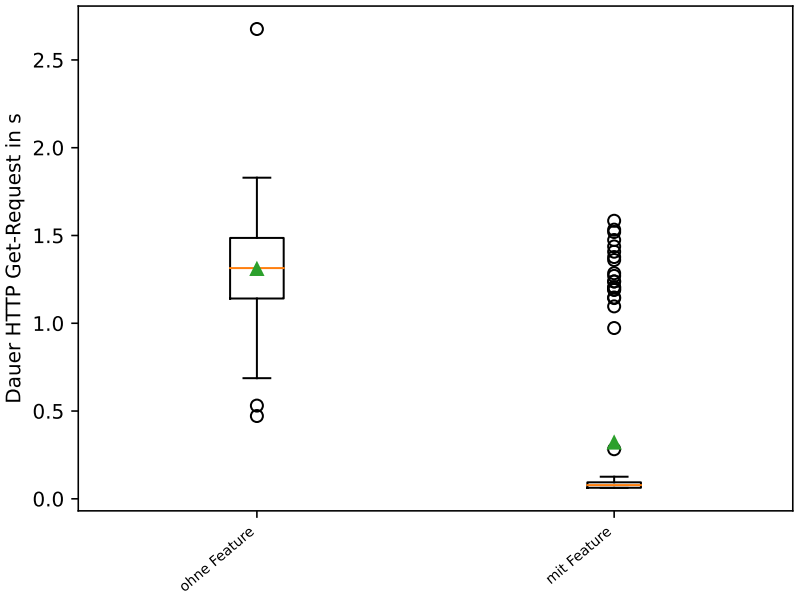


Abbildung 6.10: Ergebnis der gemessenen Zeiten mit und ohne Feature am Downlink.

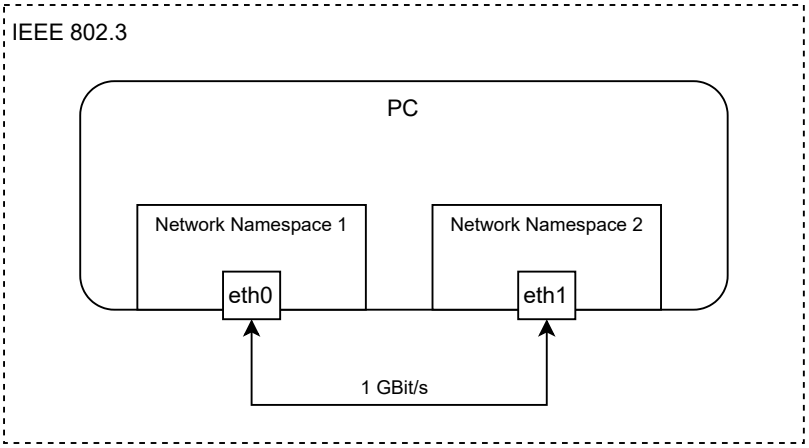


Abbildung 6.11: Setup zum Analysieren der Performanz.

werden können. Mithilfe der zwei Network Namespaces werden die Interfaces des PCs voneinander getrennt, damit die Kommunikation zwischen diesen nur über das Ethernet-Kabel möglich ist. Bei allen Messungen wird die Performanz analysiert, indem die Zeit bestimmt wird, wie lange ein UDP Paket von einem Network Namespace zum anderen benötigt. Um diese Zeit für jedes Paket bestimmen zu können, wird direkt vor dem Senden eines Paketes die aktuelle Zeit in den Payload geschrieben. Damit und mit dem aufgezeichneten Empfangszeitpunkt des Paketes kann anschließend die Verzögerungszeit für jedes Paket berechnet werden. Für die Messungen werden jeweils 100.000 Pakete gesendet, wobei zwischen dem Senden der einzelnen Pakete je 1 Millisekunde gewartet wird.

6.5.1 Scheduling

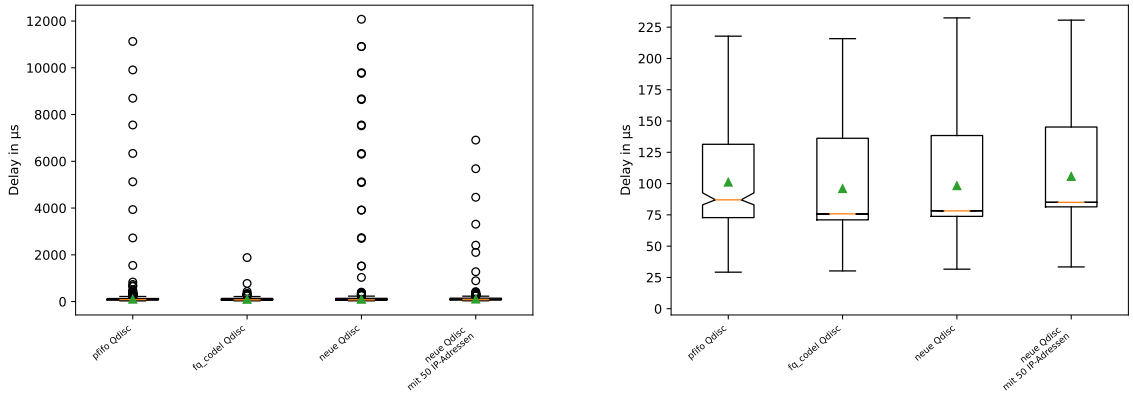
Zur Untersuchung der Performanz des Schedulings, muss die neue Qdisc betrachtet werden. Damit die Ergebnisse besser zu bewertet sind, werden auch Messungen mit zwei anderen Qdiscs durchgeführt, um dadurch einen entsprechenden Vergleich zu haben. Dazu wird die pfifo und die fq_codel Qdisc gewählt, da diese einmal eine simple Qdisc ist und die andere sozusagen die Standard Qdisc unter Linux ist. Die neue Qdisc wird so konfiguriert, dass alle gesendeten Pakete immer in Klasse 0 zugeordnet werden. Dies kann einfach sichergestellt werden, da die Paketgröße und die Anzahl an gesendeten Paketen pro Sekunde bekannt sind und somit kann der Threshold unter Berücksichtigung des betrachteten Zeitintervalls berechnet werden. Für die Messungen wird ein betrachtetes Intervall von 10 Sekunden angenommen und dementsprechend wird der Threshold von Klasse 1 auf 1.000.000 gesetzt, damit die gewünschte Zuordnung entsteht.

Um auch eine Aussage treffen zu können, ob die Anzahl der vorhandenen IP-Adressen einen Einfluss auf die Performanz hat, werden mit der neuen Qdisc zwei Messungen durchgeführt. Dazu wird bei einer Messung mit 50 vorhandenen IP-Adressen erzwungen, dass die letztendlich benötigte IP-Adresse des Senders erst an Position 51 im Array steht. Dies wird erreicht, indem die Datenstrukturen, welche eine IP-Adresse beinhalten, also die des Monitorings und des Datenverbrauchs, beim Initialisieren schon mit 50 IP-Adressen befüllt werden. Bei der anderen Messung befindet sich die IP-Adresse des Senders an einer der ersten Positionen im jeweiligen Array. Für die verschiedenen Messungen wird im Setup dann die entsprechende Qdisc an Interface *eth0* gesetzt. Die Pakete müssen somit aus dem Network Namespace 1 gesendet werden, damit diese die Qdisc auch passieren. Am Interface *eth1* werden die ankommenden Pakete dann entsprechend aufgezeichnet.

In Abbildung 6.12 sind die Ergebnisse der Messungen in Form von Boxplots aufgetragen. Dabei sind in Abbildung 6.12a auch die Ausreißer mit dargestellt. Bei den vier durchgeführten Messungen ist der Anteil der Ausreißer jeweils unter 0,1 %. Dabei hat die neue Qdisc im Vergleich zur pfifo Qdisc weniger Ausreißer, aber im Vergleich zur fq_codel Qdisc mehr, wobei diese Unterschiede in der Anzahl minimal sind. Die Zeiten der Ausreißer weichen wie in Abbildung 6.12a gesehen werden kann, nur zwischen der fq_codel Qdisc und den anderen Messungen stark ab, indem diese Werte bei der fq_codel Qdisc deutlich niedriger sind. Um die Boxplots besser sehen und somit bewerten zu können, werden diese in Abbildung 6.12b noch mal ohne die Ausreißer abgebildet. Darin kann gesehen werden, dass der Unterschiede bei den Medianen der verschiedenen Messungen minimal sind. Dabei ist der Median bei der neuen Qdisc mit den 50 IP-Adressen im Vergleich zu ohne um circa 7 Mikrosekunden schlechter, doch dies ist für die Endbenutzer letztendlich nicht spürbar. Zusammenfassend kann gesagt werden, dass die Performanz der Implementierung der neuen Qdisc im Vergleich zu den anderen annähernd gleich ist.

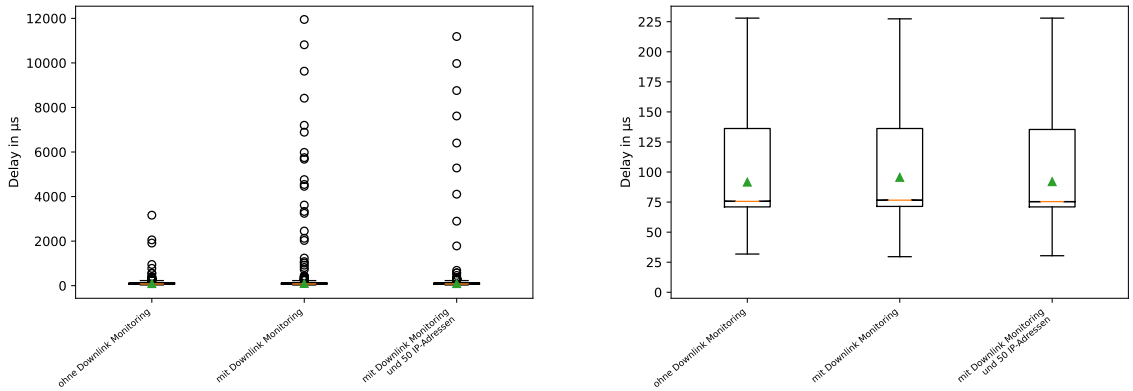
6.5.2 Downlink Monitoring

Um die Performanz des Downlink Monitorings bewerten zu können, wird auch eine entsprechende Messung ohne durchgeführt. Außerdem erfolgt auch hier eine Messung, wobei in den entsprechenden Datenstrukturen 50 IP-Adressen vorhanden sind, sodass sich die IP-Adresse des Senders erst an Position 51 des jeweiligen Arrays befindet. Dadurch kann dann analog wie oben auch eine Aussage getroffen werden, ob sich die Anzahl an vorhandenen IP-Adressen auf die Performanz auswirkt. Im Setup wird das Downlink Monitoring an Interface *eth0* gesetzt. Somit müssen bei diesen



(a) Mit Ausreißer. **(b)** Ohne Ausreißer.

Abbildung 6.12: Performanzanalyse des Schedulings.



(a) Mit Ausreißer. **(b)** Ohne Ausreißer.

Abbildung 6.13: Performanzanalyse des Downlink Monitorings.

Messungen die Pakete von dem Network Namespace 2 aus gesendet werden, damit diese das Downlink Monitoring passieren. Dementsprechend werden diese Pakete dann auch an Interface *eth0* aufgezeichnet.

In Abbildung 6.13 sind die jeweiligen Ergebnisse in Form von Boxplots aufgetragen. Dabei sind in Abbildung 6.13a auch die Ausreißer mit dargestellt. Der Anteil dieser ist mit weniger als 0,1 % sehr gering. Mit dem Downlink Monitoring werden es ungefähr 25 % mehr Ausreißer als ohne. Die Anzahl an vorhandenen IP-Adressen hat keine Auswirkung auf die Anzahl an Ausreißer. Aber durch das Downlink Monitoring werden die Zeiten der Ausreißer größer, wie in Abbildung 6.13a gesehen werden kann. Um die Boxplots besser sehen und damit bewerten zu können, sind diese in Abbildung 6.13b noch einmal ohne die Ausreißer dargestellt. Darin kann gesehen werden, dass die Mediane im Vergleich zwischen mit und ohne dem Downlink Monitoring identisch sind. Dementsprechend wird die Performanz durch das Downlink Monitoring nicht beeinflusst und die Anzahl an vorhandenen IP-Adressen hat außerdem keine Auswirkungen.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein neues Stream-basiertes, dynamisches und faires Scheduling-Verfahren für WiFi-Netzwerkverkehr entworfen und implementiert, welches die hier neu definierte Datenverbrauch-Fairness realisiert.

Zuerst wurde jedoch ein WLAN im Infrastruktur-Modus analysiert, um die Stelle des Bottlenecks zu finden. Dabei wurden letztendlich zwei Stellen identifiziert. Ein Bottleneck befindet sich am Uplink zum Internet und somit im eigenen WLAN-Netzwerk. Der andere ist am Downlink, wobei sich dieser Bottleneck beim Internet-Provider befindet und somit kann an diesem nichts direkt realisiert werden. Mit dem neuen Scheduling-Verfahren soll an diesen Stellen Fairness erreicht werden. Da es die gewünschte Art von Fairness bisher nicht gab, wurde diese unter dem Namen Datenverbrauch-Fairness zuerst definiert. Diese berücksichtigt den in einem festgelegten vergangenen Zeitintervall entstandenen Datenverbrauch, indem die Pakete von Teilnehmern, welche mehr Daten verbrauchen, eine niedrigere Priorität bekommen. Somit werden Nutzer mit einem geringen Datenverbrauch letztendlich bevorzugt und erreichen dadurch schnellere Antwortzeiten.

Das entwickelte Scheduling-Verfahren besteht dabei aus drei grundlegende Komponenten. Das ist einmal das Scheduling, welches das nächste zu sendende Paket auswählt. Außerdem wird ein Monitoring benötigt, welches alle Pakete des Up- und Downlinks bekommt. Da jedoch das Monitoring des Uplinks schon durch das Scheduling möglich ist, wird eine entsprechende Komponente nur für das Downlink Monitoring benötigt. Aus den Paketen des Monitorings kann der aktuelle Datenverbrauch jedes Teilnehmers bestimmt werden. Darüber werden dann im Scheduling die Prioritäten der Pakete festgelegt. Zur Bestimmung des aktuellen Datenverbrauchs wurden verschiedene Ansätze beschrieben und bewertet. Für die Umsetzung des Schedulings wurden zwei grundlegend verschiedene Varianten erläutert. Diese sind eine feingranulare und eine klassenbasierte Variante. Wie der Name schon sagt, werden einmal die verschiedenen Prioritäten mithilfe der Klassen erreicht. Bei der anderen Variante werden die Prioritäten feingranularer auf Geräte Basis bestimmt.

Die dritte Komponente stellt sicher, dass mit dem Scheduling-Verfahren nicht nur am Uplink, sondern auch am Downlink Datenverbrauch-Fairness erreicht werden kann. Diese realisiert dabei ein zusätzliches Feature, welches am Downlink einen bestimmten Anteil von Paketen von Endgeräten mit hohem Datenverbrauch verwirft. Dadurch reduziert der entsprechende Sender die Datenrate. Somit erhalten solche Geräte eine niedrigere Priorität und dies bedeutet in diesem Fall, dass diese eine geringere Downlink-Datenrate bekommen.

Implementiert wurde nur die klassenbasierte Variante, da die feingranulare Variante nicht skaliert, denn diese benötigt jeweils für jede vorhandene IP-Adresse eine geeignete Datenstruktur zur Speicherung der Pakete. Dabei wurde das Scheduling mithilfe einer neuen Qdisc realisiert. Das Downlink Monitoring wurde über einen tc Filter des Typs *matchall* umgesetzt, da dadurch automatisch alle Pakete berücksichtigt werden. Als Action dieses Filters wurde eine neue Action

implementiert, welche das Monitoring der Pakete umsetzt. Außerdem wurde mit dieser Action auch das zusätzliche Feature, also das anteilige Verwerfen von Paketen, realisiert. Für die neue Qdisc und Action wurde ein entsprechendes Linux-Kernel-Modul geschrieben.

Zum Schluss wurde die Implementierung des neuen Scheduling-Verfahrens noch evaluiert. Dabei wurde auch ein Vergleich mit WMM vorgenommen, da dieses QoS im WLAN ermöglicht. Es wurde gezeigt, dass mit dem neuen Verfahren Datenverbrauch-Fairness erreicht wird, wobei dies an realen Situationen verifiziert wurde. Außerdem wurde die Performanz analysiert und entsprechend verglichen. Dabei wurde kein signifikanter Unterschied zu dem Standard Linux Scheduling-Verfahren festgestellt.

In der Praxis werden IPv4- und IPv6-Adressen gleichzeitig verwendet, doch dies unterstützt die aktuelle Implementierung nicht. Dementsprechend wäre eine zukünftige Aufgabe, die vorhandene Implementierung so anzupassen, dass beide Adressierungsarten gleichzeitig unterstützt werden. Dass dies möglich ist, geht daraus hervor, da in dieser Arbeit gezeigt wurde, dass eine Implementierung jeweils separat möglich ist. Somit folgt, dass auch eine Kombination aus beiden realisierbar ist. Durch diese Anpassung wäre das Scheduling-Verfahren dann für eine Anwendung in der Praxis geeignet.

Außerdem könnte die Konfiguration der vorhandenen Parameter des Scheduling-Verfahrens verbessert werden, indem das tc-Command-Line-Programm entsprechend erweitert wird. Dadurch wäre dann die Konfiguration für beispielsweise Netzwerkadministratoren einfacher und außerdem auch schneller möglich.

Mit dem entwickelten und implementierten Feature am Downlink konnte durch Verwerfen eines bestimmten Anteils von Paketen erzwungen werden, dass die Downlink-Datenrate entsprechend reduziert wird. Doch dieses Feature bietet noch die Möglichkeit für Verbesserungen. Insbesondere könnte analysiert werden, was geeignete Werte für die Prozentsätze der unterschiedlichen Klassen sind. Dabei müsste die Auswahl dieser Werte letztendlich nicht nur von der Höhe des Datenverbrauchs abhängen, sondern es könnte beispielsweise auch die verfügbare Datenrate und die Anzahl an aktiven Geräten berücksichtigt werden. Sinnvoll wäre in diesem Kontext auch noch eine Untersuchung, ob das Verwerfen von Paketen möglicherweise bei bestimmten Anwendungen oder Transportprotokollen zu Problemen führt.

Literaturverzeichnis

[23a] *iproute2 v6.5.0 GitHub*. 6. Sep. 2023. URL: https://github.com/iproute2/iproute2/tree/v6.5.0 (zitiert auf S. 21).

[23b] *tc Manpage v6.5.0*. 6. Sep. 2023. URL: https://github.com/iproute2/iproute2/blob/v6.5.0/man/man8/tc.8 (zitiert auf S. 21).

[CL15] C. Cano, D. J. Leith. „Coexistence of WiFi and LTE in unlicensed bands: A proportional fair allocation scheme“. In: *2015 IEEE International Conference on Communication Workshop (ICCW)*. 2015, S. 2288–2293. DOI: 10.1109/ICCW.2015.7247522 (zitiert auf S. 23).

[CLGS17] C. Cano, D. J. Leith, A. Garcia-Saavedra, P. Serrano. „Fair Coexistence of Scheduled and Random Access Wireless Networks: Unlicensed LTE/WiFi“. In: *IEEE/ACM Transactions on Networking* 25.6 (2017), S. 3267–3281. DOI: 10.1109/TNET.2017.2731377 (zitiert auf S. 23).

[CTG+19] R. Corbel, S. Tuffin, A. Gravey, X. Marjou, A. Braud. „Assessing the Impact of QUIC on Network Fairness“. In: *Journal of Communications* (Jan. 2019), S. 908–914. DOI: 10.12720/jcm.14.10.908-914 (zitiert auf S. 32).

[GF07] R. Ghazizadeh, P. Fan. „Dynamic Priority Scheduling Mechanism through Adaptive InterFrame Space“. In: *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. 2007, S. 1992–1995. DOI: 10.1109/WICOM.2007.498 (zitiert auf S. 23).

[GK08] H. Gong, J. Kim. „Dynamic load balancing through association control of mobile users in WiFi networks“. In: *IEEE Transactions on Consumer Electronics* 54.2 (2008), S. 342–348. DOI: 10.1109/TCE.2008.4560097 (zitiert auf S. 28).

[Her21] J. Herrmann. „Erweiterung des Mininet-Netzwerk-Emulators um einen zeitgesteuerten Scheduling-Mechanismus“. Bachelorarbeit. Universität Stuttgart, 2021, S. 21. DOI: 10.18419/opus-11643 (zitiert auf S. 22).

[HF09] R. He, X. Fang. „A Fair MAC Algorithm with Dynamic Priority for 802.11e WLANs“. In: *2009 International Conference on Communication Software and Networks*. 2009, S. 255–259. DOI: 10.1109/ICCSN.2009.10 (zitiert auf S. 23).

[IEE05] IEEE. „IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements“. In: *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003)* (2005), S. 1–212. DOI: 10.1109/IEEESTD.2005.97890 (zitiert auf S. 19, 20).

[IEE21a] IEEE. „IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications“. In: *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), S. 1–4379. DOI: 10.1109/IEEESTD.2021.9363693 (zitiert auf S. 17).

[IEE21b] IEEE. „IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN“. In: *IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020)* (2021), S. 1–767. DOI: 10.1109/IEEESTD.2021.9442429 (zitiert auf S. 21).

[IEE22] IEEE. „IEEE Standard for Ethernet“. In: *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)* (2022), S. 1–7025. DOI: 10.1109/IEEESTD.2022.9844436 (zitiert auf S. 17).

[KLP16] H. Ko, J. Lee, S. Pack. „A Fair Listen-Before-Talk Algorithm for Coexistence of LTE-U and WLAN“. In: *IEEE Transactions on Vehicular Technology* 65.12 (2016), S. 10116–10120. DOI: 10.1109/TVT.2016.2533627 (zitiert auf S. 23).

[KMG19] C. Kramer, K. Mathews, R. Gotzhein. „Cooperative Fair Bandwidth Scaling in Contention-based Wireless Networks using Time Token Bucket“. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. 2019, S. 1–9. DOI: 10.1109/IPCCC47392.2019.8958764 (zitiert auf S. 24).

[Kol01] A. Kolarov. „Study of the TCP/UDP fairness issue for the assured forwarding per hop behavior in differentiated services networks“. In: *2001 IEEE Workshop on High Performance Switching and Routing (IEEE Cat. No.01TH8552)*. 2001, S. 190–196. DOI: 10.1109/HPSR.2001.923630 (zitiert auf S. 24).

[LDR+23] R. Laidig, F. Dürr, K. Rothermel, S. Wildhagen, F. Allgöwer. „Dynamic Deterministic Quality of Service Model with Behavior-Adaptive Latency Bounds“. In: *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2023, S. 127–136. DOI: 10.1109/RTCSA58653.2023.00024 (zitiert auf S. 15, 36).

[NMK+23] H. K. Neelakantam, B. P. Makala, P. Kommoju, D. S. R. Ogirala, P. N. Suneel, M. K. D. „LTE and WLAN fair co-existence in unlicensed bands“. In: *2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*. 2023, S. 1–5. DOI: 10.1109/ICEEICT56924.2023.10157672 (zitiert auf S. 23).

[RLPP10] A. Riza, T. C. Ling, K. Phang, H. Pee. „Improving QOS In WLAN Using Dynamic Weighted Fair Scheduling“. In: *Malaysian Journal of Computer Science* 23 (Sep. 2010). DOI: 10.22452/mjcs.vol23no2.2 (zitiert auf S. 23).

[SHB18] T. Szigeti, J. Henry, F. Baker. *Mapping Diffserv to IEEE 802.11*. RFC 8325. Feb. 2018. DOI: 10.17487/RFC8325. URL: https://www.rfc-editor.org/info/rfc8325 (zitiert auf S. 20).

[VMS+04] J. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, O. Sallent. „A hierarchical token bucket algorithm to enhance QoS in IEEE 802.11: proposal, implementation and evaluation“. In: *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*. Bd. 4. 2004, 2659–2662 Vol. 4. DOI: 10.1109/VETECF.2004.1400539 (zitiert auf S. 24).

Alle URLs wurden zuletzt am 24. 11. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift