Institut für Kommunikationsnetze und Rechnersysteme
Universität Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Diplomarbeit Nr. 3156

Entwurf und Realisierung eines Testbeds für die Untersuchung von verzögerungsbasierten Staukontrollmechanismen

Mark Hübler

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. Andreas Kirstädter
Betreuer:	Dipl.-Ing. Christian Blankenhorn Dipl.-Ing. Mirja Kühlewind
begonnen am:	02. März 2011
beendet am:	01. September 2011
CR-Klassifikation:	I.6.4, I.6.5, C.2.1, C.2.5, C.2.6

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

Hintergrund

Die Erfahrung zeigt, dass beim Zugriff auf gemeinsame Netzwerkressourcen immer wieder einzelne Netzwerkpfade überlastet werden. Zur Auflösung bzw. Vermeidung von Staus enthält das Transmission Control Protocol (TCP) einen Staukontrollmechanismus. Um die Staukontrolle zu optimieren, wurden verschiedene Varianten für diesen Mechanismus entwickelt. Dabei kann zwischen zwei Typen der Staukontrolle unterschieden werden – verlustbasierte und verzögerungsbasierte Staukontrolle. Wie der Name andeutet, beruhen verzögerungsbasierte Staukontrollmechanismen auf der Messung der Paketverzögerung. Steigende Paketverzögerungen werden als drohender Stau interpretiert. Jedoch sind Staus nicht die einzigen Einflüsse, welche die Paketverzögerung beeinflussen. Dies kann zu falschen Reaktionen der Staukontrolle führen.

Im Rahmen dieser Arbeit wurden solche Störgrößen für die verzögerungsbasierte Staukontrolle identifiziert und modelliert, um sie im Netzemulator-Framework IKR-EmuLib umzusetzen. Mit Hilfe des prototypisch entstandenen Testbeds können verschiedene Messverfahren und Staukontrollmechanismen verglichen werden.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden die unterschiedlichen Kategorien der TCP-Staukontrolle vorgestellt, verschiedene Komponenten der Paketverzögerung dargestellt und auf ihre Relevanz untersucht. In diesem Zusammenhang wird der Medienzugriff im WLAN näher betrachtet. Außerdem werden die Gründe für den Ansatz der Emulation dargelegt.

Kapitel 3 – Modellierung von Verzögerungen: Hier werden zwei Störgrößen, nämlich die Verzögerung durch Verarbeitung in Netzwerkgeräten und die Verzögerung bei WLAN-Verbindungen näher untersucht und modelliert.

Kapitel 4 – Aufbau des Testbeds: In diesem Kapitel werden das Netzemulator-Framework IKR EmuLib und der Aufbau des Testbeds vorgestellt.

Kapitel 5 – Validierung: Hier werden mittels des Testbeds Messungen durchgeführt und die Ergebnisse der Arbeit überprüft.

Kapitel 6 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt mögliche Weiterentwicklungen sowie Anwendungsmöglichkeiten vor.

2 Grundlagen

In diesem Kapitel werden die Grundlagen dieser Arbeit erläutert. Zunächst wird die grundsätzliche Funktionsweise der TCP-Staukontrolle gezeigt und zwischen verlustbasierter und verzögerungsbasierter Staukontrolle unterschieden. Um ihren Einfluss auf die verzögerungsbasierte Staukontrolle zu untersuchen, werden im Anschluss die verschiedenen Komponenten der Verzögerung in paketvermittelten Netzwerken betrachtet. Es folgt eine nähere Beschreibung der Verzögerung durch Verarbeitung in Netzwerkgeräten, sowie der Verzögerung im WLAN.

2.1 TCP-Staukontrolle

Das Transmission Control Protocol (TCP) ist ein grundlegendes Protokoll der Internetprotokollfamilie. Es bietet auf der Transportschicht paketvermittelte, zuverlässige und verbindungsorientierte Datenübertragung. [KR05]

<table>
<tr><th>OSI-Schicht</th><th>OSI-Erweiterung</th><th>TCP/IP-Schicht</th><th>Beispiele</th></tr>
<tr><td>Anwendung (7)</td><td rowspan="5"></td><td>Anwendung</td><td>DNS, POP, HTTP</td></tr>
<tr><td>Darstellung (6)</td><td rowspan="2"></td><td>X.216</td></tr>
<tr><td>Sitzung (5)</td><td>L2TP</td></tr>
<tr><td>Transport (4)</td><td>Transport</td><td>TCP, UDP</td></tr>
<tr><td>Vermittlung (3)</td><td>Internet</td><td>IP, ICMP</td></tr>
<tr><td rowspan="2">Sicherung (2)</td><td>Logical Link Control (2b)</td><td rowspan="3">Host-zu-Netzwerk</td><td rowspan="3">Ethernet, WLAN</td></tr>
<tr><td>Media Access Control (2a)</td></tr>
<tr><td>Bitübertragung (1)</td><td></td></tr>
</table>

Tabelle 2.1: Einordnung von TCP im OSI- und TCP/IP-Schichtenmodell [Tan03, ITU94, Cis07]

Ein wichtige Aufgabe von TCP ist die Überlast- oder Staukontrolle. Wenn Netzressourcen gleichzeitig genutzt werden, kann dies zur Überlastung des Netzwerkes führen. Die Staukontrolle dient dazu, eine solche Überlastung zu vermeiden oder aufzulösen. Die Staukontrolle muss dabei von der Flusskontrolle abgegrenzt werden. Während es bei der Staukontrolle darum geht, ob ein Teilnetz in der Lage ist, den angebotenen Verkehr zu bewältigen, geht es bei der Flusskontrolle darum, dass ein bestimmter Sender einen bestimmten Empfänger nicht mit mehr Daten, als dieser bewältigen kann, überschwemmt. Dafür ist die Verarbeitungsgeschwindigkeit des Empfängers entscheidend, nicht die Übertragungskapazität

des Netzes. [Tan03] In dieser Arbeit geht es um die Staukontrolle. Die TCP-Staukontrolle versucht Überlastungen von Netzwerkpfaden aufzulösen oder zu vermeiden, indem die beteiligten Stationen ihr Sendeverhalten anpassen, sobald sie einen (drohenden) Stau erkennen. Dabei gibt es zwei Kategorien von Verfahren – verlustbasierte und verzögerungsbasierte Staukontrolle.

2.1.1 Verlustbasierte Staukontrolle

Wenn eine Verbindung zu stark belastet ist, dann führt dies bei TCP zu Paketverlusten[1]. Dies liegt daran, dass bei Überlastung die Warteschlange eines Netzwerknotens so stark wachsen kann, dass der Pufferspeicher voll ist. In diesem Fall können ankommende Pakete nicht angenommen werden. Folglich werden sie fallen gelassen. [KR05] Unabhängig davon, ob ein Paket tatsächlich fallen gelassen wurde, wartet der Sender nur eine gewisse Zeit (*timeout*) auf eine Empfangsbestätigung. Bleibt diese aus, geht er davon aus, dass das Paket verloren gegangen ist und versucht es erneut. Dies hat zur Folge, dass Pakete bei Stau verstärkt mehrfach übertragen werden müssen, was die Last noch zusätzlich erhöht. Die Aufgabe der Staukontrolle ist es, das Sendeverhalten so zu ändern, dass der Stau vermieden wird oder sich auflöst. Umgekehrt soll jedoch auch die Bandbreite möglichst gut ausgenutzt und die Senderate nicht unnötig niedrig gehalten werden.

Die Grundannahme verlustbasierter Staukontrollmechanismen ist, dass die Ursache für (verstärkte) Paketverluste eine Überlastung der Verbindung ist. Im Grundsatz funktionieren diese Mechanismen so, dass die Senderate langsam erhöht wird, bis Paketverluste auftreten. Wenn Paketverluste auftreten, was der Sender an den ausbleibenden Empfangsbestätigungen (ACK) erkennt, wird die Senderate deutlich reduziert, um den Stau aufzulösen. Im Anschluss wird die Rate wieder langsam erhöht, bis erneut Paketverluste auftreten. Verlustbasierte Staukontrollmechanismen reagieren also erst auf einen mutmaßlich erkannten Stau. Dabei tastet sich TCP so lange an die mögliche Senderate heran, bis es zu Paketverlusten kommt. Es kommt also sogar bei alleiniger Nutzung der Verbindung regelmäßig zu einem „Stau“ . Dieses Verhalten ist notwendig, weil einerseits keine Bandbreite verschenkt werden soll, aber andererseits keine Möglichkeit besteht, die verfügbare Bandbreite zu ermitteln. [BP95] Das Sendeverhalten bei verlustbasierter Staukontrolle kann wie in der vereinfachten Abbildung 2.1 aussehen.

2.1.2 Verzögerungsbasierte Staukontrolle

Im Gegensatz zu verlustbasierten Staukontrollmechanismen reagieren die verzögerungsbasierten Mechanismen nicht nur auf Paketverluste sondern versuchen über die Verzögerung

[1] Korrekterweise müsste je nach TCP/IP bzw. OSI-Schicht zwischen Rahmen (*frames*), Paketen und Segmenten unterschieden werden. Diese Unterscheidung würde im Zusammenhang mit dieser Arbeit jedoch hauptsächlich Verwirrung stiften, so dass ich – wie einige der zitierten Arbeiten – fast durchgängig von Paketen sprechen werde.

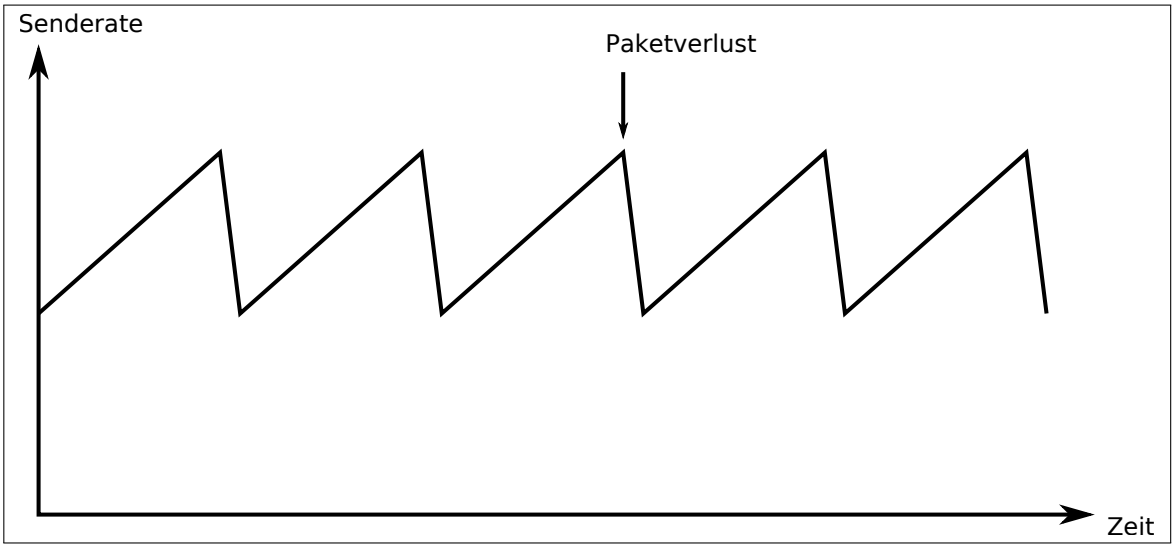

Abbildung 2.1: Prinzipskizze verlustbasierte Staukontrolle

von Paketen drohende Staus zu erkennen. Dazu wird in der Regel die sogenannte Round Trip Time (RTT) gemessen. Dies ist die Zeit vom Absenden eines Pakets bis zum Eintreffen der Empfangsbestätigung. Im Grundsatz funktionieren verzögerungsbasierte Staukontrollmechanismen so, dass eine steigende RTT als drohender Stau interpretiert wird. Entsprechend wird in diesem Fall die Senderate verringert. Wenn sich die RTT wieder erholt, wird auch die Senderate wieder erhöht. In 2.2 sieht man in einer Prinzipskizze wie sich ein verzögerungsbasierter Staukontrollmechanismus idealerweise an eine optimale Senderate heran tastet. Dies ist jedoch wirklich idealisiert. In Kapitel 5 wird sich zeigen, dass dies beim betrachteten TCP Vegas bei weitem nicht so gut funktioniert. Der Vorteil dieser Vorgehensweise ist, dass

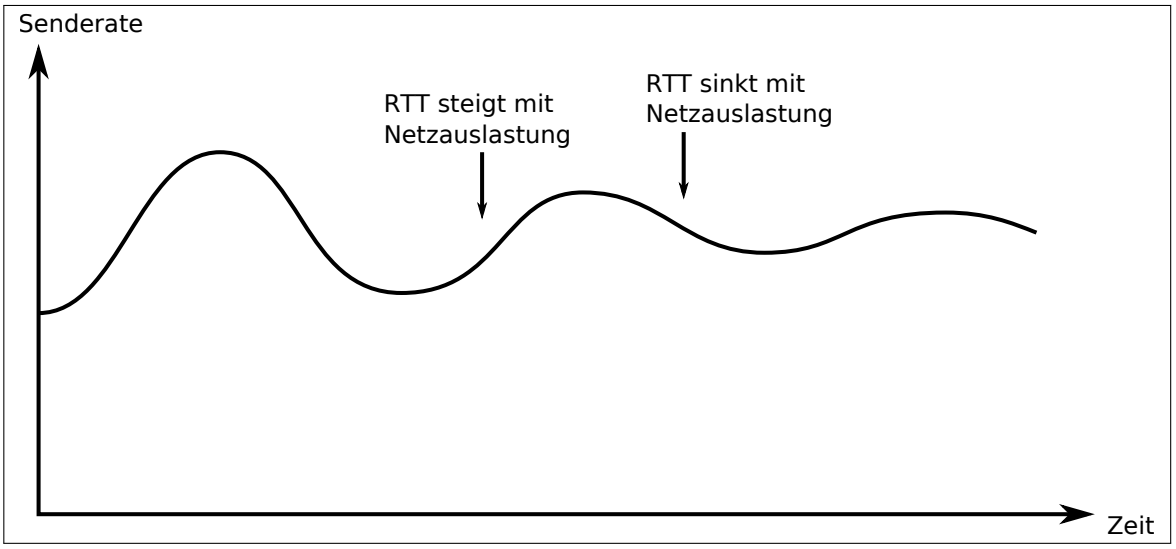

Abbildung 2.2: Prinzipskizze verzögerungsbasierte Staukontrolle

Staus und der damit einhergehende, vorübergehende Einbruch der Datenübertragungsrate vermieden werden. Voraussetzung für ein gutes Funktionieren ist aber neben schnellen und präzisen Messungen der Verzögerungen, dass die Ursache von Verzögerungsschwankungen tatsächlich mit der Belastung des Netzwerkpfades zusammen hängt oder der Mechanismus zwischen dieser und anderen Ursachen unterscheiden kann.

2.2 Ursachen für Verzögerungen im Netz

Da verzögerungsbasierte Staukontrollmechanismen auf schwankende Paketverzögerungen reagieren, ist es wichtig, Ursachen und Charakteristik von Verzögerungen zu kennen.

Die Verzögerung in einem paketvermittelten Netzwerk setzt sich aus verschiedenen Komponenten zusammen. Nach Kurose/Ross sind die vier wichtigsten Komponenten die *processing delay* (Verarbeitungsverzögerung), *queueing delay* (Warteschlangenverzögerung), *transmission delay* (Übertragungsverzögerung) und *propagation delay* (Ausbreitungsverzögerung). Diese treten an bzw. zwischen jedem Netzwerkknoten auf. [KR05]

- Die *processing delay* ist die Zeit die auf einem Netzwerkknoten benötigt wird, um das Paket zu bearbeiten. Dazu gehört es, den Paketheader daraufhin zu untersuchen, wohin das Paket weiter geschickt werden soll, aber bspw. auch die Zeit zur Überprüfung auf Bitfehler. Moderne Router erledigen dies im Mikrosekundenbereich oder schneller. Allerdings können neben der einfachen Weiterleitung (*store and forward*) von Paketen auch weitere Aufgaben für Verzögerungen sorgen. Man denke an Firewalls, Network Address Translation (NAT) oder Virtual Private Networks (VPN). In diesen Fällen kann die Verzögerung deutlich größer werden.
- Die *queueing delay* bezeichnet die Zeit, die ein Paket warten muss, bis es verschickt werden kann, weil andere Pakete vorher an der Reihe sind oder gerade übertragen werden. Im Idealfall beträgt die *queueing delay* null. Im schlechtesten Fall „ewig“ (was im Falle von TCP zur Folge hat, dass das Paket verworfen wird und neu übertragen werden muss).
- Die *transmission delay* ist die Zeit, die benötigt wird, ein Paket auf die Leitung zu schicken. Sie berechnet sich aus der Paketlänge geteilt durch die Übertragungsrate der Verbindung zum nächsten Knoten. Diese Verzögerungskomponente bewegt sich in der Größenordnung von Mikro- bis Millisekunden.
- Die *propagation delay* ist die Zeit, die ein Bit von einem Knoten zum Nächsten braucht. Sie berechnet sich als Ausbreitungsgeschwindigkeit geteilt durch die Entfernung. Die Ausbreitungsgeschwindigkeit hängt vom verwendeten Medium (Glasfaser, Kupfer, Luft etc.) ab und entspricht der Lichtgeschwindigkeit oder knapp darunter. In Wide Area Networks (WAN) bewegt sich die *propagation delay* im Millisekundenbereich.

In Tabelle 2.2 finden sich Größenordnungen der Verzögerungskomponenten für eine 200km lange 1GB/s-Leitung, ein 1250-byte-Paket und einen 100-MIPS-Prozessor.

Verzögerungsart	Einfache Paketweiterleitung	komplexe Payload-Modifikationen
Processing delay	10µs	1000µs
Queueing delay	0-∞	0-∞
Transmission delay	10µs	10µs
Propagation delay	1000µs	1000µs

Tabelle 2.2: Größenordnung der Verzögerungskomponenten im Netzwerk [RWW04]

Die *processing delay* ist normalerweise sehr gering, kann aber sehr unterschiedliche Werte annehmen. Die Höhe hängt von den durchlaufenen Anwendungen und der verwendeten Hardware auf den Knoten ab. Bei komplexen Anwendungen macht sie einen erheblichen Anteil der Verzögerung aus.

Die *queueing delay* unterliegt enormen Schwankungen von null bis unendlich. Eine steigende *queueing delay* ist ein Anzeichen für eine sich füllende Warteschlange und damit für drohende Überlastung. Deswegen ist die Korrelation zwischen Verzögerung und Last als Grundannahme in die verzögerungsbasierte Staukontrolle eingeflossen.

Die *transmission delay* nimmt nur einen geringen Anteil an der Gesamtverzögerung ein. Sie ist proportional zur Paketlänge.

Zumindest bei großen Netzwerken nimmt die *propagation delay* einen erheblichen Anteil an der Verzögerung ein. Sie ist konstant, da sie nur vom Übertragungsmedium und von der Entfernung abhängt.

Selbstverständlich können sich alle diese Verzögerungskomponenten ändern, wenn sich das Routing im Netzwerk ändert. Auf einer neuen Route findet sich eine andere Konstellation von Netzwerkgeräten, Leitungen und Entfernung, was sich auf *transmission delay*, *processing delay*, *propagation delay* und auch auf andere, untergeordnete Verzögerungsarten auswirkt. Im Falle der *processing delay* können sich sogar die auf den Knoten laufenden Anwendungen ändern. Allerdings geht es insbesondere bei den hier relevanten komplexen Anwendungen oftmals um Knoten, die in jedem Fall durchlaufen werden müssen. Sinnvollerweise kann bspw. eine Firewall nicht durch verändertes Routing umgangen werden.

Die Ursache für Routingänderungen sind allerdings häufig Überlast und Ausfälle von Verbindungen. Dementsprechend ist es oft sogar gewünscht, dass sich insbesondere die *queueing delay* mit der Route ändert. In jedem Fall muss sich der Staukontrollmechanismus auf die neue Route einstellen, ohne dass man diesen Vorgang als Störung der Funktionalität bezeichnen könnte. Deshalb wird bei den folgenden Betrachtungen von einer stabilen Route ausgegangen wird.

Wenn man von der *processing delay* absieht, dann ist die Schwankung der Paketverzögerung bei Verbindungen über Kabel ohne Überlast offenbar sehr gering ist. (Abbildung 2.3) Bei WLAN-Verbindungen sieht dies anders aus. Auch ohne Stau kommt es hier zu erheblichen Schwankungen von Paket zu Paket (Abbildung 2.4) Die Ursache dafür liegt in einer Verzögerungskomponente, die bei der Einteilung aus [KR05] nicht berücksichtigt ist, weil sie bei kabelgebundenen Netzwerken nur eine untergeordnete Rolle spielt, nämlich die Verzögerung

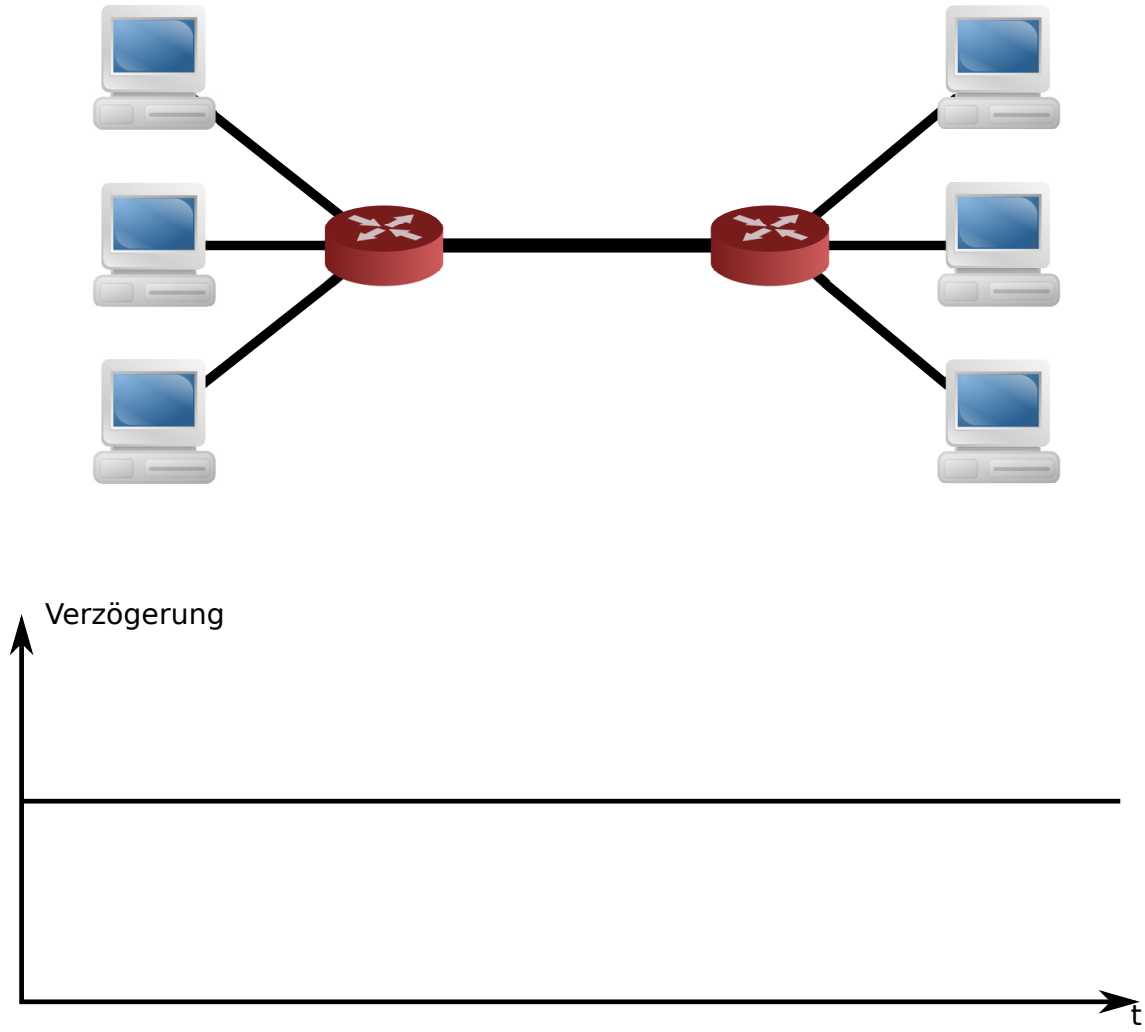

Abbildung 2.3: Paketverzögerung in kabelgebundenen Netzwerken

beim Medienzugriff (*media access delay*). Diese Verzögerung kann bei WLAN-Verbindungen erheblich schwanken und ist damit geeignet verzögerungsbasierte Staukontrollmechanismen zu stören. Die *media access delay* wird in Abschnitt 2.2.2 genauer untersucht.

2.2.1 Verarbeitungsdauer in Netzwerkgeräten

Wie Abschnitt 2.2 gezeigt hat, kann die *processing delay*, also die Verarbeitungsdauer in Netzwerkgeräten sehr unterschiedliche Werte annehmen. Dies legt einen möglichen Einfluss auf verzögerungsbasierte Staukontrollmechanismen nahe. Deshalb wird sie in dieser Arbeit näher untersucht.

Zunächst muss geklärt werden, welche Faktoren Einfluss auf die *processing delay* haben. Deren Größe hängt hauptsächlich von drei Faktoren ab:

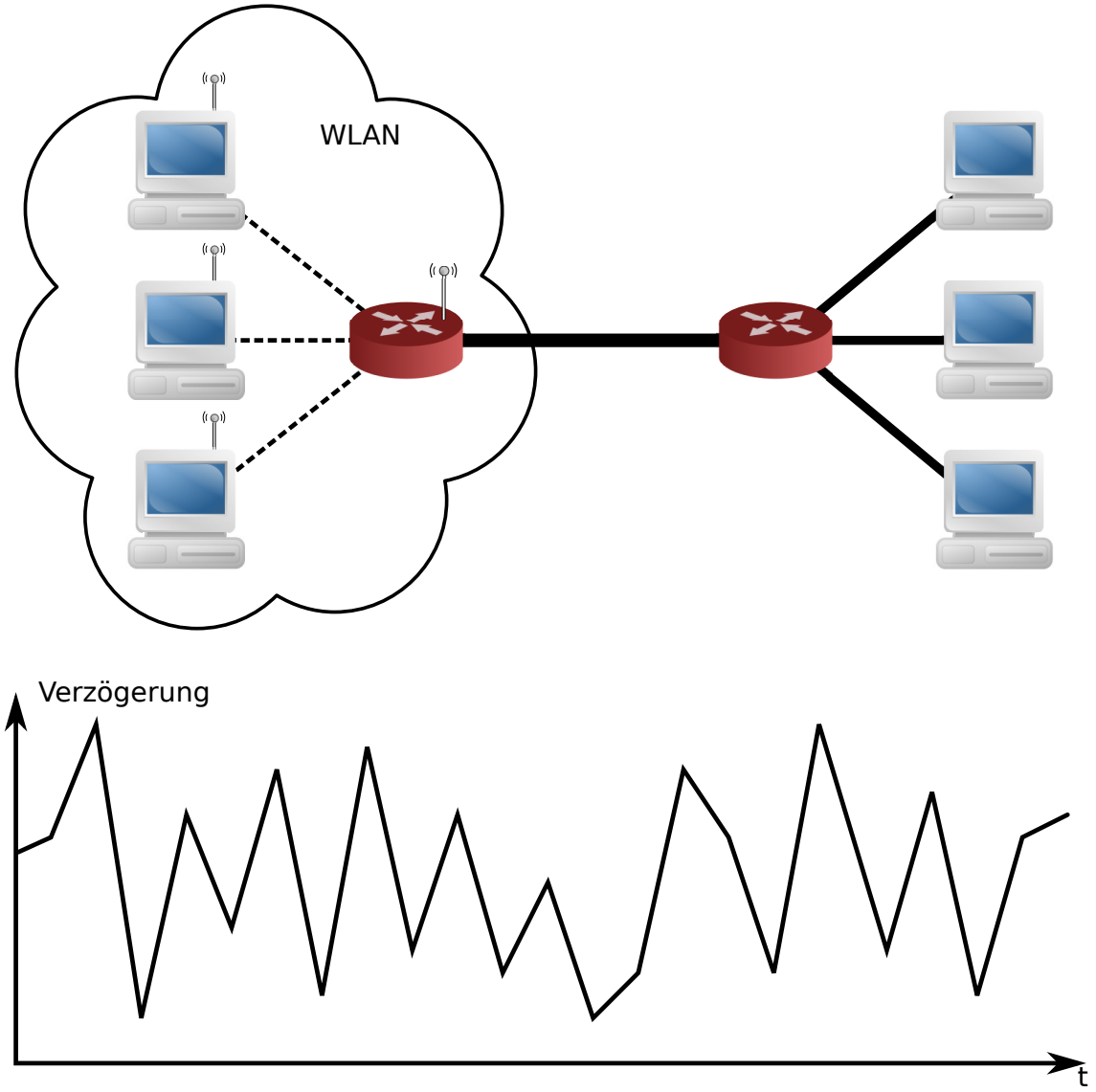

Abbildung 2.4: Paketverzögerung in kabelgebundenen Netzwerken

1. Art bzw. Komplexität der Anwendung
2. Leistungsfähigkeit des Netzwerkgerätes
3. Paketeigenschaften

In Abschnitt 2.2 wurde für die Betrachtung ausgeschlossen, dass sich das Routing ändert. Folglich bleiben die durchlaufenen Netzwerkknoten und damit die Leistungsfähigkeit der fraglichen Netzwerkgeräte, sowie die darauf laufenden Anwendungen gleich. Damit bleiben noch Paketeigenschaften, die den Rechenaufwand verändern bzw. von vorn herein eine andere Behandlung zur Folge haben können. Eine solche unterschiedliche Behandlung erfolgt

in der Praxis hauptsächlich aufgrund der Quell- und Zieladressen, die sich innerhalb einer Verbindung logischerweise nicht ändern. Wenn die Verarbeitungsdauer nicht vom tatsächlichen Inhalt des Paketes abhängt, wie dies z.B. bei einem content-based Switch [AAP+00] der Fall ist, dann bleibt als Größe, die einen Einfluss auf die Verarbeitungsdauer hat, die Paketlänge. Wie die Verzögerung in Netzwerkgeräten in Abhängigkeit von der Paketlänge in ein Modell überführt werden kann, wird in Abschnitt 3.1 gezeigt.

2.2.2 Verzögerungen im WLAN aufgrund des Medienzugriffs

Wie in Abschnitt 2.2 festgestellt wurde, kommt es bei WLAN-Verbindungen zu erheblichen Schwankungen der Paketverzögerung. Es liegt nahe, die Ursache nur bei der schlechten Qualität von WLAN-Verbindungen zu suchen. Diese führt zu Paketverlusten auf der MAC-Schicht (Media Access Control, siehe Tabelle 2.1), welche sich als Verzögerungen auf die Transportschicht übertragen und im schlimmsten Fall TCP-Neuübertragungen notwendig machen. Dies ist ein bekanntes Problem, das auch verlustbasierte Staukontrollmechanismen betrifft. [Tan03] Neben der Unzuverlässigkeit von WLAN-Verbindungen, führt aber schon die prinzipielle Funktionsweise von WLAN zu Verzögerung beim Medienzugriff (*media access delay*), die stark unterschiedlich ausfallen können.

Bei der gemeinsamen Nutzung eines Kanals, muss auf der MAC-Schicht vermieden werden, dass mehrere Stationen gleichzeitig Pakete übertragen. Andernfalls kommt es zu Kollisionen. Die Pakete werden dann nicht korrekt übertragen und müssen erneut gesendet werden. Im Falle von Kabelverbindungen über Ethernet wird dazu ein Verfahren mit Kollisionserkennung – CSMA/CD (Carrier Sense Multiple Access/Collision Detection) verwendet. Wenn eine Station anfängt zu übertragen, und dann bemerkt, dass bereits eine andere Station überträgt, stellt sie den eigenen Sendeversuch sofort ein. So kann die zeitliche Auswirkung einer Kollision minimiert werden. Voraussetzung dafür ist, dass die Station in der Lage ist, gleichzeitig zu übertragen und auf dem Kanal zu horchen. Im Falle von WLAN ist dies nicht möglich, weil hier die Stationen diese Fähigkeit nicht haben. Wenn eine WLAN-Station angefangen hat zu übertragen, dann fährt sie damit auch im Falle einer Kollision fort, bis das gesamte Paket gesendet ist. Die Auswirkung einer Kollision ist deswegen deutlich größer, weil der Kanal länger belegt ist. Aus diesem Grund wird bei WLAN, sofern keine zentrale Steuerung vorhanden ist, ein Verfahren mit Kollisionsvermeidung – CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) verwendet. [KR05]

Im Falle eines Medienzugriffs mittels der Distributed Coordination Function (DCF)[2] läuft das Verfahren folgendermaßen ab. (Abbildung 2.5 enthält eine vereinfachte Darstellung, bei der vor allem auf die Unterscheidung der verschiedenen Wartezeiten verzichtet wurde.)

[2] Bei DCF arbeiten alle Stationen unabhängig, ohne zentrale Steuerung. Der DCF-Modus muss von allen Implementierungen des IEEE 802.11-Standards unterstützt werden. Daneben gibt es auch einen optionalen PCF-Modus (Point Coordination Function), bei dem ein Zugriffspunkt alle Aktivitäten in seiner Zelle steuert und so Kollisonen verhindert. [Tan03]

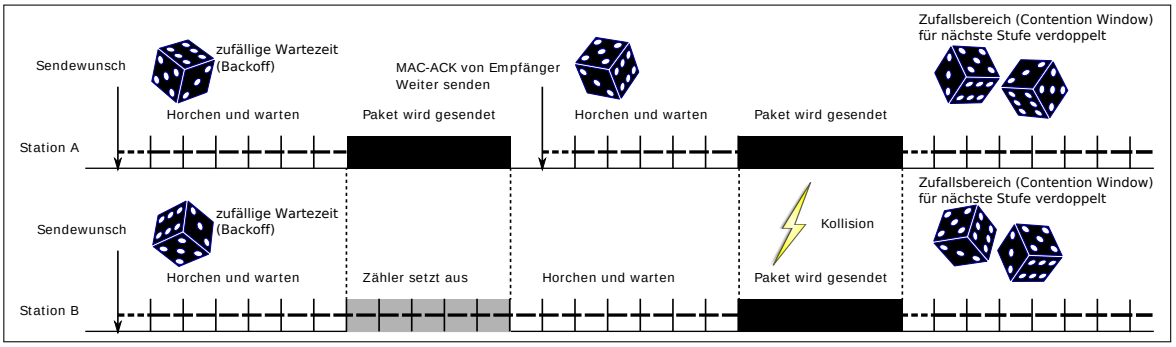

Abbildung 2.5: Vereinfachte Darstellung des WLAN-Medienzugriffs mit DCF/Basic Access

Wenn eine Station im WLAN ein Paket versenden will, dann horcht sie zunächst auf dem Kanal, ob dieser frei ist. Wenn dies eine DIFS (Distributed Coordination Function Interframe Spacing) genannte Zeitspanne lang der Fall ist, würfelt sie einen zufälligen *Backoff*-Wert[3] aus. Dieser Zähler wird nun solange der Kanal frei ist, jede *slot time* um eins reduziert. Wenn der Kanal belegt ist, wird das Herunterzählen so lange unterbrochen, bis der Kanal wieder ein DIFS lang frei ist. Sobald der Backoff-Zähler null erreicht, wird das Paket übertragen. Empfängt die Empfängerstation das Paket erfolgreich, wartet sie ein SIFS (Short Interframe Spacing) lang und sendet dann eine Empfangsbestätigung (ACK)[4] zurück. Erhält der Sender kein ACK, wird davon ausgegangen, dass das Paket verloren gegangen ist. Dazu kann es bspw. kommen wenn zwei Stationen zufällig gleichzeitig übertragen wollen. In diesem Fall setzt die Station einen *Retry-* oder *Retransmission*-Zähler[5] eins herauf. Außerdem wird der Zufallsbereich für den Backoff-Wert, das sogenannte *Contention Window* verdoppelt bis ein Maximalwert erreicht wird. Wenn der Retry-Zähler das Retry-Limit erreicht, wird das Paket verworfen. Wird das Paket erfolgreich versendet, dann wird der Retry-Zähler wieder auf null und die Größe des Contention Windows auf den Startwert gesetzt. Neben diesem einfachen Vorgehen (Basic Access) gibt es noch ein erweitertes Verfahren, das besonders bei großen Datenpaketen Anwendung findet: RTS/CTS Access (Request to Send/Clear to Send). Dabei wird statt des Datenpakets zunächst ein sehr viel kleineres RTS-Paket gesendet, um den Kanal zu reservieren. Wenn die Empfänger-Station das RTS-Paket erhält, antwortet sie nach einem SIFS mit CTS und erlaubt damit das Senden des eigentlichen Datenpakets. Der Empfang des Datenpakets wird wiederum mit einem ACK quittiert. Alle anderen Stationen warten währenddessen und senden nicht selbst. Die Länge der Kanalbelegung ist in den RTS- und CTS-Paketen enthalten. Wenn der Sender kein CTS erhält wird (wie beim fehlenden ACK) der Retry-Zähler um eins erhöht und das Contention Window vergrößert. [RVP09]

Um einen Eindruck für die Größenverhältnisse der einzelnen Zeiten zu bekommen, seien hier exemplarisch die Werte nach IEEE 802.11b[iee07] angeführt. Danach ist ein SIFS 10µs, ein DIFS 50µs und eine slot time 20µs lang. Die Größe des Contention Windows geht von

[3] engl.: to back off – sich zurückhalten, zurückweichen
[4] Dieses ACK auf MAC-Ebene darf nicht mit dem TCP-ACK aus der Transportschicht verwechselt werden.
[5] engl.: to retry – erneut versuchen / retransmission – Neuübertragung

31 bis 1023. Wie man aus diesen Größenverhältnissen leicht erkennt, hängt die Verzögerung maßgeblich vom zufällig gewählten Backoff-Wert sowie der Anzahl der notwendigen Übertragungsversuche ab. Es können große Ausreißer bei der Verzögerung entstehen. Es ist anzunehmen, dass sich diese Verzögerungen ähnlich wie Paketverluste [KR05] von der MAC-Schicht auf die höheren Netzwerkschichten übertragen. Da TCP die Ursache der Verzögerung nicht erkennen kann, besteht die Gefahr, dass verzögerungsbasierte Staukontrollmechanismen davon gestört werden.

2.3 Simulation und Emulation

Wie gezeigt wurde besteht Bedarf unterschiedliche TCP-Varianten in verschiedenen Szenarien zu untersuchen. In der Praxis stellen sich dabei diverse Probleme. Zunächst einmal stehen real existierende Netzwerke in der Regel nicht für die alleinige Nutzung zu Testzwecken zur Verfügung. Zumindest große Netzwerke können schon aus Platzgründen auch nicht ohne weiteres auf- bzw. nachgebaut werden. Zudem müsste die notwendige Hard- und Software vorhanden sein oder beschafft werden, was ggfs. mit hohen Kosten verbunden ist. Eine Änderung des Aufbaus oder der Parameter ist in der Regel mit hohem Aufwand verbunden.

Ein weiterer Faktor, der insbesondere den Fall der WLAN-Verbindungen betrifft, ist die Reproduzierbarkeit. Ein WLAN kann kaum hundertprozentig nach außen abgeschottet werden. Es besteht immer die Gefahr, dass Einflüsse von außen die Messergebnisse stören. Ein einfaches Beispiel hierfür wäre, dass nebenan ebenfalls ein WLAN betrieben wird.

Eine Lösung dieser Probleme besteht darin, ein mathematisches Modell des Netzwerks zu erstellen und es zu simulieren. Weit verbreitet ist dabei die auf diskreten Ereignissen basierende Simulation. Diskrete Ereignisse bedeutet hier, dass sich der Systemzustand nur an einer abzählbaren Menge von einzelnen Zeitpunkten unmittelbar ändern kann. Zu diesen Zeitpunkten tritt ein Ereignis (*event*) auf. [LK91] Dies klingt zunächst einfach, aber hinreichend gute Modelle können sehr komplex sein, was nicht nur bei der Berechnung, sondern auch bei der Erstellung des Modells einen hohen Aufwand bedeuten kann.

Eine Möglichkeit, den Aufwand zu reduzieren ist die Netzwerkemulation. Die Netzwerkemulation ist eine Mischung von Realität und Simulation bei der reale Elemente einer eingesetzten Netzwerkanwendung mit simulierten Elementen kombiniert werden. Es handelt sich sozusagen um eine Simulation mit Schnittstelle zur Realität. Ein wesentlicher Unterschied zwischen Simulation und Emulation ist, dass die Zeit in der Simulation virtuell und unabhängig von der wirklichen Zeit ist. Die Emulation muss dagegen aufgrund der Schnittstelle zur Realität in Echtzeit ausgeführt werden. [NGKR06, GRL05]

Für diese Arbeit wurde das auf Paketebene arbeitende Netzemulator-Framework IKR EmuLib verwendet. Dieses baut auf der ereignisbasierten Simulationsumgebung IKR SimLib auf und wird im Abschnitt 4.1 näher vorgestellt wird.

3 Modellierung von Verzögerungen

In Kapitel 2 wurden zwei Einflüsse herausgearbeitet, welche zu Schwankungen in der Paketverzögerung führen und damit die Funktion verzögerungsbasierter Staukontrollmechanismen stören können. Erstens die Verarbeitung in Netzwerkgeräten und zweitens Verbindungen über WLAN. Diese sollen im Folgenden modelliert werden. Zunächst die Verzögerung in Netzwerkgeräten.

3.1 Modell für die Verzögerung durch Verarbeitung in Netzwerkgeräten

Router und andere Netzwerkgeräte müssen weit mehr leisten als einfach nur Pakete weiter zu leiten. Firewalls, VPN, NAT, Verschlüsselung und andere Anwendungen benötigen Rechenzeit und verzögern so die Ankunft von Paketen. Es gibt einige Forschungsarbeiten dazu, wie groß der Rechenaufwand ist bzw. wie sehr die Netzwerkgeräte dadurch belastet werden. So gibt es verschiedene Benchmark-Suites für Netzwerkprozessoren [WF00, MMSH01].

Damit, wie sich dies in Paketverzögerungen übersetzt, beschäftigen sich jedoch relativ wenig Arbeiten. Drei Arbeiten, die sich mit dieser Fragestellungen befassen sind [PMF+03], [CMZ+04] und [RWW04]. Davon wird jedoch nur in [RWW04] ein Modell entwickelt, das die Verzögerung eines einzelnen Pakets in einem einzelnen, parametrisierbaren Netzwerkprozessor berechnet. Deshalb habe ich es habe ich für mein Testbed gewählt und umgesetzt. Im Folgenden wird dieses Modell vorgestellt.

Das Netzwerkprozessor-Modell geht davon aus, dass Verarbeitungskosten einer Anwendung weitgehend systemunabhängig aus der Zahl der notwendigen Prozessorbefehle und Speicherzugriffe berechnet werden kann. Deswegen beruht es auf folgenden Parametern für die jeweilige Anwendung a.

- Befehle pro Paket α_a (unabhängig von der Länge des Pakets)
- Befehle pro Byte β_a
- Speicherzugriffe pro Paket γ_a (unabhängig von der Länge des Pakets)
- Speicherzugriffe pro Byte δ_a
- Paketlänge l

Näherungsweise können die Zahl der Befehle i und die Zahl der Speicherzugriffe m für eine Anwendung a in Abhängigkeit von der Paketlänge l wie folgt berechnet werden.

$$i_a(l) = \alpha_a + \beta_a \cdot l \tag{3.1}$$
$$m_a(l) = \gamma_a + \delta_a \cdot l \tag{3.2}$$

Nun müssen die Zahl der Befehle und Speicherzugriffe noch in die benötigte Verarbeitungszeit übersetzt werden. Das Modell geht von einem RISC-Prozessor aus. RISC-Prozessoren arbeiten pro Takt einen Befehl ab. Deshalb kann die Zahl der Befehle leicht in die entsprechende Verarbeitungszeit $t_{p,a}$ für die Anwendung a auf einem Prozessor p mit der Taktfrequenz f umgewandelt werden.

$$t_{p,a}(l) = \frac{i_a(l)}{f} \tag{3.3}$$

Für den Speicherzugriff wird eine durchschnittliche Zugriffsdauer t_{mem} angenommen, so dass man die Speicherzugriffszeit $t_{m,a}$ für eine Anwendung erhält.

$$t_{m,a}(l) = m_a(l) \cdot t_{mem} \tag{3.4}$$

Damit ergibt sich die Verzögerung $t_a(l)$, die ein Paket aufgrund der Verarbeitung erfährt, als die Summe von verbrauchter Rechen- und Speicherzugriffszeit.

$$t_a(l) = t_{p,a}(l) + t_{m,a}(l) = \frac{\alpha_a + \beta_a \cdot l}{f} + \gamma_a + \delta_a \cdot l \cdot t_{mem} \tag{3.5}$$

Einschränkungen des Modells Das gezeigte Modell für die *processing delay* in einem Netzwerkgerät hat den Vorteil, dass es sehr einfach ist, aber trotzdem sowohl die Verzögerung pro Paket als auch die Verzögerung pro byte erfasst.

Es unterliegt aber auch gewissen Einschränkungen. So wird implizit ein lineares Verhalten vorausgesetzt. Dies gilt zwar für Anwendungen wie Paket-Weiterleitungen und IPSec, aber bspw. nicht für die schon in Abschnitt 2.2.1 erwähnten content-based Switche. Ein Hindernis bei der Benutzung kann es zudem sein, dass die Parameter ohne detaillierte Spezifikationen schwer herzuleiten sind. Außerdem führt die Benutzung von Co-Prozessoren und anderen Hardwarebeschleunigern zu einer heterogenen Architektur, die nicht so einfach beschrieben werden kann. [RWW04]

Anwendung a	Befehle		Speicherzugriffe	
	pro Paket (α_a)	pro Byte (β_a)	pro Paket(γ_a)	pro Byte(δ_a)
IPv4-radix	4493	0	868	0
IPv4-trie	205	0	50	0
Flow-Klassifizierung	153	0	79	0
IPsec-Verschlüsselung	-2363	294	-868	104

Tabelle 3.1: Paketverarbeitungsaufwand unterschiedlicher Anwendungen

Parameter für das Modell In [RWW04] wurden vier Anwendungsfälle untersucht, für die Parameter zur Verfügung stehen. Parameter für andere Anwendungen müssen bei Bedarf erst bestimmt werden.

- IPv4-radix: Paket-Weiterleitung nach RFC 1812 [Bak95] mit einer Patricia-Trie-Struktur (engl. radix tree) der Routing-Tabelle.
- IPv4-trie: Ähnlich wie IPv4-radix, benutzt aber eine Trie-Struktur mit kombinierter Ebenen- und Pfad-Kompression. [NK99]
- Flow-Klassifizierung: Pakete werden in Flows klassifiziert, die durch 5-Tupel (Quelladresse, Zieladresse, Quellport, Zielport, Transportprotokoll) definiert werden. Flow-Klassifizierung ist oftmals Teil von Anwendungen wie Firewall und NAT.
- IPsec-Verschlüsselung: Verschlüsselung der Paket-Payload nach dem IP Security Protokoll [KA98] mit 3DES.

Diese Parameter wurden für die untersuchten Anwendungen jeweils mit Hilfe des Tools PacketBench [RW03, RWW04] ermittelt und können der Tabelle 3.1 entnommen werden.[1]

3.2 Stochastisches Modell für die Verzögerung im WLAN

Wie in Abschnitt 2.2.2 gezeigt wurde, kann die Paketverzögerung im WLAN stark schwanken und ist damit geeignet verzögerungsbasierte Staukontrollmechanismen zu stören. Im Folgenden wird ein Modell für die Verzögerung im WLAN entwickelt, um dies untersuchen zu können. Es gibt eine große Anzahl an Forschungsbeiträgen zur Verzögerung durch WLANs. In den meisten Fällen wird dabei jedoch nur der Durchsatz oder die durchschnittliche Paketverzögerung betrachtet. Einige wenige befassen sich auch mit der Streuung der Paketverzögerung. Die meisten dieser Modelle beruhen auf dem Markov-Ketten-Modell von Bianchi [Bia00].

Für die Netzwerkemulation auf Paketebene ist das Verhalten des einzelnen Pakets wichtig. Hier bietet [RVP09] – eine der Arbeiten, die auf [Bia00] aufbauen – einen guten Ansatzpunkt

[1] Die Paketlänge enthält den Header. Dies bewirkt, dass die Pro-Paket-Werte negativ sein können. Da jedes Paket eine Mindestlänge hat, ergibt sich trotzdem ein positives Ergebnis für die Zahl der Befehle und Speicherzugriffe.

Bei der Herleitung der durchschnittlichen Paketverzögerung sowie deren Standardabweichung werden dort mathematische Formeln entwickelt, die sich auf einzelne Pakete anwenden oder dahingehend verändern lassen. Dieses Modell habe ich angepasst und erweitert und stelle es hier vor.

Im Folgenden werde sehr viele Parameter und Variablen verwendet. Zur Referenz habe ich in Tabelle 3.2 eine Übersicht der Wichtigsten erstellt.

3.2.1 Herleitung und Entwicklung des Modells

Annahmen und Vereinfachungen

In [RVP09] stellen P.Raptis, V. Vitsas und K.Paparrizos ein mathematisches Modell für die Datenübertragung im WLAN mittels DCF vor. Um die Komplexität des Modells und damit auch die erforderliche Rechenleistung zu reduzieren, haben sie folgende Annahmen und Vereinfachungen getroffen:

- **Keine versteckten Stationen.** Es wird davon ausgegangen, dass sich alle Stationen gegenseitig „hören“ können. Wenn eine Station auf dem Kanal horcht, ist also sichergestellt, dass sie tatsächlich bemerkt, wenn eine der anderen Stationen sendet.
- **Keine *Capture*-Effekte.** Der Effekt, dass eine Station den Kanal durchgehend belegt und die anderen Stationen nicht zum Zuge kommen, wird vernachlässigt. Dies kann vorkommen wenn aufgrund der höheren Sendeleistung einer Station, deren Pakete bei einer Kollision nicht verworfen werden, also gegenüber den Paketen anderer Stationen bevorzugt werden. [LZ05]
- **Fehlerfreier Kanal.** Pakete gehen nicht aufgrund einer schlechten Verbindungsqualität verloren sondern nur aufgrund von Kollisionen.
- **Fixe Anzahl an Stationen.** Es gibt eine bekannte, feste Anzahl teilnehmender Stationen n, die sich nicht ändert.
- **Sättigungszustand.** Jede Station hat ständig ein Paket zur Verfügung, das sie senden will.
- **Fixe Paketlänge.** Alle Pakete haben die gleiche Länge.
- **Alle Pakete kollidieren mit der gleichen Wahrscheinlichkeit.** Bei jedem Übertragungsversuch und unabhängig davon wie viele Neuübertragungsversuche unternommen werden müssen, kollidiert jedes Paket mit der konstanten und unabhängigen Wahrscheinlichkeit p. [Bia00]
- **Ursache der Verzögerungsschwankung.** Die Schwankung bei der Verzögerung von Paketen entsteht hauptsächlich durch die Wahl unterschiedlicher Backoff-Werte in den unterschiedlichen Stufen.

p	Die Wahrscheinlichkeit, dass ein übertragenes Paket kollidiert
τ	Die Wahrscheinlichkeit, dass eine Station in einer zufällig gewählten *slot time* (siehe σ) ein Paket übertragt
$E[slot]$	Die durchschnittliche Zeit, die eine Station in einem *slot* zögert oder anders ausgedrückt die Zeit, die es durchschnittlich dauert, bis der Backoff-Zähler um 1 herabgesetzt wird
P_{tr}	Die Wahrscheinlichkeit, dass von $n-1$ Stationen wenigstens eine in der betrachteten *slot time* überträgt
P_s	Die Wahrscheinlichkeit, dass eine Übertragung auf dem Kanal erfolgreich ist
Q_j	Die Wahrscheinlichkeit, dass ein erfolgreich übertragenes Paket in der Stufe j übertragen wird
T_s	Die Zeit, die der Kanal bei einer erfolgreichen Übertragung von den anderen Stationen als belegt erkannt wird
T_c	Die Zeit, die der Kanal bei einer erfolglosen Übertragung von den anderen Stationen als belegt erkannt wird
$SIFS$	Die Zeit, die vor dem Senden eines ACK-, RTS- oder CTS- Pakets gewartet werden muss (Short Interframe Spacing)
$DIFS$	Die Zeit, die vor dem Senden eines Pakets gewartet werden muss. (Distributed Coordination Function Interframe Spacing)
σ	Die Zeit, die gehorcht werden muss, um zu entscheiden, ob der Kanal belegt ist oder nicht (*slot time*)
δ	Verzögerung, die durch die Ausbreitung des Signals im Medium (Luft) entsteht (*propagation delay*)
j	Die Backoff-Stufe, also die Zahl der bisher unternommenen Übertragungsversuche
T_{MAC}	Die Zeit, die benötigt wird, um den MAC header zu übertragen
T_{PHY}	Die Zeit, die benötigt wird, um den Physical header zu übertragen
T_{ACK}	Die Zeit, die benötigt wird, um ein ACK zu übertragen
T_{RTS}	Die Zeit, die benötigt wird, um ein RTS zu übertragen
T_{CTS}	Die Zeit, die benötigt wird, um ein CTS zu übertragen
W_i	Größe des Contention Windows in der Stufe i. Die minimale Größe des Contention Windows ist $W_{min} = W_0$
m	Anzahl der unterschiedlichen Contention Window-Größen
R	Retry- oder Retransmission-Limit
n	Anzahl der am WLAN beteiligten Stationen

Tabelle 3.2: Verwendete Variablen für das WLAN-Modell

Durchschnittliche Verzögerung und Standardabweichung

In [RVP09] wird für dieses Modell die durchschnittliche Paketverzögerung $E[D]$ wie folgt hergeleitet. Für jede mögliche Backoff-Stufe werden die Kanalbelegungszeit bei erfolgreicher Übertragung, die Kanalbelegungszeiten der vorhergehenden Backoff-Stufen mit Kollisionen und die Zeit, die durchschnittlich gewartet werden muss, bis der Backoffzähler in der jeweiligen Stufe null erreicht, addiert. (Für den gewürfelten Backoff-Wert wird dabei einfach die halbe Größe des Contention Windows verwendet.) Das Ergebnis wird dann nach der Wahrscheinlichkeit für jede Backoff-Stufe gewichtet.

$$E[D] = \sum_{j=0}^{R} \left(\left(T_s + jT_c + E[slot] \sum_{i=0}^{j} \frac{W_i - 1}{2} \right) \frac{p^j(1-p)}{1-p^{R+1}} \right) \tag{3.6}$$

Daneben findet sich in [RVP09] noch eine umgeformte Gleichung ohne Summen-Notation.[2]

$$\begin{aligned} E[D] =& T_s + T_c \frac{A_R}{B_R} + \frac{1}{2B_R} E[slot]((W2^{m+1} - W - m - 1)(B_R - B_m) - (A_m + B_m) \\ &+ W(2H - B_m) + (W2^m - 1)(A_R - A_m - mB_R + mB_m)) \end{aligned} \tag{3.7}$$

mit den Termen zur A_i, B_i und H zur besseren Lesbarkeit.

$$A_i = \frac{p(1 - p^i - i \cdot p^i(1-p)}{(p-1)^2}, B_i = \frac{1-p^{i+1}}{1-p}, H = \frac{1-(2p)^{m+1}}{1-2p} \tag{3.8}$$

Für die Standardabweichung muss zunächst $E[D^2]$ berechnet werden. $E[D^2]$ ergibt sich mit $E[U_{j-1}]$ für die Verzögerung durch die erfolglosen Stufen mit Kollisionen als

$$E[D^2] = \sum_{j=0}^{R} \frac{(1-p)p^j}{1-p^{R+1}} \frac{1}{W_j} \sum_{i=0}^{W_j - 1} (T_s + i \cdot E[slot] + E[U_{j-1}])^2 \tag{3.9}$$

Aus $E[D]$ und $E[D^2]$ kann dann die Standardabweichung ($Jitter$) berechnet werden.

$$Jitter = \sqrt{E[D^2] - (E[D])^2} \tag{3.10}$$

Für das zu erstellende Modell sind diese Gleichungen jedoch nicht bzw. nur zur Überprüfung nutzbar. Tatsächlich werden beide Gleichungen (3.6) und (3.7) für die durchschnittliche Paketverzögerung und die daraus resultierenden Standardabweichungen in Kapitel 5 hierzu

[2] In [RVP09] haben sich bei der Gleichung (3.7) – nach dortiger Zählung Gleichung (19) – zwei Vorzeichenfehler eingeschlichen, die zu negativen Werten für die Verzögerung führen. Danke an P. Raptis für die Unterstützung bei der Fehlersuche

verwendet. Das verwendete Framework IKR EmuLib (siehe Kapitel 4) – und damit zwangsläufig auch das zu entwickelnde Modell – bearbeitet aber einzelne Pakete. Es muss die Verzögerung für ein bestimmtes Paket berechnet werden. Ein Durchschnittswert und die Standardabweichung reichen nicht aus, um die Verteilung von Zufallswerten hinreichend genau zu beschreiben. Für das Modell müsste jedoch die Verteilung der Verzögerungen bekannt sein. Benötigt wird also ein stochastisches Modell für die Verzögerung der einzelnen Pakete.

Entwicklung des stochastischen Modells

Die Berechnung meines Modells für einzelne Pakete erfolgt in drei Hauptschritten. In jedem dieser drei Schritte bestimmt ein Zufallsfaktor über die Behandlung des Pakets.

1. Bestimmung, ob ein Paket letztendlich erfolgreich übertragen wird.
2. Bestimmung der Backoff-Stufe, in welcher die erfolgreiche Übertragung stattfindet.
3. Bestimmung der Verzögerung, die ein übertragenes Paket erfährt.

Bestimmung, ob ein Paket letztendlich erfolgreich übertragen wird Im ersten Schritt wird berechnet, ob das Paket letztendlich erfolgreich übertragen wird. Alle anderen Pakete werden verworfen. Trotzdem muss für diese in Schritt 3 die Verzögerung berechnet werden, da dies die Wartezeit ist, bevor der Sendeversuch für das nächste Paket gestartet werden darf. Im Falle einer TCP-Verbindung über WLAN ist es die Aufgabe der Transportschicht, für eine erneute Übertragung der verworfenen Pakete zu sorgen.

Wenn τ die Wahrscheinlichkeit ist, dass eine Station in einer zufällig gewählten slot time überträgt, dann berechnet sich die Wahrscheinlichkeit P_{none}, dass keine von n Stationen überträgt, als

$$P_{none} = (1-\tau)^n \tag{3.11}$$

Entsprechend wird in [RVP09] wird die Wahrscheinlichkeit P_{tr}, dass mindestens eine von $n-1$ Stationen überträgt, folgendermaßen angegeben.

$$P_{tr} = 1-(1-\tau)^{n-1} \tag{3.12}$$

Dies entspricht der Wahrscheinlichkeit, dass ein Paket, welches die $n$te Station in der betreffenden slot time sendet, eine Kollision erfährt.

Weiter wird die Wahrscheinlichkeit P_s, dass eine Übertragung auf dem Kanal erfolgreich ist, berechnet – mit der Bedingung, dass eine Übertragung auf dem Kanal stattfindet.

$$P_s = \frac{(n-1)\tau(1-\tau)^{n-2}}{P_{tr}} = \frac{(n-1)\tau(1-\tau)^{n-2}}{1-(1-\tau)^{n-1}} \tag{3.13}$$

Für das zu entwickelnde Modell muss ein konkretes Paket betrachtet werden. Es wird auf jeden Fall versucht, dieses Paket zu übertragen. Dies ist nur eine Frage des Zeitpunkts. (Wie lange dies dauert, wird in den folgenden Schritten berechnet.) Also ist diese Wahrscheinlichkeit gleich eins. In der gewählten slot time darf keine andere Station übertragen, sonst kommt es zur Kollision. Diese Wahrscheinlichkeit ist mit $1 - P_{tr}$ gegeben. Die Wahrscheinlichkeit P_{drop}, dass die Übertragung nicht innerhalb des Retry-Limits R erfolgreich ist, liegt damit bei

$$P_{drop} = P_{tr}^{R+1} \qquad \text{mit } R \geq 0 \tag{3.14}$$

Daraus folgt unmittelbar die Wahrscheinlichkeit $P_{success}$, dass die Übertragung letztendlich erfolgreich ist.

$$P_{success} = 1 - P_{drop} = 1 - P_{tr}^{R+1} = 1 - (1 - (1 - \tau)^{n-1})^{R+1} \qquad \text{mit } R \geq 0 \tag{3.15}$$

Im Netzwerkmodell wird mit Hilfe von $P_{success}$ für jedes Paket zufällig entschieden, ob es erfolgreich übertragen oder doch verworfen wird, weil das Retry-Limit erreicht wurde. Für erfolgreich übertragene Pakete wird im nächsten Schritt berechnet, wie viele Versuche dazu benötigt werden. Für Pakete, die verworfen werden, folgt direkt Schritt 3 mit der Berechnung der Verzögerung.

Bestimmung der Backoff-Stufe, in welcher die erfolgreiche Übertragung stattfindet Für erfolgreich übertragene Pakete muss nun bestimmt werden, in welcher Stufe die erfolgreiche Übertragung erfolgt, um im Anschluss die Verzögerung berechnen zu können.

Dazu wird für jede Stufe j eine bestimmte Wahrscheinlichkeit Q_j berechnet, dass das Paket in genau dieser Stufe übertragen wird. Da schon fest steht, dass das Paket erfolgreich übertragen wird, beträgt die Summe dieser Wahrscheinlichkeiten 1.

$$Q_j = \frac{(1-p)p^j}{1-p^{R+1}} \qquad \text{für } 0 \leq j \leq R \tag{3.16}$$

Dies entspricht dem rechten Term aus Gleichung (3.6) für die durchschnittliche Paketverzögerung. Dieser Term hat dort für die Gewichtung der Verzögerungen nach Wahrscheinlichkeit gesorgt. Im Netzwerkmodell wird die gleiche Formel verwendet, um für jedes Paket einzeln auszuwürfeln, in welcher Stufe es übertragen wird.

Bestimmung der Verzögerung, die ein übertragenes Paket erfährt Nachdem nun fest steht, ob und in welcher Backoff-Stufe das Paket übertragen wird, kann berechnet werden, wie groß die Verzögerung dieses Pakets ist. Diese Verzögerung setzt sich aus der Verzögerung in den erfolglosen Stufen und – sofern die Übertragung erfolgreich war – der Verzögerung in der erfolgreichen Stufe zusammen.

Die Verzögerung $E[S_j]$ in der erfolgreichen Stufe j ergibt sich aus der Zeit T_s, die der Kanal bei der Übertragung belegt ist, und dem Produkt aus dem zufällig bestimmten Backoff-Wert i und der Zeit $E[slot]$, die es dauert den Backoff-Zähler um eins herab zu zählen.

$$E[S_j] = T_s + i \cdot E[slot] \qquad \text{für } 0 \leq j \leq R,\ 0 \leq i \leq W_j - 1 \tag{3.17}$$

Die Gesamtverzögerung durch Kollisionen $E[U_j]$ bis zur erfolglosen Stufe j setzt sich nach [RVP09] aus folgenden Komponenten zusammen. Erstens der Zeit T_c, die der Kanal bei einer Kollision belegt ist, multipliziert mit der Zahl der erfolglosen Stufen j. Zweitens den aufsummierten durchschnittlichen Backoff-Werten für jede erfolglose Stufe multipliziert mit der durchschnittlichen Zeit $E[slot]$, die es braucht, den Backoff-Zähler um eins herab zu setzen. Der durchschnittliche Backoff-Wert wird dabei als arithmetisches Mittel von null und der jeweiligen Größe des Contention Windows W_z berechnet.[3]

$$E[U_j] = (j+1) \cdot T_c + E[slot] \cdot \sum_{z=0}^{j} \left(\frac{W_z - 1}{2} \right) \qquad \text{für } 0 \leq j \leq R \tag{3.18}$$

An diesem Punkt habe ich das Modell gegenüber [RVP09] dahingehend verfeinert, dass für den Backoff-Wert der erfolglosen Stufen nicht nur der Durchschnittswert verwendet wird. Stattdessen wird für jede einzelne Stufe der Backoff-Wert i zufällig bestimmt. Auf diese Weise sollten die für die Untersuchung von verzögerungsbasierten Staukontrollmechanismen interessanten Ausreißer noch besser abgebildet werden. Somit ergibt sich mit dem für Stufe z zufällig gewählten Backoff-Wert i_z die veränderte Gesamtverzögerung $E_{neu}[U_j]$ durch Kollisionen bis zur erfolglosen Stufe j als

$$E_{neu}[U_j] = \sum_{z=0}^{j} (T_c + i_z \cdot E[slot]) \qquad \text{für } 0 \leq j \leq R \text{ und } i_z \text{ zufällig aus } [0, W_z - 1] \tag{3.19}$$

Diese Gleichung findet zweifach Verwendung. Einerseits muss im Falle einer insgesamt gescheiterten Übertragung $E[U_R]$ gewartet werden, bis das Retry-Limit R erreicht ist und das nächste Paket bearbeitet werden kann. Andererseits muss auch für erfolgreich übertragene Pakete berechnet werden, wie lange die Verzögerung $E[U_{j-1}]$ in den vorangegangenen erfolglosen Stufen war.

Die Gesamtverzögerung in der erfolgreichen Stufe j ergibt sich aus der Verzögerung $E[S_j]$ in der erfolgreichen Stufe j und der Verzögerung $E[U_{j-1}]$ in den vorangegangenen, erfolglosen Stufen. Wenn das Paket auf Anhieb in Stufe $j = 0$ übertragen wird, dann beträgt die Verzögerung der vorangegangenen Stufen offensichtlich null.

[3] In [RVP09] wird i statt z als Zählvariable verwendet. Da i jedoch im Folgenden als Variable für den ausgewürfelten Backoff-Wert verwendet wird, habe ich mich hier für z als Zählvariable entschieden, um die Verwechslungsgefahr zu beseitigen.

$$E[D_j] = E[S_j] + E[U_{j-1}] \qquad \text{für } 0 \leq j \leq R \qquad (3.20)$$
$$\text{mit } E[U_{-1}] = 0$$

Damit sind der logische Berechnungsablauf und die verwendeten Gleichungen im Netzwerkmodell klar. Um den Rechenaufwand im laufenden Modell gering zu halten, habe ich die Berechnung der Verzögerungen so implementiert, dass diese für alle möglichen Backoff-Werte vorab berechnet werden. Zur Laufzeit müssen so nur noch die Backoff-Werte ausgewürfelt und die dazugehörige Verzögerung nachgeschlagen werden.

Für diese Berechnungen wurden bisher einige Größen, einfach als gegeben angenommen. In den folgenden Abschnitten werden diese nun hergeleitet.

Bestimmung der durchschnittlichen Zählschrittdauer des Backoff-Zählers

Prinzipiell ist jede slot time gleich lang, aber der Backoff-Zähler wird nur herab gesetzt wenn der Kanal frei ist. Diese zusätzliche Verzögerung ist in $E[slot]$, der Zeit, die es durchschnittlich dauert, den Backoff-Zähler um eins herab zu setzen, eingerechnet. $E[slot]$ setzt sich wie folgt zusammen: die Wahrscheinlichkeit, dass keine andere Station überträgt $(1 - P_{tr})$ multipliziert mit der *slot time* σ plus die Wahrscheinlichkeit, dass jemand erfolgreich (P_s) bzw. erfolglos $(1 - P_s)$ überträgt, jeweils multipliziert mit der Zeit, die der Kanal dann als belegt erkannt wird (T_s bzw. T_c).

$$E[slot] = (1 - P_{tr} \cdot \sigma + P_{tr} \cdot P_s \cdot T_s \cdot + P_{tr} \cdot (1 - P_s) \cdot T_c \qquad (3.21)$$

P_{tr} und P_s sind schon aus den Gleichungen (3.12) und (3.13) bekannt. Die Herleitung der Kanalbelegungszeiten T_s und T_c folgt im nächsten Abschnitt.

Bestimmung der Kanalbelegungszeiten bei erfolgreicher und erfolgloser Übertragung

T_s und T_c bezeichnen die Zeit, die der Kanal bei erfolgreichem bzw. erfolglosem Senden als belegt erkannt wird. Für die Berechnung von T_s und T_c muss zwischen Basic Access und RTS/CTS Access (siehe Kapitel 2.2.2) unterschieden werden.

Basic Access Bei Basic Access sind T_s und T_c identisch. Die Belegzeit setzt sich aus dem Overhead O^{bas} für die Übertragung und der Übertragungsdauer des Pakets selbst zusammen.

$$T_s^{bas} = T_c^b as = O^{bas} + \frac{l}{C} \tag{3.22}$$

Dabei ist l die Paketlänge und C die Übertragungsrate des Kanals. Der Overhead O^{bas} ergibt sich aus der Beschreibung des Medienzugriffs mit DCF in Abschnitt 2.2.2. Es müssen einfach nur die verbrauchten Warte- und Übertragungszeiten addiert werden.

$$O^{bas} = DIFS + T_{MAC} + T_{PHY} + \delta + SIFS + T_{ACK} \tag{3.23}$$

RTS/CTS Access Bei RTS/CTS Access wird zuerst mit den kleinen RTS/CTS-Paketen versucht, den Kanal zu reservieren. Deswegen ändern sich T_s und T_c entsprechend. Die Übertragungsdauer des Datenpakets bleibt im Erfolgsfall gleich wie bei Basic Access. Nur der Overhead ändert sich.

$$T_s^{RTS} = O^{RTS} + \frac{l}{C} \tag{3.24}$$

Analog zu Basic Access bezeichnen O^{RTS} den Overhead für die Übertragung, l die Paketlänge und C die Übertragungsrate des Kanals. O^{RTS} setzt sich wie folgt aus den Übertragungsdauern und Wartezeiten zusammen.

$$O^{RTS} = DIFS + T_{MAC} + T_{PHY} + T_{RTS} + 3SIFS + 4\delta + T_{CTS} + T_{ACK} \tag{3.25}$$

Eine Kollision betrifft nur die Steuerungspakete. Die verbrauchte Zeit ist also unabhängig vom eigentlichen Paket.

$$T_c^{RTS} = DIFS + T_{RTS} + SIFS + T_{CTS} \tag{3.26}$$

Bestimmung der Wahrscheinlichkeiten für Übertragungen und Kollisionen

Zwei entscheidende Eingangsgrößen für das Modell sind die beiden Wahrscheinlichkeiten p, dass ein Paket kollidiert und τ, dass eine Station in einer zufällig gewählten *slot time* ein Paket übertragt. In [WPL+02] wurde hierfür ein Gleichungssystem hergeleitet.[4]

$$p = 1 - (1 - \tau)^{n-1} \tag{3.27}$$

[4] Die Gleichung für p ist identisch mit der Gleichung (3.12) für P_{tr}, da beides der Wahrscheinlichkeit entspricht, dass mindestens eine von $n - 1$ (anderen) Stationen (gleichzeitig) überträgt.

$$\tau = \frac{2(1-2p)(1-p^{R+1})}{W(1-(2p)^{m+1})(1-p)+1-2p)((1-p^{R+1})+W2^m p^{m+1}(1-p^{R-m}))} \quad (3.28)$$
mit $R > m$

Dabei ist n die Zahl der am WLAN beteiligten Stationen. W steht für W_{min} und bezeichnet die die minimale Größe des Contention Windows. R bezeichnet das Retry-Limit und m die Backoff-Stufe bis zu welcher das Contention Window verdoppelt wird. In den darauf folgenden Backoff-Stufen bleibt die Größe des Contention Windows konstant.

Die beiden Gleichungen (3.27) und (3.28) bilden ein nicht-lineares Gleichungssystem mit eindeutiger Lösung, das eine eindeutige Lösung hat und numerisch gelöst werden kann. Im Rahmen dieser Arbeit wurde dazu MATLAB R2009b verwendet. [Mat] Für die Berechnung wurde das Gleichungssystem dazu wie in Listing 3.1 in die MATLAB-eigene Sprache umgesetzt und in einer M-Datei nle.m gespeichert.

```
function f=nle(x)

W=32;
R=6;
m=5;
n=5;

f=[(1-(1-x(1))^(n-1))-x(2); (2*(1-2*x(2))*(1-x(2)^(R+1)) /
    (W*(1-(2*x(2))^(m+1))*(1-x(2))+(1-2*x(2)) *
    ((1-x(2)^(R+1))+W*(2^m)*(x(2)^(m+1))*(1-x(2)^(R-m)))))-x(1)]
```

Listing 3.1: Matlab-Code zur Bestimmung von p und τ

Die Werte für W, R, m n sind dabei selbstverständlich nur Beispielwerte, die für verschiedene Szenarien verändert werden können. Am MATLAB-Prompt müssen nun Startwerte für die Bestimmung von p und τ eingegeben, sowie die Lösung des Gleichungssystems gestartet werden. Optional kann mit *format long* die Zahl der angezeigten Dezimalstellen erhöht werden.

```
>> format long;
>> x0 = [0.9 0.1];
>> x = fsolve ('nle', x0)
```

Als Ergebnis erhalten wir in unserem Beispiel

```
x = 0.047852257321780   0.178103546648486
```

also

$\tau = 0.047851368751331$
$p = 0.178099982535998$

Bestimmung der Parameter für das WLAN-Modell

Alle weiteren Parameter können dem WLAN-Standard [ieee07] entnommen bzw. selbst festgelegt werden. Zu beachten ist dabei, dass im vorgestellten Modell der Backoff-Wert im Bereich $[0, W_j - 1]$ liegt, wobei der Wert von W_j 2er-Potenzen von W_0 annimmt. Im WLAN-Standard sind die Minimal- und Maximalwert für das Contention-Window um eins niedriger gewählt, also 1023 statt 1024 und 31 statt 32. Für die Verwendung im Netzwerkmodell müssen die Werte aus dem Standard also um eins erhöht werden. Mit den angepassten Werten kann dann die Anzahl der unterschiedlichen Contention-Window-Größen m berechnet werden:

$$m = log_2 \left(\frac{W_{max}}{W_{min}} \right) \tag{3.29}$$

Ähnlich sieht es beim Retry-Limit aus. Hier muss der Wert gegenüber dem Standard um eins veringert werden, weil der Wertebereich mit null und nicht mit eins beginnt. Zudem geht das Modell von nur einem Retry-Limit aus. Der Standard sieht jedoch ein *short retry limit* für Basic Access und ein *long retry limit* für RTS/CTS Access vor.

Bei der Wahl der Anzahl beteiligter WLAN-Stationen muss beachtet werden, dass das Modell erst ab zwei beteiligten Stationen korrekt arbeitet. Bei logischer Betrachtung kann es jedoch ohnehin kein WLAN mit weniger als zwei Stationen geben, da zumindest eine Partnerstation vorhanden sein muss, um Daten zu senden.

In Tabelle 3.3 finden sich die angepassten Parameter für IEEE 802.11b wie sie in [RVP09] verwendet werden.

3.2.2 Erweiterung des Modells für unterschiedlich lange Pakete

Für eine Übertragung mit TCP ist die Annahme einer festen Paketlänge unrealistisch. Denn neben den Datenpakten müssen auch die sehr viel kürzeren SYN, FIN und vor allem ACK-Pakete übertragen werden. (Nicht zu verwechseln mit den ACK-Paketen in der MAC-Schicht, die in Abschnitt 2.2.2 betrachtet wurden.)

Deswegen habe ich versucht, das WLAN-Modell für Pakete mit variabler Länge aus [RVBP07] mit diesem Modell zusammen zu führen. Dabei hat sich das Problem gestellt, dass

T_{MAC}	224bits/11Mbit/s	Zeit, um den MAC header zu übertragen
T_{PHY}	192bits/1Mbit/s	Zeit, um den Physical header zu übertragen
T_{ACK}	112bits/11Mbit/s + PHY	Zeit, um ein ACK zu übertragen
T_{RTS}	160bits/1Mbit/s + PHY	Zeit, um ein RTS zu übertragen
T_{CTS}	112bits/1Mbit/s + PHY	Zeit, um ein CTS zu übertragen
δ	1µ	propagation delay
σ	20µ	slot time
$SIFS$	10µ	SIFS
$DIFS$	50µ	DIFS
W_{min}	32	Minimale Größe des Contention Windows (W_0)
m	5	Anzahl der Contention Window-Größen
R	6	Retry-Limit
n	≥ 2	Anzahl der am WLAN beteiligten Stationen

Tabelle 3.3: WLAN-Parameter

in [RVBP07] davon ausgegangen wird, dass die möglichen Paketlängen und deren Wahrscheinlichkeit bekannt sind. Aus diesem Grund habe ich folgende Vereinfachung getroffen. In meinem Modell wird nur zwischen langen Paketen und kurzen Paketen unterschieden. Die Abweichung durch diese Vereinfachung dürfte unwesentlich bleiben, da sowohl die Länge der Datenpakete als auch der Steuerungspakete in der Regel keinen großen Schwankungen unterliegen. Das bisherige Modell muss hauptsächlich in einem Punkt modifiziert werden, nämlich bei der Berechnung der Kanalbelegungszeiten T_s und T_c. Der größte Anpassungsaufwand liegt dabei darin, dass anders als in [RVBP07] möglichst wenig Durchschnittswerte, sondern zum aktuell verarbeiteten Paket passende Werte verwendet werden sollen. Dabei muss immer beachtet werden, ob sich ein bestimmter Wert auf die betrachtete Station oder auf die übrigen Stationen bezieht.

Für das erweiterte Modell werden einige Wahrscheinlichkeitswerte in Bezug auf die Länge der beteiligten Pakete benötigt. Die Wahrscheinlichkeit, dass ein Paket die Länge l hat ist mit $P_L(l)$ gegeben. Dabei geht es um die Pakete der emulierten anderen Stationen. Um zu entscheiden, ob die betrachtete Station gerade ein langes oder ein kurzes Paket versenden will, wird zur Laufzeit die jeweilige Paketlänge bestimmt. Dieser Wert muss letztlich experimentell bestimmt werden. Für eine Größenordnung kann man jedoch davon ausgehen, dass für jedes angekommene Datenpaket mindestens ein ACK zurück gesendet wird. Der Wert sollte also – wenn es nur um die Unterscheidung von Steuerungs- und Datenpaketen geht – nahe bei 0.5 liegen.

Die Wahrscheinlichkeit P_k, dass exakt k von n Stationen an einer Kollision teilnehmen, beträgt nach [RVBP07]

$$P_k = \binom{n}{k} \cdot \frac{\tau^k (1-\tau)^{n-k}}{q_{tr}(1-q_s)} \qquad \text{mit } k \geq 2 \tag{3.30}$$

Dabei sind q_s und q_{tr} analog zu den Gleichungen (3.12) und (3.13) die Wahrscheinlichkeiten, dass von n Stationen mindestens eine überträgt, sowie die Wahrscheinlichkeit, dass eine Übertragung, die auf dem Kanal stattfindet, erfolgreich ist. Mehr dazu im Absatz 3.2.2.

Um zu berechnen, wie lange der Kanal bei einer Kollision belegt ist, wird die Summe der Wahrscheinlichkeiten P_y, dass Pakete kürzer als oder gleich lang wie das längste Paket in einer Kollision sind, benötigt.

$$P_y = \sum_{h \in L | h < l} P_L(h) \tag{3.31}$$

Damit kann die Wahrscheinlichkeit $P_{l,k}$, dass ein Paket mit der Länge l bei einer Kollision mit k involvierten Stationen, das Längste ist, bestimmt werden.

$$P_{l,k} = \sum_{r=1}^{k} (-1)^{r+1} \binom{k}{r} (P_L(l))^r P_y^{k-r} \qquad \text{mit } 2 \leq k \leq n \tag{3.32}$$

Mit diesen Wahrscheinlichkeiten können die Kanalbelegungszeiten T_s und T_c bestimmt werden.[5] Benötigt werden jeweils drei Werte. Ein Wert für kurze Pakete, ein Wert für lange Pakete und ein Durchschnittswert. Der Durchschnittswert wird für die Bestimmung der Zeit $E[slot]$, die es dauert, den Backoff-Zähler herunter zu zählen, benötigt, weil es dabei nicht um das Paket der betrachteten wartenden Station geht, sondern um die Pakete der anderen Stationen. Deshalb wird dort – und nur dort – für die Variablen T_s und T_c der Durchschnittswert eingesetzt.

Wie beim bisherigen Modell müssen auch hier Basic Access und RTS/CTS Access getrennt behandelt werden.

Basic Access Für erfolgreiche Übertragungen setzt sich T_s aus der Summe des Paket-Overheads und der Payload zusammen.

Nach [RVBP07] ist der Durchschnittswert für die Kanalbelegung beim erfolgreichen Senden T_s analog zu Gleichung (3.22).

$$T_{s,avg} = O^{bas} + \frac{E[l]}{C} \tag{3.33}$$

Dabei ist $E[l]$ die nach Wahrscheinlichkeit gewichtete, durchschnittliche Länge eines Pakets. Der Overhead O^{bas} wird wie schon in Gleichung (3.23) für das Modell mit fester Paketlänge bestimmt.[6] Bei Betrachtung einer bestimmten Station und eines bestimmten Pakets ändert sich an der Gleichung nur, dass die Paketlänge l dieses Pakets verwendet wird.

$$T_s = O^{bas} + \frac{l}{C} \tag{3.34}$$

[5] Falls nötig wird bei der Bezeichnung zwischen langen und kurzen Paketen unterschieden, also l_{big} und l_{small}, $T_{s,big}$ und $T_{s,small}$, sowie $T_{cs,big}$ und $T_{c,small}$.

[6] in [RVBP07] geht in O^{bas} fälschlicherweise 2δ statt nur δ ein

Für den Fall einer Kollision ist die Länge des längsten in die Kollision verwickelten Pakets entscheidend, denn so lange dieses übertragen wird, wird der Kanal als belegt erkannt. Der Durchschnittswert kann hier wieder [RVBP07] entnommen werden.

$$T_c = O^{bas} + \frac{1}{C} \cdot \sum_{l \in L} \left(l \cdot \sum_{k=2}^{n} P_k P_{l,k} \right) \tag{3.35}$$

Bei nur zwei möglichen Paketlängen ergibt sich daraus.

$$T_c = O^{bas} + \frac{1}{C} \cdot \left(\left(l_{small} \cdot \sum_{k=2}^{n} P_k P_{l_{small},k} \right) + \left(l_{big} \cdot \sum_{k=2}^{n} P_k P_{l_{big},k} \right) \right) \tag{3.36}$$

Die Berechnung von $T_{c,big}$ ist trivial, weil in diesem Fall keine anderen Pakete länger sein können.

$$T_{c,big} = O^{bas} + \frac{l_{big}}{C} \tag{3.37}$$

Für den Fall eines kurzen Pakets muss die Wahrscheinlichkeit berechnet werden, dass dieses Paket das längste in die Kollision verwickelte Paket ist. Diese beträgt nach Gleichung (3.32)

$$P_{l_{small},k} = \sum_{r=1}^{k} (-1)^{r+1} \binom{k}{r} (P_L(l_{small}))^r P_{y,small}^{k-r} \tag{3.38}$$

$P_{y,small}$ entspricht der Wahrscheinlichkeit, dass nur kurze Pakete an der Kollision beteiligt sind.

$$P_{y,small} = P_L(l_{small})^{k-1} \tag{3.39}$$

RTS/CTS Access Für erfolgreiche Übertragungen sieht der Fall bei RTS/CTS Access ähnlich aus wie bei Basic Access, nur dass für den Overhead O^{RTS} nach Gleichung (3.25) verwendet werden muss. Für den Durchschnittswert muss die Länge l wieder durch den gewichteten Durchschnittswert $E[l]$ ersetzt werden.

$$T_s = O^{RTS} + \frac{l}{C}$$
$$O^{RTS} = DIFS + T_{MAC} + T_{PHY} + T_{RTS} + 3 \cdot SIFS + 4 \cdot \delta + T_{CTS} + T_{ACK}$$

Kollisionen können bei RTS/CTS Access nur beim Senden der RTS- und CTS-Pakete auftreten. Entsprechend gilt

$$T_c = T_c^{RTS} = DIFS + T_{RTS} + SIFS + T_{CTS} \tag{3.40}$$

Durchschnittliche Zählschrittdauer des Backoff-Zählers In [RVBP07] werden statt den Gleichungen 3.12 für die Wahrscheinlichkeit P_{tr}, dass eine von $n-1$ Stationen überträgt und 3.13 für die Wahrscheinlichkeit, dass eine Übertragung auf dem Kanal erfolgreich ist, die folgenden beiden Gleichungen verwendet. Dadurch ändert sich die durchschnittliche Zeit $E[slot]$, für das Herunterzählen des Backoff-Zählers.

$$q_{tr} = 1 - (1-\tau)^n \tag{3.41}$$

$$q_s = \frac{n\tau(1-\tau)^{n-1}}{q_{tr}} = \frac{n\tau(1-\tau)^{n-1}}{1-(1-\tau)^n} \tag{3.42}$$

$$E[slot] = (1 - q_{tr} \cdot \sigma + q_{tr} \cdot q_s \cdot T_s \cdot + q_{tr} \cdot (1-q_s) \cdot T_c \tag{3.43}$$

Diese Gleichungen unterscheiden sich von denen aus [RVP09] dadurch, dass n anstatt $n-1$ Stationen betrachtet werden. In [RVP+05] wird argumentiert, dass eine Station, welche gerade zögert, nicht gleichzeitig um den Kanal konkurriert. Dies wird bei meinem Modell, welches eine bestimmte Station betrachtet noch deutlicher. Wenn die betrachtete Station darauf wartet, dass der Backoff-Zähler null erreicht, ist es logisch ausgeschlossen, dass sie gleichzeitig ein Paket übertragt, damit den Kanal belegt und dies selbst als Anlass nimmt, den Backoff-Zähler nicht herunter zu setzen. Insofern ist die Benutzung der Gleichungen nach [RVP09] in diesem Zusammenhang logisch.

Für die Berechnung von $E[slot]$ bedeutet dies folgendes. $E[slot]$ hängt nicht von der Paketlänge der betrachteten Station ab, sondern von den Paketlängen der anderen Stationen. Deswegen wird hier zwar die Gleichung (3.21) nach [RVP09] verwendet, aber es werden wie schon erwähnt die Durchschnittswerte für die Kanalbelegung T_s und T_c verwendet, nicht die oben berechneten Werte für lange oder kurze Pakete.

$$E[slot] = (1 - P_{tr} \cdot \sigma + P_{tr} \cdot P_s \cdot T_s \cdot + P_{tr} \cdot (1-P_s) \cdot T_c \tag{3.44}$$

Auch die Gleichung (3.30) für die Anzahl der an einer Kollision beteiligten Stationen muss für diesen Fall so angepasst werden, dass nur $n-1$ Stationen betrachtet werden.

$$P_k = \binom{n-1}{k} \cdot \frac{\tau^k (1-\tau)^{n-1-k}}{P_{tr}(1-P_s)} \qquad \text{mit } k \geq 2 \tag{3.45}$$

Hybrid Access Basic Access und RTS/CTS Access müssen sich nicht gegenseitig ausschließen. In der Praxis wird oftmals für kurze Pakete Basic Access und für Pakete ab einer gewissen Länge RTS/CTS Access verwendet. Deswegen wird in [RVBP07] auch der sogenannte Hybrid Access betrachtet. Allerdings wurde dort nicht berücksichtigt, dass für Basic Access und RTS/CTS Access in der Regel unterschiedliche Retry-Limits gelten. Dies hat zur Folge, dass für beide Fälle unterschiedliche Werte für p und τ und die zahlreichen davon abgeleiteten Parameter gelten. Dies zu berücksichtigen würde die Komplexität des Modells erheblich erhöhen, vielleicht sogar ein nahezu komplett neues Modell notwendig machen, weswegen ich diesen Fall von vorn herein nicht berücksichtigt habe.

Fazit Das entwickelte erweiterte Modell bildet unterschiedliche Paketlängen zwar nicht hundertprozentig ab, sollte aber hinreichend genau, um die deutlich unterschiedlichen Paketlängen von Daten- und Steuerungspaketen zu berücksichtigen. Dabei ist der Realismus bei Basic Access besser gegeben, weil es wahrscheinlicher ist, dass nur Basic Access verwendet wird als, dass RTS/CTS Access auch für sehr kleine Pakete angewendet wird.

4 Aufbau des Testbeds

Wie bereits ausgeführt, sollen die in Kapitel 3 beschriebenen Modelle emuliert werden. In diesem Kapitel werden nun das verwendete Netzemulator-Framework und der Aufbau des Testbeds beschrieben.

4.1 Das Netzemulator-Framework IKR EmuLib

Für die Emulation wird das Framework IKR EmuLib benutzt. IKR EmuLib setzt auf der ereignisgesteuerten Simulationsumgebung IKR SimLib auf. Ein Simulationsmodell in IKR SimLib besteht aus einem Netzwerkmodell mit einem Generator (Quelle) und einem Sink (Abfluss) für den Netzwerkverkehr. [NGKR06]

Bei EmuLib wird dies dahingehend erweitert, dass das System als IP-Paket-Router arbeitet. Das heißt, dass man über ein Netzwerk-Interface echte Pakete in das Modell schicken kann, die dort verarbeitet werden und dann wieder über ein Netzwerk-Interface weitergeleitet werden. (Abbildung 4.1)

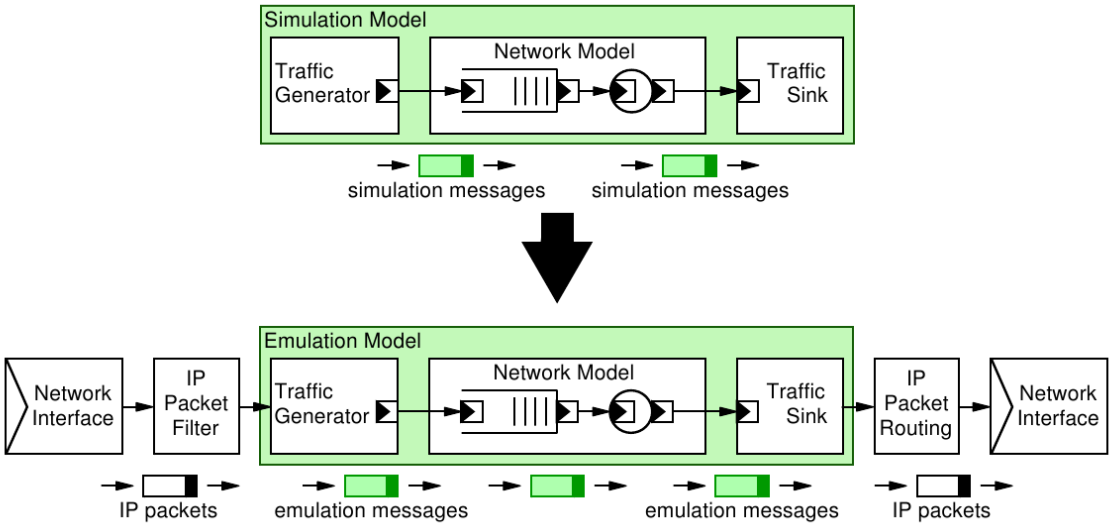

Abbildung 4.1: Von SimLib zu EmuLib [NG05]

Da bei der Emulation Elemente der realen Welt integriert werden, muss EmuLib in Echtzeit arbeiten. Für die Erstellung des Modells bedeutet dies, dass es so effizient sein muss, dass

es mindestens in Echtzeit abgearbeitet werden kann. Es darf also – in unserem Fall – nicht passieren, dass es länger dauert die Verzögerung zu berechnen, als die Verzögerung selbst beträgt.

4.2 Aufbau

Für das Testbed wurde ein einfacher Aufbau aus drei Rechnern gewählt. Je ein Rechner dienen als Sender und Empfänger. Alle Daten, die der Sender an den Empfänger schickt werden über den dritten Rechner geleitet, auf dem der Emulator mit dem Netzwerkmodell läuft.

Verwendet wurden 3 Rechner mit jeweils 4GB Arbeitsspeicher und 2x3GHz-Prozessoren. Auf allen Rechnern lief Ubuntu 10.04 (Maverick Meerkat) und das OpenJDK 6 Java Runtime Environment. Als Grundlage der Implementuerung dienten IKR SimLib in der Version 2.8.0 Beta2 und IKR EmuLib 1.0 Beta1, sowie eine gepatchte Version von Jpcap 0.7 [Fuj]. Jpcap ist eine von SimLib benötigte Java-Bibliothek für das Capturing und Senden von Netzwerkpaketen. Die Änderungen beheben folgende Probleme: Erstens werden Pings in der Standardversion nicht durchgeleitet. Zweitens verwirft Jpcap wie sich im Laufe der Arbeit herausstellte, die TCP-Optionen, insbesondere das Feld Maximum Segment Size (MSS), welches für die Aushandlung der Maximum Transmission Unit (MTU) notwendig ist. Ohne den Patch wurde in der Folge stets eine MSS von nur 576 (statt 1500) byte ausgehandelt. Das Netzwerk-Interface eth1 von Sender und Empfänger wurde jeweils mit dem Emulationsrechner verbunden. Beim Sender wurde eth1 die private IP-Adresse 192.168.1.1 zugewiesen, beim Empfänger 192.168.2.1. Der Emulations-Rechner dazwischen hat die beiden IP-Adressen 192.168.2.1 auf eth1 und 192.168.2.254 auf eth2. Alle Verbindungen von 192.168.1.1 ins Subnetz 192.168.2.0/24, sowie alle Verbindungen von 192.168.2.1 ins Subnetz 192.168.1.0/24 wurden über den Emulationsrechner geroutet.

Alle Pakete, die zwischen Sender- und Empfängerrechner ausgetauscht werden – und nur diese – durchlaufen das Netzwerkmodell. IKR EmuLib erlaubt es, verschiedene Modell-Komponenten miteinander zu verbinden. Vor das eigentliche Netzwerkprozessor- bzw. WLAN-Modell wurde jeweils ein begrenzter FIFO-Puffer eingefügt, der sicher stellt, dass Pakete, die nicht sofort verarbeitet werden können, nicht verworfen werden. Da echte Netzwerkgeräte einschließlich WLAN-Router ebenfalls einen solchen Puffer besitzen, entspricht dieses Verhalten der Realität. Bei der Größe des Puffers kann man sich entsprechend an real existierenden Geräten orientieren.

Abbildung 4.2 zeigt den Aufbau am Beispiel des WLAN-Modells. Der Aufbau mit Netzwerkprozessor sieht entsprechend aus. Die beiden Modelle können auch kombiniert und gleichzeitig genutzt werden. Ebenso wäre es ohne weiteres möglich, den Aufbau um weitere Sender und Empfänger zu erweitern, um z.B. das Zusammenspiel verschiedener Staukontrollmechanismen im Hinblick auf Fairness zu untersuchen.

Für die Validierung wurde jedoch jeweils nur ein Sender, ein Empfänger und eines der beiden Modelle plus FIFO-Puffer verwendet. So können die Effekte des jeweiligen Modells am besten beobachtet werden.

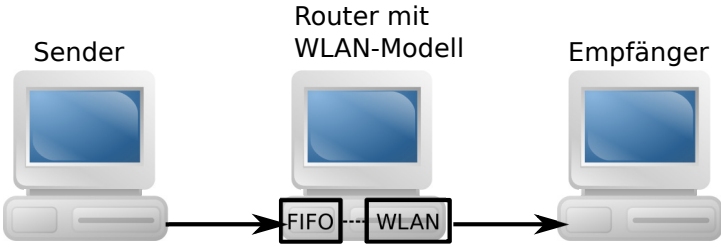

Abbildung 4.2: Aufbau des Testbeds mit WLAN-Modell

Für die Erzeugung von TCP-Verbindungen wurden auf den jeweiligen Rechnern selbst geschriebene Sender und Empfänger verwendet. Der Sender baut (über den Emulationsrechner) eine TCP-Verbindung zum Empfänger auf und versucht dann möglichst schnell und viel Daten an den Empfänger zu schicken. Dieser macht nichts weiter als die Daten anzunehmen und in Intervallen mitzuschreiben wie viel Daten empfangen wurden, und daraus die Übertragungsgeschwindigkeit für das Intervall zu errechnen. Auf dem Emulationsrechner stehen zudem die SimLib-eigenen Messmöglichkeiten zur Verfügung. Darunter bspw. der *DistTimeMeter*, welcher zur Messung von Verzögerungen genutzt werden kann und die dazugehörigen statistischen Werte ermittelt. Unglücklicherweise ist das Auswertungstool SimTree nur für SimLib, nicht aber für EmuLib nutzbar, so dass die Auswertung der Emulation aufwendiger ist als sie es bei der Simulation wäre.

4.3 Übertragung der Modelle auf das Testbed

Bevor die Modelle im Testbed verwendet werden können, müssen einige Dinge beachtet werden, die nicht das Modell an sich, sondern die Umsetzung und Verwendung in EmuLib betreffen.

4.3.1 Modell für die Verzögerung in Netzwerkgeräten

Die Formel (3.5) Verzögerungen durch die Verarbeitung in Netzwerkgeräten konnte ich für mein Testbed fast direkt implementieren. Zu beachten war, dass das Modell nur auf Datenpakete, nicht auf Steuerungspakete angewendet werden soll, weil nur auf diesen ggfs. komplexe Payload-Modifikationen durchgeführt werden. Gelöst habe ich dies über die Paketlänge. Es wird nur für Pakete mit einer festgelegten Mindestlänge eine Verzögerung berechnet. Für kürzere Pakete kann eine feste Verzögerung vorgegeben werden.

4.3.2 Modell für die Verzögerung im WLAN

Das entwickelte Modell für die Paketverzögerung im WLAN arbeitet auf der MAC-Schicht. Das zur Emulation eingesetzte IKR-EmuLib arbeitet jedoch als IP-Paketrouter, also auf der Internetschicht. Deswegen müssen bei der Umsetzung des Modells einige Dinge vorausgesetzt und bedacht werden.

- Ein IP-Paket muss einem Paket (oder korrekterweise Rahmen) auf MAC-Ebene entsprechen. Dies ist in der Realität normalerweise gegeben [KR05] und wird durch das Modell implizit vorausgesetzt.
- Als Paketlänge wird die Länge des Rahmens auf MAC-Ebene erwartet, der die TCP- und IP-Header umfasst. Dies muss bei der Wahl der Parameter und ggfs. bei Testläufen mit vorgegebener Paketlänge beachtet werden.
- Im Gegensatz zum Modell für die Verzögerung in Netzwerkgeräten sollen im WLAN-Modell TCP-Steuerungspakete (ACK, SEQ, FIN)[1] durch das Modell verzögert werden, weil sie auch in der Realität ganz normal über das WLAN geschickt werden. Da diese Pakete das Netzwerkmodell durchlaufen, ist dies gegeben.

4.3.3 Konfiguration

Die Konfiguration des Testbeds erfolgt an zwei Stellen. Einerseits können die unterschiedlichen Modell, sowie Messmodule von SimLib/EmuLib im Java-Quellcode zusammengebaut werden. In Listing 4.1 sind als Beispiel die Generatoren, das Netzwerkprozessor- und das WLAN-Modell mit ihren jeweiligen Puffer, und der Sink hintereinander geschalten. Die generatoren und der Sink stellen die Schnittstelle zu den Netzwerk-Interfaces dar. Am Sink ist dann noch ein Zähler angebracht, welcher die Zahl der übertragenen Nachrichten (Pakete) zählt und die Simulation bzw. in diesem Fall Emulation nach einer gewissen Anzahl beenden kann.

```
mux.connect("output", npQueue, "input");
npQueue.connect("output", np, "input");
np.connect("output", wlanQueue, "input");
wlanQueue.connect("output", wlan, "input");
wlan.connect("output", sink, "input");
controlCountMeter.attachInput(sink, "input");
```

Listing 4.1: Verbinden von Modellen in SimLib/EmuLib

Andererseits können die Parameter der verwendeten Modelle in einer Konfigurationsdatei wie sie in Listing 4.2 als Beispiel zu sehen ist konfiguriert werden. In diesem Fall sind sowohl das Netzwerkprozessormodell als auch das WLAN-Modell aktiviert.

[1] Das TCP-ACK darf nicht mit dem MAC-ACK verwechselt werden. Letzteres wird im Modell bei der Berechnung des Overheads O^{bas} bzw. O^{RTS} berücksichtigt.

```
{
  Batches = 4;
  BatchPacketCount = 25000;
  TransientPacketCount = 1000;

  WlanBufferSize = 65536;
  NPBufferSize = 65536;

  WlanPhase WP {
    ServiceTimeDist {
      WlanDelay {
        PacketLength = 8184;         // in bit
        SmallPacketLength=80*8;      // in bit
        PacketLengthThreshold = 576*8; // in bit
        SmallPacketProbability = 0.6; // Nicht definiert oder feste Paketlaenge
        ChannelBitrate = 11e6;
        T_MAC = 224/11e6;
        T_PHY = 192/1e6;
        T_ACK = 112/11e6 + 192/1e6;
        //T_RTS = 160/1e6 + 192/1e6; // Nicht definiert oder 0 bedeutet Basic Access
        //T_CTS = 112/1e6 + 192/1e6; // Nicht definiert oder 0 bedeutet Basic Access
        PropagationDelay = 1e-6;
        SlotTime = 20e-6;
        SIFS = 10e-6;
        DIFS = 50e-6;

        W_min=32;
        R=6;
        m=5;
        n=5;

        tau = 0.047851368751331;
        p = 0.178099982535998;
      }
    }
  }

  NetworkProcessor NP {
    ServiceTimeDist {
      NetworkProcessorDelay {
        ProcessorFrequency = 233e6;
        AverageMemoryAccessTime = 170e-9;
        PerPacketProcessingCost = 153;
        PerByteProcessingCost = 0;
        PerPacketMemoryAccesses = 79;
        PerByteMemoryAccesses = 0;
        DataPacketThreshold = 200; // in byte
        ControlPacketDelay = 0.000001;
      }
    }
  }

  Generator Generator_1 {
    Device = "eth1";
  }
```

```
  Generator Generator_2 {
    Device = "eth2";
  }

  Sink Sink {
  }
}
```

Listing 4.2: Konfigurationsdatei sim.par

5 Validierung

In den vorigen Kapiteln wurden Modelle für die Paketverzögerung entwickelt und der Aufbau des Testbeds beschrieben. Nun soll überprüft werden, ob die Emulation plausible Ergebnisse liefert. Dazu werden beide Modelle auf zwei Fragen hin überprüft.

1. Ergeben sich plausible Verzögerungswerte für die Pakete?
2. Arbeitet das Netzwerkmodell schnell genug?

Außerdem wird überprüft, wie sich die Netzwerkmodelle mit verschiedenen TCP-Varianten verhalten. Als Beispiel eines verlustbasierten Staukontrollmechanismus dient dabei TCP Reno. Stellvertretend für die verzögerungsbasierten Mechnanismen wird TCP Vegas betrachtet.

5.1 Überprüfung der Verzögerungswerte

5.1.1 Netzwerkprozessor-Modell

Bei den Netzwerkprozessoren stellt sich das Problem, dass Vergleichswerte fehlen. Trotzdem habe ich für die vier Anwendungen aus Abschnitt 3.1 Verzögerungswerte ermittelt. Als Prozessor wurde dabei der in [RWW04] erwähnte Intel IXP1200 mit einer Taktfrequenz von 233 MHz und einer durchschnittlichen Speicherzugriffszeit von 170ns angenommen. Zur Bestimmung der Verzögerungswerte habe ich Pings in unterschiedlicher Größe verwendet. Die gemessenen Durchschnittswerte finden sich in Tabelle 5.1. Die Standardabweichungen bewegen sich in der Größenordnung von $0,0$ bis 10^{-10} sind also vernachlässigbar. Man kann leicht erkennen, dass die Paketlänge nur bei IPsec eine Rolle spielt. Die anderen Anwendungen sind, wie schon aus Tabelle 3.1 erkennbar war, von der Paketlänge unabhängig. Der minimale Unterschied in der letzten angegebenen Nachkommastelle bei der Flow-Klassifizierung ist vernachlässigbar und kommt wohl durch die nie vollständig vermeidbare Varianz in der Rechen- und Übertragungszeit zustande.

Weiter ist zu erkennen, dass die Verzögerung bei IPsec leicht mehr als proportional mit der Paketlänge zunimmt.

Die Berechnung des Modells funktioniert anscheinend einwandfrei. Zur Güte des Modells kann leider mangels Referenzwerten wenig gesagt werden. Die Verzögerungen liegen aber in der allgemeinen Größenordnung von 10µ für einfache Paketweiterleitungen und 1000µ für komplexe Payload-Modifikationen wie sie in 2.2 angegeben sind.

Anwendung	Paketlänge in byte	Paketverzögerung in s
IPv4-radix	1500	$1,92832618 \cdot 10^{-5}$
	1000	$1,92832618 \cdot 10^{-5}$
	500	$1,92832618 \cdot 10^{-5}$
IPv4-trie	1500	$8,79828326 \cdot 10^{-7}$
	1000	$8,79828326 \cdot 10^{-7}$
	500	$8,79828326 \cdot 10^{-7}$
Flow-Klassifizierung	1500	$6,56652361 \cdot 10^{-7}$
	1000	$6,56652360 \cdot 10^{-7}$
	500	$6,56652361 \cdot 10^{-7}$
IPsec-Verschlüsselung	1500	$1,88510060 \cdot 10^{-3}$
	1000	$1,25167571 \cdot 10^{-3}$
	500	$6,20774421 \cdot 10^{-4}$

Tabelle 5.1: Paketverzögerung im Netzwerkprozessor

Die *emulation delay*, also die Verzögerung gegenüber der Echtzeit liegt in der Größenordnung von$10^{-4}s$, was eigentlich nicht besonders hoch ist, aber im Verhältnis zu den berechneten Paketverzögeruungen relativ viel ist. Sie summiert sich jedoch nicht auf, so dass der Rückstand gegenüber der Echtzeit insgesamt konstant bleibt und das Modell offenbar schnell genug berechnet wird.

5.1.2 WLAN-Modell

Für die beiden Varianten des WLAN-Modell wurden für verschiedene Parameter die berechneten Verzögerungen überprüft. Im ersten Schritt wurden dabei die Paketverzögerugen für das Modell mit fester Paketlänge auf unterschiedliche Weise bestimmt. Erstens die Berechnung des Modells: darunter sind die Werte, die das Netzwerkmodell für die Verzögerung berechnet, zu verstehen. Diese wurden bei der Berechnung im Modell mitgeschrieben und mit LibreOffice Calc ausgewertet. Zweitens die Messungen im Modell: dazu wurden die Verzögerungen mit SimLib-eigenen Mitteln gemessen. Ein sogenannter *DistTimeMeter* wurde genutzt, um die Zeit, die ein Paket vom Eingang bis zum Ausgang des Modells benötigt, sowie die dazugehörigen statistischen Werte, zu bestimmen. Außerdem wurden die zu erwartenden Werte mit den Gleichungen nach [RVP09] berechnet. Dabei gibt es wie schon erwähnt zwei Gleichungen, um die durchschnittliche Paketverzögerung zu berechnen, welche dann auch in die Standardabweichung einfließt. Hier werden beide Werte angegeben, um zu zeigen, in welchem Genauigkeitsbereich sich das Modell bewegt, und wie sehr schon die unterschiedlichen Berechnungsweisen das Ergebnis beeinflussen. Obwohl beide Berechnungen mathematisch gleichwertig sind, fließen Rundungsfehler unterschiedlich stark in das Ergebnis ein. Diesen Effekt muss man auch beachten wenn man die Parameter, insbesondere die numerisch bestimmten Parameter p und τ, rundet. Bei diesen ist die Auswirkung besonder groß, weil sie in den verwendeten Gleichungen potenziert werden.

Maßzahl	Modellberechnung	Modellmessung	Raptis Gl. 17	Raptis Gl. 19
5 Stationen				
Mittelwert	0,00712929	0,00713048	0,00699598	0,00713811
Standardabw.	0,00824698	0,00823040	0.00843358	0,00831363
Minimum	0,00121955	0,00121955	–	–
Maximum	0,36408406	0,36408406	–	–
10 Stationen				
Mittelwert	0,01484846	0,01485399	0,01472019	0,01497932
Standardabw.	0,02713351	0,02705680	0,02750688	0,02736663
Minimum	0,00121955	0,00121954	–	–
Maximum	0,83835756	0,83835756	–	–
50 Stationen				
Mittelwert	0,07918279	0,07831765	0,07902675	
Standardabw.	0,16924886	0,16531442	0,16497662	
Minimum	0,00121955		–	–
Maximum	1,84702384		–	–

Tabelle 5.2: Paketverzögerung in Sekunden im WLAN-Modell

In Tabelle 5.2 finden sich die Ergebnisse mit den Parametern nach Tabelle 3.3 bei Basic Access. Für die Messung wurden Pings mit der Größe 1023 bytes (inklusive aller Header) durch das Netzwerkmodell geschickt. Die Messung lief für 100000 Pakete also 50000 Pings und ihre Antwort.

Wie man sieht sind die Ergebnisse des Netzwerkmodells sehr nahe an denen nach [RVP09]. Teilweise sind die Ergebnisse sogar zwischen den Werten nach den beiden Gleichungen. Die Größenordnungen passen zudem jeweils zu den – nur ungenau ablesbaren – dortigen Schaubildern. Dies spricht für eine korrekte Umsetzung. Bei der durchschnittlichen Paketverzögerung lässt sich wie dort eine Proportionalität zur Anzahl der Stationen im WLAN beobachten. Auch für RTS/CTS Access oder bei der Änderung einzelner Parameter wie der Größe des Contention Windows ergeben sich keine wesentlichen Abweichungen.

Desweiteren sind die Unterschiede von Messung und Berechnung gering, was darauf schließen lässt, dass die Performance des Netzwerkmodells hinreichend gut ist. Dies bestätigen auch Messungen der *emulation delay*, also der Verzögerung des Modells gegenüber der Echtzeit. Im Idealfall wäre diese null. Negativ kann oder darf sie nicht werden. Wäre dies der Fall würde dies einen Fehler in EmuLib bedeuten. Für das WLAN-Modell bewegt sie sich im der Größenordnung von 10^{-4} bis 10^{-5} und damit in der Regel mindestens eine Zehnerpotenz niedriger als die berechnete Verzögerung. Sie tritt also gegenüber der Paketverzögerung zurück. Es war auch kein Ausummieren der *emulation delay* zu beobachten, was ebenfalls dafür spricht, dass die Berechnung des WLAN-Modells schnell genug erfolgt.

Neben den Maßzahlen aus Tabelle 5.2 ist auch die Verteilung der Paketverzögerung interessant. Zu erwarten ist hier, dass die meisten Pakete eine geringer Verzögerung erfahren,

es aber einige mit sehr hoher Verzögerung gibt. Dies sieht man in den Abbildungen 5.1 und 5.2. für 5 bzw. 50. Stationen. Die Verteilung ist in beiden Fällen ähnlich, jedoch ist der Wertebereich für die berechnete Paketverzögerung unterschiedlich. (Man beachte die Skala.) Auf der fast leer wirkenden Achse verteilen sich bis zu den ermittelten Maximalwerten einige wenige Pakete. Diese Verteilung entspricht auch ungefähr derjenigen, die in [ZKF04] gezeigt wird.

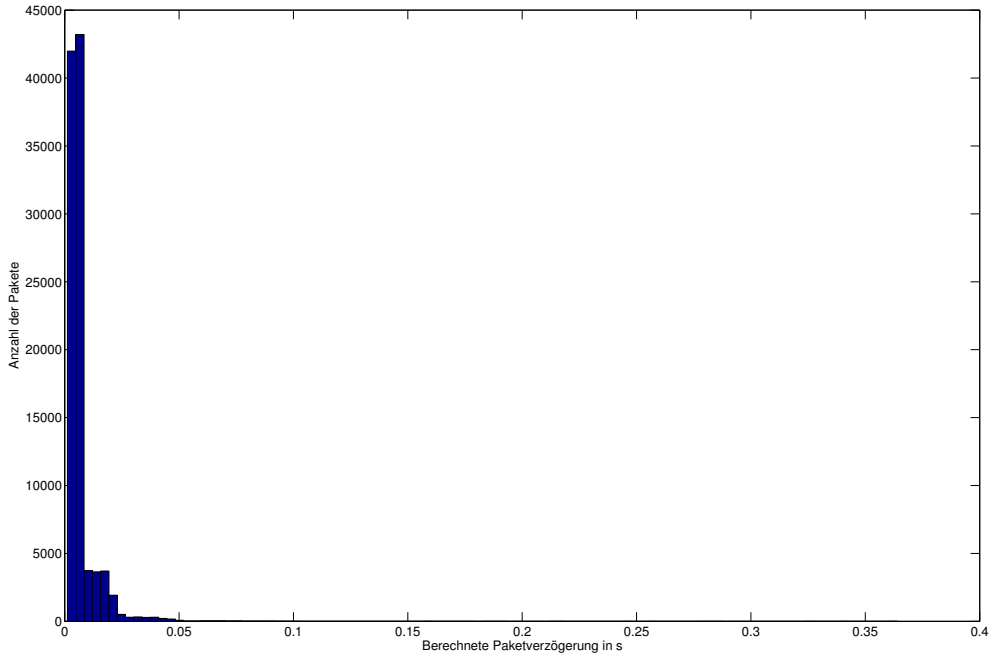

Abbildung 5.1: Verteilung der berechneten Paketverzögerung für 5 Stationen

Nachdem das Grundmodell offenbar korrekte Ergebnisse liefert, muss das erweiterte Modell für unterschiedliche Paketlängen überprüft werden. Dazu habe ich verglichen, ob es für den Fall, die obige Konstellation die gleichen Ergebnisse liefert. Hierfür habe ich die Wahrscheinlichkeit für kleine Pakete sehr niedrig (0,0000001) eingestellt. Korrekt wäre eigentlich null gewesen, aber dies habe ich als Erkennung dafür, dass das Modell mit fester Paketlänge verwenden soll implementiert. Auch hier ergaben sich fast die gleichen Werte wie in Tabelle 5.2. Für das ursprüngliche Modell wurde also offenbar kein Fehler eingeführt.

Ob die Werte für die tatsächliche Verwendung von kurzen und langen Paketen tatsächlich korrekt sind, lässt sich leider mangels Vergleichswerte schlecht beurteilen. Bei den Testläufen im nächsten Abschnitt mit TCP-Verbindungen wurden aber niedrigere Werte für die durchschnittliche Paketverzögerung gemessen, obwohl die langen Pakete 1500 byte anstatt 1023 byte hatten. Bei 5 Stationen und einem Anteil von 60% kurzen Paketen betrug sie nur 0,00627466s. Erstaunlicherweise ist auch die Standardabweichung geringer, nämlich nur 0,00695920s. Sowohl die maximale Verzögerung (0.30458026s) als auch die minimale Verzö-

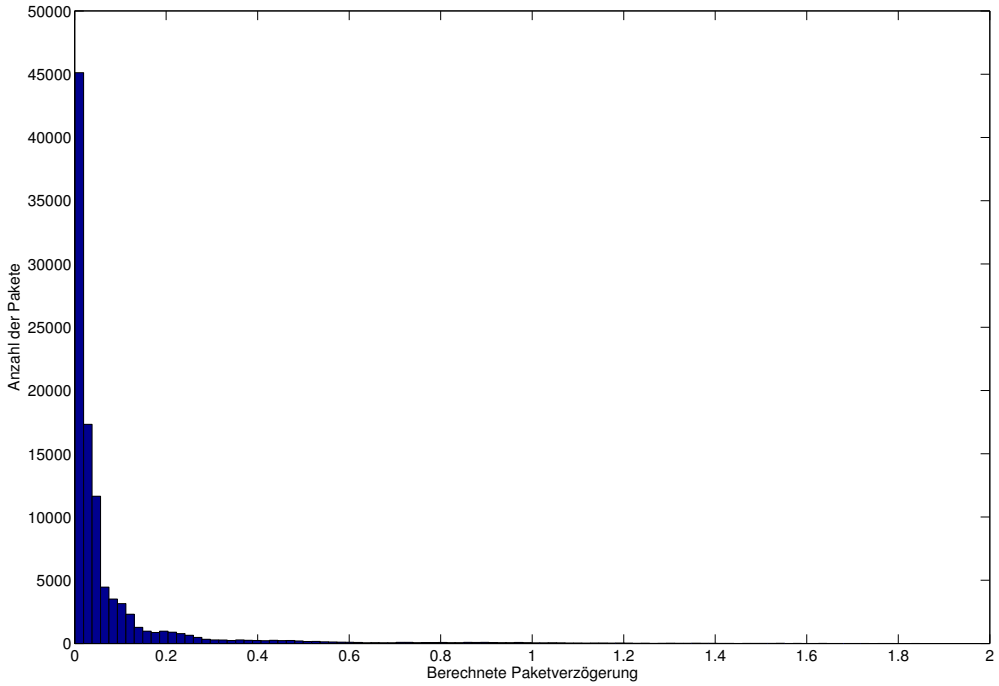

Abbildung 5.2: Verteilung der berechneten Paketverzögerung für 50 Stationen

gerung (0.000533733s) sind ebenfalls geringer als für feste Paketlängen gemessen. Insgesamt sie die Werte in jedem Fall plausibel, denn sie bestätigen die erwartete bessere Perfomance Ich hätte allenfalls eine höhere Standardabweichung erwartet.

5.2 Überprüfung des Verhaltens verschiedener TCP-Varianten

Nachdem die Modelle plausible Werte für die Paketverzögerung liefern, sollen nun tatsächlich TCP-Datenströme hindurch geleitet werden. Dabei geht es weniger um Aussagen über die entsprechenden TCP-Varianten als darum, ob das Verhalten plausibel ist. Gemessen wurde jeweils die ankommende Datenrate beim Empfänger.

5.2.1 Netzwerkprozessor-Modell

Wie schon festgestellt wurde, erzeugt das Netzwerkprozessor-Modell nur unterschiedliche Verzögerungen wenn die Paketlänge schwankt. Mangels Möglichkeit einen TCP-Datenstrom mit willkürlich festgelegten Paketlängen zu erzeugen, konnte ich für das Netzprozessor-Modell kein Szenario testen, das besondere Ansprüche an die Staukontrolle stellt. Da das Modell für den Netzwerkprozessor keine Zufallskomponente enthält, sind hier keine

ungewöhnlichen Ergebnisse zu erwarten. Trotzdem sind die Ergebnisse interessant, nicht zuletzt als Referenz für die Überprüfung des WLAN-Modells, da man sieht, wie sich TCP im Normalfall bei einem einzelnen Sender auf einer Leitung und konstanter Verzögerung verhält.

In den Abbildungen 5.3 und 5.4 finden sich die Ergebnisse für einen Fall eine Fall ohne Einfluss der Paketlänge (FLow-Klassifizierung). Im Endeffekt läuft dies auf eine feste Verzögerung pro Paket hinaus. Das hat den Vorteil, dass man sehr gut das normale Verhalten von TCP Reno und TCP Vegas sieht. Die Charakteristik des Durchsatzverlaufs ist zwar unterschiedlich, aber man sieht in beiden Fällen Sägezähne, die eigentlich für verlustbasierte Staukontrolle typisch ist. Ein kleiner Unterschied besteht aber doch. Bei TCP Vegas sind die Sägezähne nicht so gerade wie bei TCP Reno und flachen immer mehr ab je näher die Datenrate der maximalen Rate kommt. Trotzdem konnte offenbar der Paketverlust nicht vermieden werden, so dass die Datenrate – wie bei Reno – deutlich herunter gesetzt werden musste.

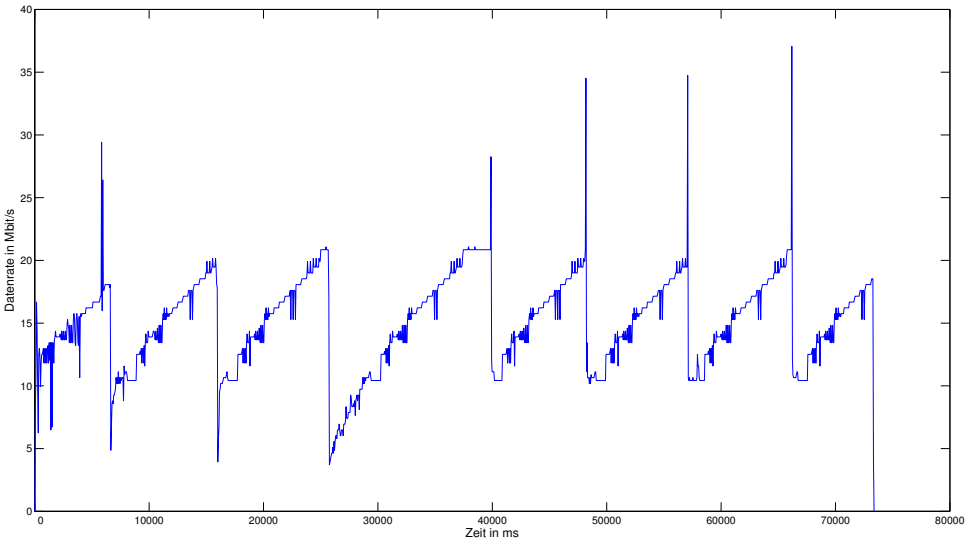

Abbildung 5.3: Durchsatz von TCP Reno im Netzwerkprozessor-Modell (Flow)

Bei IPSec als Beispiels für eine Anwendung mit Einfluss der Paketlänge (5.5 und 5.6) geht der Durchsatz gegenüber der Flow-Klassifizierung deutlich herunter. Der Durchsatz sieht wie gedeckelt mit leichten Abweichungen nach oben und unten aus. Auch hier sehen sich TCP Reno und Vegas relativ ähnlich. Jedoch gibt es bei Reno wieder teils erhebliche Ausschläge bei der Datenrate. Obwohl sie teilweise (siehe Flow-Klassifizierung) auch bei TCP Vegas auftreten, ist das Phänomen bei TCP Reno sehr viel stärker. Das unterschiedliche Verhalten von TCP Reno und Vegas könnte also einen Einfluss darauf haben.

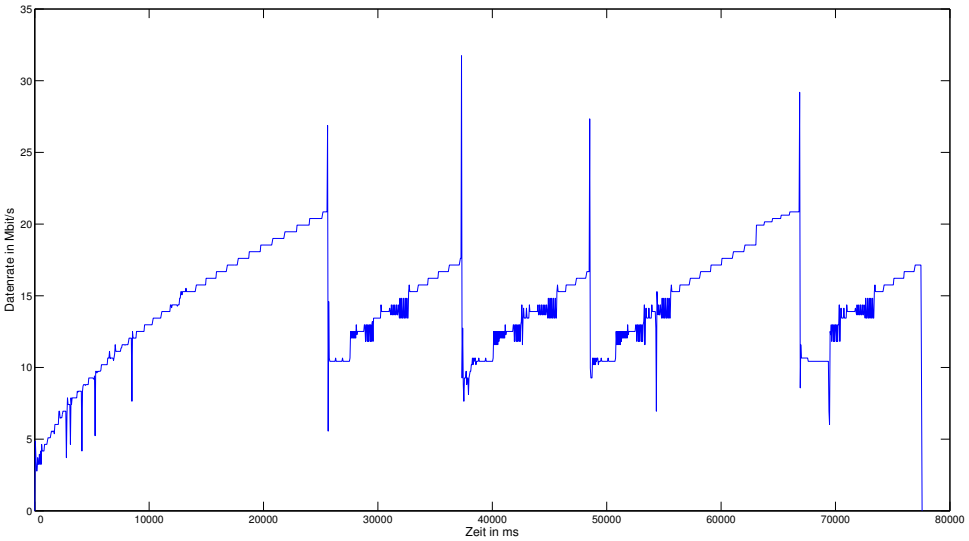

Abbildung 5.4: Durchsatz von TCP Vegas im Netzwerkprozessor-Modell (Flow)

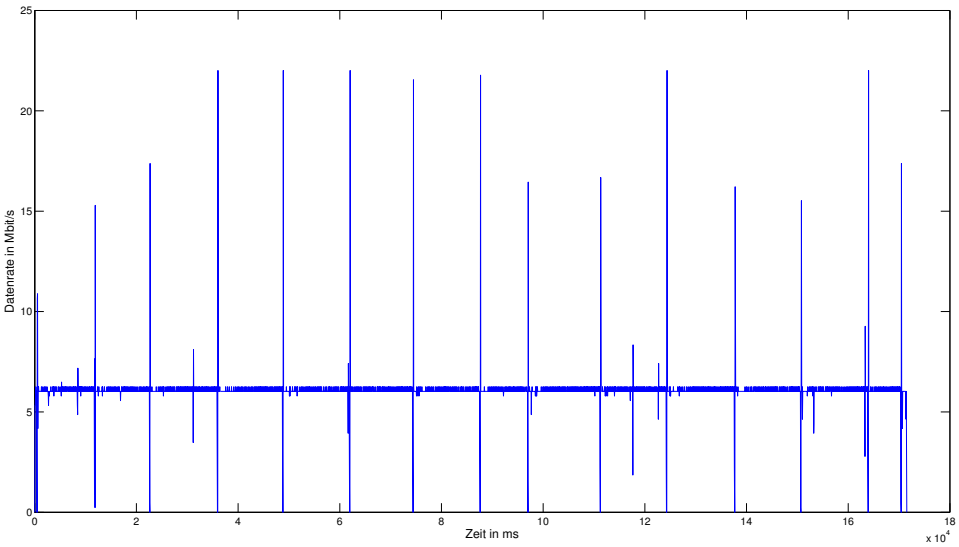

Abbildung 5.5: Durchsatz von TCP Reno im Netzwerkprozessor-Modell (IPSec)

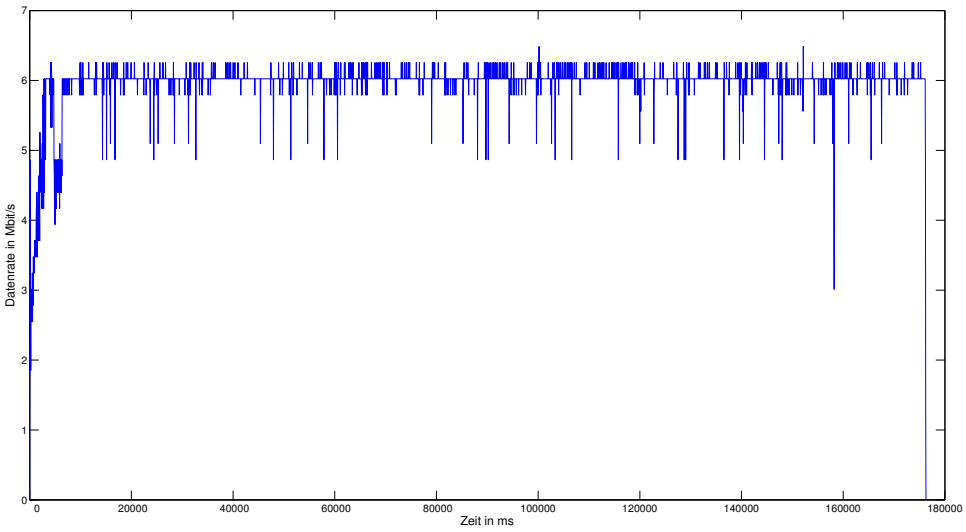

Abbildung 5.6: Durchsatz von TCP Vegas im Netzwerkprozessor-Modell (IPSec)

5.2.2 WLAN-Modell

Durch die zufällig schwankende Paketverzögerung im WLAN-Modell ist das Verhalten von TCP beim Durchlaufen wesentlich interessanter als beim Netzwerkprozessor-Modell. Als Parameter wurden wiederum diejenigen aus Tabelle 3.3 verwendet. Die Paketlänge für große Pakete lag bei 1500 byte, was der MTU im Ethernet und damit auch der gängigen Paketgröße im Internet entspricht. Auch die tatsächlich gemessene Paketlänge lag bei 1500 byte. Die Länge der kleinen Pakete wurde mit 80 byte festgelegt. Es wurde angenommen, dass 60% der Pakete klein sind. Da zu jedem Datenpaket mindesens ein ACK gehört, sollte der Wert realistisch sein.

Der Verlauf der Datenrate ist in den Abbildungen 5.7 für TCP Reno und 5.8 für TCP Vegas dargestellt. Das Verhalten der beiden TCP-Varianten ist relativ ähnlich. Erstaunlich sind die – wie beim Netzwerkprozessor-Modell – auftretenden starken Ausschläge bei TCP Reno. TCP Vegas verhält sich dagegen „normal“ .

In beiden Fällen eine stark schwankende Übertragungsrate. Aufällig ist der stufig wirkende Verlauf der Durchsatzkurven, der auf den Versuch die maximale Bandbreite zu bestimmen zurück zu führen sein könnte. Die typische Charakteristik mit Sägezähnen ist nur noch mit viel Phantasie erkennbar. Beide TCP-Varianten haben also offenbar Schwierigkeiten die optimale Bandbreite zu finden. Trotzdem kommt ein annehmbarer Durchsatz zustande.

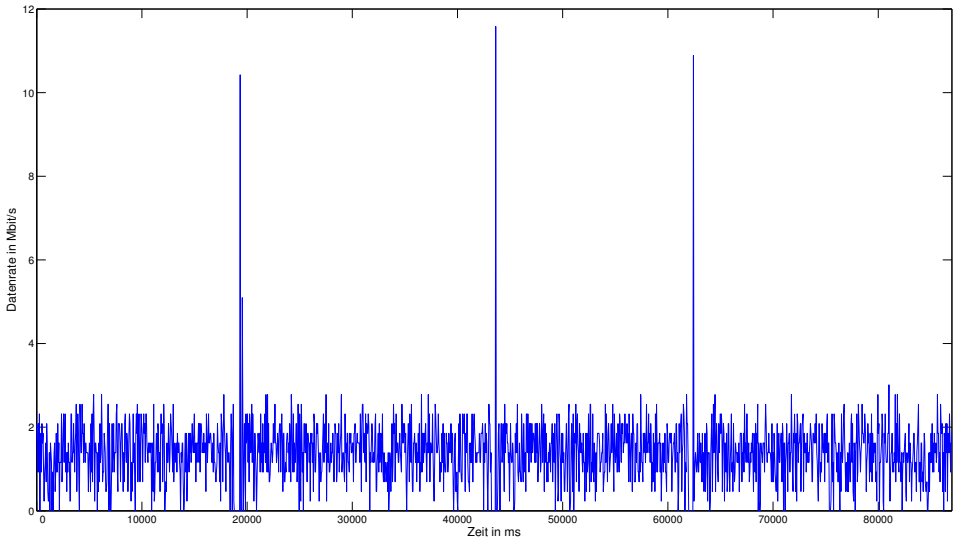

Abbildung 5.7: Durchsatz von TCP Reno im Wlan-Modell

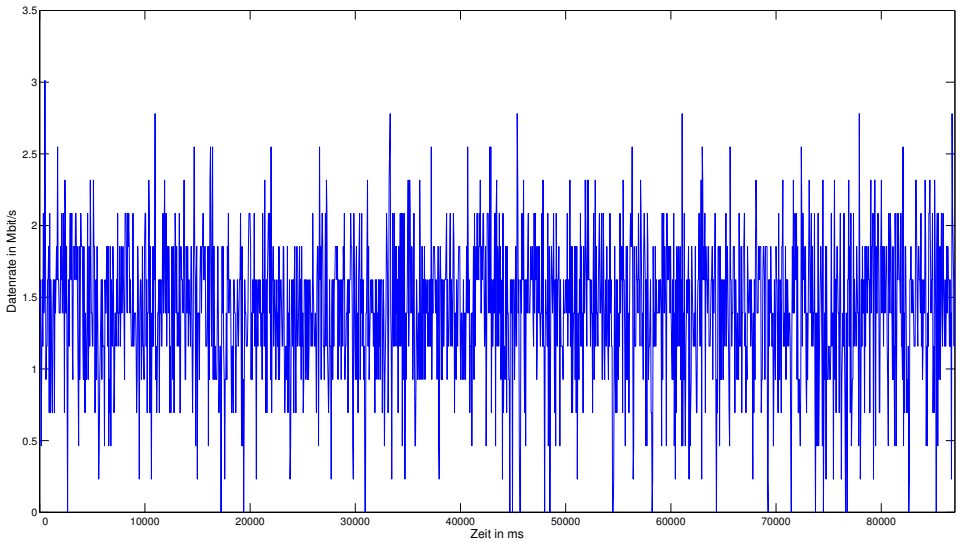

Abbildung 5.8: Durchsatz von TCP Vegas im Wlan-Modell

5.3 Bewertung der Messergebnisse

Zusammenfassend kann gesagt werden, dass die Verzögerungen in den beiden Modellen offenbar korrekt und hinreichend schnell berechnet werden. Im Falle des WLAN-Modells kann durch die Ergebnisse in [RVP09] bestätigt werden, dass die berechneten Werte nahe an der Relität liegen. Für das Netzwerkprozessor-Modell kann leider nur die Größenordnung als plausibel bezeichnet werden, weil konkrete Referenzwerte zur Überprüfung fehlen.

Zu klären bleibt, welche Ursache die auftretenden Ausschläge bei der Datenrate besonders von TCP Reno haben. Diese erscheinen mir wenigstens zu hoch. Durch Vergrößerung der Messintervalle konnte ich zwar Größe der Ausschläge verringern, aber meine Vermutung, dass sie nur durch ungünstige Intervallgrenzen entstehen hat sich nicht bestätigt. Abgesehen davon sehen die Verläufe realistisch aus.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, dass verzögerungsbasierte Staukontrollmechanismen von TCP in manchen Szenarien mangelhaft arbeiten. Dies weckt Bedarf für eine Möglichkeit, verschiedene TCP-Varianten zu vergleichen. Dazu wurde ein Testbed entwickelt, welches solche Szenarien emuliert.

Zunächst wurden zwei wichtig erscheinende Störgrößen für die Funktionalität von verzögerungsbasierten Staukontrollmechanismen herausgearbeitet – die Verarbeitungszeit in Netzwerkgeräten und die schwankende Verzögerung bei WLAN-Verbindungen.

Für die Verzögerung in Netzwerkgeräten wurde ein Modell von [RWW04] in Netzemulator-Framework IKR EmuLib umgesetzt. Das Modell liefert plausible Ergebnisse, die jedoch mangels Referenzwerten nicht detailliert überptüft werden können. Zumindest kann aber gesagt werden, dass die Größenordnung stimmt. Relevant in Bezug auf verzögerungsbasierte Staukontrolle ist das Netzwerkprozessor-Modell bei schwankenden Paketlängen. Ansonsten ist die Verzögerung konstant, da eine Zufallskomponente fehlt.

Für WLAN-Verbindungen wurde das Modell für die Paketverzögerung aus [RVP09] angepasst, so dass es auch auf einzelne Pakete angewendet werden kann. Außerdem wurde die zufällige Verzögerung bei Paketkollisionen genauer abgebildet. In einem weiteren Schritt wurde das Modell nach [RVBP07] erweitert, um unterschiedliche Paketlängen zu berücksichtigen. Beide stochastischen Modelle wurden in IKR EmuLib umgesetzt. Messungen haben die erzeugten Paketverzögerungen des erstellten Modells sehr gut bestätigt.

In Verbindung mit TCP-Verbindungen zeigen sowohl das Netzwerkprozessor-Modell als auch das WLAN-Modell plausibles Verhalten. Allerdings treten bei TCP Reno Ausschläge bei der Datenübertragungsrate auf, deren Ursache noch ungeklärt ist. Diese müssen noch genauer untersucht werden, um ein Problem mit den Modellen oder dem Messvorgang auszuschließen.

Alles in allem erscheint das Testbed geeignet, mit Hilfe die modellierten Störgrößen, einen Beitrag zur Analyse unterschiedlicher Staukontrollmechanismen zu leisten.

Ausblick

Mit Hilfe des entworfenen Testbeds können verschiedene TCP Staukontrollmechanismen verglichen werden. Neben der Performance der jeweiligen Staukontrollmechanismen dürfte

vor allem auch die Fairness gegenüber anderen Varianten zu untersuchen sein. Es wäre inakzeptabel, dass ein neuer Staukontrollmechanismus zu deutlich höheren Bandbreiteanteilen gegenüber anderen Mechanismen führt.

Trotzdem besteht noch Verbesserungspotential. So könnten WLAN-Verbindungen noch genauer modelliert werden. Dabei muss jedoch berücksichtigt werden, dass das Modell schnell genug berechenbar bleiben muss, um in Echtzeit zu arbeiten. Möglichkeiten zur Verbesserung wären die Berücksichtigung der genauen Paketlänge anstatt nur einer Unterscheidung zwischen großen und kleinen Paketen. Interessant wäre auch das Verhalten eines nicht-gesättigten WLANs, sowie der Mischbetrieb verschiedener WLAN-Standards. Es ist allerdings zweifelhaft, wenn auch nicht ausgeschlossen, dass diese genauere Modellierung ein grundsätzlich anderes Verhalten in Bezug auf die Staukontrolle bewirkt.

Interessanter dürfte es sein, Paketverluste aufgrund der unzuverlässigen WLAN-Verbindung zu modellieren. In einem ersten Schritt könnte man dazu einfach zufällig Pakete verwerfen. Realistischer wäre es aber wohl, zwischen Phasen mit vielen korrekt übertragenen Bits in Folge (*runs*) und Phasen mit vielen fehlerhaften Bits in Folge (*burst*) zu unterscheiden. [KWK03] Nicht korrigierte Bitfehler übertragen sich dann in fehlerhafte Pakete, die verworfen werden müssen. Interessant wäre dieses Szenario, weil in diesem Fall die Reaktion der TCP-Staukontrolle genau falsch ist. Richtig wäre es, verlorene Pakete so schnell wie möglich neu zu übertragen. Eine Verlangsamung der Übertragung verschlimmert die Situation noch. [Tan03] Dies ist allerdings ein Problem, das nicht spezifisch für verzögerungsbasierte Staukontrolle ist, sondern mindestens genauso stark verlustbasierte Staukontrolle betrifft.

Schließlich könnten noch weitere, bisher unberücksichtigte Störgrößen identifiziert und umgesetzt werden. Naheliegend wäre dabei eine Untersuchung von anderen Funkübertragungstechniken wie Bluetooth oder UMTS.

Ich bin zuversichtlich, dass das Testbed eine gute Grundlage für die Untersuchung von Staukontrollmechanismen bietet. Insbesondere der WLAN-Teil dürfte hilfreich sein, da er trotz Vereinfachungen die Charakteristik von WLAN-Verbindungen mit schwankenden Verzögerungen gut abbildet.

Literaturverzeichnis

[AAP+00] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradham, D. Saha. Design, implementation and performance of a content-based switch. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1117–1126. 2000. doi: 10.1109/INFCOM.2000.832470. (Zitiert auf Seite 15)

[Bak95] F. Baker. Requirements for IP Version 4 Routers. RFC 1812, Network Working Group, 1995. (Zitiert auf Seite 20)

[Bia00] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *Selected Areas in Communications, IEEE Journal on*, 18(3):535 –547, 2000. doi:10.1109/49.840210. (Zitiert auf den Seiten 20 und 21)

[BP95] L. Brakmo, L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *Selected Areas in Communications, IEEE Journal on*, 13(8):1465 –1480, 1995. doi:10.1109/49.464716. (Zitiert auf Seite 9)

[Cis07] Cisco Systems, Inc. *Cisco Active Network Abstraction Technology Support and Information Model Reference Manual: Layer 2 Tunnel Protocol "L2TP"*, 3.6 edition, 2007. URL http://www.cisco.com/en/US/docs/net_mgmt/active_network_abstraction/3.6/master_tech/9l2tp.html. (Zitiert auf Seite 8)

[CMZ+04] B.-K. Choi, S. Moon, Z.-L. Zhang, K. Papagiannaki, C. Diot. Analysis of point-to-point packet delay in an operational network. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pp. 1797–1807. 2004. doi:10.1109/INFCOM.2004.1354590. (Zitiert auf Seite 18)

[Fuj] K. Fujii. Jpcap – a Java library for capturing and sending network packets. Official Website. URL http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/. (Zitiert auf Seite 37)

[GRL05] S. Guruprasad, R. Ricci, J. Lepreau. Integrated network experimentation using simulation and emulation. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pp. 204–212. 2005. doi:10.1109/TRIDNT.2005.21. (Zitiert auf Seite 17)

[ieee07] IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2007. doi:10.1109/IEEESTD.2007.373646. (Zitiert auf den Seiten 16 und 30)

[ITU94] ITU. X.216: Information technology - Open Systems Interconnection - Presentation service definition, 1994. URL http://www.itu.int/rec/T-REC-X.216/. (Zitiert auf Seite 8)

[KA98] S. Kent, R. Atkinson. Security architecture for the internet protocol. RFC 2401, Network Working Group, 1998. (Zitiert auf Seite 20)

[KR05] J. F. Kurose, K. W. Ross. *Computer Networking: A top-down approach featuring the internet*. Pearson/Addison-Wesley, 3rd edition, 2005. (Zitiert auf den Seiten 8, 9, 11, 12, 15, 17 und 39)

[KWK03] A. Kopke, A. Willig, H. Karl. Chaotic maps as parsimonious bit error models of wireless channels. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pp. 513–523. 2003. doi:10.1109/INFCOM.2003.1208702. (Zitiert auf Seite 53)

[LK91] A. M. Law, W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 2nd edition, 1991. (Zitiert auf Seite 17)

[LZ05] X. Li, Q.-A. Zeng. Performance Analysis of the IEEE 802.11 MAC Protocol over a WLAN with Capture Effect. *IPSJ Digital Courier*, 1:545–551, 2005. doi:10.2197/ipsjdc.1.545. (Zitiert auf Seite 21)

[Mat] Matlab. Official German Mathworks Website. URL http://www.mathworks.de/products/matlab/. (Zitiert auf Seite 29)

[MMSH01] G. Memik, W. H. Mangione-Smith, W. Hu. NetBench: a benchmarking suite for network processors. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, ICCAD '01, pp. 39–42. IEEE Press, 2001. (Zitiert auf Seite 18)

[NG05] M. Necker, C. Gauger. IKR EmuLib: A Library for Seamless Integration of Simulation and Emulation. ITG FG 5.2.1 Simulationsworkshop, Mittweida, Germany, 2005. URL http://www.ikr.uni-stuttgart.de/Content/Publications/View/FullFrame.html?36462. (Zitiert auf Seite 36)

[NGKR06] M. C. Necker, C. M. Gauger, S. Kiesel, U. Reiser. IKR EmuLib: A Library for Seamless Integration of Simulation and Emulation. *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2006 13th GI/ITG Conference*, pp. 1–18, 2006. (Zitiert auf den Seiten 17 und 36)

[NK99] S. Nilsson, G. Karlsson. IP-address lookup using LC-tries. *Selected Areas in Communications, IEEE Journal on*, 17(6):1083 –1092, 1999. doi:10.1109/49.772439. (Zitiert auf Seite 20)

[PMF+03] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, C. Diot. Measurement and analysis of single-hop delay on an IP backbone network. *Selected Areas in Communications, IEEE Journal on*, 21(6):908 – 921, 2003. doi:10.1109/JSAC.2003.814410. (Zitiert auf Seite 18)

[RVBP07] P. Raptis, V. Vitsas, A. Banchs, K. Paparrizos. Delay Distribution Analysis of IEEE 802.11 with Variable Packet Length. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pp. 830–834. 2007. doi:10.1109/VETECS.2007.180. (Zitiert auf den Seiten 30, 31, 32, 33, 34 und 52)

[RVP+05] P. Raptis, V. Vitsas, K. Paparrizos, P. Chatzimisios, A. C. Boucouvalas, P. Adamidis. Packet Delay Modeling of IEEE 802.11 Wireless LANs. In *Proceedings of the 2nd International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2005)*, volume 1, pp. 71–76. 2005. (Zitiert auf Seite 34)

[RVP09] P. Raptis, V. Vitsas, K. Paparrizos. Packet Delay Metrics for IEEE 802.11 Distributed Coordination Function. *Mobile Networks and Applications*, 14:772–781, 2009. doi:10.1007/s11036-008-0124-7. (Zitiert auf den Seiten 16, 20, 21, 23, 24, 26, 30, 34, 43, 44, 51 und 52)

[RW03] R. Ramaswamy, T. Wolf. PacketBench: a tool for workload characterization of network processing. In *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, pp. 42–50. 2003. doi:10.1109/WWC.2003.1249056. (Zitiert auf Seite 20)

[RWW04] R. Ramaswamy, N. Weng, T. Wolf. Characterizing network processing delay. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 3, pp. 1629–1634. 2004. doi:10.1109/GLOCOM.2004.1378257. (Zitiert auf den Seiten 12, 18, 19, 20, 42 und 52)

[Tan03] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 4th edition, 2003. (Zitiert auf den Seiten 8, 9, 15 und 53)

[WF00] T. Wolf, M. Franklin. CommBench-a telecommunications benchmark for network processors. In *Performance Analysis of Systems and Software, 2000. ISPASS. 2000 IEEE International Symposium on*, pp. 154–162. 2000. doi:10.1109/ISPASS.2000.842295. (Zitiert auf Seite 18)

[WPL+02] H. Wu, Y. Peng, K. Long, S. Cheng, J. Ma. Performance of reliable transport protocol over IEEE 802.11 wireless LAN: analysis and enhancement. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pp. 599–607. 2002. doi:10.1109/INFCOM.2002.1019305. (Zitiert auf Seite 28)

[ZKF04] H. Zhai, Y. Kwon, Y. Fang. Performance analysis of IEEE 802.11 MAC protocols in wireless LANs. *Wireless Communications and Mobile Computing*, 4(8):917–931, 2004. doi:10.1002/wcm.263. URL http://dx.doi.org/10.1002/wcm.263. (Zitiert auf Seite 45)

Alle URLs wurden zuletzt am 04.08.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Mark Hübler)