Adaptive Verfahren zur Lastkontrolle im Internet

Annabel de la Torre Cevallos

Dissertation,

genehmigt von der Fakultät für Informatik

der Universität Fridericiana zu Karlsruhe

Tag der mündlichen Prüfung: 04.06.2004

Erster Gutachter: Prof. Dr.-Ing. Werner Zorn

Zweiter Gutachter: Prof. Dr.rer.nat. Wilfried Juling

Mein Dank für die Betreuung dieser Arbeit möchte ich meinem Doktorvater, Herrn Prof. Dr.-Ing. Werner Zorn zum Ausdruck bringen. Seine Anregungen, Fachkenntnisse und die mir entgegengebrachte Offenheit waren sehr wertvoll für das Gelingen dieser Arbeit. Herrn Prof. Dr.rer.nat. Wilfried Juling danke ich für die Übernahme des Korreferats und die damit verbundene Betreuung.

Herrn Prof. Dr.-Ing. Siegfried Wendt und Herrn Alfred Klein des Hasso-Plattner-Instituts (HPI) möchte ich für die Bereitstellung der Arbeitsmöglichkeiten beim HPI danken.

Herrn Dipl.Inform. Thomas Brandel aus der Universität Karlsruhe möchte ich für seine Unterstützung bei den Messungen danken.

Dem KAAD danke ich für die finanzielle Unterstützung der ersten zwei Jahren.

Meinem Ehemann Jens Bohnert danke ich für die anstrengende Korrektur des Manuskripts, die geleistete Motivation und die mir entgegengebrachte Geduld.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Überlastkontrolle (engl. Congestion-Control) ist eine ständige unabdingbare Aufgabe in Rechnernetzen, da Netzüberlastung unabhängig von der Geschwindigkeit der Netze auftritt. Netzüberlastung verschwindet nicht, indem die Komponenten der Rechnernetze wie Speicher, Prozessoren und Leitungen schneller werden (siehe z.B. [Rai90]), denn die Engpässe werden dabei i.a. immer nur verlagert. Eher im Gegenteil ist es die Disparität zwischen den Übertragungskapazitäten der Leitungen, zu kleine oder zu große Puffer, oder zu langsame Prozessoren, die das Leistungsverhalten negativ beeinflussen und Netzüberlastung verursachen.

Die Leistungskenngrößen der Datenübertragung des Internet können sehr unterschiedlich sein [Flo91, FJ92, Her95, Pax97a, Pax99, SC01], da das Internet Rechnernetze mit diversen Techniken verbindet. In [Pax97a] und [Pax99], einer Studie mit 20 000 TCP-Datenübertragungen und 35 Hosts, wurde eine hohe Varianz in der Verteilungsfunktion der Paketverlustrate in Abhängigkeit von der Tageszeit festgestellt. Größere Variationskoeffizienten der Leistungskenngrößen in Gigabit-Netzen wurden in der Studie von [SC01] festgestellt. Ergebnisse in experimentellen Netzen wie z.B. Internet2 [ST01] zeigen ebenfalls hohe Varianz des Durchsatzes in TCP. Benutzer von Internet-Diensten wie FTP, Telnet bzw. SSH, HTTP usw. bestätigen hohe Schwankungen der Antwortzeiten und Durchsätze zwischen unterschiedlichen Netzen, aber auch in einzelnen Netzen zu unterschiedlichen Tageszeiten.

Ende der 90er Jahren nutzten 95% der gesendeten Bytes und 90% der Verbindungen im Internet-Verkehr TCP als Transportprotokoll [TMW97], wobei TCP-Reno die am meisten benutzte TCP-Variante war [Ste97]. Bis Anfang der 90er Jahre waren die meisten TCP-Datenübertragungen auf Grund der dominierenden FTP-Applikationen hochvolumig [BRS99]. Durch den Einsatz des HTTP-Protokolls ist die Mehrzahl der TCP-Datenübertragungen extrem kurz geworden und umfassen manchmal nur wenige KB [TMW97]. Diese kurzen konkurrierenden Datenübertragungen enden, bevor die in TCP vorgesehenen Mechanismen sich an die Netzbedingungen anpassen können. Dies hat seinen Grund darin, dass TCP nicht für kurzzeitige Datenübertragungen entwickelt wurde.

1.2 Problem und existierende Ansätze

Das Problem der Überlastung sei anhand der Abbildung 1.1 veranschaulicht. Idealerweise sollten alle Sender den gleichen oder einen fest einstellbaren Anteil an der Kapazität des Engpasses erhalten. Abbildung 1.1 (a) zeigt den Fall, bei welchem `Sender2` keinen Vorteil gegenüber `Sender1` hat. Die Leitung zwischen `Router` und `Empfaenger` wird fair aufgeteilt, unabhängig von der Tatsache, dass `Sender2` die doppelte Leitungskapazität gegenüber `Sender1` zum `Router` hat. In der Realität geschieht dieses nicht, sondern einige Akteure *verdrängen* andere überproportional. Abbildung 1.1 (b) zeigt, dass `Sender2` mehr Kapazität am Engpass als `Sender1` erhält. Diese Verdrängung kommt durch eine schnellere Anbindung und durch die daraus resultierenden kürzeren Antwortzeiten zu Stande. Würde die Kapazität der gemeinsamen Ausgangsleitung des `Routers` auf 3 Mb erhöht, könnte es trotzdem zur Überlastung kommen, wenn `Router` nicht in der Lage wäre, alle ankommenden Pakete zu verarbeiten und dann selbst wieder die neue Engpasskomponente bilden würde. Mögliche Gründe könnten langsame Prozessoren oder mangelnde Pufferkapazität sein. Inzwischen sind die Leitungen nicht der Engpass, sondern die Hosts oder die Router[1], da in den letzten Jahren die Preisleistungs-Verhältnisse der Leitungskapazität viel stärker gefallen sind als die der Prozessoren.

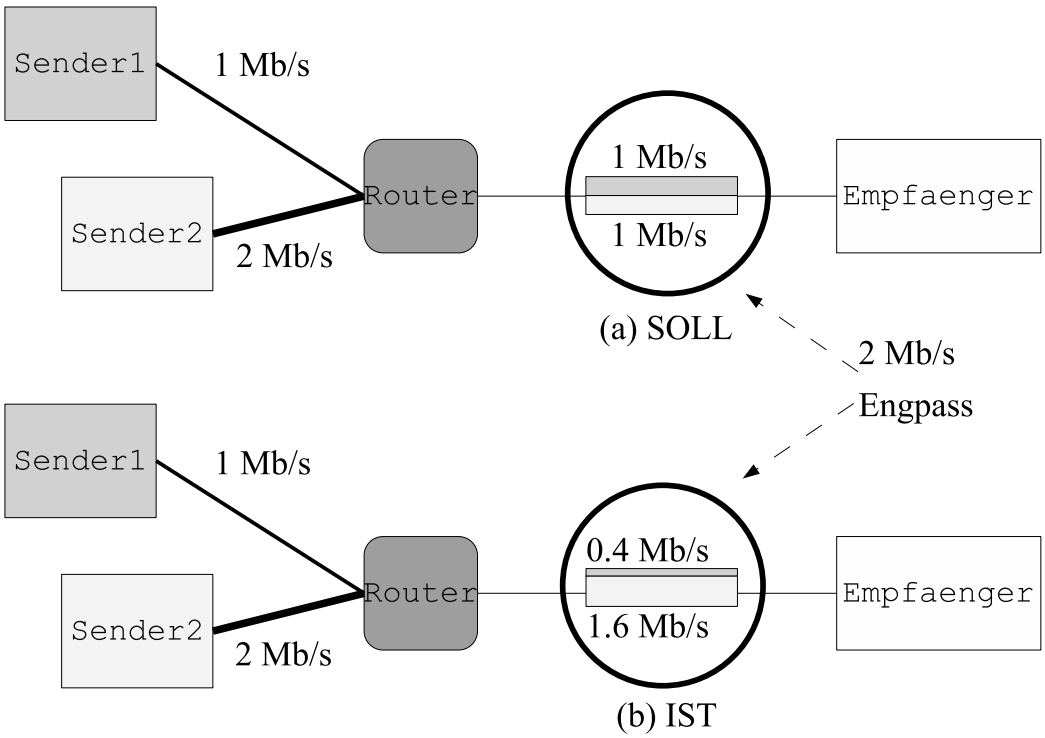


Abbildung 1.1: Teilung der Leitungskapazität (a) SOLL - Engpass zwischen `Sender1` und `Sender2` fair geteilt, (b) IST - `Sender2` verdrängt `Sender1` beim Engpass

Die Ursachen der in Abbildung 1.1 dargestellten Verdrängung sind unterschiedlicher Natur. Welche Mechanismen für die Pufferbehandlung und für das Scheduling in den Routern eingesetzt werden, ist dabei ebenso wichtig wie die Größe der Senderate auf Transport- und/oder Applikationsschicht. Angenommen `Sender1` und `Sender2` wenden die gleichen Protokolle an, so ist es möglich, dass `Router` vorwiegend Pakete von `Sender1` verwirft, sobald seine Puffergröße nicht mehr ausreicht.

[1] Der Begriff Host bezeichnet hier ein Endsystem in Rechnernetzen. Der Term Router wird in der Literatur unterschiedlich angewendet. Hier sind die Zwischensysteme der Rechnernetze gemeint, d.h. die Rechner, die Hosts verbinden.

Dieses Verhalten ist letztlich durch die Netzüberlastung begründet. Netzüberlastung ist in den Routern beobachtbar, sobald mehr Verkehr ankommt als von diesem verarbeitet werden kann. Dabei stauen sich Pakete in den Warteschlangen und führen bei IP-basierten Netzen zum Paketverlust. Paketverluste wirken sich in den Transport- und Applikationsprotokollen noch gravierender aus und reichen von der erneuten Übertragung eines Pakets oder größerer Transporteinheiten über Sender-Deaktivierung bis hin zum Verbindungsabbruch.

Der Internet-Verkehr ist stochastisch, Internet selbst ein chaotisches System. Chaos impliziert in deterministischen nichtlinearen Systemen instabiles Verhalten [Med92]. Hierbei ist es unmöglich, längerfristige Vorhersagen zu treffen, während jedoch kurzfristige durchaus gültig sind. So fand z.B. [Pax97b] dass, eine Vorhersage von Paketverlusten *nur* für die nächsten Minuten nützlich ist. In chaotischen Systemen können ähnliche Prozesse einen ähnlichen Verlauf haben, jedoch nur solange ein Ereignis eintrifft, in Folge dessen die Prozesse divergieren. Im Internet wird zunehmend dieser Effekt beobachtet [ENW96].

In [HL97] findet man die Erklärung des Phänomens der so genannten *Gewitter* im Internet. Gewitter sind plötzlich auftretende Perioden mit extrem hoher Last. Gewitter entstehen dadurch, dass die kurze Antwortzeit während der Perioden mit geringer Last dazu führt, dass die Benutzer das Netz intensiver nutzen. Dieser positive Rückkopplungseffekt führt zu einem Werteanstieg der Last, solange bis Überlast eintritt. Mit höherer Last verschlechtert sich die Antwortzeit, durch welche Verbindungen abgebrochen werden und die Benutzer das Netz verlassen. Im Ergebnis ist der Internet-Verkehr oszillierend, wobei kleine Änderungen große Auswirkungen haben können, was typisch für chaotische Systeme ist.

Die Aufgabe besteht darin, geeignete Mechanismen zu finden, um diese Schwankungen zu reduzieren. Dabei bereitet die hohe Anzahl von Akteuren und die Eigendynamik des Internet besondere Schwierigkeiten. Dabei besteht eine wesentliche zusätzliche Anforderung darin, dass solche Überlastkontrollen in einem Arbeitsbereich von einigen Kb/s bis hin zu Mb/s oder sogar Gb/s -Übertragungsraten robust wirken sollten.

Der Zyklus der Überlastkontrolle besteht nun darin:

(i) den Verkehr zu beobachten und Überlast festzustellen,
(ii) entscheiden, was geändert werden soll,
(iii) die Änderungen zu befolgen,
(iv) das geänderte Verhalten des Systems erneut zu beobachten.

Die Mechanismen zur Überlastkontrolle müssen von der Natur der Sache her im Netz verteilt sein, z.B. in den Routern oder in den Hosts. Routerbasierte Überlastkontrolle (engl. Gateway Congestion-Control) setzt voraus, dass darauf liegende Transport- oder Applikationsprotokolle durch Ende-zu-Ende-Überlastkontrolle (engl. End-to-End Congestion-Control) auf ihre Aktionen reagieren. Ein weiterer wichtiger Aspekt ist festzustellen, für welche Verkehrsarten sie eingesetzt werden sollen. Diese Arbeit wird vor allem auf den Einsatz innerhalb von *Best-Effort*-Netzen eingehen. Best-Effort bedeutet, dass der Durchsatz nicht garantiert werden kann. Typisch dafür ist die Übertragung einer Datei mit FTP oder HTTP über TCP/IP. Die Anforderungen an solche Dienste unterscheiden sich von Echtzeit-Diensten, wie Audio- oder Video-Übertragung, in zwei zentralen Punkten: der Dienst ist nicht zeitkritisch, verlangt jedoch die zuverlässige Übertragung des ganzen Datenstroms. Das Problem der Überlastkontrolle ist so alt wie TCP/IP (d.h. etwa 20 Jahre) und in zahllosen Arbeiten untersucht worden. Dies hat zu mehreren RFC geführt, die zum Teil implementiert wurden.

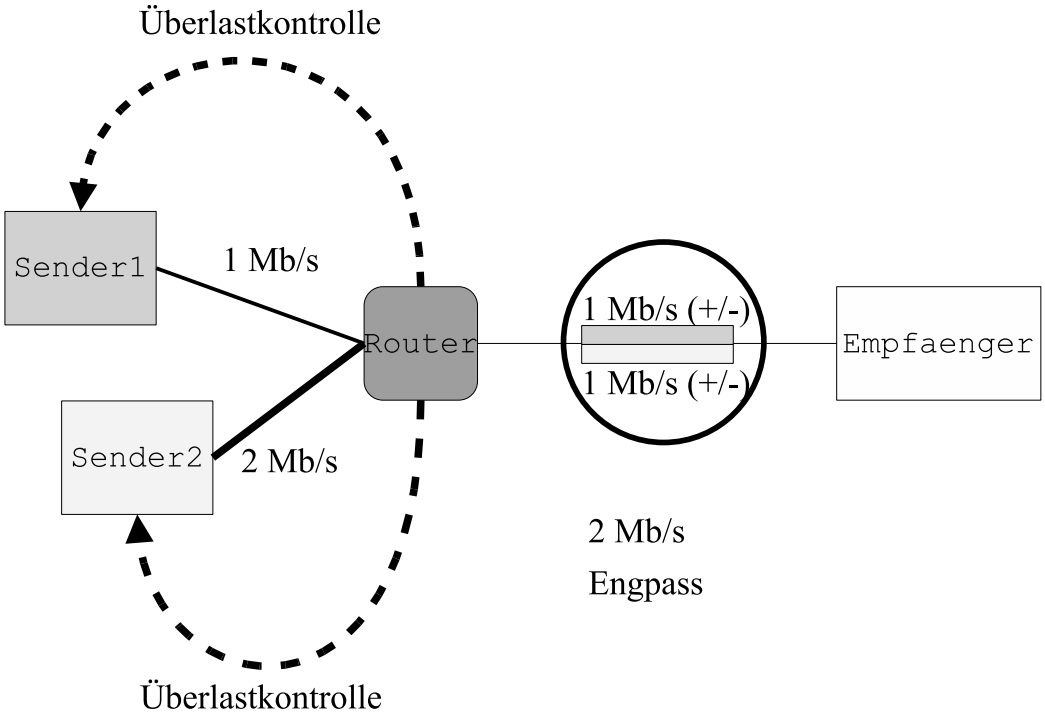


Abbildung 1.2: Einsatzort der Überlastkontrolle

Die Evolution von routerbasierten Verfahren lässt die geänderten Anforderungen an die Netze erkennen. Packet-Discarding (PD) [Tan81] z.B. benutzt eine Warteschlange für alle Verbindungen, was zu Folge haben kann, dass die Leitungskapazität nicht fair geteilt wird (siehe Abbildung 1.1). Um die Benutzer voneinander zu isolieren, verwenden u.a. Fair-Queuing (FQ) [Nag85], Bit-Round Fair-Queuing (BRFQ) [DKS89] und Stochastic Fair-Queuing (SFQ) [McK90] mehrere Warteschlangen. Die Kosten der so gewonnenen Fairness sind hohe Komplexität und höherer Protokollaufwand. Random Early Detection (RED) [FJ93] nimmt die Prinzipien von PD wieder auf und führt folgende Idee ein: durch gezieltes Verwerfen von Paketen können die Transportprotokolle auf eine sich bildende Netzüberlastung in statu nascendi reagieren. Pakete von denjenigen Sendern, die zu Lasten anderer Sender mehr Leitungskapazität verlangen, werden bevorzugt und in Abhängigkeit der Auslastung der Warteschlange verworfen. Variationen von RED sind FRED [LM97], SRED [OLW99], REM [Ath01], PI [HMTG01] und ARED [FGS01]. Der Unterschied zwischen ihnen ist das Kriterium zum Verwerfen eines Pakets bzw. die Mechanismen zur Erkennung von *aggressiven* Sendern.

Die meisten für TCP vorgeschlagenen Verfahren sind vom Typ Ende-zu-Ende. Standard TCP-Varianten setzen auf Slow-Start [Jac88] auf. Slow-Start verursacht auf Grund seines exponentiell wachsenden Senderfensters instabile Senderaten, welche zu Timeouts bei Paketverlusten führt, und nach welchen anschließend das Fenster auf den minimalen Anfangswert von einem Segment gesetzt wird. In den Routern entsteht dabei burst-artiger Verkehr mit großen Lastspitzen. Dieses Verfahren vermeidet die Netzüberlastung daher nicht.

Im Laufe der Zeit sind Recovery-Mechanismen zu TCP hinzugefügt worden. Sie behandeln die erneute Übertragung fehlender Pakete und sie zielen auch darauf hin, Timeouts zu verhindern, da sie den Durchsatz stark negativ beeinflussen. Dabei wird versucht, die Stabilität der Datenübertragung dadurch sicherzustellen, dass eine erneute Übertragung verloren gegangener Pakete vom Sender auch

vorgenommen wird, wenn noch kein Timeout eingetreten ist. Kriterien hierfür sind wiederholte Anforderungen von fehlenden Paketen.

In den Verfahren CARD [Rai89], Tri-S [WC91], DUAL [WC92], VEGAS [BOP94] wird die Antwortzeit als Maß zur Einschätzung der Netzauslastung verwendet. Das Vergrößern/Verkleinern des Fensters hängt vom Wert der Antwortzeit ab. Dies ist nichts anderes als die Variationen der Warteschlangen der Router abzuschätzen und das Fenster anzupassen.

Neue Ende-zu-Ende-Überlastkontrolle sind TCP-ähnliche senderatenbasierte Verfahren, die auch für Echtzeit-Verkehr einsetzbar sind: TCP Friendly Rate Control Protocol (TFRCP) [PKTK98] und Equation-Based Congestion-Control (FRTP) [FHPW00]. Durch die Einstellung einer stabileren Senderate wird die hohe Varianz der Antwortzeit kontrolliert.

Schließlich gibt es Verfahren, die sowohl Ende-zu-Ende- als auch routerbasierte Überlastkontrolle einsetzten. Die ursprüngliche Idee stammt aus DECBit [JRC87, RR88] und besteht darin, bei hoher Last ein Bit im Paket-Header zu setzen. Ein Bit wird in jedem Router angehängt und beim Sender zum Vergrößern/Verkleinern des Fensters angewendet. Weiterentwicklungen davon sind Selective-Binary-Feedback [RJC91] und Q-bit [Ros93]. Eine Verallgemeinerung dieses Ansatzes ist XCP [KHR02]. Anstelle von Bits wird die notwendige Veränderung des Fensters in einem Paket-Header mitgeteilt. Der Paket-Header ist dadurch größer, aber die Information zum Vergrößern/Verkleinern des Fensters ist präziser.

Es ist ersichtlich, dass Paketverwurf und deren Behandlung eine große Rolle in der Überlastkontrolle für Best-Effort spielen. Aktuelle Standards setzen auf RED und auf den Einsatz von Recovery-Mechanismen und Slow-Start auf. Dies impliziert, dass falls die Recovery-Mechanismen nicht eingesetzt werden, das Verkehrsprofil weiterhin stark schwankend wird. Um gute Durchsätze zu ermöglichen, muss der Sender diese Mechanismen möglichst mit geringer Zeitverzögerung anwenden. Ebenso neue, auf stabile Senderaten basierende Ende-zu-Ende-Verfahren benutzen weiterhin die Paketverlustrate zur Überlastkontrolle. Dies kann zu niedrigen Durchsätzen führen, zumal in kurzen Datenübertragungen der Einfluss einer kleinen Paketverlustrate groß sein kann. DECBit-ähnliche Verfahren sind die Alternative, wobei fast alle von ihnen nur unzureichende Feedback-Signale zur Einstellung einer geeigneten Senderate geben.

Das Problem der Überlastkontrolle in Best-Effort-Netzen ist bisher nicht abschließend gelöst worden. Dies hat seinen Grund darin, dass die Rahmenbedingungen sich ständig ändern, darunter die Art, Anzahl und Dauer der Verbindungen sowie die Leitungskapazitäten. Da die Überlastkontrolle gute Durchsätze ermöglichen soll, stellt sich die Frage, ob Mechanismen, die für Megabit-Netze entworfen wurden, auch in Gigabit-Netzen gute Ergebnisse erzielen. Siehe z.B. Slow-Start in TCP: Da dabei lediglich wenige Pakete ausgetauscht werden, führt seine Anwendung zu einer niedrigen Auslastung der Leitung. Wird zusätzlich der Einfluss von Paketverlust betrachtet, ist es evident, dass dieses Verfahren gravierende Leistungsprobleme mit sich bringt. Liegen Sender und Empfänger weit von einander entfernt, gibt es zudem den Nachteil, dass die benötigte Zeit für die Ausbreitung eines Feedback-Signals lang ist. Ein weiterer Grund ist, dass TCP ein komplexes Protokoll ist. Es existieren nicht viele unterschiedliche Implementierungen, da sie meisten auf dem so genannten 4.x BSD System basieren. Aus Mangel an Transparenz sind nur geringfügige Optimierungen vorgenommen worden, vor allem Recovery-Mechanismen.

Daher besteht die Möglichkeit einen Beitrag zu leisten, und die aktuellen Ansätze und Rahmenbedingungen erneut zu überprüfen. Vor allem in Gigabit-Netzen gibt es Perspektive für DECBit-ähnliche Verfahren, wobei Mechanismen für die Generierung präziser Feedback-Signale gesucht werden müssen.

1.3 Ziel

Die Grundhypothese ist, dass DECBit-ähnliche- und Ende-zu-Ende-Überlastkontrolle nicht ausreichen, um hohe Durchsätze zu erreichen und gleichzeitig die Netzstabilität zu gewährleisten. Das Netz muss ebenfalls die Möglichkeit haben, den einen oder den anderen Sender zu bremsen, um die Last zu reduzieren. Es sollen Protokollmechanismen eingesetzt werden, die für einen stabileren Betrieb sorgen. Die großen Schwankungen der Senderate der Verbindungen sollen vermieden werden. Um hohe Durchsätze und Fairness zwischen den Verbindungen zu erhöhen, sollte das Verwerfen von Paketen als letztes Mittel eingesetzt werden.

In dieser Dissertation wird ein Modell zur adaptiven Überlastkontrolle für Best-Effort-Verkehr entwickelt, das diesen Anforderungen genügt. Dies erfordert Mechanismen zur Kontrolle der Last an der Quelle und zum frühen Erkennen der Überlast in den Routern. Dafür ist es notwendig, dass die Feedback-Signale aus den Routern nicht nur Information über die Existenz von Netzüberlastung beinhalten, sondern auch über das gewünschte Lastniveau der jeweiligen Verbindungen.

1.4 Gliederung und Vorgehensweise

Nach dem vorliegenden Kapitel 1 werden in Kapitel 2 die allgemeinen leistungsanalytischen Grundlagen und Kenngrößen in Rechnernetzen behandelt.

In Kapitel 3 werden die speziellen Ursachen und Folgen der Netzüberlastung erläutert. Ihr Einfluss auf die Kenngrößen in Rechnernetzen sowie die Grundprinzipien der Überlastkontrolle werden vorgestellt. Die wichtigsten der zahlreichen existierenden Ansätze werden behandelt und näher untersucht. Der Akzent wird auf die Analyse der Verfahren zur Überlastkontrolle für den Best-Effort-Verkehr gesetzt.

Kapitel 4 beinhaltet eine Analyse des Einflusses von Netzüberlastung auf die Datenübertragung in TCP. Die Mechanismen zur Überlastkontrolle werden in einem FMC-basierten [FMC] Petrinetz-Zustandsdiagramm [Zor02] dargestellt. Dieses Zustandsdiagramm veranschaulicht die Einbettung der Überlastkontrolle in TCP innerhalb des Protokollstacks. Die Folgen auf die Leistungskenngrößen von zufällig eintreffenden Ereignissen sowie die Effekte auf parallele Datenübertragungen werden in einer ergänzenden Fallstudie aufgezeigt. Dabei wird demonstriert, wie sich ein einziger Paketverlust in zwei TCP-Datenübertragungen in Abhängigkeit ihrer Zustandsvariablen völlig unterschiedlich auswirken kann.

In Kapitel 5 wird das hier entwickelte ANDINA-Modell (**A**daptive **N**etwork **D**atastream **IN**line **A**lgorithm) zur Überlastkontrolle vorgestellt. Die Ergebnisse der Analyse der Überlastkontrolle in TCP sowie die Untersuchung von existierenden Ansätzen weisen darauf hin, dass kombinierte Ende-zu-Ende- mit routerbasierten Mechanismen am stabilsten arbeiten. Es ist weiterhin notwendig, dass durch genauere Feedback-Signale die Senderaten erhöht und abgesenkt werden können. Der Ansatz ist: jeder Router bestimmt anhand seiner Last ein lokales Optimum zur Minimierung von Paketverlust und Maximierung der Auslastung der Leitung. Dieses Optimum ist eine Zwischenankunftszeit (IAT), die von der Anzahl der abgeschätzten aktiven Verbindungen abhängt. Die drei Komponenten, Sender, Empfänger und Router, kommunizieren mittels eines Paket-Headers. Der Router generiert eine Soll-IAT und schreibt diese in den Paket-Header. Der Empfänger kopiert den Paket-Header und sendet ihn an den Sender unverändert zurück. Der Sender benutzt die IAT zur Generierung der Senderate.

Die Eigenschaften dieses Verfahrens erfordern die Analyse durch Simulation, da es im Rahmen dieser Arbeit nicht möglich war, ANDINA innerhalb eines realen Routers zu implementieren. Mit der Simulation ist es möglich Netzmodelle in komplexen Umgebungen zu untersuchen. Das eingesetzte Simulations-Programm, *Network Simulator* (`ns`), ist ein Simulator für diskrete Ereignisse. Es ist in einer objektorientierten Version von `Tcl`, `OTcl`, und `C++` implementiert und ermöglicht, Überlastkontrolle unter vielen verschiedenen Bedingungen zu untersuchen. Ein Prototyp von ANDINA wurde in `ns` implementiert, und in vielen zahlreichen typischen Test-Szenarios untersucht. Die Stabilität unter unterschiedlichen Rahmenbedingungen wie Leitungskapazität, unterschiedlichen Pfadlänge, Dauer der Verbindungen und Auslastungsgrad wurde bewertet. Der Vergleich erfolgt mit den Simulator-Versionen der derzeit weltweit bevorzugt im Einsatz befindlichen TCP-Varianten Reno, Newreno und SACK und den Warteschlangen PD, ARED, REM und PI.

Kapitel 6 enthält eine Zusammenfassung der wichtigsten Ergebnisse sowie einen Ausblick.

Im Anhang finden sich ein Glossar, ein mathematisches Glossar, sowie Tabellen der Messergebnisse und der verwendeten FMC-Notation.

Kapitel 2

Grundlagen der Leistungsanalyse

Das Internet ist ein offenes paketvermitteltes Rechnernetz. Da Rechnernetze als Warteschlangensysteme modelliert werden können, sollen zuerst einige Grundlagen der Warteschlangentheorie vorgestellt werden. Einige Kenngrößen der Leistungsanalyse von Rechnernetzen sowie Grundmodelle der Datenübertragung in der Transport-, Internet- und Netzschicht werden anschließend vorgestellt und erläutert.

2.1 Warteschlangensystem (WS-System)

2.1.1 Struktur und Kenngrößen

Gegeben sei ein elementares WS-System (siehe Abbildung 2.1) bestehend aus einem WS-Bediener und einer Warteschlange, mit:

- N Anzahl der Aufträge im System, d.h. die Last des Systems ist N Aufträge
- N_q Anzahl der Aufträge in der Warteschlange
- N_s Anzahl der Aufträge beim Bediener
- A Ankunftsrate der Aufträge
- X Bedienzeit beim Bediener
- W Wartezeit der Aufträge in der Warteschlange, wenn sie nicht sofort bedient werden
- D Durchsatz des Systems
- R Reaktionszeit, d.h. die gesamte Zeit der Aufträge im System

Die Reaktionszeit besteht aus Wartezeit und Bedienzeit:

$$R = X + W \tag{2.1}$$

Folgende Grundgesetze gelten im eingeschwungenen Zustand, vorausgesetzt, dass sich der Prozess im stationären Zustand befindet. Der eingeschwungene Zustand schließt die Anfangsphase eines Prozesses aus. Ein Prozess ist stationär, wenn seine charakteristische Funktion zeitunabhängig ist:

$$f(x) = f(x + \Delta t) \tag{2.2}$$

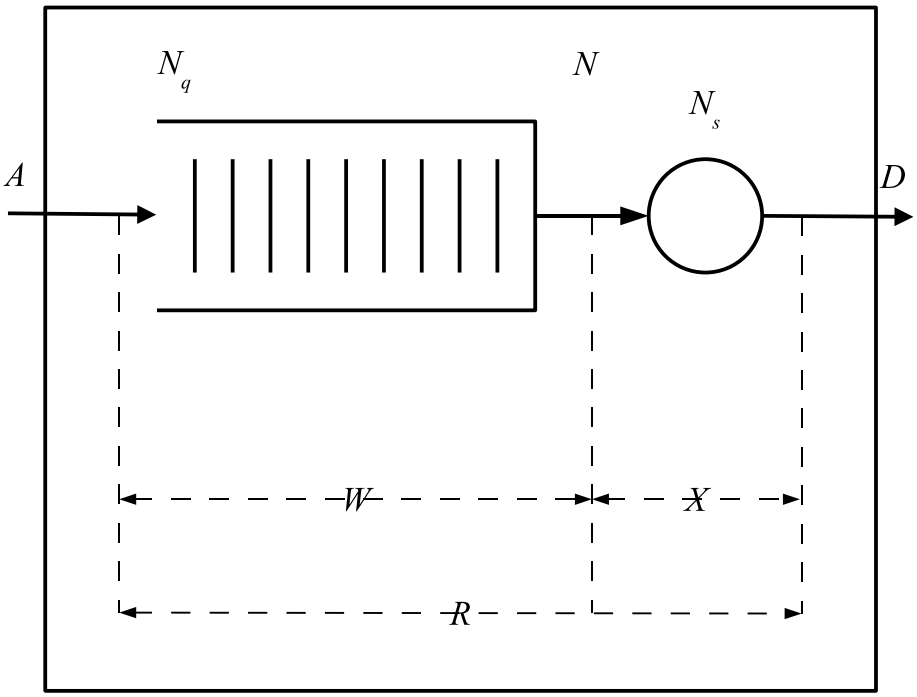


Abbildung 2.1: Bediener mit Warteschlange

Maximaldurchsatz [HZ95]. Der Maximaldurchsatz D_{max} wird erreicht, wenn:

$$D_{max} = \frac{1}{X} \tag{2.3}$$

Auslastung [HZ95]. Der Zeitanteil währenddessen der Bediener beschäftigt ist, ist die Auslastung U des Systems. In stabilen Systemen ist die Auslastung definiert als:

$$U = \frac{D}{D_{max}} = N_s \qquad 0 \leq U < 1 \tag{2.4}$$

Stabilitätsbedingung [HZ95]. Ein System ist stabil, wenn es gilt:

$$A \leq D_{max} \tag{2.5}$$

d.h. wenn im Laufe der Zeit seine Warteschlange nicht bis ins Unendliche wächst. Dafür ist es notwendig, dass der ankommende Auftragsfluss kleiner oder gleich dem ausgehenden Auftragsfluss ist. Daraus folgt, dass die Stabilität des Systems nur dann gewährleistet ist, wenn die Ankunftsrate kleiner oder gleich der Bedienrate ist. Durch Verletzung der Stabilitätsbedingung können nicht alle ankommenden Aufträge aufgenommen werden. Diese Aufträge werden aus dem System *verworfen*.

Satz von Little [Lit61]. Das Satz von Little (siehe Abbildung 2.2) sagt aus, dass die mittlere Anzahl der Aufträge im System das Produkt aus Ankunftsrate und Reaktionszeit ist. Analog dazu ist die mittlere Anzahl der Aufträge in der Warteschlange das Produkt aus Ankunftsrate und Wartezeit.

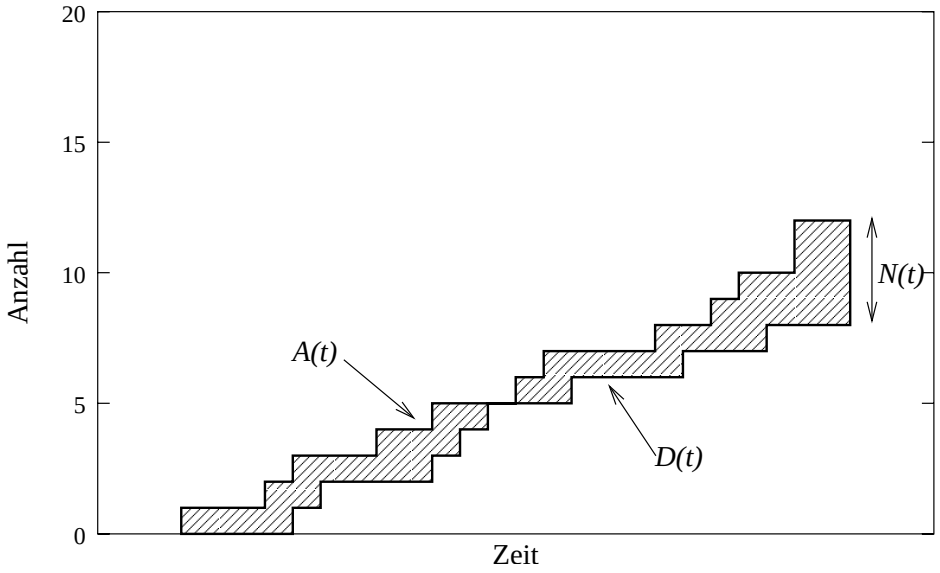


Abbildung 2.2: Satz von Little (bas. auf [Kle75])

$$N = A \cdot R \tag{2.6}$$

mit

$$A = \lim_{t \to \infty} A(t) \tag{2.7}$$

$$R = \lim_{t \to \infty} R(t) \tag{2.8}$$

Im Gleichgewicht ist $A = D$ und daher:

$$N = D \cdot R \tag{2.9}$$

$$N_q = D \cdot W \tag{2.10}$$

$$N_s = D \cdot X \tag{2.11}$$

2.1.2 Deterministische WS-Systeme

In deterministischen WS-Systemen sind den Kenngrößen zu jedem Zeitpunkt eindeutig festgelegte Werte zugewiesen [HZ95], d.h. die Zeitpunkte, die Dauer von Zuständen und die Zeitpunkte der Zustandsübergänge sind a priori bekannt. Ein dynamisches System lässt sich dadurch exakt beschreiben. Die Kenngrößen des Systems werden als Mittel- oder als Extremwerte dargestellt. Daraus resultieren insbesondere zwei Arten für die Analyse: Mittelwert- und Extremwertanalysen. Mit der Extremwertanalyse können obere und untere Grenzen der Leistungskenngrößen festgestellt werden.

2.1.3 Stochastische WS-Systeme

Stochastische WS-Systeme sind durch zwei Zufallsprozesse beschrieben. Diese Prozesse sind die Zwischenankunftszeit und die Bedienzeit der Aufträge. Die Wechselwirkung zwischen ihnen beschreibt die Wartezeit (und Reaktionszeit) im WS-System. Zufallsprozesse werden durch die Verteilungs- und Dichtefunktion der entsprechenden Zufallsvariablen angegeben.

Kenngrösse	Stochastisches-WS-System	Deterministisches WS-System
Unabhängige Variablen		
Mittlere Zwischenankunftszeit	$\overline{t} = \frac{1}{\lambda}$	
Mittlere Ankunftsrate	λ	A
Verteilungsfunktion	$PDF: \quad a(t) = 1 - e^{-\lambda t}$	
Dichtefunktion	$pdf: \quad A(t) = \lambda \cdot e^{-\lambda t}$	
Mittlere Bedienzeit	$\overline{x} = \frac{1}{\mu}$	
Mittlere Bedienrate	μ	D
Verteilungsfunktion	$PDF: \quad b(t) = 1 - e^{-\mu t}$	
Dichtefunktion	$pdf: \quad B(t) = \mu \cdot e^{-\mu t}$	
Abhängige Variablen		
Systemkenngrössen		
Wartezeit	$W = \frac{\rho/\mu}{1-\rho}$	W
Reaktionszeit	$R = \frac{1/\mu}{1-\rho}$	R
Lastkenngrössen		
Auslastung	$\rho = \frac{\lambda}{\mu}$	U
Aufträge im System	$\overline{N} = \frac{\rho}{1-\rho}$	N
Aufträge in der Warteschlange	$\overline{N_q} = \overline{N} - \rho$	N_q
Aufträge beim Bediener	$\overline{N_s} = \overline{N} - \overline{N_q} = \rho$	N_s

Tabelle 2.1: Kenngrößen in $M/M/1$ und in deterministischen Systemen

Ein WS-System mit exponentieller Verteilungsfunktion der Zwischenankunftszeit und der Bedienzeit und einem Bediener wird als $M/M/1$ bezeichnet. Eine exponentielle Verteilungsfunktion wird durch ihren Mittelwert charakterisiert. In diesem Fall ist $1/\lambda$ die mittlere Zwischenankunftszeit und $1/\mu$ die mittlere Bedienzeit. Die Kenngrößen eines $M/M/1$ Systems und die äquivalenten Kenngrößen in deterministischen Systemen zeigt Tabelle 2.1 [Kle75]:

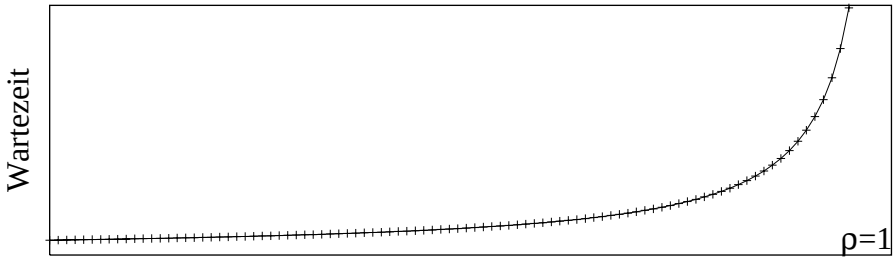


Abbildung 2.3: Wartezeit als Funktion der Auslastung

Je näher ρ zu 1 ist, desto größer ist die Warteschlange. Mit anderen Worten, je näher die Ankunftsrate zu der Bedienrate ist, desto größer wird die mittlere Wartezeit sein. Der Faktor $1/(1-\rho)$ sagt aus, dass die Wartezeit in der Nähe von $p \approx 1$ exponentiell steigt, d.h die Kosten einer hohen Auslastung wirken sich in der längeren Wartezeit aus (siehe Abbildung 2.3).

Abbildung 2.4 zeigt die Reaktionszeit in Abhängigkeit der Last. Um eine Auswahl zwischen den Lastniveaus zu treffen, wird eine zusammengesetzte Kenngröße, die Power [Kle79], angewendet. Die Power ist definiert als das Verhältnis aus Durchsatz und Reaktionszeit:

$$P = \frac{D^{\alpha}}{R}$$

Mit dem Exponent α kann entweder der Durchsatz oder die Reaktionszeit priorisiert werden. In der Regel wird $\alpha = 1$ genommen. Die maximale Power ist auch der *optimale* Arbeitspunkt, denn gesucht ist ein hoher Durchsatz mit möglichst kurzer Reaktionszeit. Die Power wird mit einer Warteschlangenlänge von eins maximiert [Kle76].

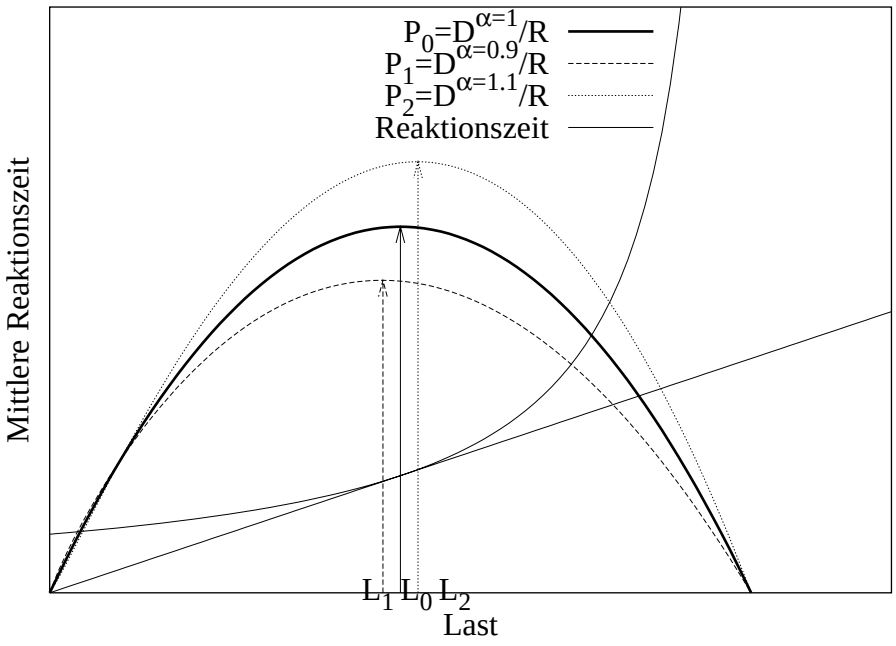


Abbildung 2.4: Reaktionszeit, Power und Last

In Abbildung 2.4 ist L_0 die Last, mit der die Power mit $\alpha = 1$ maximiert wird. Die Ursprungsgerade tangiert die Reaktionszeitfunktion an L_0. Für alle L_i mit $L_i \ll L_0$ sind die Reaktionszeiten viel kürzer als in L_0, aber auch die Durchsätze sind viel geringer, denn es gilt $A \ll D^{\alpha=1}$. Im Gegensatz dazu sind für alle L_i mit $L_i \gg L_0$, die Reaktionszeiten und Durchsätze viel größer als in L_0, denn A ist in der Nähe zu $D^{\alpha=1}$. Die Punkte L_1 bzw. L_2 sind außerdem die Lastniveaus zur Maximierung der Power mit $\alpha = 0.9$ bzw. $\alpha = 1.1$. Je größer α ist, desto größer wird die optimale Last.

Um ein stochastisches WS-System zu definieren, wird die Notation $A/B/m$ angewendet, wo A und B die Verteilungen der Zwischenankunftszeit bzw. der Bedienzeit darstellen, und m die Anzahl von Bedienern ist. Der allgemeine Fall wird als $G/G/m$ bezeichnet, nämlich die Verteilungsfunktionen können beliebig sein. Die ersten zwei Lastkenngrößen der Tabelle 2.1 gelten auch in allen $G/G/m$ WS-Systemen. Darüber hinaus wird die Wartezeit in allen WS-System vom Faktor $(1-\rho)$ bestimmt.

2.2 Warteschlangennetze (WS-Netze)

Ein Warteschlangennetz (WS-Netz) besteht aus mehreren WS-Systemen, die miteinander gekoppelt sind. Abhängig von der Art der Last können offene und geschlossene WS-Netzen unterschieden werden [Bol89, HZ95].

Offenes WS-Netz. Die Anzahl der Aufträge N ist nicht konstant, denn sie werden von außen aufgenommen und wieder nach außen abgegeben. Gegeben ist die Ankunftsrate λ_i in jedem WS-System. Dadurch wird die Last bestimmt. Das System wird durch Folgendes charakterisiert (siehe Abbildung 2.5):

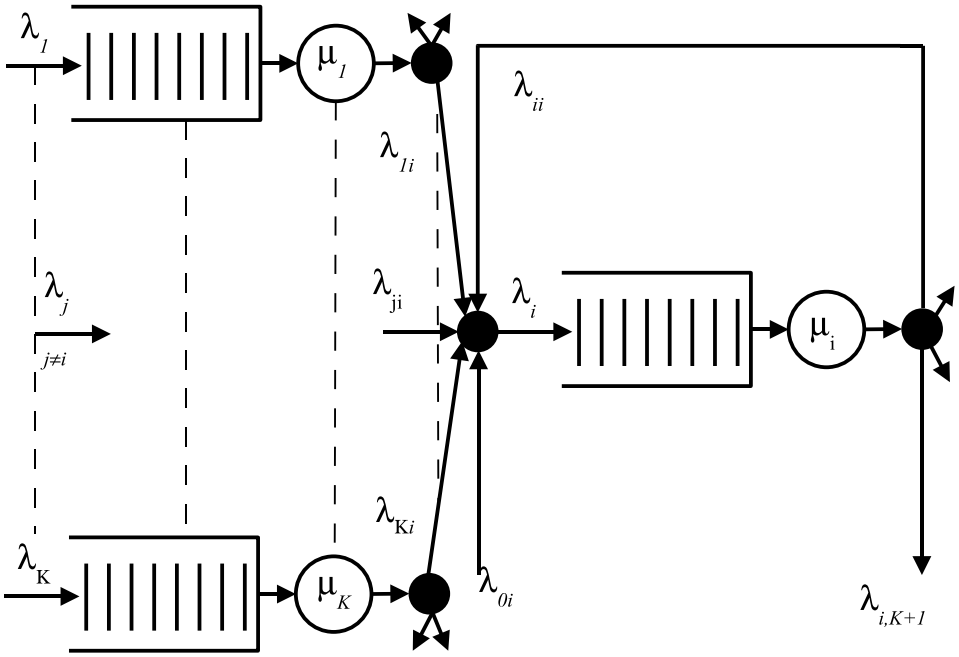


Abbildung 2.5: Offenes WS-Netz [ZL98])

mit

K	Anzahl der Knoten
N	Anzahl der Aufträge
m_i	Anzahl der Bediener
μ_i	Bedienrate des Bedieners i
λ_i	Ankunftsrate beim Knoten i
λ_{0i}	Ankunftsrate aus externen Quellen
$\lambda_{i,K+1}$	Ankunftsrate zu externen Senken
p_{ij}	Übergangswahrscheinlichkeit vom Knoten i zum Knoten j
k_i	Anzahl der Aufträge im Knoten i
$\underline{k}$,	Zustand des WS-Netzes, $\underline{k} = (k_1 ... k_i ... k_K)$
$p(k_1 ... k_i ... k_K)$	Wahrscheinlichkeit, dass das WS-Netz im Zustand $\underline{k}$ ist
ρ_i	Auslastung des Knotens i

Es gilt dann:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{K} \lambda_j \cdot p_{ij} \qquad \text{Eingangsfluss}$$

$$\lambda_{ij} = \lambda_{i,K+1} + \sum_{j=1}^{K} \lambda_j \cdot p_{ij} \qquad \text{Ausgangsfluss}$$

$$\rho_i = \frac{\lambda_i}{\mu_i}$$

$$p_{i,K+1} = 1 - \sum_{j=1}^{K} p_{ij}$$

Die Wahrscheinlichkeit, dass das WS-Netz im Zustand $p(k_1...k_i...k_K)$ ist, wird durch das Produkt der Zustandswahrscheinlichkeiten der einzelnen Knoten i gegeben [Jac57]:

$$p(k_1, ...k_i...k_K,) = \prod_{i=1}^{K} p_i(k_i) \quad \forall \underline{k} \tag{2.12}$$

Geschlossenes WS-Netz. Gegeben ist die Anzahl der Aufträge im System, d.h. im WS-Netz zirkulieren N Aufträge. Weder kommen neue Aufträgen von außen hinzu, noch verlassen Aufträge das WS-Netz. Dies bedeutet, dass es keine externen Quellen ($\lambda_{0i} = 0$) und Senken ($\lambda_{i,K+1} = 0$) gibt. Weiter gilt:

$$N = \sum_{i=1}^{K} k_i$$

$$\lambda_i = \sum \lambda_j \cdot p_{ij} \quad i = 1...K$$

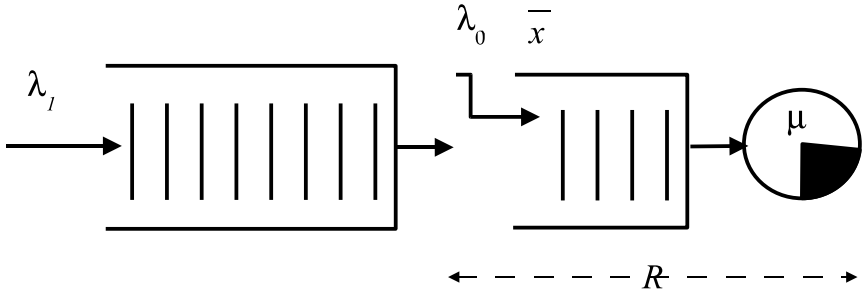
2.3 Hierarchischer Bediener

In [ZL98] wird ein hierarchisches Warteschlangen-Modell dargestellt. Dabei werden die Kenngrößen eines Systems als Funktion der Kenngrößen des darunter liegenden Systems ausgedrückt.

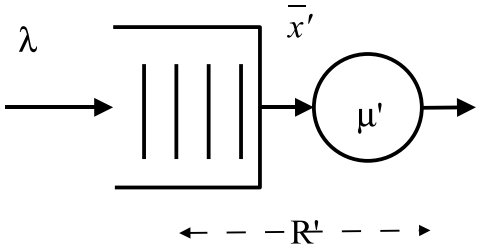
Aus Tabelle 2.1 werden die Bedienzeit $\overline{x}$, die Reaktionszeit R und die Anzahl der Aufträge $\overline{N}$ für einen Bediener der Schicht (n) genommen. Es ist nahe liegend, dass die Bedienzeit der Schicht (n) durch die Reaktionszeit der Schicht (n-1) bestimmt wird. Der Dehnfaktor $DF = 1/(1 - \rho)$ stellt den Anteil von μ der Schicht (n-1) dar, der für die Schicht (n) nicht zur Verfügung steht. Sei $\rho' = \lambda/\mu'$, dann gilt Folgendes für den Bediener der darauf liegenden Schicht:

$$\overline{x'} = R = DF \cdot \overline{x}$$
$$\mu' = \frac{1}{\overline{x'}} = \frac{1}{DF \cdot \overline{x}}$$
$$R' = \frac{1}{1-\rho'} \cdot \overline{x'} = \frac{DF}{1 - DF \cdot \rho} \cdot \overline{x} \tag{2.13}$$

Die Reaktionszeit für zwei Systeme wird in Abbildungen 2.6 und 2.7 dargestellt.



(a) Schicht (n-1)



(b) Schicht (n)

Abbildung 2.6: Hierarchischer Bediener (a) Ersatzschaltbild, (b) Nächste-Schicht-Bediener [ZL98]

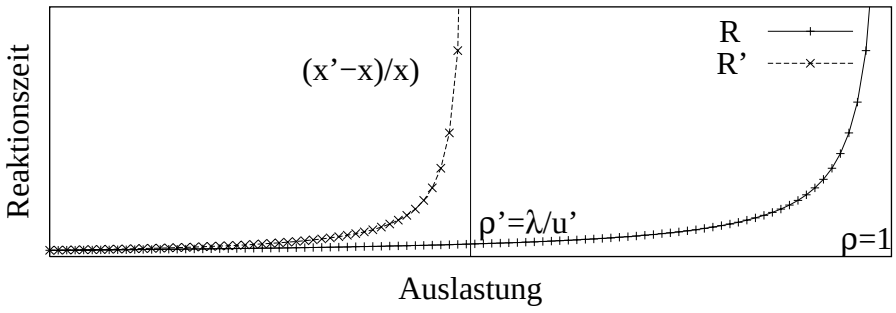


Abbildung 2.7: Reaktionszeit in hierarchischen Modellen

Aus 2.3, 2.6 und 2.13 geht hervor, dass die Reaktionszeit als Funktion des Durchsatzes der darunter liegenden Schicht ist:

$$R' = \frac{1}{1 - 2 \cdot \frac{D}{D_{max}}} \cdot \frac{1}{D_{max}}$$

Der maximale Durchsatz ist dann:

$$D' = N' \cdot \frac{1}{1 - 2 \cdot \frac{D}{D_{max}}} \cdot \frac{1}{D_{max}} \tag{2.14}$$

2.4 Leistungsmodellierung von Rechnernetzen

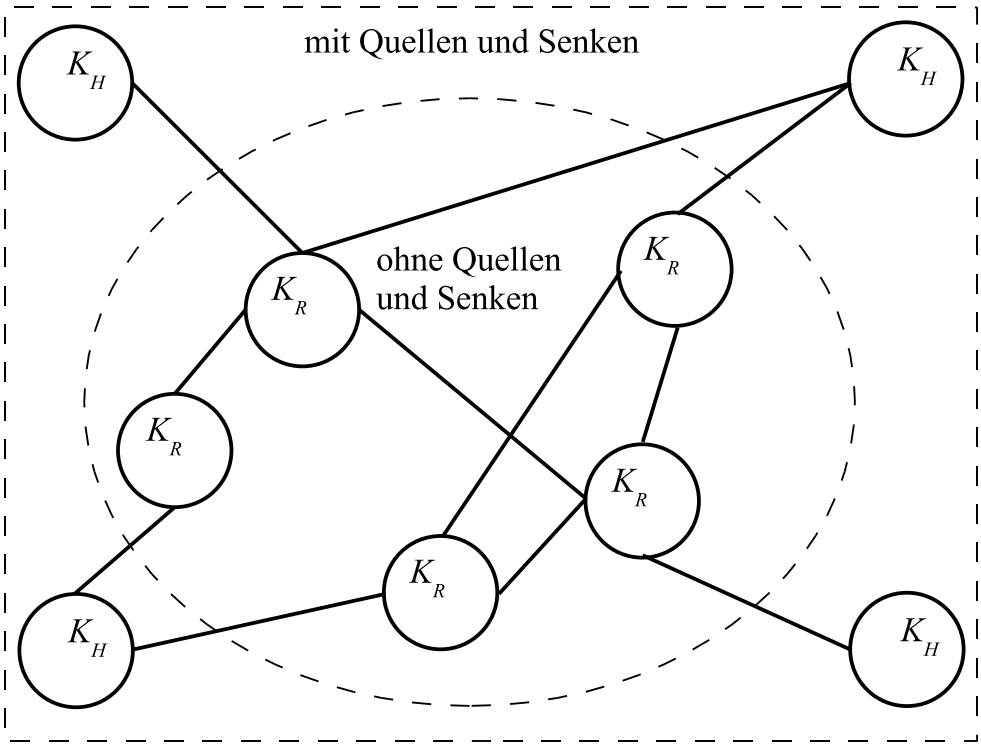


Abbildung 2.8: Rechnernetz

2.4.1 Definition

Rechnernetze bestehen aus Knoten, die durch Kanäle oder Leitungen miteinander gekoppelt sind (siehe Abbildung 2.8). Knoten der Netze sind Hosts (K_H, Endsysteme) und Router (K_R, Zwischensysteme). In paketvermittelten Rechnernetzen werden Pakete zwischen den Knoten transportiert. Die Router leiten die Pakete an die nächsten Router oder Hosts weiter, oder speichern sie in Warteschlangen, wenn die Leitung besetzt ist.

2.4.2 Grundmodelle und Kenngrößen

Zur Modellierung von Rechnernetzen werden WS-Netze angewendet, da u.a. [Bol89]:

- Rechnernetze bestehen aus mehreren unabhängigen Bedienern (Rechnern). Die Last wird durch die Datenübertragungen verursacht.
- Eine Datenübertragung beansprucht die Bediener sequentiell.
- Ein Bediener kann gleichzeitig von mehreren Datenübertragungen in Anspruch genommen werden.

Die Knoten der Rechnernetze können durch WS-Systeme modelliert werden. Modelle für die Übertragungszeit in der Leitung werden im Folgenden dargestellt. Die Übertragungsstrecke kann mit Hilfe der Übertragungsrate modelliert werden. Diese Modelle sind von Interesse, da sie die Rahmenbedingungen der Leistungskenngrößen zeigen unter deren Berücksichtigung ein Verfahren zur Überlastkontrolle entwickelt wird.

Übertragungsrate. Die Übertragungsrate B ist die gesendete Datenmenge b pro Zeit t.

$$B = \frac{b}{t} \tag{2.15}$$

Die Datenmenge kann je nach Schicht in Bits, Frames, Pakete, usw. angegeben werden. Auf der Schicht 2 wird der Begriff Datenrate angewendet.

Datenrate. Die Datenrate B wird in Bits/s angegeben, und stellt die Leitungskapazität dar, und ist daher vom Übertragungsmedium abhängig. Es ist nahe liegen, dass zwei direkt verbundene Knoten nicht schneller als die Datenrate dieser Leitung kommunizieren können. Die Datenrate stellt somit eine obere Schranke für den jeweiligen Durchsatz dar.

Signalausbreitungsgeschwindigkeit. Die Signalausbreitungsgeschwindigkeit v stellt die Ausbreitung einer Störung entlang der Leitung der Länge E dar.

$$v = \frac{E}{t} \tag{2.16}$$

Sie ist vom Übertragungsmedium abhängig. Ein Verkürzungsfaktor gibt die Abweichung von der Lichtgeschwindigkeit an. In Glasfaser ist der Verkürzungsfaktor 0.67, in verdrilltem Kabel 0.6 und in Koaxialkabel 0.77. Somit beträgt die Signalausbreitungsgeschwindigkeit 200000 km/s in Glasfaser, 180000 km/s in verdrilltem Kabel und 231000 km/s in Koaxialkabel [Sie03].

2.4.3 Modellierung von Übertragungsstrecken

Die Zeit R, die eine Datenmenge der Größe b zur Übertragung über einer Leitung der Länge E mit der Datenrate B benötigt, wird üblicherweise nach folgender Formel berechnet ([Ste95, Zit00]):

$$R = \frac{b}{B} + \frac{E}{v} \tag{2.17}$$

Die Übertragungsstrecke lässt sich dabei als Serienschaltung zweier Bediener (siehe Abbildung 2.9) modellieren [Zor03]:

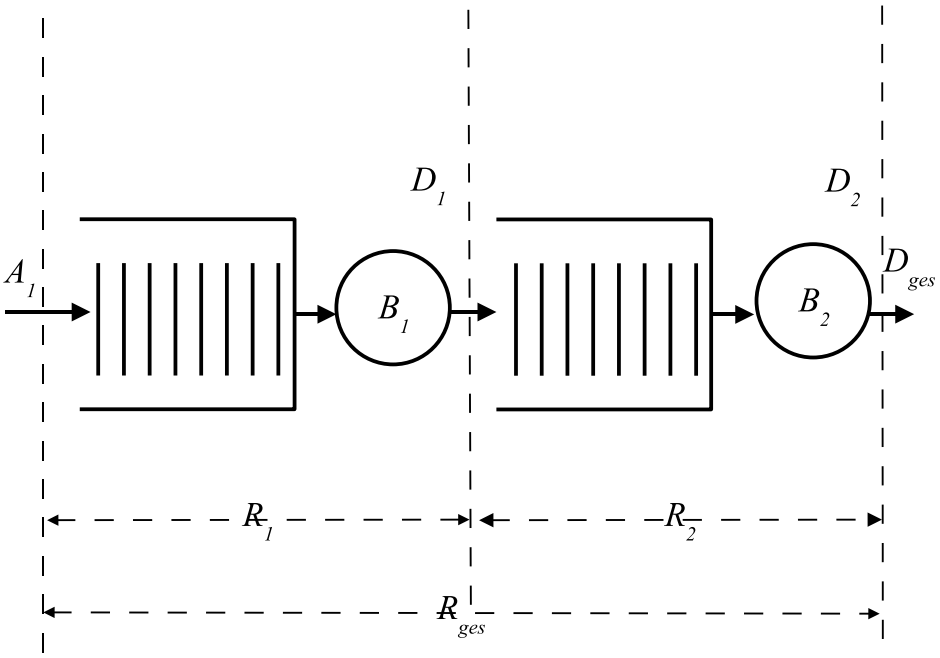


Abbildung 2.9: Modellierung einer Übertragungsstrecke - 2 Bediener

mit

X_1	Bedienzeit des Senders in s mit $X_1 = b/B$
b	Größe des zu übertragenden Pakets in bits
B	Senderate des Senders in bits/s
X_2	Bedienzeit des Transportsystems in s mit $X_2 = E/v$
E	Länge der Übertragungsstrecke in km
v	Geschwindigkeit des Transportsystems in km/s

Woraus sich die Gesamtzeit für das Senden und Übertragung eines Pakets im allgemeinen Fall ergibt:

$$\begin{aligned} R_{ges} &= R_1 + R_2 \\ &= (W_1 + X_1) + (W_2 + X_2) \end{aligned}$$

mit W_1 und W_2 als Wartezeiten in den Warteschlangen der Bediener 1 und 2.

Bei der Ermittlung der übrigen Leistungskenngrößen sind folgende Überlegungen anzustellen: Im Sonderfall eines abgestimmten Systems im Gleichgewicht, bei welchem die Ankunftsrate von außen gleich der Bedienrate des Senders ist, d.h. $A_1 = B_1$ werden die ankommenden Pakete ohne Wartezeit direkt bedient, so dass $W_1 = 0$ gilt.

An der Schnittstelle zwischen Sender und Transportsystem gilt im allgemeinen Fall:

$$D_1 = A_2 \neq B_2$$

Dies sei an folgendem Beispiel einer Satellitenübertragung über eine T1-Leitung veranschaulicht mit:

$$X_1 = \frac{8 \text{ bit/Byte} \cdot 1500 \text{ Byte}}{1.544 \text{ Mb/s}} = 7.772 \text{ ms}$$
$$X_2 = \frac{36\,000 \text{ km}}{300\,000 \text{ km/s}} = 120 \text{ ms}$$

voraus sich als Bedienraten für den Sender und für den Transportsystem mittels elektromagnetischen Wellen ergeben:

$$B_1 = 128.667 \text{ Pakete/s}$$
$$B_2 = 8.333 \text{ Pakete/s}$$

Was einen Unterschied der Bedienrate um den Faktor $B_1/B_2 = 15.44$ entsprechen würde und in Widerspruch zur zuvor postulierten Gleichgewichtbedingung stünde. Der Widerspruch löst sich auf, wenn man das Transportsystem als Infinite-Server (IS) modelliert, wie es in Abbildung 2.10 dargestellt ist.

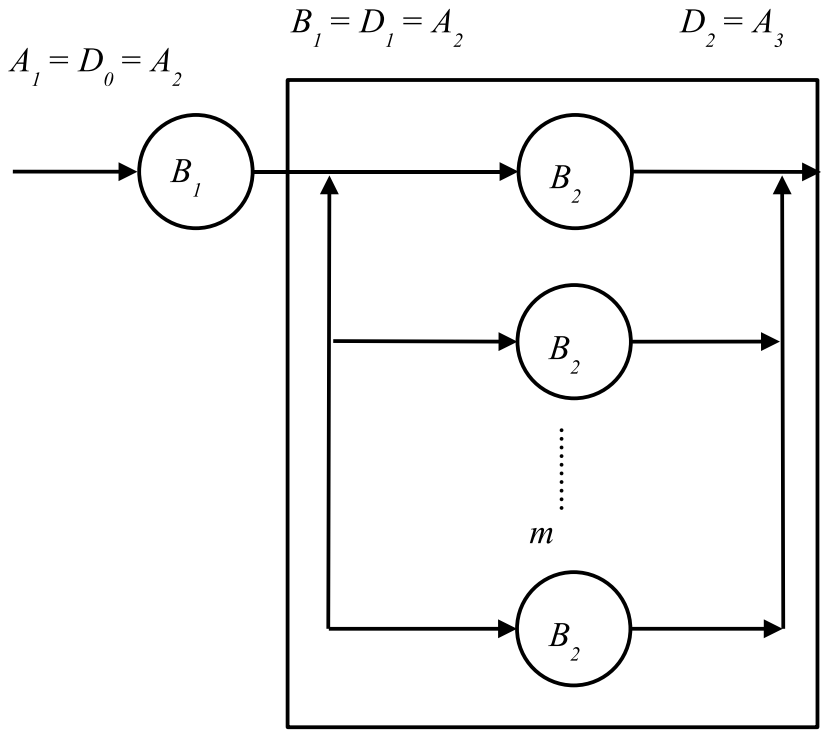


Abbildung 2.10: Modellierung einer Übertragungsstrecke - Infinite-Server

Hieraus ergibt sich:

$$D_{ges} = A_1 = B_1 = m_2 \cdot B_2 = D_2$$
$$= 128.667 \text{ Pakete/s}$$
$$N = 16.44 \text{ Pakete}$$

wovon sich befinden:

N_1 1 Paket im Sender
N_2 15.44 Pakete im Transportsystem

Andere Transportsysteme, welche nicht auf den Eigenschaften elektromagnetischer Wellen berufen, wie. z.B. solche für mechanischen Transport (durch LKWs, usw.) stellen keine Infinite-Server dar und erfordern die Modellierung mittels Warteschlangen-Bedienern.

2.5 Hierarchische Modelle

Rechnernetze sind hierarchisch aufgebaut, daher begrenzen die Eigenschaften einer Schicht das Verhalten von allen darauf liegenden Schichten: Leistungskenngrößen der Netzschicht wirken sich in der Internet-, Transport- und Applikationsschicht aus.

2.5.1 Round-Trip-Time (RTT) einer Transportverbindung

Abbildung 2.11 zeigt ein Modell einer hierarchischen Datenübertragung in einem paketvermittelten Netz. Zwischen den Knoten K_S und K_E besteht eine Verbindung, deren Weg Verbindungspfad heißt. Dazwischen liegen n Router, die durch Kanäle verbunden sind. Die Datenübertragung in der Transportschicht oder Applikationsschicht ist eine Akteuren-Kommunikation, auch Ende-zu- Ende-Kommunikation genannt.

Ist die Verbindung unidirektional und fließt die Datenübertragung von K_S nach K_E, dann ist K_S der Quell-Host und K_E der Ziel-Host. Während der Verbindung sendet Host K_S n Pakete der Länge b zu Host K_E. Für jedes gesendete Paket kann eine Antwortzeit gemessen werden, die Round-Trip-Time (RTT) genannt wird. RTT ist die Zeit, die ein Paket von K_S nach K_E benötigt, zusätzlich der Zeit, die die Quittung von K_E nach K_S braucht (siehe Abbildung 2.12).

Für symmetrische Pfade lässt sich die RTT folgendermaßen ausdrucken:

$$rtt = 2 \times \left[\sum_{i=1}^{n} \left(X_i + W_i \right) + \sum_{i=0}^{n+1} \left(M_i + Pd_i \right) \right] \tag{2.18}$$

mit

X_i Verarbeitungszeit der Pakete im Knoten i (Router oder Host)
W_i Wartezeit im Knoten i (Router oder Host)
M_i Paketierungszeit b/B zum Knoten $i + 1$ (Router oder Host)
Pd_i Signallaufzeit E/v zum Knoten $i + 1$ (Router oder Host)

Die Verarbeitungszeit der Pakete X_i in einem Knoten des Pfades hängt von den Leistungskenngrößen des Rechners und von der Komplexität des Protokolls ab. Operationen wie Fehlerkorrektur und

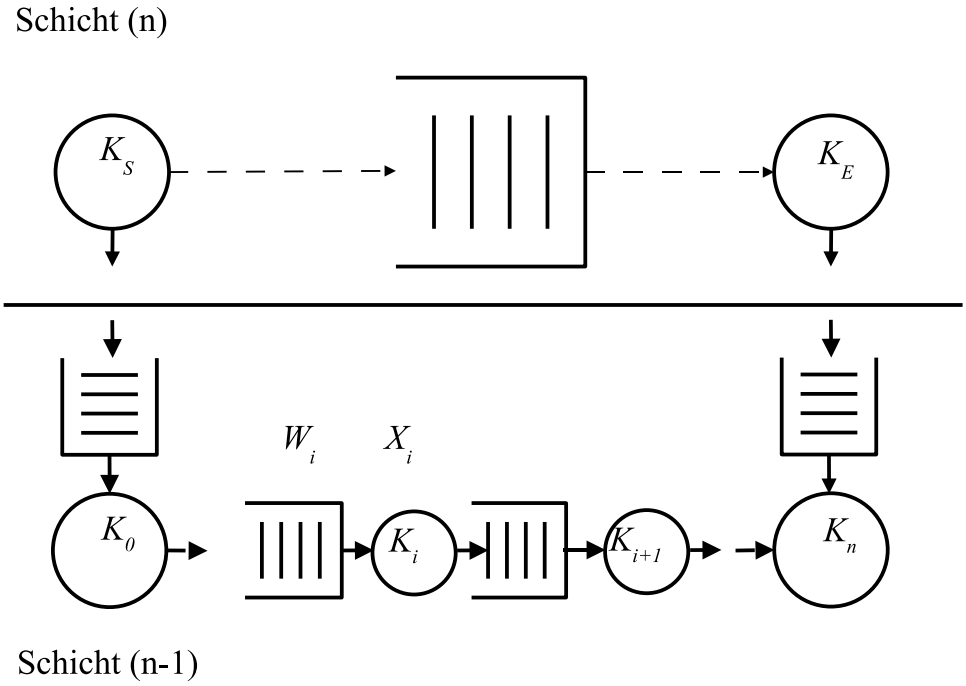


Abbildung 2.11: Modellierung einer Transportverbindung

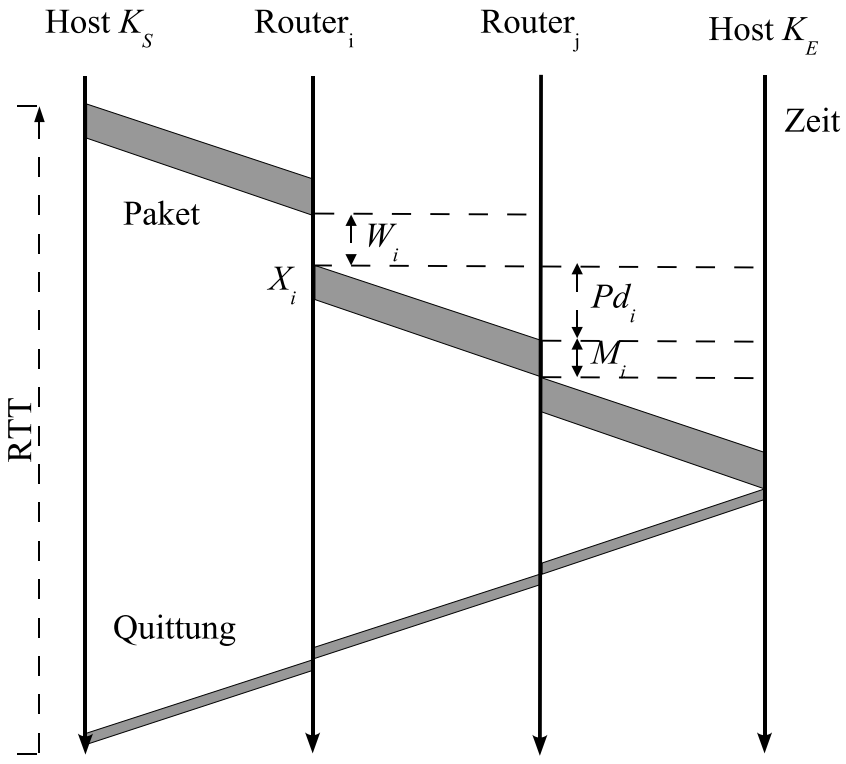


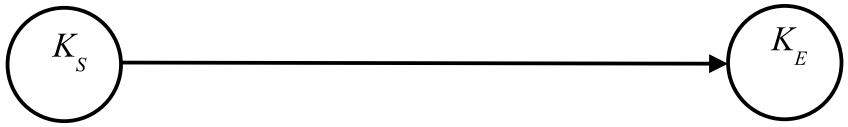
Abbildung 2.12: Zeitdiagramm der Zusammensetzung der RTT

-behebung erhöhen die Komplexität, aber auch die Zuverlässigkeit eines Protokolls. Als Beispiel betrachtet man UDP und TCP: UDP ist schneller aber nicht zuverlässiger als TCP, da UDP u.a. nicht verbindungsorientiert ist, und somit keine Kontrolle von Paketverlust hat. Als Folge muss diese Funktionalität auf dem darauf liegenden Protokoll verlagert werden.

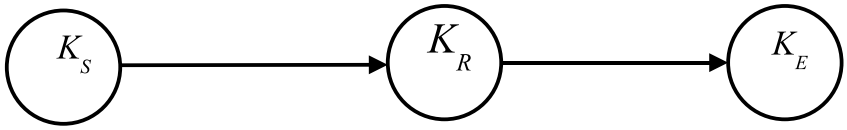
Die Wartezeit W_i in den Routern hängt von der jeweiligen Warteschlangenlänge ab. Die Warteschlangenlänge ist die Speicherkapazität in Bytes, die als Puffergröße bezeichnet wird.

Der Term $M_i + Pd_i$ der Gleichung 2.18 kann während einer Verbindung ohne Rerouting als konstant betrachtet werden, da er die Übertragungszeit des Pakets ist. Der Term $X_i + W_i$ von 2.18 entspricht der Reaktionszeit eines WS-Systems (Gleichung 2.1). Daher wird die minimale RTT erreicht, wenn die Warteschlangen klein bzw. leer sind. Es ist zu erwarten, dass die Variation der RTT die Variationen der Wartezeit auf den Routern des Verbindungspfades entspricht. In den Routern sammeln sich die Pakete von mehreren Verbindungen, daher spiegelt die Variation der RTT nicht nur die Fluktuationen der eigenen Senderate, sondern die Dynamik des gesamten Verkehrs wieder. Aus 2.18 ist es zu erwarten, dass die RTT proportional zur Auslastung variiert: ($\sigma_{RTT} \propto 1/(1-\rho)$) [Com95].

2.6 Netze mit Datenverlust



(a) Direkt verbunden



(b) Indirekt verbunden

Abbildung 2.13: Netze mit Paketverlust (a) direkt verbunden (b) indirekt verbunden

Bis hier hin sind lediglich Netze ohne Datenverlust betrachtet worden. Dies ist nur eine ideale Situation, da in der Realität Paketverlust in der Datenübertragung auftritt. Als Folge sinkt der Durchsatz, da diese Einheiten erneut übertragen werden müssen. Datenverlust entsteht z.B. durch Ausfall eines Anschlusses (Netzschicht) oder unzureichende Puffergröße in den Routern oder in den Hosts (Internet- oder Transportschicht).

Abhängig von der Stelle, wo Paketverlust auftritt, wird Flusskontrolle oder Überlastkontrolle eingesetzt. Die Flusskontrolle sorgt dafür, dass ein schneller Sender einen langsamen Empfänger nicht überrollt. Paketverlust entsteht in den Hosts, wenn der Datenfluss zu groß für die Leistung des Prozessors bzw. für die Puffergröße ist. Die Abbildung 2.13 zeigt ein Modell des zu betrachtenden Systems bei der Fluss- bzw. Überlastkontrolle. In der Flusskontrolle werden die Knoten als direkt verbunden betrachtet und als indirekt verbunden in der Überlastkontrolle. Flusskontrolle darf nicht mit Überlastkontrolle verwechselt werden. Obwohl beide als Ziel haben, Paketverlust zu vermeiden, ist der Verlustsort im ersten Fall ein Host und im zweiten Fall ein Router. Überlastkontrolle wird im nächsten Kapitel ausführlich behandelt.

In diesen Netzen sind zwei Kenngrößen von Interesse: die Paketverlustrate und die Fairness.

Paketverlustrate. Die Paketverlustrate ist das Verhältnis zwischen Verlorene- b_P und Gesamtdaten b:

$$p = \frac{b_P}{b} \tag{2.19}$$

Fairness. Obwohl es sehr viele und teilweise komplexe Definitionen von Fairness eines Systems gibt, siehe z.B. [JCH84], werden hier die Standardabweichung sowie der Variationskoeffizient angewendet. Sie spiegeln die Eigenschaften der Verteilungsfunktionen wieder und sind einfach zu ermitteln.

Kapitel 3

Überlastkontrolle - eine Übersicht

In diesem Kapitel werden die wesentlichen Prinzipien der Überlastkontrolle erläutert. Lastkontrolle beschäftigt sich mit der Behandlung von Netzüberlastung (*Congestion*). Die Ursachen und Folgen von Netzüberlastung werden diskutiert. Anschließend werden existierende Ansätze zur Überlastkontrolle näher untersucht.

3.1 Netzüberlastung

In paketvermittelten Rechnernetzen wird Netzüberlastung als der Zustand bezeichnet, in welchem als Folge hoher Auslastung der Ressourcen das Leistungsverhalten des Systems absinkt. Mögliche Ursachen sind:

- Hardware-Konfiguration. Große Unterschiede der Leistungskenngrößen führen zur Überlastung der Engpass-Komponente: Große Datenraten der Leitungen bringen keinen Vorteil, wenn die Prozessoren der Router nicht schnell genug sind. Ein anderes Beispiel: Eine Leitung mit einer geringeren Datenrate innerhalb eines Gigabit-Netzes wird häufig überlastet sein.

- Systemparameter. Dazu zählen u.a. Puffergrößen und Senderaten. Die Puffergrößen können anhand von Modellen ermittelt werden, die für den stationären Zustand der Netze gültig sind. Ändert sich das Verkehrsprofil im Laufe der Zeit, kann es vorkommen, dass die Puffergrößen falsch dimensioniert sind.

- Last. Zu Spitzenzeiten kann die Last ein Vielfaches dessen betragen, wozu das Netz ausgelegt ist. Ein anderes Problem ist der so genannte burst-artige Verkehr. Er kommt vor, wenn mehrere Pakete einer Verbindung innerhalb kurzer Zeit an einem Router ankommen. Diese Verbindungen haben eine große Pfadkapazität (siehe Abschnitt 3.3.3). Hier ist die übertragene Datenmenge pro RTT eher klein im Verhältnis zur Pfadkapazität. So entstehen in den Routern alternierende Perioden mit mal niedriger und mal hoher Auslastung, die als inaktive bzw. aktive Periode genannt werden. Diese kurzen Perioden mit hoher Auslastung stellen transiente Netzüberlastungen dar.

Die zentrale Frage lautet: Wie kann Netzüberlastung vermieden werden? Dazu ist es erforderlich, dass die Netzüberlastung überhaupt erst einmal erkannt werden muss. Sie kann aus unterschiedlichen

Sichten betrachtet werden:

- Aus der IP-Sicht: Die Wartezeit steigt, da die Warteschlangen länger werden. Bei Pufferüberlauf werden anschließend Pakete verworfen. Dies bedeutet, dass ein Router anhand der Auslastung der Warteschlangen eine sich bildende Netzüberlastung erkennen kann.

- Auf der TCP-Sicht: Die RTT steigt und der Durchsatz sinkt kontinuierlich. Paketverlust wird beobachtet und Rücksetzmechanismen müssen eingesetzt werden. Hier gilt: eine steigende RTT kann bedeuten, dass sich eine Netzüberlastung bildet; Paketverluste treten auf, wenn sich die Netzüberlastung bereits gebildet hat. Wirken Rücksetzmechanismen nicht, dann werden Verbindungen abgebrochen.

Netzüberlastung wird vermieden, indem die Stabilitätsbedingung der Warteschlangentheorie (Gleichung 2.5) eingehalten wird, d.h. die Zwischenankunftsrate darf nicht größer als die Bedienrate sein.

Eine unkontrollierte Netzüberlastung führt letztendlich zum *Congestion-Collapse* [Jac88]. Im Congestion-Collapse kann sich das System nicht mehr erholen und der gesamte Durchsatz geht gegen Null. Diese Situation kann entstehen, wenn mehrere Datenübertragungen mit hohen Senderaten über die gleichen Router gehen. Die Router werden früher oder später einige Pakete verwerfen müssen. Dadurch wird jedoch die Last nicht geringer, da diese Pakete vom Sender[1] erneut übertragen werden müssen. Die Router erwarten, dass die Sender auf den Paketverlust reagieren, und ihre Senderaten absenken. Sind keine Kontrollmechanismen vorhanden, geschieht dies nicht. Sind die Kontrollmechanismen ungeeignet, werden die Senderaten langsamer als erforderlich abgesenkt. Die Sender versuchen, verlorene Pakete erneut zu senden, solange es ihre Rücksetzmechanismen erlauben. Bleiben die Ankunftsraten der Router höher als die Bedienraten, werden die Sender ihre Verbindungen im Laufe der Zeit abbrechen müssen. Die Folgen eines solchen Congestion-Collapse sind dann [Jac88, Lai02]:

- Verschwendung von Ressourcen an unzustellbare Pakete.
- Zunahme von Kontrollpaketen im Netz.
- Paketverlust in den Routern.
- Retransmission in der Transportschicht.
- Verbindungsabbruch in der Applikationsschicht

3.2 Überlastkontrolle

Die Überlastkontrolle basiert auf Protokollmechanismen zum Vermeiden und zur Behandlung von Netzüberlastung. Darüber hinaus ist es wichtig, die Auslastung der Ressourcen zu optimieren und hohen Durchsatz und Fairness für konkurrierende Sender zu garantieren. In dieser Aufgabe sind alle Hosts und Router und die dort vorhandenen Prozesse wie Routing, Scheduling u.a.m. involviert. Die Anforderungen an die Überlastkontrolle sind:

- Stabilität in Bezug auf folgende Eigenschaften der Rechnernetzen:
    - Datenrate kann von 9.6 Kb/s bis zu 10 Tb/s betragen

[1] Der Begriff Sender wird hier als Synonym für eine Transportinstanz angewendet. Eine Transportinstanz bezeichnet einen Endpunkt einer logischen Verbindung, d.h. die Kommunikation zwischen zwei Prozessen in den Hosts. In der englischsprachigen Literatur zur Überlastkontrolle wird häufig *flow*, *source* oder *user* angewendet.

- RTT bewegt sich zwischen einigen Mikrosekunden bis zu Sekunden je nach LAN, WAN, Satellit, usw.
- Paketverlust kann zwischen 0-100 % variieren
- Änderung des Pfades während der Übertragung
- Unterschiedliche Protokolle
- Unterschiedliche Applikationen (Best-Effort-, Echtzeit-Verkehr)

- Verteilte Kontrolle innerhalb mehrerer zu koordinierender Akteure.

3.3 Klassifizierung der Überlastkontrolle

Um die Eigenschaften der Überlastkontrolle zu verdeutlichen, werden hier vier Kriterien zur Klassifizierung untersucht: der Lösungsansatz, der Einsatzort im Netz, der Sendemechanismus des Transportprotokolls und die Auslastung des Netzes.

3.3.1 Nach dem Lösungsansatz

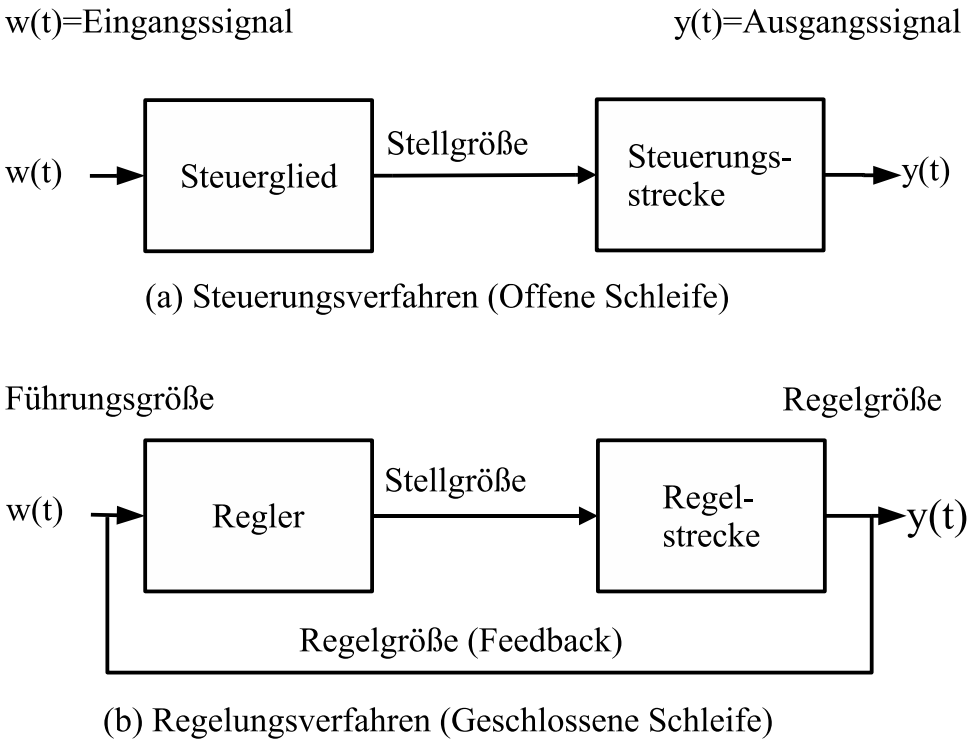


Abbildung 3.1: Steuerungs- und Regelungsverfahren

In Anlehnung an die Steuerungs- und Regelungstheorie wird in [YR95] eine Taxonomie zur Klassifizierung der Überlastkontrolle nach dem Lösungsansatz eingeführt. Im Wesentlichen gibt es zwei Möglichkeiten: Kontrolle ohne und mit Rückkopplung (Feedback). Sie werden als Steuerungs- (offene Schleife) und Regelungs- (geschlossene Schleife) Verfahren genannt (siehe Abbildung 3.1). Die vollständige Klassifizierung zeigt Abbildung 3.2.

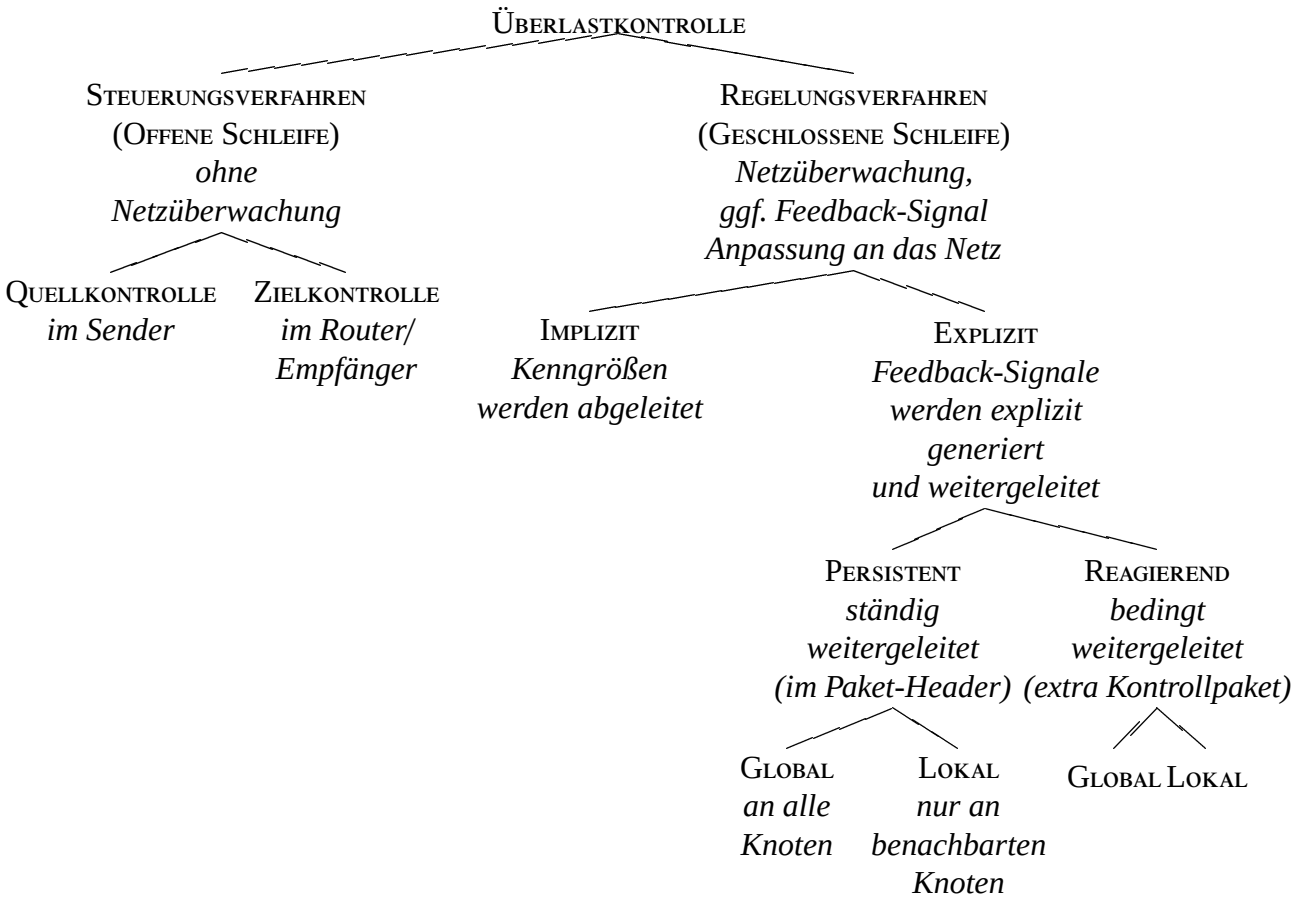


Abbildung 3.2: Klassifizierung der Ansätze zur Überlastkontrolle nach dem Lösungsansatz in [YR95]

Steuerungsverfahren versuchen, die Netzüberlastung durch geeigneten Komponenten- und Protokollentwurf zu bewältigen. Der Netzzustand wird nicht überwacht. Ihre Kontrolle setzt nur auf die Kenntnis der Komponente über sich selbst, z.B. die Datenrate oder die Puffergröße. Zur Stabilisierung des Verkehrs wird u.a. Zugangskontrolle eingesetzt oder es werden Pakete verworfen. Gesucht sind dabei geeignete Entscheidungen bezüglich Zugangskontrolle, Scheduling und/oder dem Verwerfen von Paketen.

Steuerungsverfahren lassen sich in Quell- und Zielkontrolle unterteilen, je nachdem, wo die Kontrolle stattfindet. Im ersten Fall erfolgt die Kontrolle an der Quelle des Datenflusses, z.B. beim Sender. Der Zweck ist, dass unter bestimmen Bedingungen keine Pakete mehr zugelassen werden, für deren Übertragung ohne Überlastungs-Risiko das Netz nicht garantieren kann. Die Kontrollinstanz bei der Zielkontrolle ist der Ziel-Host oder ein Router, wobei die Last auf den Verbindungspfaden kontrolliert wird.

Regelungsverfahren verfolgen das Ziel, die Leistungskenngrößen des Netzes dynamisch zu bewerten, entstehende Netzüberlastung festzustellen, und sie zu kontrollieren. Dafür sind folgende Schritte erforderlich:

(i) Überwachung der Last im Netz durch Beobachtung geeigneter Größen wie Warteschlangenlängen, Auslastung, Paketverlustrate, RTT.
(ii) Generierung und Weitergabe dieser Größen als Feedback-Signale an die Regler. Die Feedback-Signale können implizit oder explizit sein.
(iii) Anpassung der Komponenten an die Last.

Verfahren mit impliziten Feedback-Signalen setzen voraus, dass die Regler in der Lage sind, die Netzauslastung zu erkennen. Der Ablauf eines Timers, die Quittungen, die Änderung der RTT oder des Durchsatzes werden häufig als implizite Signale angewendet.

Explizite Feedback-Signale können als zusätzliche Information in den Paket-Headern oder als eigenständige Kontrollpakete weitergeleitet werden. Hier entstehen Kosten durch die Generierung dieser Information im Messsystem und ihre Bearbeitung im Regler.

Für die Generierung und Weiterleitung expliziter Feedback-Signale muss festgesetzt werden: 1) wie häufig: z.B. in jedem Paket, nur bei Netzüberlastung, 2) an wenn: z.B. von Router zu Router, von Router zu Host, von Host zu Host und 3) wie: z.B. im Paket-Header, als Kontrollpaket Feedback-Signale gesendet werden.

Nach der Häufigkeit des Sendens lassen sich Signale in persistente und reagierende unterteilen. Persistente Signale werden konstant und unabhängig vom Netzzustand weitergeleitet, d.h. Pakete mit Applikationsdaten tragen sie mit sich im Paket-Header oder Kontrollpakete werden periodisch gesendet. Reagierende Signale werden nur unter gewissen Umständen generiert, z.B. *die mittlere Warteschlangenlänge übersteigt eine Schwelle*. Dies bringt den Nachteil mit sich, dass, gerade wenn die Auslastung hoch ist, zusätzliche Pakete gesendet werden.

Nach der Flussrichtung können Signale in globale und lokale klassifiziert werden. Bei globalen Feedback-Signalen fließt die Information vom Host (Sender) zu Host (Empfänger). Lokale Feedback-Signale gehen nur an die benachbarten Knoten.

3.3.2 Nach dem Einsatzort im Netz

Mögliche Einsatzorte der Überlastkontrolle sind:

- Host: Die Kontrolle findet in den Transport- oder Applikationsprotokollen eines Hosts statt. Diese Verfahren werden Ende-zu-Ende-Überlastkontrolle (engl. End-to-End- (E2E) Congestion-Control) genannt und haben den Vorteil, dass sie einfach einsetzbar sind, da sie nur in den einzelnen Hosts implementiert werden müssen.
- Router: Die Kontrolle findet in den Routern des Netzes statt. Diese Verfahren werden als routerbasierte Überlastkontrolle (engl. Gateway Congestion-Control) bezeichnet. Bei vorhandener Netzüberlastung wird eine Aktion ausgelöst, damit die Sender ihre Senderate senken.
- Host und Router: Die Kontrolle findet auf der Netzschicht statt. Diese Verfahren werden als Hop-Überlastkontrolle (engl. Hop-Congestion-Control) bezeichnet und werden häufig als Virtual-Circuit realisiert.

Diese Möglichkeiten schließen sich nicht gegenseitig aus, denn ihre Aufgaben ergänzen sich. Obwohl sie unabhängig voneinander implementiert werden können, ist ein gewisses Maß an Koordination notwendig, wie anhand des Beispiels des Congestion-Collapse gezeigt wurde.

3.3.3 Nach dem Sendemechanismus des Transportprotokolls

Das Senden der Pakete kann durch zwei Mechanismen kontrolliert werden: Senderate oder Fenstergröße.

Senderatenbasierte Protokolle stellen die Anzahl n an zu sendenden Paketen der Paketgröße b während eines Zeitintervalls Δt ein. Nach Ablauf einer Periode Δt wird die Senderate sr_i überprüft und gegebenenfalls neu gesetzt.

$$sr_i = \frac{b \cdot n}{\Delta t} \tag{3.1}$$

Fensterbasierte Protokolle stellen zum Zeitpunkt t_i eine Anzahl n an zu sendenden Bytes oder Paketen ein, auch Fenster genannt. Das Fenster entspricht dann der Datenmenge, die sich auf der virtuellen Leitung befinden kann. Das Fenster wird auch Flusskontrollfenster, Flusssteuerungsfenster, oder Window (WND) genannt. Nach Auftreten eines bestimmten Ereignisses zum Zeitpunkt t_{i+1} wird das Fenster neu gesetzt. Dieses Ereignis kann z.B. eine Quittung oder der Ablauf eines Timers sein. Ist das Ereignis eine positive Quittung, kann eine RTT_{i+1} berechnet und gegebenenfalls das Fenster vergrößert werden. Ist das Ereignis der Ablauf eines Retransmission-Timers (RT) nach *rto* Zeiteinheiten, wird das Fenster verkleinert bzw. zurückgesetzt. Das Fenster zur Flusskontrolle, *rwnd*, wird ebenfalls berücksichtigt:

$$wnd_{i+1} = f(wnd_i, rtt_i, rto_i, rwnd_i ...) \tag{3.2}$$

Die Senderate ist somit keine Stellgröße. Sie kann nur indirekt und erst zum darauf folgenden Zeitpunkt t_{i+1} festgestellt werden.

$$sr_{i+1} = \frac{wnd_{i+1} - wnd_i}{rtt_{i+1} - rtt_i} \tag{3.3}$$

Welche Kriterien gibt es für die Einstellung der Senderate bzw. des Fensters? Der ideale Mechanismus sollte Flusskontrolle (Begrenzung durch den Sender/Empfänger) aber auch die Pfadkapazität einer Verbindung (Begrenzung durch das Netz) berücksichtigen.

Pfadkapazität einer Verbindung. Die Pfadkapazität [Ste95] einer Verbindung K, auch *Produkt Datenraten-Verzögerung*[2] genannt, ist die maximale im Transit gehaltene Datenmenge, d.h. die Anzahl der Bits, welche die virtuelle Leitung zwischen den Hosts A und B füllt. Sie ist von der kleinsten Datenrate B_{min} des Verbindungspfades und von der RTT abhängig:

$$K = B_{min} \cdot rtt \tag{3.4}$$

B_{min} wird als der Engpass des Verbindungspfades bezeichnet. Dies schließt nicht aus, dass ein anderer Faktor der Engpass der Datenübertragung sein kann, etwa ein langsamer Prozessor oder eine vom Kommunikationsprotokoll abhängige Zustandsvariable, usw.

Je größer die RTT, desto größer ist die Pfadkapazität. Das gleiche trifft für die Datenrate zu, wird die Datenrate erhöht, erhöht sich auch die Pfadkapazität.

In fensterbasierten Protokollen ist der theoretisch höchst mögliche Durchsatz erreicht, wenn das Fenster der Pfadkapazität entspricht, denn dann ist die virtuelle Leitung zu 100 % ausgelastet. Entsprechend ist das Ziel solcher Protokolle, das Fenster dynamisch an die sich ändernde RTT anzupassen. In der Tat ist es sehr schwierig, dieses Maximum zu halten. Erstens ändert sich die RTT ständig. Für lange Verbindungspfade kann dies dementsprechend lange dauern, bis die Information der Änderung der RTT den Sender erreicht. Zweitens gibt es Quittungen, auf die gewartet werden muss. Geht ein Paket in der Mitte des Fensters verloren, dann entstehen Lücken in dem zu übertragenden Datenfluss, die die Übertragung stoppen können.

Ist das Fenster kleiner als die Pfadkapazität, müssen Durchsatzeinbußen in Kauf genommen werden; ist es dagegen zu hoch, werden Paketverluste verursacht.

3.3.4 Nach der Auslastung des Netzes

Eine steigende Auslastung beeinflusst die Leistungskenngrößen des Netzes negativ. Es gibt allerdings zwei ausgezeichnete Betriebszustände: der *Knee*- und der *Cliff*-Punkt (siehe Abbildung 3.3). Knee- und Cliff-Punkt bezeichnen mittels Faustregeln bestimmte Punkte der Unter- bzw. Überlast und deren Folgen. Messtechnisch sind diese Punkte sehr schwer zu ermitteln. Sie geben einen ersten Eindruck des Verhaltens der Kenngrößen in Abhängigkeit von der Last. Daher sind folgende Formeln als eine Annäherung zu verstehen.

Vor dem Knee-Punkt:

- die RTT kann stabil bleiben ($drtt/dL \approx 0$),
- der Durchsatz kann linear steigen ($dD/dL \approx c_1$), und
- die Power kann linear steigen ($dP/dL \approx c_2$).

Zwischen Knee- und Cliff-Punkt:

- die RTT steigt schnell,
- der Durchsatz steigt langsamer und
- die Power sinkt, nachdem das Maximum am Knee erreicht wurde ($dP/dL = 0$ und $d^2P/dL^2 < 0$).

[2] Verzögerung ist ein Synonym für RTT. Der Begriff Produkt Datenraten-Verzögerung ist gängiger.

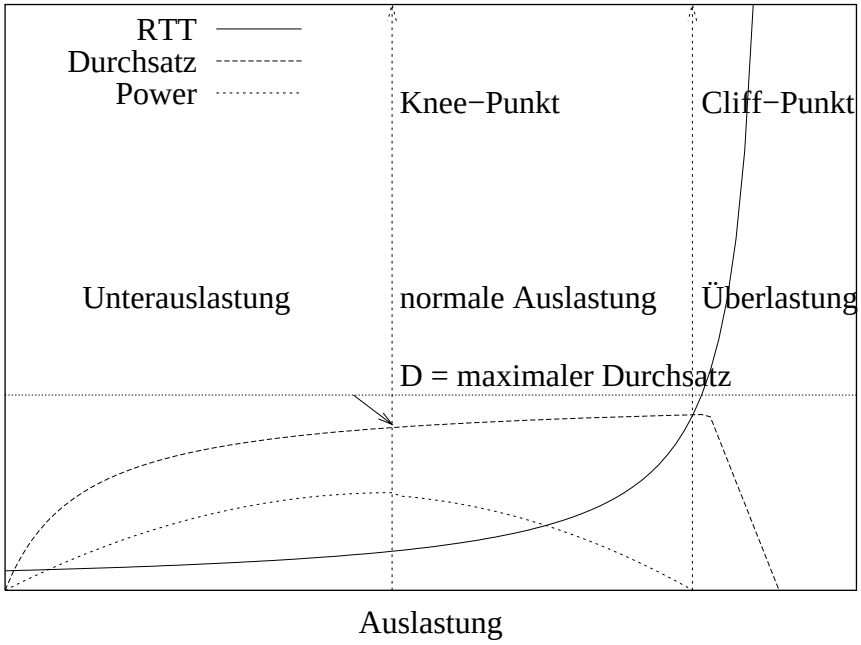


Abbildung 3.3: Netzauslastung und Netzüberlastung

Nach dem Cliff-Punkt:

- die RTT strebt gegen Unendlich ($drtt/dL \rightarrow \infty$),
- der Durchsatz sinkt gegen Null ($D \rightarrow 0$),
- die Power ist Null ($P = 0$).

Je nachdem, ob der Arbeitspunkt beim Knee- oder beim Cliff-Punkt ausgewählt wird, lässt sich die Überlastkontrolle in präventiv und reaktiv unterteilen. Es ist wichtig anzumerken, dass Steuerungsverfahren per Definition präventiv sind, wobei Regelungsverfahren sowohl präventiv als auch reaktiv sein können. Abbildung 3.4 zeigt ein Schema der Entwicklung der Warteschlangenlängen bei präventiven und reaktiven Verfahren.

Präventive Verfahren arbeiteten am Knee-Punkt, sie vermeiden die Netzüberlastung und sind wie folgt charakterisiert (siehe Tabelle 3.1):

- Aus der IP-Sicht: Niedrige Pufferauslastung. In dem $\rho \ll 1$ gilt, wird das Verwerfen von Pake-

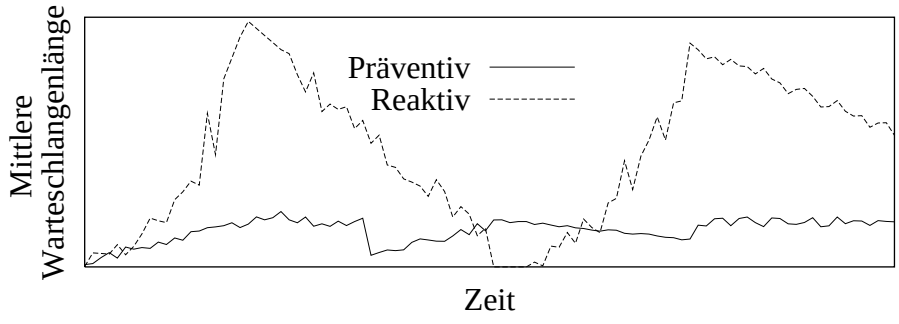


Abbildung 3.4: Schemata der präventiven und reaktiven Überlastkontrolle

ten vermieden. Die mittlere Warteschlangenlänge $\bar{N}$ und die mittlere Reaktionszeit R variieren um ein Optimum.

- Aus der TCP-Sicht: Häufige Benutzung der RTT und RTT-bezogenen Messgrößen als Überlastungssignal. Der Anstieg der RTT ist das Indiz, dass die Warteschlange wächst. Ein kleiner Anstieg der RTT ($rtt_i > rtt_{i-1}$ und $\Delta rtt \approx$ klein) bewirkt das Verkleinern des WND ($wnd_{i+1} < wnd_i$).

Zeit	**PRÄVENTIVE VERFAHREN**		**REAKTIVE VERFAHREN**	
	im ROUTER	im HOST	im ROUTER	im HOST
t_i	$\rho \ll 1$ $R_i > R_{i-1}$ $\rightarrow$	$rtt_i > rtt_{i-1}$ $wnd_{i+1} < wnd_i$	$\rho \approx 1$ $\bar{N}_{qi} = \bar{N}_{q_{MAX}}$ $\rightarrow$	$wnd_{i+1} = wnd_{min}$
t_{i+1}	$\rho \ll 1$ $\lambda_{i+1} < \lambda_i$ $R_{i+1} < R_i$ $\bar{N}_{qi+1} \approx \bar{N}_{qi}$ $\rightarrow$	$rtt_{i+2} < rtt_{i+1}$ $wnd_{i+2} > wnd_{i+1}$	$\rho \ll 1$ $\lambda_{i+1} \ll \lambda_i$ $R_{i+1} \ll R_i$ $\bar{N}_{qi+1} \ll \bar{N}_{qi}$ $\rightarrow$	$wnd_{i+2} \gg wnd_{i+1}$

Tabelle 3.1: Präventive und reaktive Überlastkontrolle

Reaktive Verfahren arbeiten mit einer Last etwas unterhalb des Cliff-Punktes. Netzüberlastung wird dabei nicht vermieden, so dass die Aktionen dieser Verfahren erst nach dem Eintreffen einer Netzüberlastung aktiviert werden. Ihre Eigenschaften sind:

- Aus der IP-Sicht: Hohe Pufferauslastung, die zum Verwerfen von Paketen führt. Dies bewirkt hohe Lastspitzen, denn die Warteschlange wird im ersten Moment kleiner, aber der Prozess fängt von neuem an zu senden. Die Folge ist eine ungleiche Verteilung der Kapazität des Engpasses, da Verbindungen mit kleinen Pfadlängen schneller auf die Änderungen reagieren können. Ist die Auslastung zum Zeitpunkt t_i hoch ($\rho_i \approx 1$) und die Warteschlange voll ($\bar{N}_{qi} = \bar{N}_{q_{MAX}}$), werden die nächsten Pakete verworfen. Die Sender reagieren auf diesen Verlust, und verkleinern ihre WND stark. Die Ankunftsrate und die Auslastung der Router werden absinken ($\lambda_{i+1} \ll \lambda_i$ und $\rho_{i+1} \ll \rho_i$). Als Folge wird die Warteschlange kleiner ($\bar{N}_{qi+1} \ll \bar{N}_{qi}$) und die Reaktionszeit viel kürzer ($R_{i+1} \ll R_i$). Nach der Verbesserung der Reaktionszeit zum Zeitpunkt t_{i+1}, werden diejenigen Sender diese Information nutzen, die sich am nächster zum Router befinden.

- Aus der TCP-Sicht: Benutzung von Paketverlust als Überlastungssignal. Die Sender erfahren eine hohe RTT-Varianz als Folge der Variation der Warteschlangenlänge. Im vorherigen Beispiel, wird die RTT steigen ($rtt_i > rtt_{i-1}$), wenn die Auslastung steigt. Da die Sender die RTT jedoch nicht bewerten, ignorieren sie die Variationen der Auslastung im Router. Erst nachdem die ersten Paketverluste festgestellt werden, wird das WND gesenkt ($wnd_{i+1} = wnd_{min}$).

3.4 Existierende Ansätze

Um einen Überblick über die gängigsten Verfahren zu gewinnen, sind diese in Abbildung 3.5 entsprechend der Klassifizierung nach dem Lösungsansatz in [YR95] zusammengestellt. Basierend darauf, werden sie nach ihren wichtigen Ansätzen gruppiert und ihre Vor- und Nachteile diskutiert.

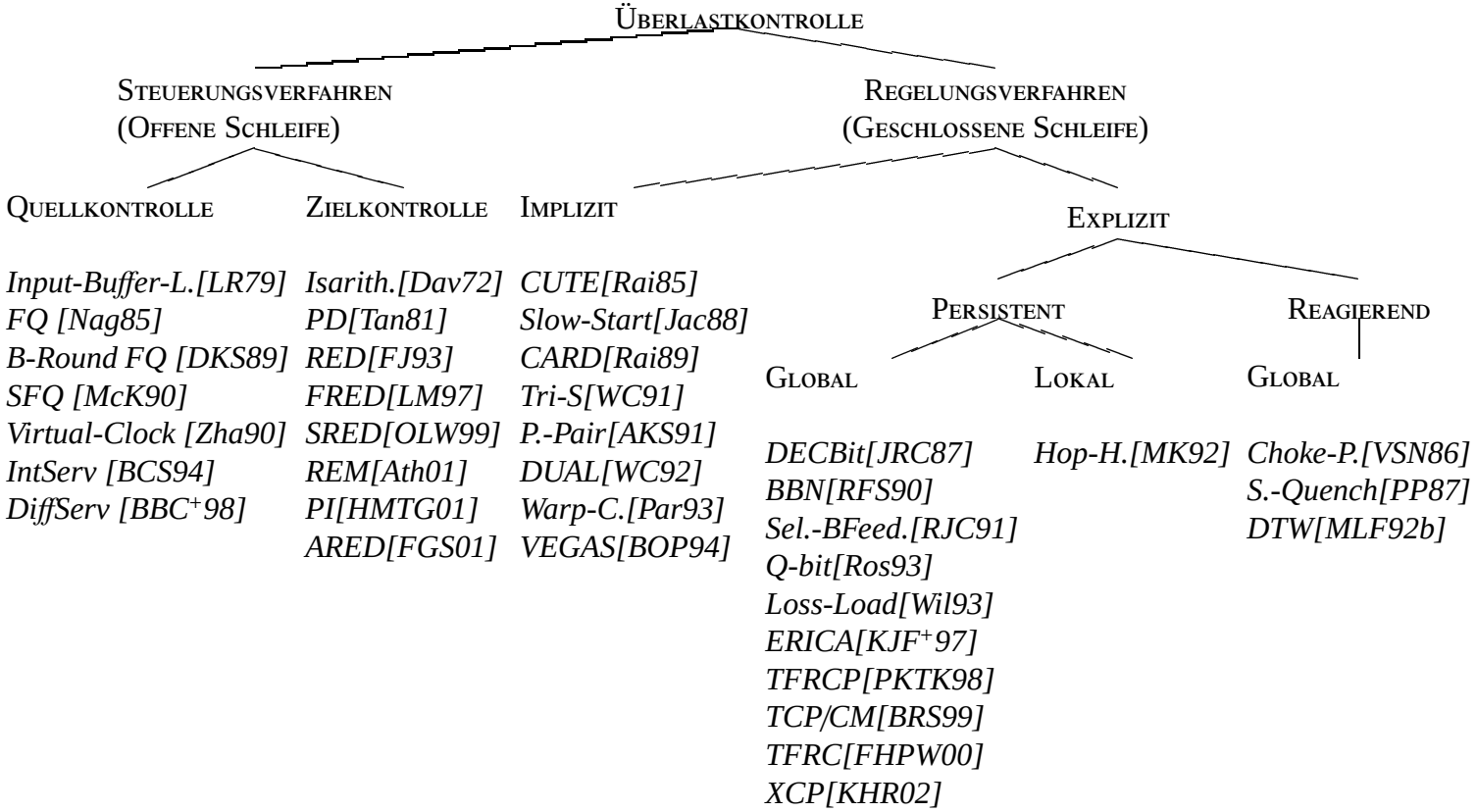


Abbildung 3.5: Existierende Ansätze klassifiziert nach dem Lösungsansatz in [YR95]

3.5 Steuerungsverfahren

Folgende Mechanismen werden hier betrachtet: Pufferbehandlung, Zugangskontrolle und Scheduling. Im ersten Fall handelt es sich um Verfahren, die durch Verwerfen von Paketen aus der Warteschlange ihre Last lokal reduzieren. Im zweiten Fall handelte es sich um Strategien zur Lastkontrolle an der Quelle. Scheduling bezieht sich auf die Bedienstrategien der Warteschlangen. Mit mehreren Warteschlangen ist Scheduling ein Mechanismus, dass zur Priorisierung des Verkehrs in einem Mehrklassenmodell eingesetzt werden kann.

3.5.1 Pufferbehandlung

Die Schwerpunkte dieser Verfahren sind die Festlegung der Puffergröße und die Pufferbehandlung in den Routern (engl. Active-Queue-Management, kurz AQM). Die Idee ist, dass in Abwesenheit von Zugangskontrolle die Last durch das Verwerfen von Paketen reduziert werden kann. Dann sind die zu treffenden Entscheidungen, wann und welche Pakete verworfen werden. Dabei wird vorausgesetzt, dass Transport- oder Applikationsprotokolle den Paketverlust feststellen und die Last reduzieren können. Ist dies nicht der Fall, ist Netzüberlastung nicht zu vermeiden. Alle Verbindungen werden über eine einzelne Warteschlange mittels Multiplexen bedient. Durch Multiplexen kann ein Router auf eine transiente Netzüberlastung besser reagieren. Ein weiterer Vorteil ist, dass die Algorithmen in einzelnen Routern implementiert werden können.

Die Feststellung der Puffergröße ist eine schwierige Aufgabe. Zu große Puffer erhöhen die RTT bis hin zu unannehmbaren Werten. Zu kleine Puffer verursachen häufigen Paketverlust. In der Vergangenheit wurde sie meist anhand von Poisson-Modellen festgestellt. Real weicht die Last in Rechnernetzen jedoch von diesen Modellen ab, da sowohl kurzfristig als auch längerfristig der Verkehr burst-artig sein kann [WTSW97, WP98]. Anwendungen wie Remote-Login oder FTP sind gut mit Poisson-Prozessen zu modellieren [PF95], andere jedoch, wie z.B. TELNET, weichen davon ab. Dies bekräftigt das Argument, unterschiedliche Warteschlangen für unterschiedliche Verkehrsarten einzusetzen.

Isarithmatic-Verfahren [Dav72]. Der Ansatzpunkt dieses Verfahrens ist, dass Netzüberlastung in den Teilnetzen dadurch entsteht, dass zu viele Pakete zirkulieren. Folglich sollte ihre Anzahl limitiert werden, um zu vermeiden, dass die Anzahl der Pakete über alle Grenzen steigt. Hierzu wird jedem Router eine Anzahl von *Permits* zugewiesen, die zum Weiterleiten eines Pakets berechtigen. Bevor einem Paket den Netzzugang gewährt wird, muss ein Permit vorhanden sein. Der Empfänger des Pakets gibt das Permit frei. Fehlende Permits führen zu Paketverlusten in den Routern.

Dieses Verfahren hat zumindest zwei Schwächen: Erstens kann ein einzelner Router überlastet sein, wenn eine Leitung intensiv benutzt wird. Dies ist der Fall, wenn viele Benutzer die Dienste des gleichen Servers verlangen. Zweitens ist es schwierig, gerechte und effiziente Mechanismen zur Verteilung der Permits zwischen den Routern zu finden.

Packet-Discarding (PD) [Tan81]. Bei dieser Methode ist die Puffergröße der Warteschlange fest und eine Reservierung findet nicht statt. Wenn der Puffer voll ist, werden Pakete verworfen. Die Warteschlange kann nach gewissen Kriterien bedient werden, z.B. FCFS (DropTail) oder LCFS (DropFront). PD zeichnet sich durch ihre geringe Prozesszeit aus.

Ein Nachteil ist, dass keine Fairness garantiert werden kann, denn ohne zusätzliche Mechanismen erzielen aggressive Sender mit kleinen RTT bessere Durchsätze als kooperative Sender mit großen RTT. Darüber hinaus ist in dieser Warteschlange der Verlust mehrerer hintereinander gesendeter Pakete korreliert, und muss in den höheren Schichten als ein einziges Ereignis bewertet werden. Falls das erste Paket zu einem Zeitpunkt ankommt, in dem der Puffer bereits voll ist, dann ist die Wahrscheinlichkeit hoch, dass die darauf folgenden Pakete auch verworfen werden [FJ93, FF96].

Random Early Detection (RED) [FJ93]. Die Warteschlange wird durch gezieltes Verwerfen klein gehalten. Ein Router markiert ankommende Pakete bevor der Puffer überläuft, und zwar dann, wenn die mittlere Warteschlangenlänge einen Schwellwert überschreitet. Markieren bedeutet das Verwerfen des Pakets oder das Setzen eines Bits im Paket-Header. Wird die zweite Alternative verwendet, dann können RED und RED-Varianten wie die im Abschnitt 3.7.2 beschriebenen Router-Mechanismen verwendet werden.

Die mittlere Warteschlangenlänge avg_i stellt einen exponentiell gewichteten gleitenden Durchschnitt dar:

$$avg_i = (1 - w_q) \cdot avg_{i-1} + w_q \cdot q_i$$

Beim Überschreiten bzw. Unterscheiten der Schwellwerte min_{th} und max_{th} finden folgende Aktionen statt:

$$\text{markiere} \begin{cases} \text{keine Pakete} & \text{falls } avg_i < min_{th} \\ \text{alle Pakete} & \text{falls } avg_i \geq max_{th} \\ \text{alle Pakete mit } p_{a,i} & \text{falls } min_{th} \leq avg_i < max_{th} \end{cases}$$

mit

w_q	Gewichtungsfaktor für die Warteschlangenlänge (i.a. $w_q = 0.002$)
q_i	aktuelle Warteschlangenlänge
$p_{a,i}$	Wahrscheinlichkeit zum Markieren eines Pakets

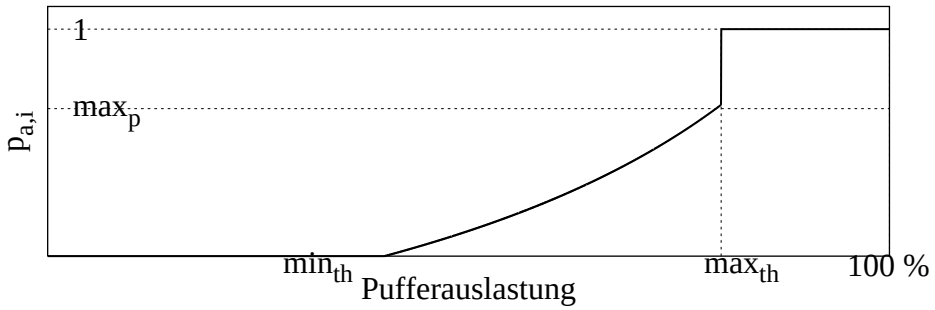


Abbildung 3.6: RED-Wahrscheinlichkeit zum Verwerfen eines Pakets (bas. auf [Arm00])

Mit anderen Worten: Solange $avg_i < min_{th}$ gilt, werden keine Pakete markiert. Ist $avg_i \geq max_{th}$ werden alle Pakete markiert. Zwischen min_{th} und max_{th} werden die Pakete mit der Wahrscheinlichkeit $p_{a,i}$ markiert (siehe Abbildung 3.6).

Der Parameter w_q dient zum Filtern transienter Netzüberlastung. Je näher w_q zu eins ist, desto größer ist der Einfluss eines kurzfristigen Wachstums der Warteschlange auf avg_i.

Die Wahrscheinlichkeit p_a steigt mit der Anzahl der empfangenen Pakete *count* seit dem letzten Markieren (d.h. mit dem Grad der Netzüberlastung) und mit p_b an. Die Variable p_b ist proportional zur Paketgröße, zu einem gegebenen Parameter max_p und zur Variation von *avg*:

$$p_{b,i} = \frac{max_p \cdot (avg_i - min_{th})}{(max_{th} - min_{th})} \cdot \frac{PacketSize_i}{MaxPacketSize}$$
$$p_{a,i} = \frac{p_{b,i}}{1 - count_i \cdot p_{b,i}}$$

Die Grundidee ist, je mehr Pakete einer Verbindung angehören, desto wahrscheinlicher das Verwerfen eines dieser Pakete wird. Zunehmend ersetzen RED-Router die traditionellen PD-Warteschlangen-Systemen [FP01]. Was den Durchsatz betrifft, reagiert RED sehr empfindlich auf den Wert von max_{th} [HTM02].

Flow Random Early Drop (FRED) [LM97]. FRED ist eine Variante von RED. Es wurde beobachtet, dass RED eine Tendenz zu *Unfairness* hat, wenn die Sender der Verbindungen unterschiedliche Reaktionen auf Feedback-Signale haben. Bei FRED werden drei Typen von Verbindungen differenziert, je nachdem wie die Sender reagieren:

- Nicht adaptive Verbindungen, falls die Sender auf Paketverlust nicht reagieren
- Stabile Verbindungen, falls die Sender schnell auf Paketverlust reagieren, da die RTT klein ist
- Schwache Verbindungen, falls die Sender langsam auf Paketverlust reagieren, da die RTT groß ist

Mit zunehmender Netzüberlastung können nicht adaptive Verbindungen das Wachsen der Warteschlange bis über die obere Schwelle (max_{th}) verursachen. Dadurch werden Pakete der anderen Verbindungstypen verworfen, obwohl sie auf die Feedback-Signale bereits reagiert haben. Daher ist es notwendig, einen gewissen Verbindungskontext zu pflegen, um zwischen den Typen von Verbindungen unterscheiden zu können. Im Gegensatz zu RED werden die Pakete Verbindungen zugeordnet, damit das Verwerfen von Paketen nicht nur von der Benutzung der Warteschlange abhängt. FRED schätzt die Anzahl von aktiven Verbindungen ab und stellt die Wahrscheinlichkeit zum Verwerfen eines Pakets in Abhängigkeit der Pufferauslastung jeder Verbindung fest.

Bei FRED werden neue globale Variablen eingeführt: min_q, max_q und *avgcq*. Sie sind das Minimum, Maximum und Durchschnitt von zugelassenen Paketen, die jede Verbindung in der Warteschlange haben darf. Pro Verbindung werden zwei Variablen gehalten: die Anzahl der Pakete in der Warteschlange ($qlen_j$) und die Anzahl der fehlgeschlagenen Versuche auf ein Feedback-Signal zu reagieren ($strike_j$). Zum Verwerfen der Pakete werden bevorzugt Verbindungen mit $qlen_j > avgcq$ oder $strike_j > 1$ genommen. Jede Verbindung darf min_q Pakete in der Warteschlange haben, bevor es zum Verwerfen kommt. Diese Bedingungen werden zu den normalen RED-Bedingungen (RED_i) hinzugefügt, so dass beim Eintreffen eines Pakets der Verbindung *j* zum Zeitpunkt *i* Folgendes geprüft wird:

$$\text{Verwerfe} \begin{cases} \text{dieses Paket} & \text{falls } qlen_{j,i} \geq max_q \vee \\ & \quad avg_i \geq max_{th} \wedge qlen_{j,i} > 2 \cdot avgcq \vee \\ & \quad qlen_{j,i} \geq avgcq \wedge strike_{j,i} > 1 \\ \text{dieses Paket (mit } p_{a,i}) & \text{falls } RED_i \wedge qlen_{j,i} \geq max(min_q, avgcq) \\ \text{keine Pakete} & \text{falls } avg_i < min_{th} \\ \text{alle Pakete} & \text{falls } avg_i \geq max_{th} \end{cases}$$

Es ist anzumerken, dass FRED die Vorteile der Anwendung von mehreren Warteschlangen anstrebt, ohne dabei für jede Verbindung die notwendige Pufferkapazität zu reservieren.

Stabilized Random Early Drop (SRED) [OLW99]. SRED ist eine andere Optimierungsvariante von RED. Es wird die Anzahl von aktiven Verbindungen abgeschätzt, damit die mittlere Warteschlangenlänge in Abhängigkeit davon optimiert werden kann. Eine aktive Verbindung wird durch die Existenz von Paketen in der Warteschlange festgestellt. Im Gegensatz zu FRED werden keine Statistiken pro Verbindungen gesammelt, jedoch anhand der im Puffer voraus gegangenen Pakete festgestellt, welche Verbindungen aktiv sind. Dabei wird eine Liste mit N_H Paketen gehalten. Die Größe N_H der Liste kann als das *Gedächtnis* der Router betrachtet werden. Wenn ein Paket ankommt, wird es mit k Paketen der Liste verglichen. Die Auswahl der Pakete folgt einem Zufallsprozess. Wenn zwei Pakete der gleichen Quelle angehören, wird ein *Hit* festgestellt. Der Begriff Quelle wird ausdrücklich offen gelassen und kann u.a. eine Verbindung oder eine Quell-Adresse bedeuten. Die Liste beinhaltet die Identifikation der Quelle, einen Zeitstempel und die Anzahl der Pakete. Beim Hit wird der Paket-Zähler der Verbindung erhöht und der Zeitstempel aktualisiert. Falls kein Hit stattfindet, wird das Element der Liste mit der Information des neuen Pakets mit einer Wahrscheinlichkeit p ersetzt.

Hits werden zur Abschätzung der aktiven Verbindungen und zur Identifizierung von aggressiven Sendern eingesetzt. Vorzugsweise werden Pakete aus diesen Verbindungen verworfen. Die eingesetzten Gleichungen für die Berechnung der Wahrscheinlichkeit zum Verwerfen des Pakets i sind:

$$Hit_i = \begin{cases} 0 & \text{falls no-Hit} \\ 1 & \text{falls Hit} \end{cases}$$

$$F_i = (1 - w_q) \cdot F_{i-1} + w_q \cdot Hit_i \qquad 0 < w_q < 1$$

$$p_{zap} = p_{sred}(q) \cdot min\left(1, \frac{1}{256 \cdot F_i^2}\right)$$

mit

F_i	Hit-Frequenz beim Ankommen von Paket i
w_q	Gewichtungsfaktor für die Hit-Frequenz-Funktion (i.a. $w_q = 1/N_H$)
p_{sred}	Funktion der Pufferbelegung q
p_{zap}	Wahrscheinlichkeit zum Verwerfen des Pakets i

Für die Abschätzung der aktiven Verbindungen wird der Kehrwert von F_i genommen. Denn mit N_C aktiven Verbindungen, deren Ankunftswahrscheinlichkeit $(\pi_i)_{i=1}^{N_C} = 1$ ist, gilt:

$$F\{Hit_i = 1\} \approx \sum_{i=1}^{N_C} \pi_i^2 \lessapprox \frac{1}{N_C}$$

Im Gegensatz zu RED hängt die Wahrscheinlichkeit zum Verwerfen der Pakete nur von der aktuellen Pufferbelegung q und von der Anzahl $1/F_i$ von aktiven Verbindungen ab. Die Parameter werden so gewählt, dass die Fenster der TCP-Verbindungen optimal gesetzt werden.

Random Exponential Marking (REM) [Ath01]. Die Grundidee vom REM ist, die Messung der Überlast von der Messung der Leistungskenngrößen zu entkoppeln, denn Erstere muss die Anzahl von Verbindungen berücksichtigen, während die zweite unabhängig davon ist. Dabei soll das Verfahren eine hohe Auslastung bei niedrigem Paketverlust erreichen. REM versucht, die Ankunftsrate an die Datenrate der Leitung anzupassen, gleichzeitig soll die Warteschlangenlänge um ein Optimum variieren.

Als Maß der Überlastung wird die Variable *price* gehalten. Mit dem Wert von *price* wird die Wahrscheinlichkeit zum Markieren eines Pakets angegeben. Basierend auf der Differenz zwischen der Ankunftsrate und der Datenrate der Leitung, und zwischen der Warteschlangenlänge und dem Sollwert für die Warteschlangenlänge wird *price* erhöht oder verringert. Falls die gewichtete Summe dieser Differenzen positiv ist, wird *price* erhöht, andernfalls verringert. Die Summe wird positiv wenn entweder die Ankunftsrate höher ist als die Datenrate, oder die Warteschlangenlänge größer als ihr Sollwert ist. Diese Differenzen werden größer, wenn die Anzahl von Verbindungen höher ist. Die Berechnung von *price* ist dann:

$$price_{i+1} = max(0, price_i + \gamma \cdot (\alpha \cdot (q_i - q^*) + x_i - c_i))$$
$$p_i = 1 - \phi^{-price_i}$$

mit

- q_i Warteschlangenlänge zum Zeitpunkt i
- q^* Sollwert der Warteschlangenlänge
- x_i Ankunftsrate der Warteschlange zum Zeitpunkt i
- c_i zur Verfügung stehende Datenrate für die Warteschlange zum Zeitpunkt i
- p_i Wahrscheinlichkeit zum Markieren eines Pakets

Die Parameter γ und α sollen klein und positiv sein. In der Regel wird $\gamma = 0.001$ und $\alpha = 0.1$ gewählt. Der Parameter ϕ wird auf 1.001 gesetzt.

Diese Gleichungen sollen gewährleisten, dass, wenn die Anzahl von Verbindungen zunimmt, die Wahrscheinlichkeit zum Markieren eines Pakets exponentiell mit der Pufferauslastung steigt. Gleichzeitig wird die Warteschlangenlänge auf den Sollwert stabilisiert, denn es ist die Differenz zum Sollwert, die für die Ermittlung der Wahrscheinlichkeit verwendet wird.

Proportional Controller (PI) [HMTG01]. Wie bei REM wird die Warteschlangenlänge mit einem Referenzwert, q_{ref}, reguliert. Die Warteschlangenlänge wird mit einer gewissen Periodizität abgetastet und diese Werte für die Berechnung der Wahrscheinlichkeit zum Markieren verwendet:

$$p_{i+1} = \alpha \cdot (q_{i+1} - q_{ref}) - \beta \cdot (q_i - q_{ref}) + p_i$$

mit

- p_i Wahrscheinlichkeit zum Verwerfen eines Pakets zum Zeitpunkt der Abtastung i
- q_i momentane Warteschlangenlänge

Daraus resultiert, dass die Wahrscheinlichkeit zum Markieren innerhalb der Abtastungsperiode steigt, wenn die momentane Warteschlangenlänge größer als der Sollwert ist oder größer als die vorangegangene abgetastete Warteschlangenlänge ist. Die Parameter α und β hängen sowohl von der Datenrate der Leitung, als auch von der höchsten erwarteten RTT und von der erwarteten Anzahl von Verbindungen ab.

Adaptive RED (ARED) [FGS01]. Adaptive RED stellt dynamisch die Parameter w_q und max_p in Abhängigkeit der Datenrate der Leitung und der aktuellen Pufferauslastung fest. Das Ziel ist, die Auslastung der Leitung zu optimieren. Es wurde beobachtet, dass die Geschwindigkeit zum Bilden einer Netzüberlastung von der Anzahl der Verbindungen abhängt. Ein kleiner max_p kann gut bei niedriger Auslastung aber schlecht bei hoher Auslastung sein. Daher wird max_p in Abhängigkeit von der Pufferauslastung festgesetzt. Des weiterem muss die Berechnung der mittleren Warteschlangenlänge Rücksicht auf die Leitungskapazität nehmen. Zuerst wird w_q in Abhängigkeit der Datenrate der Leitung festgesetzt. Dann, wenn die mittlere Warteschlangenlänge (avg) unter die untere Schwelle (min_{th}) absinkt, wird die Wahrscheinlichkeit zum Verwerfen des Pakets (max_p) konservativer berechnet. Somit gilt:

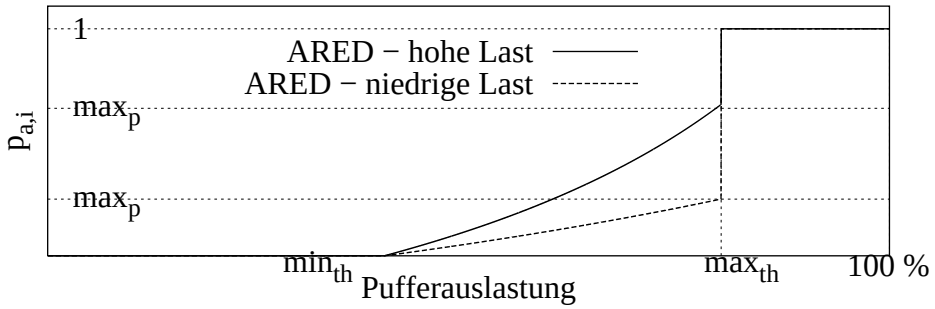


Abbildung 3.7: Wahrscheinlichkeit zum Verwerfen eines Pakets in ARED (bas. auf [Arm00])

$$w_q = 1 - exp(1/B)$$

$$max_{p,i} = \begin{cases} max_{p,i-1} + \alpha & \text{falls } avg_{i-1} > q \wedge max_{p,i-1} \leq 0.5 \\ \beta \cdot max_{p,i-1} & \text{falls } avg_{i-1} < q \wedge max_{p,i-1} \geq 0.01 \end{cases}$$

$$q = min[min_{th} + 0.4 \cdot (max_{th} - min_{th}),\ min_{th} + 0.6 \cdot (max_{th} - min_{th})]$$

mit

- B Datenrate der Leitung in Pakete/s mit durchschnittlicher Paketgröße
- α Vergrößerungsfaktor für $max_{p,i}$ (i.a. $\alpha = min(0.01, max_{p,i-1}/4)$)
- β Verkleinerungsfaktor für $max_{p,i}$ (i.a. $\beta = 0.9$)
- q Soll-Warteschlangenlänge

Die Variable $max_{p,i}$ liegt somit zwischen 1 % und 50 %. Je höher die Last, desto größer wird max_p. Diese Änderung der Last kann sowohl durch die Zunahme der Anzahl der Verbindungen als auch durch Ignorieren von Feedback-Signalen verursacht werden. Da Fairness nur indirekt behandelt wird, gibt es auch keine Garantie dafür.

3.5.2 Zugangskontrolle und Scheduling

Während die bisher vorgestellten Verfahren auf das Verwerfen von Paketen aufbauen, schränken die im Folgenden beschriebenen Verfahren die Last im Netz durch Zugangskontrolle ein. Die Zugangskontrolle kann auf mehreren Ebenen stattfinden: von globalen Verkehrsarten bis hinunter auf einzelne Verbindungen. Dabei wird der Verkehr in mehrere Klassen geteilt und häufig jede Klasse durch ihre eigene Warteschlange bedient. Mechanismen zur Klassifizierung der Pakete, zur Auswahl der zu sendenden Pakete (Scheduling) und zur Aufteilung der Leitungskapazität (Link-Sharing) sind notwendig. Nachteil ist der notwendige Aufwand für die Behandlung mehrerer Warteschlangen. Dieser Aufwand umfasst die Klassifizierung der Pakete, die Einrichtung neuer Warteschlangen, die Suche der entsprechenden Warteschlange beim Ankommen von Paketen und das Löschen von Warteschlangeninhalten.

Der Einsatz von mehreren Warteschlangen verlangt die Reservierung von getrennten Speicherbereichen in den Routern, was die nachteilige Folge hat, dass für kurzfristig erhöhte Anforderungen die Ressourcen für eine Klasse eventuell nicht ausreichen. Mit Scheduling ist es möglich, eine gewisse Dienstgüte (engl. Quality of Service, kurz QoS) für eine bestimmte Klasse zu gewährleisten. Denn dabei werden die Priorisierung der Klassen und die Aufteilung der Leitungskapazität ermöglicht.

Die neueren Ansätze wie IntServ und DiffServ stellen Verfahren zur Gewährleistung von Dienstgüte dar. Im Vordergrund steht die Anforderung an das Internet, mehrere Verkehrsklassen des Best-Effort- und Echtzeit-Verkehrs übertragen zu können.

Input-Buffer Limit [LR79]. Dieses Verfahren basiert auf der Fähigkeit der Router, zwischen Input- und Transit-Verkehr zu unterscheiden. Die Puffergröße für den Input-Verkehr ist ein Bruchteil des Transit-Verkehrs. Im Fall einer Netzüberlastung wird der Input-Verkehr durch Reduzierung ihrer Pufferbereiche gestoppt. Der dadurch freiwerdende Puffer wird für den Transit-Verkehr zur Verfügung gestellt.

Fair-Queuing (FQ) [Nag85]. Mit dem Ziel die Fairness zu gewährleisten, hält jeder Router eine Warteschlange pro Quell/Ziel-Host Paar. Die Warteschlangen werden mit einem Round-Robin-Algorithmus bedient. Aggressive Sender haben keinen Vorteil, da sie nur ihre eigenen Warteschlangen füllen. Da die Paketgröße unterschiedlich sein kann, gibt es trotzdem einen Vorteil für die Quell/Ziel-Host Paare, deren Verbindungen zu Applikationen gehören, die große Paketlängen einsetzen.

Bit-Round Fair-Queuing [DKS89]. Dieses Verfahren basiert auf FQ mit zusätzlicher Berücksichtigung der Paketgröße. Das nächste zu sendende Paket wird aus derjenigen Warteschlange ausgewählt, welche den kleinsten Round-Wert angenommen hatte, nachdem das vorherige Paket gesendet wurde. Bei Überlastung wird ein Paket aus der Warteschlange mit dem höchsten Round-Wert verworfen.

Die gleiche Datenrate pro Quell/Ziel-Host wird garantiert, jedoch nicht für die einzelnen Verbindungen innerhalb des Paares Quell/Ziel-Host: z.B. eine TELNET-Verbindung hat keinen Vorrang in der

gleichen Warteschlange gegenüber einer FTP-Verbindung. Die Folge ist, dass ein Host mit wenigen Verbindungen einen höheren Durchsatz als ein Host mit vielen Verbindungen erreicht. Eine andere Variante der gleichen Autoren ist Weighted-Fair-Queuing (WFQ). In WFQ werden die einzelnen Quell/Ziel-Host Paare mit einem Faktor gewichtet und dadurch priorisiert.

Stochastic Fair-Queuing (SFQ) [McK90]. Je höher die Anzahl von aktiven Verbindungen in FQ, desto aufwändiger wird die Paketauswahl. Dieses Skalierungsproblem wird gelöst, indem pro Quell/-Ziel-Host eine fixe Anzahl von Warteschlangen bereitgestellt wird. Obwohl dies mehr Speicher beansprucht, wird die Suche schneller. Die passenden Warteschlangen werden mittels einer Hash-Funktion gefunden.

Virtual-Clock [Zha90]. Die Router weisen jeder Verbindung einen Timer zu, der nach Ankommen jedes Pakets aktualisiert wird. Dadurch kann die Zwischenankunftszeit der Verbindung ermittelt werden, die zur Bestimmung ihrer Ist-Senderate dient. Die Differenz zwischen der Zwischenankunftszeit der Verbindung und der Systemzeit zeigt die Abweichung von der Soll-Senderate. Die Senderate soll nach Überschreiten eines Schwellwertes reduziert werden. Zu diesem Zweck senden die Router Kontrollpakete an die Sender. Somit werden aggressive Sender gebremst.

Ein Nachteil von Virtual-Clock ist, dass reservierte jedoch nicht benutzte Ressourcen zu niedriger Auslastung führen. Außerdem erhöhen die Kontrollpakete im Überlastfall die ohnehin schon hohe Last.

Integrated Services Architecture (IntServ) [BCS94]. IntServ teilt die Applikationen in Echtzeit- und Best-Effort, und bietet einen entsprechenden Dienst an. Das Ziel ist, die Leitungskapazität zwischen unterschiedlichen Organisationen aufteilen zu können. Der Echtzeit- hat Priorität gegenüber den Best-Effort-Verkehr, und wird durch Zugangskontrolle zugelassen oder abgelehnt. Dienstgüte für den Echtzeit-Verkehr kann durch Ressource-Reservierung ermöglicht werden. Dies impliziert, dass Mechanismen zur Isolierung der Verbindungen sowie zum Speichern von Verbindungszuständen vorhanden sein müssen. Die Komponenten dieses Modells sind:

- *Paket Scheduling (PS)* zur Weiterleitung der Pakete. Dies erfolgt entweder unter Verwendung einer Warteschlange pro Klasse oder durch WFQ-Warteschlangen.
- *Bandwidth Enforcement* zur Kontrolle der Senderate jeder Verbindung.
- *Classifier* zur Zuordnung der Pakete zu der entsprechenden Warteschlange.
- *Admission Control* zum Annehmen oder Ablehnen einer Dienstanforderung.
- *Reservation Setup Protocol (RSVP)* zur Sicherstellung der Ressourcen.

Differentiated Services (DiffServ) [BBC+98, NBBB98]. DiffServ ist eine Architektur, um skalierbare differenzierbare Services im Internet zu ermöglichen. Das Speichern von Verbindungszuständen in den Routern ist dabei nicht notwendig. Eine Service-Klasse wird über eine eigene Warteschlange bedient und kann als eine Kombination aus folgenden Faktoren realisiert werden:

- Das Setzen von Bits im Feld Type-Of-Service (TOS) in IPv4 oder im Feld Traffic-Class in IPv6. Dieses Feld wird in DiffServ als das DS-Feld bezeichnet.
- Diese Bits können zur Weiterleitung des Pakets ausgewertet werden.

- Diese Bits können zur Markierung des Pakets genutzt werden.

Mechanismen zur Klassifizierung sowie zur Pufferbehandlung und Scheduling sind vorhanden. Somit muss ein Router in der Lage sein, basierend auf dem DS-Feld im Paket-Header, die angeforderte Service-Klasse zu identifizieren und die Pakete an die entsprechende Warteschlange weiterzuleiten. Die meiste Komplexität steckt in den Routern am Rande einer Domäne. Diese Router sind zuständig für die Zugangskontrolle und die Klassifizierung der Pakete. Da die Verbindungen einer Service-Klasse aggregiert werden, werden für die einzelnen Verbindungen keine Dienstgüte garantiert. Eine wichtige Eigenschaft von DiffServ ist, dass das Weiterleiten von dem Routing getrennt ist.

3.6 Regelungsverfahren - Ende-zu-Ende-Überlastkontrolle

In dieser Kategorie fallen Verfahren mit Überlastkontrolle in den Endpunkten der Verbindungen. Die Sender und die Empfänger können nur auf die Feedback-Signale des Kommunikationspartners zugreifen. Sie bestehen aus einer Sender- und aus einer Empfängerkomponente, deren Aufgabe es ist, das Fenster oder die Senderate zu setzen. Da das Netz als Blackbox betrachtet wird, kann der Einfluss von anderen Verbindungen nur bedingt eingeschätzt werden. Die Verfahren können mit impliziten oder expliziten Signalen arbeiten. Eingesetze Mechanismen zur Regelung sind hierbei Timeouts, die Nutzung der RTT oder der Paketverlustraten.

3.6.1 Implizites Feedback - TCP

Diese Verfahren behandeln die Überlastkontrolle in TCP, wobei ihre Prinzipien auch in anderen fensterbasierten Protokollen angewendet werden können. In TCP ist Überlastkontrolle mit Flusskontrolle kombiniert. Der Sender limitiert die Einspeisung von neuen Paketen ins Netz mittels eines Fensters, genannt Congestion-Window (CWND). Der Einfluss von anderen Verbindungen im Netz kann nur an der Variation der RTT festgestellt werden. Wichtig sind die Initialisierungs-Werte, sowie die Algorithmen zum Vergrößern bzw. Verkleinern des CWND.

Der Sender hat hierbei zwei wichtige Entscheidungen zu treffen: wie viel das CWND vergrößert oder verkleinert wird und wie häufig dies geschehen soll.

Das AIMD-Prinzip (*additive-increase/multiplicative decrease*) [CJ89] gibt Richtlinien an: Zum Gewährleisten der Stabilität muss das Vergrößern bzw. das Verkleinern des CWND additiv bzw. multiplikativ erfolgen:

$$cwnd_i = \begin{cases} cwnd_{i-1} + \alpha & \alpha > 0 \quad \text{CWND vergrößern} \\ \beta \cdot cwnd_{i-1} & \beta < 1 \quad \text{CWND verkleinern} \end{cases}$$

Das Vergrößern des CWND sollte bewirken, dass die gesamte Last nach und nach erhöht wird. Noch wichtiger ist, dass andere konkurrierende Sender Zeit haben, diese erhöhte Last festzustellen, und sich daran anzupassen. Das Verkleinern des CWND muss schnell erfolgen, da die Warteschlangen eine längere Erholungszeit benötigen, bis sie wieder die gewünschte Länge erreichen. Durch die Auswahl der Parameter sollten die minimalen Schwankungen der CWND erreicht werden.

Das AIAD-Prinzip (*additive-increase/additive-decrease*) [CJ89] ist eine andere Alternative, deren Ziel ist, die Zeit zum Erreichen des Knee-Punktes und Fairness zu minimieren.

Zur Häufigkeit der Anpassung des CWND wird empfohlen, nach jeder zweiten Quittung das CWND erneut festzusetzen. Der Grund ist, dass die Effekte der Änderung der Last nicht sofort wirken. Dies betrifft die Verfahren CUTE und Slow-Start nicht, da die RTT dort nicht zur Anpassung der Last eingesetzt wird. Aus dem gleichen Grund sind CUTE und Slow-Start reaktive Verfahren. Dies impliziert einen Betrieb nahe dem Cliff-Punkt. Die anderen Verfahren benutzen die RTT und den Durchsatz, um im Knee-Punkt zu arbeiten. Dadurch erreichen sie eine bessere Leistung als die reaktiven TCP-Varianten.

CUTE [Rai85, Rai86]. Der Ablauf eines Timers impliziert den Verlust eines Pakets, und wird als implizites Überlastungssignal interpretiert. Der Sender besitzt einen unteren und oberen Schwellwert für das CWND. Das kleine initiale CWND soll den Verkehr stabil halten. Das CWND wird um eins erhöht, wenn die Anzahl von Quittungen seit der letzten Änderung gleich der Größe des aktuellen CWND ist. Dies führt zum linearen Wachstum des CWND mit der Zeit. Die Quittungen treffen alle t Zeiteinheiten beim Sender ein. Falls so viele Quittungen ankommen, wie Pakete gesendet werden, kann die Reihe $cwnd_i, i = 1, 2, 3, ..n$ als eine Zeitfunktion $cwnd(t)$ dargestellt werden. Diese Funktion nimmt folgende Werte an: $cwnd(t = 1) = 1$, $cwnd(t = 2) = 2$, $cwnd(t = 3) = 3$, $cwnd(t = 4) = 4..$, usw. Nach Ablauf eines Timers wird das CWND auf das Minimum zurückgesetzt.

$$cwnd_i = \begin{cases} cwnd_{min} & \text{falls } i = 1 \ \vee \ \text{RT} \\ cwnd_{i-1} + 1 & \text{falls } cwnd_{i-1} < cwnd_{max} \ \wedge \ cwnd_i\text{- Quittungen} \end{cases}$$

mit

$cwnd_{min}$ Minimales CWND (i.a. $cwnd_{min} = 1$)
$cwnd_{max}$ Maximales CWND (i.a. $cwnd_{max} = min(rwnd, 3 \cdot hops)$)

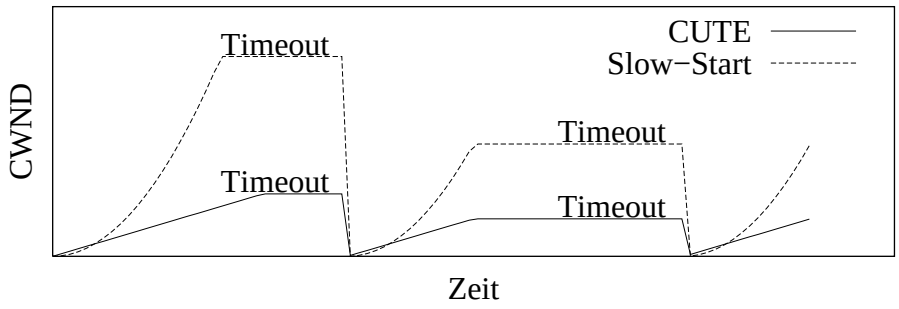


Abbildung 3.8: CUTE- und Slow-Start Modelle

Slow-Start [Jac88]. Slow-Start ist ein TCP-Mechanismus zur Initialisierung und Zurücksetzung der Datenübertragung. Er soll nur bis zum ersten Paketverlust angewendet werden. Es wurde vorgeschlagen, um den *Congestion-Collapse* [Jac88] in ARPANET zu vermeiden. Das CWND wird wie in CUTE auf einen kleinen Wert initialisiert. Immer dann, wenn eine neue Quittung ankommt, wird das CWND um einen Faktor erhöht. Dies bewirkt einen exponentiellen Anstieg des CWND mit der Zeit. Somit hat die Funktion $cwnd(t)$ folgende Werte: $cwnd(t = 1) = 1$, $cwnd(t = 2) = 2$, $cwnd(t = 3) = 4$, $cwnd(t = 4) = 8...$, usw.

$$cwnd_i = \begin{cases} cwnd_{min} & \text{falls } i = 1 \ \vee \ \text{RT} \\ cwnd_{i-1} + \alpha & \text{falls } cwnd_{i-1} < cwnd_{max} \ \wedge \ \text{Quittung} \end{cases} \tag{3.5}$$

Der Wert vom $cwnd_{max}$ ist das Minimum aus *rwnd* und dem Schwellwert des CWND. Nach jedem Ablauf des Retransmission-Timers wird $cwnd_{max}$ halbiert[3]. In TCP wird Slow- Start mit den Werten $cwnd_{min} = 1$ oder $cwnd_{min} = 2$ und $\alpha = 1$ eingesetzt.

CUTE und Slow-Start haben das Problem, dass das Vergrößern des Fensters erst durch den Ablauf eines Retransmission-Timers gestoppt wird. Werden PD-Warteschlangen eingesetzt, wird das Netz überlastet sein, bevor die Sender einen Paketverlust feststellen.

CARD [Rai89]. Vorgeschlagen als Ersatz für Slow-Start in TCP, kann CARD jedoch auch während der gesamten Datenübertragung angewendet werden. Solange kein Paketverlust festgestellt wird, wird das CWND nach dem AIMD-Prinzip gesetzt. Das Ziel ist, die Power zu maximieren und Fairness für alle Teilnehmer zu gewährleisten. Anhand des normierten Gradients der RTT, *NDG*, definiert als:

$$NDG = \frac{cwnd_i + cwnd_{i-i}}{cwnd_i - cwnd_{i-i}} \cdot \frac{rtt_i - rtt_{i-1}}{rtt_i + rtt_{i-1}}$$

wird das CWND geändert. Das CWND wird vergrößert, wenn *NDG* negativ oder Null ist:

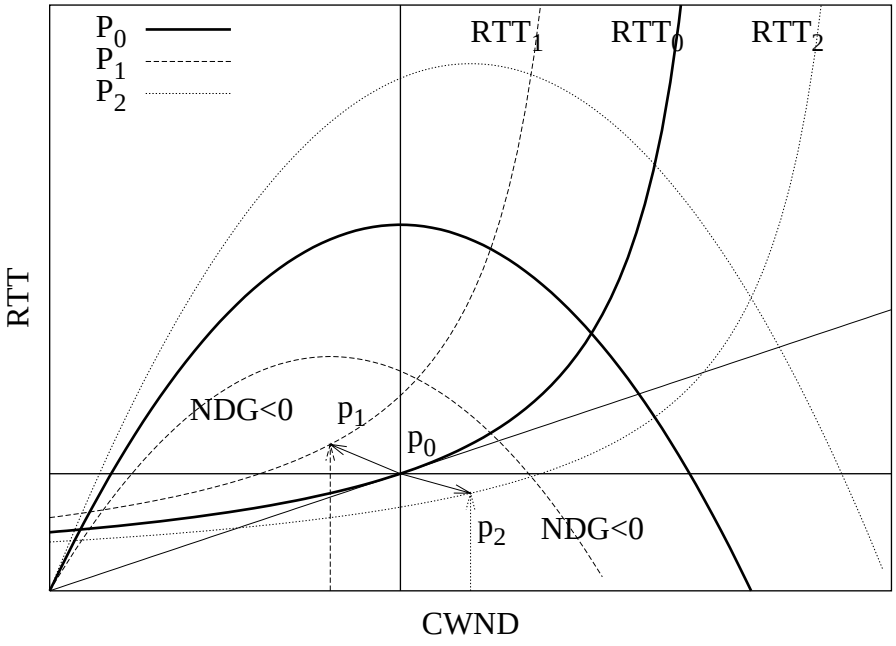


Abbildung 3.9: CARD Modell

$$cwnd_i = \begin{cases} cwnd_{min} & \text{falls } i = 1 \\ cwnd_{i-1} + 1 & \text{falls } NDG \leq 0 \ \vee \ cwnd_i = cwnd_{min} \quad (\text{i.a. } cwnd_{min} = 1) \\ 7/8 \cdot cwnd_{i-1} & \text{falls } NDG > 0 \ \vee \ cwnd_i = cwnd_{max} \quad (\text{i.a. } cwnd_{max} = rwnd) \end{cases}$$

[3] Siehe Congestion-Avoidance im Abschnitt 4.2.3

Dies bedeutet, dass die Reduzierung der gemessenen RTT nur dann zum Vergrößern des CWND führt, wenn die Ressourcen von anderen Verbindungen freigegeben werden. Dies impliziert, dass eine Verbesserung der RTT auf Grund der Reduzierung des eigenen CWND nicht berücksichtigt wird. In Abbildung 3.9 befindet sich das System am Punkt p_0. Bei den Bewegungen zu den Quadranten der Punkte p_1 und p_2 ist $NDG < 0$. Dabei kann die Power noch maximiert werden.

TRI-S [WC91]. Das Ziel von TRI-S ist, den optimalen Betriebspunkt schnell zu erreichen, immer wenn große Laständerungen eintreffen, z.B. nach Starten und Beenden von neuen Verbindungen. Zur Abschätzung der Last wird die Größe NTG definiert. NTG ist der normierte Gradient des Durchsatzes und variiert je nach Last zwischen 0 und 1. Mit hoher Last tendiert NTG gegen Null.

$$NTG_i = \frac{TG_i}{TG_1}$$
$$TG_i = \frac{D_i - D_{i-1}}{cwnd_i - cwnd_{i-1}}$$
$$D_i = \frac{cwnd\_fly_i}{rtt_i}$$

Die Ermittlung des Durchsatzes basiert auf der ausstehenden Datenmenge, da in TCP das CWND nicht die Anzahl der gesendeten Segmente in jeder Periode ist, sondern die unquittierte Datenmenge ($cwnd\_fly$).

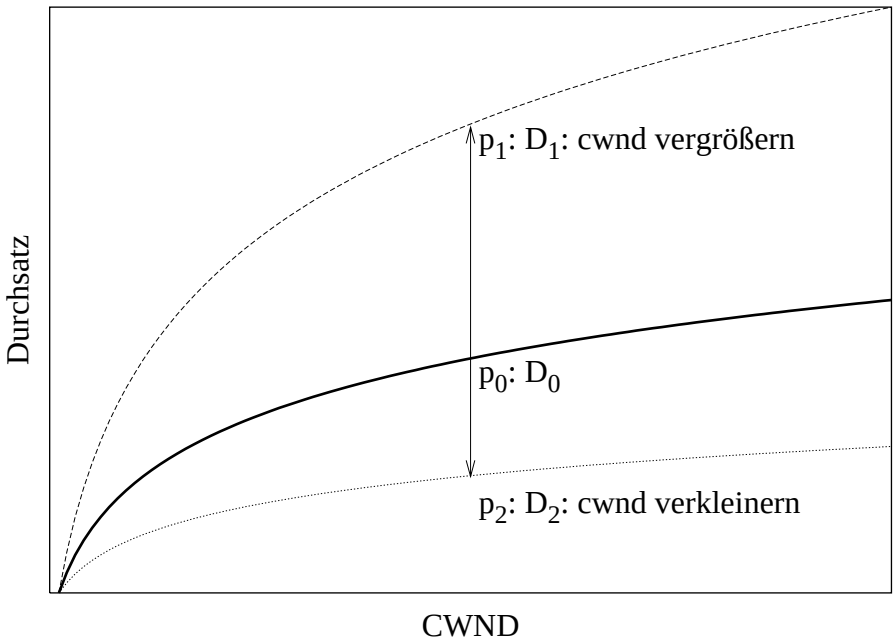


Abbildung 3.10: TRI-S Modell

TRI-S startet im Slow-Start. Nach Ablauf des Retransmission-Timers wird CWND nach dem AIAD-Prinzip folgenderweise gesetzt:

$$cwnd_i = \begin{cases} cwnd_{i-1} + \frac{1}{cwnd_{i-1}} & \text{falls } NTG_i > NTG_d \quad \text{(i.a. } NTG_d = 0.5\text{)} \\ cwnd_{i-1} - 1 & \text{falls } NTG_i < NTG_d \\ cwnd_{i-1} & \text{sonst} \end{cases}$$

Die Bewegung von p_0 zu p_1 bedeutet, dass Ressourcen freigegeben wurden. Von p_0 zu p_2 muss das *cwnd* verkleinern werden, da neue Verbindungen gestartet wurden.

DUAL [WC92]. Die beiden Grundideen sind: Erstens, da die RTT von der Wartezeit in den Routern abhängt, spiegelt die minimale (bzw. die maximale) RTT die minimale (bzw. die maximale) Warteschlangenlänge im Pfad wieder. Zweitens, da ab einer gewissen Warteschlangenlänge die Wahrscheinlichkeit einer Netzüberlastung hoch ist, muss der Sender, falls diese Schwelle überschritten ist, das CWND verkleinern.

Folglich ermittelt der Sender mittels rtt_{min} und rtt_{max} einen Schwellwert $\widehat{rtt}$, um ein Gespür für die Warteschlangenlängen zu gewinnen. Die Anpassung des CWND nach dem AIMD-Prinzip ist dann:

$$cwnd_i = \begin{cases} cwnd_{min} & \text{falls } i = 1 \quad \text{(i.a. } cwnd_{min} = 1\text{)} \\ 7/8 \cdot cwnd_{i-1} & \text{falls } rtt_i > \alpha \cdot rtt_{max} + (1-\alpha) \cdot rtt_{min} \end{cases}$$

Es wird $\alpha = 0.5$ genommen, d.h $\widehat{rtt}$ ist der Mittelwert der RTT.

VEGAS [BOP94]. In VEGAS wird der aktuelle mit dem erwarteten Durchsatz verglichen und die Differenz als Kriterium zum Vergrößern oder Verkleinern des CWND angewendet. Wie in DUAL, dient die minimale RTT zur Feststellung des optimalen Durchsatzes. VEGAS startet im Slow-Start aber das Vergrößern des CWND erfolgt nur jede zweite RTT. Wenn der aktuelle Durchsatz unter den erwarteten Wert absinkt, wird das CWND nach dem AIAD-Prinzip folgendermaßen angepasst:

$$\Delta D = \frac{cwnd_i}{rtt_{min}} - \frac{cwnd_i}{rtt_i}$$

$$cwnd_i = \begin{cases} cwnd_{min} & \text{falls } i = 1 \quad \text{(i.a. } cwnd_{min} = 1\text{)} \\ cwnd_{i-1} + 1 & \text{falls } \Delta D < \Delta D_{min} \quad \text{(i.a. } \Delta D_{min} = 2\text{)} \\ cwnd_{i-1} - 1 & \text{falls } \Delta D > \Delta D_{max} \quad \text{(i.a. } \Delta D_{max} = 4\text{)} \\ cwnd_{i-1} & \text{sonst} \end{cases}$$

Außerdem erkennt VEGAS den negativen Einfluss des Ablaufs des Retransmission-Timers auf den Durchsatz. Die Wartezeiten sollen verkürzt werden, indem die Timestamp-Option angewendet wird. Durch die Option wird das Paket um 10 Bytes größer, dafür kann aber der Sender den Verlust eines Pakets schneller feststellen. Dies ist eigentlich ein Recovery-Mechanismus, der aber indirekt den Durchsatz verbessert. Falls die Differenz zwischen der aktuellen Zeit und dem Zeitstempel der Quittung größer als der Retransmission-Timer ist, wird ein fehlendes Paket sofort neu übertragen.

3.6.2 Implizites Feedback - Andere Verfahren

Diese Verfahren sind allgemeine Ansätze für Ende-zu-Ende-Überlastkontrolle. Dabei geht es um die Abschätzung der Leitungskapazität in Abhängigkeit von der aktuellen Last.

Packet-Pair [AKS91] . Packet-Pair basiert auf folgendem Ansatz: Falls das Zeitintervall Δt zwischen zwei hintereinander gesendeten Paketen kleiner als die Bedienzeit X des Engpasses ist, dann ist die Zwischenankunftszeit beim Empfänger genau $X = b/B_{min}$ mit b als Paketgröße und B_{min} als Datenrate des Engpasses (siehe Abbildung 3.11). Wenn $\Delta t < X$, dann sind die RTT aller während der Zeit Δt übertragenen Pakete korreliert.

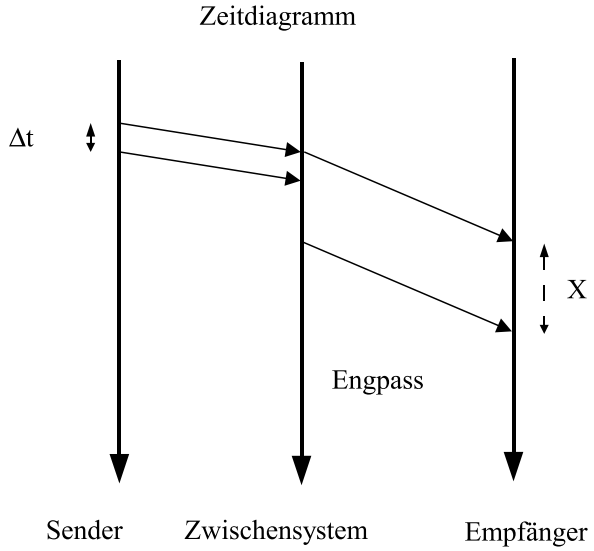


Abbildung 3.11: Packet-Pair Modell

Der Sender folgt der Überlegung: Sei $1/X$ die Bedienrate des Engpasses, dann wird der optimale Arbeitspunkt der Verbindung erreicht, wenn die Anzahl der ausstehenden Pakete $wnd = rtt \cdot 1/X$ ist.

Packet-Pair hat drei Phasen: *Starten*, *Queue-Priming* und *normale Übertragung*.

Während der Phase *Starten* werden *rtt* und $1/X$ abgeschätzt. Dazu werden zwei Pakete hintereinander gesendet. Wenn ihre Quittungen ankommen, werden $rtt_e = rtt_1$ und $x_e = rtt_2 - rtt_1$ gesetzt. Der Wert von x_e ist eine Approximation an X. Der Sender kalkuliert $wnd_e = rtt_e/x_e$ als eine Approximation an *wnd*.

Während der Phase *Queue-Priming* können bis zu wnd_e Pakete nacheinander gesendet werden.

Während der Phase *normale Übertragung* werden jeden $2 \cdot x_e$ Zeiteinheiten zwei Pakete erneut gesendet. Beim Eintreffen von je zwei Quittungen werden x_e und wnd_e aktualisiert. Falls $wnd_i < wnd_{i-1}$ gilt, wird ein Leerlauf-Intervall t_{skip} kalkuliert. Während dieses Intervalls darf nicht gesendet werden:

$$t_{skip} = max\left(\frac{wnd_{i-1} - wnd_i}{2}, 0\right)$$

Falls $wnd_i > wnd_{i-1}$ gilt, werden sofort $wnd_i > wnd_{i-1}$ Pakete gesendet. Diese Pakete werden als nicht Packet-Pair markiert und ihre RTT nicht berücksichtigt.

Packet-Pair setzt die Anwendung von FQ-Warteschlangen voraus, da sie den Einfluss des übrigen Datenverkehrs auf die RTT wieder spiegeln. Durch diese RTT kann die Verbindung ihren Anteil an der Datenrate des Engpasses abschätzen. Dies kann ein Implementierungsproblem werden, wenn die Granularität von FQ klein ist und pro Verbindung eine Warteschlange gehalten werden muss.

Die Ermittlung der Datenrate des Engpasses mit Packet-Pair verlangt, folgende störende Faktoren zu berücksichtigen [Pax99]:

- Out-of-Order Pakete oder Pakete, die nicht in der gleichen Reihenfolge beim Empfänger eintreffen, wie sie vom Sender gesendet wurden.
- Ungenauigkeit der internen Systemuhr.
- Änderungen der Datenrate des Engpasses aus Grund von Rerouting. Die Wahrscheinlichkeit von Rerouting innerhalb einer Datenübertragung ist gering, muss aber berücksichtigt werden [Pax97c].

Warp Control [Par93]. Das implizite Feedback-Signal ist die Antwortzeit, auf deren Basis die Maximierung der Auslastung ρ der Leitung gesucht wird. Die Zugangskontrolle erfolgt beim Quell-Host, indem pro Ziel-Host ein *Warp* definiert wird. Warp ist eine auf Zeitstempeln basierende Größe zur Abschätzung der Auslastung in PD-Warteschlangen. Quell- und Ziel-Host müssen die Zeitdifferenz ihrer Systemuhren feststellen[4], denn die Pakete werden vom Empfänger mit einem Zeitstempel gekennzeichnet. Der Zeitstempel wird beim Eintreffen von Quittungen im Quell-Host ausgewertet und zur Optimierung der Senderate herangezogen:

$$warp(t) = \frac{1}{c} \cdot \frac{l - last\_in}{timestamp - last\_out} \approx \rho$$

$$sr(t) = sr(t-1) + \epsilon \cdot warp(t) \qquad 0 \leq sr(t) \wedge \epsilon > 0$$

mit

$sr(t)$	Senderate zum Zeitpunkt t
c	Zeitdifferenz der Systemuhren
l	Zeit des Senders beim Eintreffen der letzten Quittung
$last\_in$	Zeit des Senders beim Eintreffen der aktuellen Quittung
$last\_out$	Zeitstempel der letzten Quittung
$timestamp$	Zeitstempel der aktuellen Quittung

3.6.3 Explizites Feedback - TCP-Friendly

Diese Verfahren sind für die Datenübertragung von Echtzeit-Applikationen vorgeschlagen. Sie sind Alternativen zu Verfahren ohne routerbasierte Überlastkontrolle. Die Datenübertragung in Echtzeit ist zeitkritisch und verlangt konstante Senderaten. Solche Applikationen werden in der Regel mit UDP zusammen mit einem anderen Applikationsprotokoll durchgeführt. Hauptziel ist, die Senderate

[4] d.h falls ein Ereignis beim Quell-Host Δi und beim Ziel-Host Δj Zeiteinheiten benötigt, dann gibt es eine Konstante $c = \Delta i / \Delta j$, die beiden Hosts bekannt ist.

stabil zu halten. Wichtig für die Stabilität ist, die Korrelation zwischen mehreren Paketverlusten zu erkennen und dies als zusammen hängende Ereignisse einer Überlastungsperiode zu bewerten. Eine Möglichkeit ist, eine TCP-ähnliche Senderate zu ermittelt. Die TCP-ähnliche Senderate ist eine Approximation an die stationäre Senderate einer TCP-Datenübertragung und basiert auf dem erwarteten Durchsatz. Ein gängiges Modell [PFTK98] ermittelt den TCP-Durchsatz D als Funktion der Fenstergröße des Empfängers, der Antwortzeit, des Paketverlusts und des Timeouts[5]:

$$D \approx f(rwnd, rtt, p, t_{rto})$$

Es gibt einige Richtlinien für diese Verfahren. Erstens, die Senderate muss als Funktion der Paketverlustrate langsam erhöht bzw. gesenkt werden. Zweitens, die Senderate wird erst nach Empfangen von mehreren negativen Feedback-Signalen (Paketverlust) halbiert. Drittens, der Empfänger soll mindestens einmal pro RTT ein Feedback-Signal senden. Viertens, die Senderate muss gesenkt werden, falls nach mehreren RTT kein Feedback-Signal empfangen wird.

TCP-Friendly Rate Control Protocol (TFRCP) [PKTK98] . Der Sender arbeitet in *rounds* oder Perioden der Länge Δt. Δt ist das Intervall für die Anpassung der Senderate. Am Anfang der Periode i wird die Senderate sr_i und die Anzahl von Paketen $n_i = sr_i \cdot \Delta t$ kalkuliert. Jedes Paket hat eine Sequenznummer und einen Zeitstempel. Der Empfänger quittiert dieses Paket und fügt einen 8-stelligen Bit-Vektor im Paket-Header hinzu. Jedes Bit dieses Vektors gibt an, ob die vorherigen 8 Pakete empfangen werden. Der Sender benutzt diesen Bit-Vektor für die Berechnung der Senderate:

$$p_i = \frac{y_i}{x_i + y_i}$$

$$sr_i = \begin{cases} sr_0 & \text{falls } i = 0 \\ 2 \cdot sr_{i-1} & \text{falls } y_{i-1} = 0 \\ f(rwnd, rtt, p_{i-1}, B) & \text{falls } y_{i-1} \neq 0 \end{cases}$$

mit

- p_i Paketverlustrate der Periode i
- x_i Anzahl der empfangenen Pakete in Periode i
- y_i Paketverlust-Vektor der Periode i
- sr_0 Initialisierungs-Wert (i.a. $sr_0 = 40$ Pakete/s)

Equation-based TCP-Friendly Protocol (TFRC) [FHPW00] . Es werden die gleichen Prinzipien wie im TFRCP angewendet, darunter die Notwendigkeit die Senderate anstatt des CWND zu benutzen. Die Paketverlustrate wird beim Empfänger berechnet und in den Paket-Header geschrieben. Der Sender berechnet daraus die Senderate:

berechne sr_{max}

$$sr_i = \begin{cases} sr_{max} & \text{falls} \quad sr_{i-1} > sr_{max} \\ sr_{i-1} + \Delta sr & \text{falls} \quad sr_{i-1} < sr_{max} \end{cases}$$

[5] Im Abschnitt 4.5.1 wird auf die Details dieser Funktion eingegangen.

3.6.4 Explizites Feedback - Andere Verfahren

TCP/CM [BRS99]. TCP/CM ist eine Architektur zur Ende-zu-Ende-Überlastkontrolle mit dem Ziel, die Datenübertragungen aller Applikationen eines Hosts zu koordinieren. Die Datenübertragungen können HTTP und FTP über TCP oder Echtzeit-Verkehr über UDP sein. Dafür gibt es in jedem Host einen Congestion-Manager (CM), der Information sammelt, Statistiken über den Netzzustand pflegt und den ausgehenden Verkehr reguliert. CM besteht aus einer Sender- und einer Empfänger-Komponente.

Der Sender-CM besteht aus: *Congestion-Controller*, *Flow-Scheduler* und *Prober*.

Der Congestion-Controller schätzt die Last ab und koordiniert die Senderaten zwischen Sendern und Empfängern. Feedback-Signale werden aus den Applikationen und aus dem Prober gewonnen. Die Autoren charakterisieren dieses Verfahren als eine hybride CWND-Senderate Methode, denn das CWND jeder Verbindung wird mit Hilfe der abgeschätzten Senderate angepasst. Die Feedback-Signale sind Paketverlust, DECBit-Signale und das Empfängerfenster.

Der Flow-Scheduler koordiniert die Arbeit der Applikationen.

Der Prober sendet zweimal pro RTT ein Kontrollpaket zur Abschätzung der Datenrate. Dieses Paket beinhaltet eine eindeutige steigende Sequenznummer. Der Empfänger antwortet mit einer Quittung, die diese Sequenznummer, die letzte Sequenznummer und die Anzahl der empfangenen Pakete zwischen den zwei Nachrichten beinhaltet. Nach Empfang dieser Nachricht berechnet der Prober die Paketverlustrate zwischen den zwei Proben. Die Paketverlustrate wird nur für die Überlastkontrolle verwendet.

Der Empfänger-CM besteht aus: *Loss-Detector*, *Responder* und *Hints-Dispatcher*. Der Loss-Detector pflegt die Statistiken über Paketverlust mit der Information des Paket-Headers. Der Responder pflegt die Statistiken der empfangenen Datenmenge pro Verbindung. Der Hints-Dispatcher bearbeitet die Befehle zur Datenübertragung der Applikationen.

3.7 Regelungsverfahren - Ende-zu-Ende- und routerbasierte Überlastkontrolle

Diese Verfahren sind durch in den Routern generierte explizite Feedback-Signale charakterisiert. Im Gegensatz zu den Steuerungsverfahren liegt der Schwerpunkt in der Weiterleitung der Signale an die Quelle. Die Last wird durch eine koordinierte Aktion der Router, der Sender und der Empfänger kontrolliert, was wiederum ein Nachteil darstellen kann, wenn dies nicht möglich ist. Diese Verfahren können jedoch Robustheit und Stabilität erreichen. Hier werden drei Arten von Feedback-Signalen betrachtet: ein Kontrollpaket, ein Bit im Paket-Header und quantitatives Feedback-Signal.

3.7.1 Kontrollpaket als Feedback-Signal

Die Feedback-Signale werden von einem Router mittels eines Kontrollpakets weitergeleitet, und zwar nur dann, wenn gewisse Bedingungen erfüllt sind. Diese Bedingungen beziehen sich auf die Auslastung des Routers oder der Leitung. Der Nachteil dieser Verfahren ist, dass gerade wenn die Netzauslastung hoch ist, zusätzlicher Verkehr erzeugt wird. Je nach Auslastungsgrad ist dies vernachlässigbar.

Problematischer ist, wenn diese Pakete auf Grund der hohen Auslastung selbst verloren gehen oder zu spät ankommen.

Choke-Packet [VSN86]. Bei der Überschreitung eines Schwellwertes der Auslastung (z.B. 75%) der Leitung wird der Router in einen Alarmzustand versetzt. Die Auslastung ρ der Leitung wird dynamisch festgestellt, mit:

$$\rho_i = w_q \cdot \rho_{i-1} + (1 - w_q) \cdot f_{i-1}$$

und w_q als ein Parameter, der aussagt, wie schnell der Router seine Vorgeschichte vergisst. Der Wert von f ist entweder 0 oder 1 und entspricht der aktuellen Auslastung der Leitung. Kommen danach weitere Pakete an, wird ein Kontrollpaket zum Quell-Host gesendet. Das Paket, welches das Senden eines Kontrollpakets verursacht, wird markiert, damit es in keinem anderen Router weitere Kontrollpakete erzeugt werden.

Die Quell-Hosts reduzieren ihre Senderate, wenn Kontrollpakete empfangen werden.

Source-Quench [PP87]. Kontrollpakete werden in den Routern oder in einem Ziel-Host generiert, wenn die Puffer nicht ausreichen oder wenn die Pufferauslastung eine Schwelle überschreitet. Pro verworfenes Paket kann ein Kontrollpaket gesendet werden. Nach Empfang eines Kontrollpakets reduziert der Quell-Host seine Senderate. Weitere Kontrollpakete werden für eine bestimmte Zeitdauer ignoriert. Falls keine zusätzlichen Kontrollpakete eintreffen, kann die Senderate eventuell erhöht werden.

Eine mögliche Implementierung kann durch ICMP-Pakete [Pos81] erfolgen[6]. Ein Nachteil ist, dass der Sender nicht wissen kann, unter welchen Umständen Kontrollpakete erzeugt worden sind: war die Ursache eine entstehende, eine bestehende oder eine transiente Netzüberlastung?

Ein weiterer Nachteil ist die Reaktionszeit des Systems: ist t die Zeit zwischen dem ausgelasteten Router und dem Host und B die Datenrate, werden weitere $t \cdot B$ Daten gesendet, bevor der Quell-Host reagiert. Vor allem in Gigabit-Netzen ist dies problematisch, denn B kann sehr groß sein und der Router wird keine schnelle Wirkung auf sein Feedback-Signal erfahren.

Hop-By-Hop Control [MK92]. Jeder Router sammelt Information über die Pufferauslastung pro Verbindung. Überschreitet die Auslastung einen Schwellwert, wird ein Kontrollpaket an den nächsten *Upstream-Router* gesendet. Upstream-Router ist der Nachbar des Routers gegen den Datenfluss. Der Upstream-Router passt die Bedienrate der Verbindung als Funktion der Variation der Auslastung dynamisch an. Empfängt ein Upstream-Router ein Kontrollpaket, würde er seine Senderate senken und seine Warteschlangen würden wachsen. Dennoch würde der Router, der das Kontrollpaket gesendet hat, sofort entlastet. Diese Aktion wiederholt sich in jedem Router, bis ein Kontrollpaket den Quell-Host erreicht.

[6] Nach der Spezifikationen des RFC1716 sollten Kontrollpakete nicht generiert werden.

3.7.2 Bit im Paket-Header als Feedback-Signal

In den folgenden Verfahren wird unter gewissen Auslastungskriterien und pro Router ein Bit im Paket-Header gesetzt. Das Auslastungskriterium kann z.B. sein, dass die mittlere Warteschlangenlänge einen Schwellwert überschreitet. Somit haben die Sender auch nicht mehr Information über die Auslastungsniveaus als mit den vorherigen Verfahren, aber sie können auf die Information von mehreren Routern zugreifen und sich damit ein Gesamtbild ermitteln.

DECBit (Binary-Feedback) [JRC87, RR88]. Der Schwellwert der Warteschlangen wird so gewählt, dass die Router im Knee-Punkt arbeiten, d.h. dass die Power maximiert wird.

Für die Berechnung der mittleren Warteschlangenlänge soll eine Periode zum Ausfiltern transienter Netzüberlastung genommen werden: das Intervall besteht aus einem Wiederherstellungszyklus und aus der aktuellen aktiven Periode. Ein Wiederherstellungszyklus besteht aus einer aktiven und aus einer inaktiven Periode.

Der Sender erhält einen Vektor mit n Bits, mit jeweils einem Bit pro Router. Falls der Anteil der gesetzten Bits den Wert $p_n = 0.5$ überschreitet, wird das CWND verkleinert.

$$cwnd_i = \begin{cases} cwnd_{i-1} + \alpha & \text{falls } p_n \leq 0.5 \\ \beta \cdot cwnd_{i-1} & \text{falls } p_n > 0.5 \end{cases}$$

Mit $\alpha = 1$ erfolgt das Vergrößern des CWND wie bei Slow-Start. Mit $\beta = 7/8$ wird ein höherer Durchsatz als bei Slow-Start erreicht, da das CWND früher, aber weniger verkleinert wird.

Die Auswertung der Bits findet entweder beim Empfänger oder beim Sender statt. Im ersten Fall wird mittels den Bits ein neues Empfängerfenster berechnet und in den Paket-Header geschrieben. Im zweiten Fall kopiert der Empfänger die Bits des Paket-Headers des ankommenden Pakets in den Paket-Header der Quittung, damit der Sender sie auswerten kann.

Selective-Binary-Feedback [RJC91]. Dieses Verfahren ist DECBit ähnlich, versucht jedoch mehr Fairness zu erreichen bei gleichzeitigem Minimieren der Anzahl von Feedback-Signalen. Die Pakete werden pro Quell/Ziel-Host in Klassen aggregiert und nur diejenigen Klassen, die aggressive Senderaten anwenden, werden aufgefordert, sie zu senken, sobald der Router über den Knee-Punkt arbeitet. Dafür ist es notwendig, sie zu identifizieren. Dies erfolgt indem in jedem Router ein Vektor gehalten wird, der die Anzahl an ankommenden Paketen pro Klasse während eines mittleren Intervalls beinhaltet. Dies ist eine Approximation für den Durchsatz. Andere Alternativen zur Aggregation der Pakete sind:

- Pro Input-Output Link: Der Router hat eine niedrige Prozessorbelastung, aber Fairness für Verbindungen mit langen Pfaden ist nicht garantiert.
- Pro Transportprotokoll: In diesem Fall kann der Router zwischen den unterschiedlichen Applikationen unterscheiden.

Nachteile dieses Verfahrens sind die Prozesszeit in den Routern und die fehlende Garantie für Fairness innerhalb einer Klasse.

Q-bit [Ros93]. Basiert auf DECBit, ist jedoch nicht fenster- sonder senderatenbasiert. Die Entscheidung, Senderaten statt Fenster anzuwenden, setzt auf der Analyse der Reaktionszeit des Protokolls beim Empfangen eines Feedback-Signals auf. Ein weiterer Unterschied ist der Zeitpunkt zum Setzen des Q-bits. Dies erfolgt sobald das Paket nicht unmittelbar weitergeleitet wird und in einer Warteschlange gepuffert wird.

3.7.3 Quantitatives Feedback-Signal

Bei diesen Verfahren ist das von den Routern gesendete Feedback-Signal eine quantitative Angabe, die der Sender zur Einstellung der Senderate verwenden kann. Um diese Feedback-Signale zu generieren, setzen die Router Mechanismen zur Abschätzung der Auslastung und der Anzahl der Verbindungen ein.

BBN [RFS90]. Durch den Einsatz eines Update-Protokolls wird die Verbreitung von Feedback-Signalen im Netz angestrebt. Sie werden an die Quell-Hosts gesendet.

Periodisch wird in den Routern die geeignete *Resource-Fraction* für jeden Datenfluss berechnet. In diesem Kontext umfasst ein Datenfluss den ganzen Verkehr zwischen zwei Hosts, d.h. alle Verbindungen zwischen ihnen. Die Resource-Fraction ist die höchste Senderate mit minimaler Wahrscheinlichkeit von Überlastung.

Im Router wird die nächste Resource-Fraction sr_soll_i mit der gewünschten sr_{opt} und der gemessenen Resource-Fraction der letzten Periode sr_ist_{i-1} berechnet. Diese Information wird in den Paket-Header des Update-Protokolls hinzugefügt.

$$sr\_soll_0 = sr_{opt}$$
$$sr\_soll_i = min\left(sr_{opt}, \frac{sr_{opt}}{sr\_ist_{i-1}} \cdot sr\_soll_{i-1}\right)$$

Jeder Quell-Host bewertet die Resource-Fraction im Paket-Header und kalkuliert für jeden Datenfluss eine *Dataflow-Fraction*. Die Dataflow-Fraction ist dabei das Minimum aller empfangenden Resource-Fraction.

Dynamic-Time-Windows (DTW) [MLF92b]. DTW ist ein Verfahren zur Zugangskontrolle, das mit einer Variante von Virtual-Clock, genannt Pulse-Queuing [MLF92a] arbeitet. DTW basiert auf Ressourcen-Reservierung innerhalb von Virtual-Circuit (VC). Jeder Verbindung ist einem VC zugeordnet. Im VC haben alle Pakete den gleichen Pfad.

Jedem VC wird ein maximales *Time-Window* (MTW) tw_{max} zugewiesen, dessen Größe während der Verbindung dynamisch variieren kann. Ein Time-Window ist ein Intervall zur Berechnung der mittleren Senderate. Nach Starten bzw. Beenden von neuen Verbindungen wird der neue Wert allen Sendern mitgeteilt.

Der Sender spezifiziert beim Verbindungsaufbau die Senderate sr, die maximale Senderate sr_{max} und die Time-Window-Fraction r. Das Time-Window tw dieses VC ist:

$$tw = r \cdot tw_{max} \qquad 0 \leq r < 1$$

Dies ergibt eine maximale Senderate von $sr_{max} \cdot tw$. Der Sender überwacht den Durchsatz nicht, versucht jedoch, seine Varianz innerhalb des Time-Window stabil zu halten. Dies stellt eine indirekte Kontrolle der Warteschlangenlängen dar.

Vorteile von DTW sind das Minimieren von Paketverlust und das Erreichen von hohen Durchsätzen in Verbindungen mit burst-artigem Verkehr.

Loss-Load-Curve [Wil93]. Die Router überwachen die Auslastung ihrer Warteschlangen und ermitteln für jede Verbindung eine Paketverlust-Auslastungs-Funktion. Diese Information wird dem Sender mitgeteilt, der eine Abschätzung einer Durchsatz-Funktion durchführt. Der Sender hat dann die Verantwortung, die geeignete Senderate auszuwählen. Dadurch wird die Abhängigkeit von der RTT und von der Anzahl der Router im Pfad erreicht, wobei eine schnelle Anpassung bei Veränderungen der RTT erfolgt.

Explicit Rate Indication for Congestion Avoidance (ERICA) [KJF+97]. ERICA ist ein Verfahren für den ABR-Service von ATM. ABR steht für *Available Bit Rate*. Ziel ist eine hohe Auslastung der Leitungskapazität, die für die ABR-Klasse zur Verfügung steht. Die Auslastung wird mittels eines Auslastungsfaktors ρ_z gemessen, welches der Quotient zwischen ABR-Ankunftsrate und ABR-Kapazität ist. Der Router stellt periodisch den Auslastungsfaktor und die Anzahl von aktiven Verbindungen fest. Da für jede Verbindung Speicherkapazität reserviert wird, muss die Ermittlung der Anzahl von aktiven Verbindungen genau sein. Dies erfolgt unter Einsatz der Technik des *bidirectional counting*. Dabei werden Zellen in beide Richtungen gezählt. Anschließend wird die Senderate für jede Verbindung festgestellt und an den Sender zurückgemeldet.

Explicit Control Protocol (XCP) [KHR02]. Das Explicit Control Protocol (XCP) ist eine Verallgemeinerung der Prinzipien von DECBit. Der Ansatz ist, dass TCP instabil und ineffizient wird, sobald die Pfadkapazität einer Verbindung steigt, unabhängig davon, welches AQM-Verfahren angewendet wird. Die Instabilität entsteht durch den Einsatz von Slow-Start und dessen Auswirkungen auf den Router: Exponentiell wachsende Senderaten führen zum Paketverlust. Ineffizienz entsteht, da die benötigte Zeit für die Bewertung des Feedback-Signals von der RTT der Verbindung abhängt. Ist die Pfadkapazität der Verbindung groß, so ist die Anpassung der Verbindung dem sprechenden langsam. Es wird dafür plädiert, dass eine effiziente Überlastkontrolle präzise Feedback-Signale erzeugen soll, und dass Paketverlust kein geeignetes Überlastungssignal darstellt. Der Grund für letzteres ist die Anwendung innerhalb von Mobilfunk-Netzen, wo der Verlust mit der Netzüberlastung nicht gekoppelt ist.

XCP besteht aus einem fensterbasierten Sender, einem TCP-ähnlichen Empfänger und einem Router. Das Feedback-Signal wird im Paket-Header weitergeleitet. Der Paket-Header wird um drei Felder erweitert: `H_cwnd`, `H_rtt` und `H_Feedback`. Die ersten zwei Felder sind das CWND und die abgeschätzte RTT der Verbindung, welche vom Sender gesetzt und vom Empfänger kopiert werden. Das dritte Feld ist die vom Router gewünschte Änderung des CWND (Vergrößern bzw. Verkleinern) zur Maximierung der Effizienz und Fairness bei sich selbst. Mit einer Paketgröße von b wird der Sender beim Empfang einer Quittung sein CWND nach folgender Gleichung bestimmen:

$$cwnd_i = max(cwnd_{i-1} + feedback, b)$$

Da ein weiterer Ansatz die Trennung der Aufgaben eines Routers zwischen Effizienz- und Fairness-Kontrolle ist, besteht der Router aus zwei Komponenten: Efficiency-Controller (EC) und Fairness-Controller (FC). Der EC maximiert die Auslastung der Leitung und der FC sorgt für eine gerechte Aufteilung der Leitungskapazität zwischen den Verbindungen. Der EC berechnet einen Parameter ϕ. $\phi > 0$ impliziert eine nicht vollständig ausgelastete Leitung, $\phi < 0$ eine überlastete Leitung. Basierend auf der mittleren RTT $\overline{rtt}$, der freien Leitungskapazität B_F, der Warteschlangenlänge q, sowie zwei Parametern $\alpha = 0.4$ und $\beta = 0.226$, kann ϕ folgendermaßen berechnet werden:

$$\phi = \alpha \cdot \overline{rtt} \cdot B_F - \beta \cdot q$$

Dann kann der FC ein positives $feedback\_p_i$ oder ein negatives Feedback $feedback\_n_i$ pro Verbindung i erzeugen:

$$feedback\_p_i = \xi_p \cdot \frac{rtt_i^2}{cwnd_i}$$
$$feedback\_n_i = \xi_n \cdot rtt_i$$

Das positive Feedback ist umkehrt proportional zum aktuellen Durchsatz und direkt proportional zur RTT der Verbindung. Somit vergrößern Sender mit wenig Durchsatz und/oder längeren RTT ihr CWND schneller. Ein negatives Feedback ist proportional zur aktuellen RTT.

Die Konstanten ξ_p und ξ_n errechnen sich wie folgt:

$$\xi_p = \frac{h + max(\phi, 0)}{\overline{rtt} \sum \frac{rtt_i}{cwnd_i}}$$
$$\xi_n = \frac{h + max(-\phi, 0)}{\overline{rtt} \cdot N}$$
$$h = max(0,\ \gamma \cdot N - \mid \phi \mid)$$

Mit N als die Anzahl von empfangenen Paketen in einer durchschnittlichen Periode und $\gamma = 0.1$ wird die Umverteilung von 10% des Verkehrs jede RTT durchgeführt.

XCP kann große Stabilität in Gigabit-Netzen erreichen und hat den Vorteil, dass keine Zustandsvariablen pro Verbindung gehalten werden müssen. Allerdings ist die Änderung des CWND proportional zu seinem aktuellen Wert. Bei einem positiven Feedback konvergieren die Durchsätze von Sendern mit unterschiedlichen RTT langfristig. Da das Vergrößern des CWND vom aktuellen Durchsatz abhängt, kann diese Anpassungszeit für kurze Datenübertragungen zu lang sein, um positive Resultate zu liefern.

3.8 Bewertung der verschiedenen Überlastkontrolle-Verfahren

Es gibt einen grundsätzlichen Zielkonflikt zwischen der Qualität und den Kosten zum Erzeugen von Feedback-Signalen: je knapper die Feedback-Signale sind, desto weniger Information über den Netzzustand bereitgestellt wird.

Die Ziele der Überlastkontrolle können sich widersprechen, denn die beste Antwortzeit bedeutet nicht zwangsläufig den besten Durchsatz, oder der beste Durchsatz für eine Verbindung kann Mangel an

Fairness für das gesamte Netz bedeuten.

In der Analyse der Verfahren zur Überlastkontrolle sind einige grundsätzliche Fragen analysiert worden: Welche Vorteile bringen Steuerungs- gegenüber Regelungsverfahren? Soll Ende-zu-Ende-Überlastkontrolle dem Vorzug gegenüber routerbasierte Überlastkontrolle gegeben werden? Welche Art von Feedback-Signalen soll gesendet werden und wann?

Es gibt keine Standard-Antworten auf diese Fragen für alle möglichen Anwendungsszenarios, zumal die Technik sich ständig ändert, die Anforderungen wachsen und neue Applikationen hinzukommen. Die Wahl eines Verfahrens hängt primär von den gesetzten Zielen, der Verkehrsart und der Protokollschicht ab. Routerbasierte und Ende-zu-Ende-Überlastkontrolle können sich ergänzen, wobei sie in der Regel getrennt entwickelt werden. So zielt die routerbasierte Überlastkontrolle darauf ab, eine allgemeine Strategie zur Pufferbehandlung und zur Aufteilung der Leitungskapazität zu realisieren.

Eine Zusammenfassung der Eigenschaften der hier dargestellten Verfahren findet man in den Tabellen 3.2, 3.3 und 3.4. Diese Tabellen sind wie folgt strukturiert: Klassifizierung, angewendete Lösungsansätze, eingesetzte Feedback-Signale und summarische Bewertung.

	Klassifizierung					**Ansatz**							**Bewertung**		
METHODE	**reaktiv präventiv**	**nach [YR95]**	**Host Router**	**Senderate WND**		**RS**	**Zugangskontrolle**	**Puffergröße**	**Scheduling**	**AQM**	**Warteschlangen**	**KP**	**Fairness-V**	**Fairness-K**	**E.agg.Sender**
Isarithmatic	x	Z	x				x				1				
PD	x	Z	x					x			1				
RED	x	Z	x							x	1				
FRED	x	Z	x							x	1				x
SRED	x	Z	x							x	1				x
REM	x	Z	x							x	1				x
PI	x	Z	x							x	1				x
ARED	x	Z	x							x	1				
Input-Buffer-L.	x	Q	x				x	x	x		n			x	
FQ	x	Q	x						x	x	n			x	
B-Round FQ	x	Q	x						x	x	n			x	
SFQ	x	Q	x						x	x	n			x	
Virtual-Clock	x	Q	x x	x		x	x		x		n	x	x		
IntServ	x	Q	x			x	x		x		n		x	x	
DiffServ	x	Q	x			x			x	x	n			x	
LEGENDE:	**Q=Quellkontrolle, Z=Zielkontrolle, RR=Ressource-Reservierung, KP=Kontrollpaket, Fairness-V=Fairness für jede Verbindung Fairness-K=Fairness für jede Klasse E.agg.Sender=Erkennung von aggressiven Sendern**														

Tabelle 3.2: Eigenschaften von Steuerungsverfahren

	Klassifizierung				Ansatz	Feedback					Bewertung		
METHODE	**reaktiv präventiv**	**nach [YR95]**	**Host Router**	**Senderate WND**	**RS**	**Timeout**	**RTT**	**Paketverlust**	**Auslastung**	**KP**	**Durchsatz opt.**	**Auslastung absch.**	**Stabile Senderate**
Implizites Feedback													
CUTE	x		x	x		x							
Slow-Start	x		x	x		x							
CARD	x		x	x		x	x				x		
Tri-S	x		x	x		x	x				x		
P.-Pair	x		x	x		x	x						x
DUAL	x		x	x		x	x		x			x	
Warp-C.	x		x	x		x	x		x			x	
VEGAS	x		x	x		x	x		x		x		x
LEGENDE:	**P=Persistent, R=Reagierend, G=Global, L=Lokal RR=Ressourcen-Reservierung, KP=Kontrollpaket**												

Tabelle 3.3: Eigenschaften von Regelungsverfahren mit impliziten Feedback-Signalen

3.8.1 Steuerungs- oder Regelungsverfahren

Wegen der großen Pfadkapazität in Gigabit-Netzen wird oftmals argumentiert, dass Steuerungs- gegenüber Regelungsverfahren geeigneter seien. Denn mit Regelungsverfahren erreichen die Feedback-Signale den Regler nicht schnell genug, um schnell eine hohe Auslastung der Leitung zu erzielen. Dabei werden die Verbindungen mit großer RTT benachteiligt.

Probleme der Steuerungsverfahren liegen in der Schwierigkeit, gültige Modelle bzw. geeignete Parameter zu finden. Es ist eine komplexe Aufgabe, adäquate Werte für die Warteschlangenlängen, ihre Anzahl, usw. und dies bei ständiger sich ändernder Last, wie dies für das Internet typisch ist. Die Entscheidung, wie viele Warteschlangen welchen Typs notwendig sind, bestimmt, inwieweit Fairness erreicht wird. Hier gilt jedoch: Je mehr Klassen, desto höher der Aufwand im Router.

Ein wichtiger Mechanismus der Steuerungsverfahren ist die Ressourcen-Reservierung. Sie ist für den Echtzeit-Verkehr vorteilhaft, denn diese Verkehrsart ist planbar. Dementsprechend wird die Auslastung hoch sein. Bei burst-artigem Verkehr werden demgegenüber oftmals reservierte Ressourcen nicht in Anspruch genommen.

3.8.2 Ende-zu-Ende- oder routerbasierte Überlastkontrolle

Ende-zu-Ende-Überlastkontrolle hat den Nachteil, dass die Information immer erst vom Sender zum Empfänger fließen muss, bevor der Empfänger eine Rückmeldung geben kann. Dadurch kann die Anpassungszeit langsam sein. Wie oben erwähnt, kann dies eine niedrige Auslastung in Gigabit-Netzen verursachen.

Ende-zu-Ende-Überlastkontrolle ist ein notwendiger, jedoch nicht ausreichender Mechanismus zur

Überlastkontrolle. Die Begrenzung der Senderate beim Sender ist eine präventive Maßnahme zur Vermeidung von Netzüberlastung. Sie kann durch Strategien zur routerbasierten Überlastkontrolle ergänzt werden.

Für routerbasierte Verfahren spricht, dass sie die effektivste Kontrolle durchführen können und bei geeigneter Implementierung stabiles Laufzeitverhalten aufweisen. Router können zwischen Übertragungszeit und Wartezeit unterscheiden. Sie haben als einzige Komponenten genauen Überblick über die Warteschlangenlängen als Funktion der Zeit. Dadurch können sie den Grad der Netzüberlastung kurzfristig erkennen. Ein weiterer Vorteil der routerbasierten Überlastkontrolle besteht darin, das Fairness erreicht werden kann. Der Nachteil dieser Verfahren liegt jedoch in der Notwendigkeit, Änderungen in allen beteiligten Routern vorzunehmen.

3.8.3 Verfahren für Best-Effort-Datenübertragung

Ansätze zur Überlastkontrolle für Best-Effort-Datenübertragungen können in drei Gruppen unterteilt werden. Die erste Gruppe umfasst die Regelungsverfahren mit Pufferbehandlung wie zum Beispiel RED, FRED, SRED, REM, PI und ARED. Diese Verfahren verwerfen Pakete in Warteschlangen gezielt, falls deren Länge vorgegebene Grenzen überschreiten. Damit soll der Sender frühzeitig ei-

	Klassifizierung						**Ansatz**	**Feedback**					**Bewertung**		
METHODE	**reaktiv**	**präventiv**	**nach [YR95]**	**Host**	**Router**	**Senderate**	**WND**	**RS**	**Timeout**	**RTT**	**Paketverlust**	**Auslastung**	**KP**	**Fairness-V**	**Fairness-K**	**E.agg.Sender**
Explizites Feedback																
Choke-P.	x		R-G	x	x							x	x			
S.-Quench	x		R-G	x	x							x	x			
DECBit		x	P-G	x	x		x					x				
BBN		x	P-G		x	x						x	x		x	
Sel.BFeed.		x	P-G	x	x	x						x			x	
Hop-H.		x	P-L	x	x								x			
DTW		x	R-G		x	x		x					x	x		
Q-bit		x	P-G	x	x	x						x				
Loss-Load		x	P-G	x	x	x					x	x		x		
ERICA		x	P-G	x	x		x		Senderate					x		
TFRCP		x	P-G	x		x			x	x	x					
TCP/CM		x	P-G	x		x	x		x		x	x	x			
TFRC		x	P-G	x		x			x		x					
XCP		x	P-G	x	x		x		WND Änderung				x			x

LEGENDE:	**Q=Quellkontrolle, Z=Zielkontrolle, RR=Ressource-Reservierung, KP=Kontrollpaket, Fairness-V=Fairness für jede Verbindung Fairness-K=Fairness für jede Klasse E.agg.Sender=Erkennung von aggressiven Sendern**

Tabelle 3.4: Eigenschaften von Regelungsverfahren mit expliziten Feedback-Signalen

ne sich bildende Netzüberlastung erkennen. Die zweite Gruppe zieht weiterhin die Anwendung von Ende-zu-Ende-Überlastkontrolle vor. Zu dieser Gruppe zählen VEGAS, TFRCP, TCP/CM und FRTP. Die dritte Gruppe stellt eine Kombination aus Router- und Ende-zu-Ende-Verfahren dar. Diese sind meist komplex, bieten jedoch den Vorteil eines sehr stabilen Betriebs. Vertreter dieser Gruppe sind DECBit, Q-bit und die RED-Varianten im Markierungs-Modus. Bei diesen Verfahren teilt jeder Router entlang des Pfades seine Überlastung durch ein Bit im Paket-Header den Sendern mit. Die Sender werten die Bits aus und verändern dementsprechend ihre Senderate bzw. ihre Fenstergröße. Die Verfahren ERICA und XCP setzen anstatt des Bits die gewünschte Senderate bzw. die Änderung des Fensters explizit im Paket-Header ein und erreichen dadurch mehr Fairness.

Aktuelle Standards der Überlastkontrolle für Best-Effort-Verkehr basieren auf dem Einsatz von Warteschlangendisziplinen des Typs PD oder RED in den Routern. Das Erreichen von hohen Durchsätzen durch RED-Varianten ist weiterhin fraglich, zumal niedrige Paketverlustraten ausreichen können, um den Durchsatz drastisch zu senken.

Verfahren, welche Ende-zu-Ende- mit routerbasierter Überlastkontrolle kombinieren, erscheinen viel versprechend. Sie weisen sehr gute Ergebnisse in der Simulation ebenso wie in der mathematischen Analyse auf. Die Schwierigkeit liegt darin, dass Änderungen in sämtlichen Komponenten (Host und Router) des gesamten Netzes erforderlich wären, und diese während der Entwicklung der Verfahren nur schwer oder praktisch gar nicht im Labor implementiert werden können. Dadurch ist man gezwungen, die notwendigen Entwurfentscheidungen überwiegend auf Analyse und Simulation abzustützen. Dies erfordert aufwendige Untersuchungen, bevor in realen Netzen Tests durchgeführt werden können. Die Implementierung spielt für die Laufzeitsysteme in Hosts und Router eine große Rolle. Obwohl diese Verfahren mehr Kosten verursachen, ist anzunehmen, dass sie den komplexen Anforderungen innerhalb des Internet am ehesten gerecht werden.

Kapitel 4

Überlastkontrolle in TCP

TCP, das am meisten benutzte Transportprotokoll, ist in RFC793 [Pos85] spezifiziert. Die Implementierung ist in RFC2581 [APS99] ausführlich beschrieben. Die Standard-Implementierung wird als 4.x BSD System bezeichnet und wird als Quelle für viele andere Implementierungen benutzt. Einige TCP-Varianten gelten als Industrie-Standard (TCP-Reno, TCP-Tahoe, TCP-SACK, TCP-Newreno), und andere als akademische Vorschläge (TCP-CARD, TCP-TRI-S, TCP-DUAL und TCP-VEGAS (siehe Abschnitt 3.6.1). Die TCP-Varianten unterscheiden sich von einander vor allem in der Art, wie die Überlastkontrolle realisiert wird und wie der Paketverlust verwaltet wird (Recovery-Phase). In diesem Kapitel werden Eigenschaften des Protokolls untersucht, die relevant für die Überlastkontrolle sind. Anschließend werden Leistungs- und Funktionsmodelle gezeigt, ergänzt durch eine Fallstudie.

4.1 Wichtige Merkmale und Zustandsvariable

TCP ist ein komplexes fensterbasiertes verbindungsorientiertes Protokoll, dessen Funktionalität u.a. Fehlerkorrektur, zuverlässigen Datentransport und Vollduplex-Verkehr mit einschließt. Folgende Eigenschaften bestimmen die Leistungskenngrößen einer TCP-Datenübertragung.

4.1.1 Maximum Segment Size (MSS)

Die Transporteinheit in TCP wird Segment genannt. Als maximale Segmentgröße (MSS) wird jedoch die Größe der Applikationsdaten im TCP-Header bezeichnet. Der Wert von MSS wird basierend auf dem Vorgabewert für MTU (Maximum Transmission Unit) in der Verbindungsaufbauphase mitgeteilt. MTU ist das größte IP-Datagramm, das ohne IP-Fragmentierung übertragen werden kann und hängt vom Übertragungsmedium ab. Ein Datagramm kann dabei in mehrere IP-Fragmente geteilt werden. Die Fragmentierung sollte vermieden werden, da dabei der Durchsatz verringert wird. Denn die Zunahme von Datagrammen in den Routern erhöht die RTT und daraus die Wahrscheinlichkeit zum Verwerfen eines Datagramms. Außerdem muss nach dem Verlust eines einzelnen IP-Fragments das ganze TCP-Segment erneut gesendet werden. Da IP- und TCP-Header insgesamt 40 Bytes haben, wird mit der Benutzung $MSS = MTU - 40$ Bytes die Fragmentierung vermieden. Es muss berücksichtigt werden, dass durch die Anwendung von TCP-Optionen der Paket-Header größer wird.

4.1.2 Receiver-Window (RWND)

Das Receiver-Window (RWND) wird vom Empfänger zur Flusskontrolle angeboten. Detaillierte Forschungsergebnisse über den Einfluss auf den Durchsatz können [Mog93, PP93] entnommen werden.

Das 16-Bit lange Options-Feld des TCP-Headers ermöglicht ein RWND von höchstens 65535 Bytes. Bei einer großen Pfadkapazität kann die Leitung wesentlich größere Datenmengen aufnehmen. Die WindowScale-Option ermöglicht das Vergrößern des RWND mittels eines Feldes im Paket-Header, das als Exponent benutzt wird. Da der Wert dieses Exponents zwischen 0 und 14 liegt, kann das RWND auf $65535 \cdot 2^{14}$ erhöht werden.

Die Datenübertragung kann vom Empfänger temporär gestoppt werden, indem er das RWND auf Null setzt. Der Sender geht dadurch in den Persist-Modus: Periodisch wird ein Byte gesendet, um zu überprüfen, ob der Empfänger eventuell wieder bereit ist, das RWND zu erhöhen und die Datenübertragung fortzufahren.

4.1.3 Congestion-Window (CWND)

Das Congestion-Window (CWND) ist die vom Sender gesetzte Schwelle und wird zur Überlastkontrolle verwendet. Da es in Segmenten gemessen wird, ist ihr Byte-Zuwachs größer, wenn Segmente mit großer MSS gesendet werden.

TCP vergrößert und verkleinert das CWND durch zwei Mechanismen: Slow-Start und Congestion-Avoidance [Jac88]. Sowohl während Slow-Start wie auch während Congestion-Avoidance kann der Sender nur so viele Segmente *in Transit halten*, wie das Minimum zwischen CWND und RWND. Dadurch berücksichtigt der Sender gleichzeitig die Fluss- und die Überlastkontrolle.

Der Schwellwert für CWND, SSTH genannt, wird zur Auswahl des Mechanismus zum Vergrößern von CWND angewendet:

$$\begin{cases} \text{Slow-Start} & \text{falls} \quad cwnd < ssth \\ \text{Congestion-Avoidance} & \text{falls} \quad cwnd \geq ssth \end{cases}$$

4.1.4 Round-Trip-Time (RTT)

Die RTT wird in Standard-Implementierungen zur Paketverlustbehandlung durch den Mechanismus des Retransmission-Timers (RT) angewendet. In CARD, TRI-S, DUAL und VEGAS wird sie zusätzlich zur Anpassung des CWND eingesetzt.

Nur für das erste Segment des Fensters wird ein Retransmission-Timer gesetzt. Solange für dieses Segment der RT gesetzt wurde, wird die RTT nach Eintreffen der Quittung aktualisiert. Tatsächlich wird durch eine periodische Unterbrechung, die spätestens jede $\Delta t = 500$ ms stattfindet, ein Zähler um eins erhöht. Die gemessene RTT ist deswegen nicht genau, da sie ein Mehrfaches von Δt darstellt:

$$rtt_i = \Delta t \cdot (rtt_{count} + 1)$$

mit

rtt	RTT der Periode i
Δt	Dauer der Unterbrechung
rtt_{count}	Anzahl der Unterbrechungen

Die RTT kann mit und ohne Timestamp-Option [JBB92] berechnet werden. Dabei schreibt der Sender die Systemzeit jedes übertragenen Segments in den Paket-Header. Der Empfänger kopiert diese Information in die Quittung, so dass der Sender die RTT für mehrere Segmente des Fensters messen kann. Diese Option benötigt zusätzliche 10-Bytes im Paket-Header. Die Timestamp-Option ist vor allem bei Datenübertragungen mit großer Pfadkapazität von Bedeutung, da sonst die Stichprobe der Messwerte der RTT zu klein wird.

Da nur Segmente mit Applikationsdaten zur Berechnung der RTT berücksichtigt werden, wird die RTT nur vom Sender ermittelt. Zusätzlich dazu wird der Karn-Algorithmus [KP91] benutzt. Dieser besagt, dass Quittungen für Segmente die mehrmals gesendet worden sind, nicht berücksichtigt werden dürfen, denn damit kann keine eindeutige Aussage gemacht werden, welches Segment quittiert worden ist.

4.1.5 Retransmission-Timeout (RTO)

Der RTO dient zur Feststellung der Zeit, nach der eine erneute Übertragung eines bereits gesendeten aber noch nicht quittierten Segments erfolgen muss, da es verloren gegangen sein könnte. Der Retransmission-Timer wird auf den Wert des RTO gesetzt und sein Ablauf als ein RT-Ereignis bezeichnet. Ein zu kleiner RTO führt zu unnötigem Senden von Segmenten mit großer RTT. Anderseits führt ein zu großer RTO zur Reduktion des Durchsatzes, da unnötig lange gewartet wird, bevor eine Aktion gegen Paketverluste unternommen wird. Diese Wartezeit wird als Leerlaufzeit bezeichnet.

Um den mittleren RTO abzuschätzen, wird zuerst eine geglättete RTT, genannt SRTT, mittels folgender Überlegung berechnet[1]: Ohne Überlastung würde eine Warteschlange nur die in der Periode gesendeten Segmente ($N_i = P$) beinhalten. Falls Überlastung vorhanden ist, erhöht sich diese Länge um einen Bruchteil der in der letzten Periode gesendeten Segmente ($N_i = P + y \cdot N_{i-1}$ mit $N_i > y \cdot N_{i-1}$). Da die RTT proportional zur Warteschlangenlänge ist, kann die aktuelle geglättete RTT $srtt_i$ als Funktion der geglätteten RTT der letzten Periode $srtt_{i-1}$ und der aktuell gemessenen RTT rtt_i nach folgender Gleichung angenähert werden:

$$srtt_i = \alpha \cdot srtt_{i-1} + (1 - \alpha) \cdot rtt_i$$

Und

α	Gewichtungsfaktor für die aktuelle rtt_i (i.a = 0.9)
$srtt_{i=0}$	initiale RTT

Damit der RTO schneller an die Fluktuation der Netzauslastung angepasst werden kann, muss die Varianz[2] der RTT verwendet werden [Jac88]:

[1] basiert auf Kleinrock's Gleichung der Warteschlangenlänge.

[2] Eigentlich wird die mittlere Abweichung und nicht die Standardabweichung benutzt, denn die Erstere kann mit Ganzzahlen berechnet werden im Gegensatz zur Standardabweichung, die Wurzel-Operationen verlangt.

$$\begin{aligned} \Delta rtt_i &= srtt_i - rtt_i \\ srtt_i &= srtt_{i-1} + g \cdot \Delta rtt_i \\ rttvar_i &= rttvar_{i-1} + h \cdot (|\Delta rtt_i| - rttvar_{i-1}) \\ rto_i &= srtt_i + 4 \cdot rttvar_i \end{aligned}$$

Und

$rttvar_i$	Schätzwert für die geglättete mittlere Abweichung innerhalb der Periode i
g	Gewichtungsfaktor für den RTO (i.a. = 1/8)
h	Gewichtungsfaktor für den RTO (i.a. = 1/4)

Für die Verbindungsaufbauphase wird $rto_0 = srtt_0 + 2 \cdot rttvar_0$ gewählt, mit $srtt_0 = 0$ und $rttvar_0 = 3$. Mit einer Unterbrechung alle 500 ms würde rto_0 auf 3 Sekunden gesetzt. Läuft ein Retransmission-Timer ab, wird der Wert exponentiell erhöht: rto_i wird mit $2^n, n \in \mathbf{N}, n \leq 7$, multipliziert. Für die Verbindungsaufbauphase impliziert dies folgende Werte für rto_i= 3, 6, 24, 48, ... Sekunden.

Die erste RTT wird bei Ankommen der Quittung für das erste Segment mit Applikationsdaten berechnet. Anschließend wird der RTO nach folgender Gleichung berechnet:

$$rto_1 = 3 \cdot rtt_1$$

4.1.6 TCP-Kontrollblock zur Fluss- und Überlastkontrolle

Die Zustandsvariablen von TCP werden in dem TCP-Kontrollblock `tcpcb` gehalten. Während der Übertragung werden Zeiger auf die relevanten Fensterpositionen in der zu übertragenden Datei behalten (siehe Abbildung 4.1). Sie sind `snd_una`, `snd_nxt` und `snd_max` für den Sender, und `rcv_nxt` und `rcv_max` für den Empfänger. Die Variable `snd_wnd` entspricht dem CWND des Senders, die Variable `rcv_wnd` ist das RWND des Empfängers.

4.2 Mechanismen zur Fluss- und Überlastkontrolle

Im Folgenden werden die wesentlichen Mechanismen beschrieben, mit denen TCP die Überlastkontrolle (siehe Abbildung 4.2) und Recovery realisiert. Ein Beispiel-Zeitdiagramm dieser Mechanismen ist in Abbildung 4.3 dargestellt.

4.2.1 Quittungen (ACK)

Anhand der Quittierung teilt der Empfänger dem Sender Information über empfangene oder fehlende Segmente mit. Die Anforderung dieses Mechanismus sind in RFC1122 [Bra89] beschrieben. Jedem Segment wird eine Sequenznummer zugewiesen, die dem Empfänger im Paket-Header mitgeteilt wird. Die Quittierung sieht vor, dass der Empfänger das bereits erhaltene Segment dadurch positiv quittiert, indem er die Sequenznummer des als nächsten zu erwartenden Segments mitteilt. Dies kann ein neues oder ein verloren gegangenes Segment sein. Hat der Empfänger z.B. *mss* Bytes mit Sequenznummer *n* empfangen, so sendet er eine Quittung (ACK) mit Sequenznummer $n + mss$.

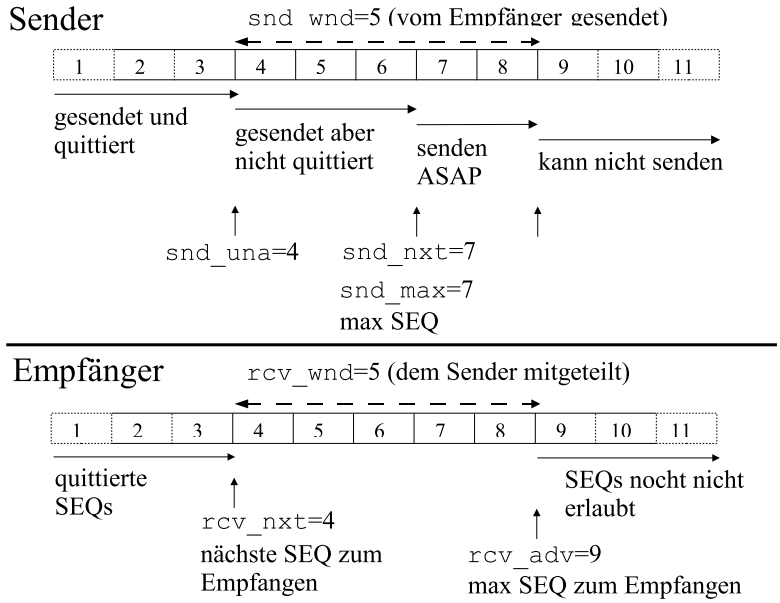


Abbildung 4.1: TCP-Variablen zur Fluss- und Überlastkontrolle (aus [WS95])

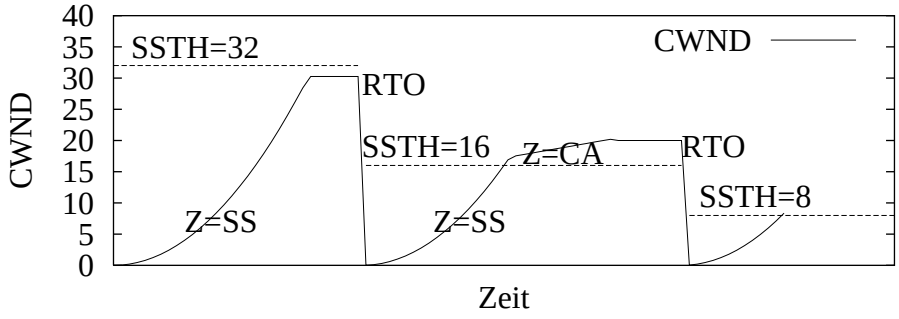


Abbildung 4.2: TCP-Mechanismen zur Überlastkontrolle: SS und CA (aus [Lai02])

Diese vom Empfänger in den Paket-Header geschriebene Sequenznummer heißt beim Sender Quittungsnummer.

Die Kriterien zum Generieren von Quittungen sind:

- Das erste Segment der Datenübertragung wird sofort quittiert.
- Jedes $b-$Segment wird quittiert. Normalerweise ist b auf zwei gesetzt. Dieser Mechanismus wird als kumulative Quittung bezeichnet.
- Nach Ablauf des DELACK-Timers. Die Empfehlung von RFC1122 ist, ein DELACK-Timer kleiner als 500 ms zu halten. Die meisten TCP-Varianten benutzen 200 ms.
- Beim Eintreffen von Out-Of-Order Segmenten (OOO-Segmente) wird die fehlende Sequenznummer mitgeteilt.

4.2.2 Slow-Start (SS)

Slow-Start ist der Mechanismus zur Approximation des Fensters. Dabei wird das CWND auf eins oder zwei Segmente initialisiert. Pro quittiertes Segment wird das CWND mit $\alpha = 1$ erhöht (siehe

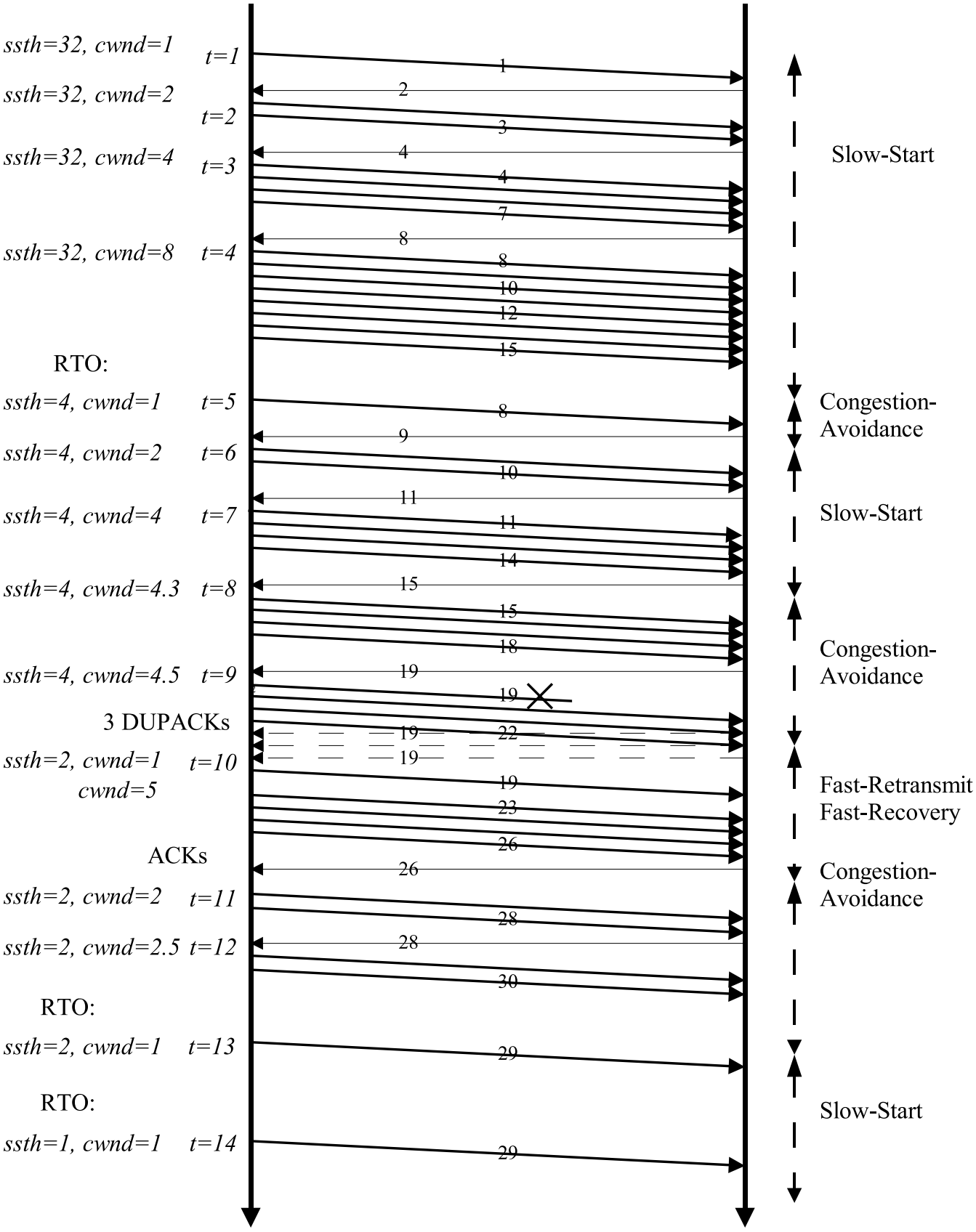


Abbildung 4.3: Zeitdiagramm der TCP-Zustände während der Datenübertragung

Abschnitt 3.6.1). Da CWND nicht in Segmenten, sondern in Bytes berechnet wird, ergibt sich:

$$ssth_1 = 32 \cdot mss$$

$$cwnd_i = \begin{cases} 1 \cdot mss & \text{falls i = 1,} \\ cwnd_{i-1} + mss & \text{falls ACK} \end{cases}$$

Mit

$cwnd_i$	CWND der Periode i
$ssth_i$	Schwellwert des CWND der Periode i
$rwnd_i$	RWND der Periode i

Ein Paketverlust wird vom Sender in einer der folgenden Bedingungen angenommen:

- Ablauf des Retransmission-Timers,
- Ankommen von 3 duplizierten Quittungen (DUPACK) oder
- Ankommen von SACK-Segmenten

4.2.3 Congestion-Avoidance (CA)

Nach erfolgter Initialisierung des CWND schaltet TCP von Slow-Start- auf dem Congestion-Avoidance-Modus um. Congestion-Avoidance wird anschließend zum Vergrößern des CWND eingesetzt, nachdem Paketverlusten festgestellt wurden. Die erste Aktion beim Paketverlust ist die Halbierung des SSTH mit Mindestwert zwei und das Setzen des CWND auf eins. Danach, wenn weitere Quittungen eintreffen, wird das CWND um seinen Kehrwert erhöht. Diese Operationen, in Bytes, sind [WS95]:

$$ssth_i = min\left(2, \frac{min(cwnd_{i-1}, rwnd_i)}{2 \cdot mss}\right) \cdot mss \qquad \text{falls RT}$$

$$cwnd_i = \begin{cases} mss & \text{falls RT} \\ cwnd_{i-1} + \frac{mss \cdot mss}{cwnd_{i-1}} & \text{falls ACK} \end{cases}$$

Er ist ersichtlich, dass mit Congestion-Avoidance das CWND um eins pro RTT erhöht wird, d.h. wenn alle Segmente des Fensters quittiert worden sind. Der Anstieg des CWND mit der Zeit erfolgt somit linear.

4.2.4 Fast-Retransmit (FR) und Fast-Recovery (FC)

Der Nachteil der bisher erläuterten Mechanismen besteht darin, dass der Ablauf des Retransmission-Timers die einzige Möglichkeit ist, Paketverluste festzustellen. Dies verlängert die Leerlaufzeit der Datenübertragung und reduziert dadurch den Durchsatz. Daher wurden in [Jac90] die Fast-Retransmit und Fast-Recovery Mechanismen vorgeschlagen.

Fast-Retransmit ermöglicht die schnelle Übertragung eines Segments, das verloren gegangen ist. Er setzt voraus, dass der Empfänger eine Quittung mit der kleinsten fehlenden Sequenznummer sendet, sobald OOO-Segmente empfangen werden. Nach dem Empfang von n Quittungen mit der gleichen Quittungsnummer (DUPACK), wird das fehlende Segment mit der entsprechenden Sequenznummer

übertragen (Retransmit), ohne auf den Ablauf des Retransmission-Timers zu warten. Fast-Retransmit kann den Verlust von jeweils nur einem Paket behandeln. Üblicherweise ist n auf $rexmtth = 3$ gesetzt.

Fast-Retransmit kommt nicht zum Einsatz, wenn das CWND sehr klein ist, denn in diesem Fall ist der Empfänger nicht in der Lage, die erforderlichen drei DUPACK zu senden. Beispielweise sei das CWND auf drei gesetzt. Falls nun eins von diesen Segmenten verworfen wird, kann der Sender höchstens noch zwei DUPACK empfangen. In diesem Fall wird der Sender zwangsläufig auf einen Retransmission-Timer warten müssen, was zur Folge hat, dass das CWND wieder auf eins gesetzt wird.

In Fast-Retransmit wird das fehlende Segment neu übertragen und der SSTH halbiert:

$$
\begin{aligned}
&\text{if DUPACK} = rexmtth\\
&\quad ssth_i = min\left(2, \frac{min(cwnd_{i-1}, rwnd_i)}{2 \cdot mss}\right) \cdot mss\\
&\quad cwnd_i = mss\\
&\quad rxtmit(seq)\\
&\quad cwnd_i = ssth_i + 3 \cdot mss
\end{aligned}
$$

Fast-Recovery wird nach der Retransmission eingesetzt und kontrolliert das Vergrößern des CWND nach der Übertragung des verlorenen Segments solange bis eine Quittung für neue Segmente ankommt. Als Beispiel nehme man an, der Sender übertrug die Segmente 3 bis 15, von denen das Segment 5 verloren ging. Die Retransmission von Segment 5 findet nach Ankommen der drei DUPACK statt (Fast-Retransmit), aber solange dieses Segment beim Empfänger nicht ankommt, können bis zu 7 weitere DUPACK für Segment 5 (Segmente 9 bis 15) beim Sender eintreffen. Bis der Sender das generierte ACK für neue Segmente empfängt (für Segmente ab 16) wird er Fast-Recovery anwenden.

In Fast-Recovery wird mit jedem neuen DUPACK das CWND linear erhöht, solange bis eine Quittung für neue Segmente ankommt. Dabei wird das CWND auf den alten SSTH gesetzt und der Zähler für duplizierte ACK zurückgesetzt:

$$
cwnd_i = \begin{cases} cwnd_{i-1} + \frac{mss \cdot mss}{cwnd_{i-1}} & \text{falls DUPACK} \\ ssth_{i-1} & \text{falls ACK} \end{cases}
$$

Das in Abbildung 4.3 dargestellte Beispiel des Zusammenspiels aller dieser Mechanismen wird im Folgenden erläutert.

(i) Von Zeit $t = 1$ bis $t = 4$ befindet sich der Prozess im Slow-Start-Modus, denn es gilt $cwnd \leq ssth$.

(ii) Zum Zeitpunkt $t = 5$ läuft der Retransmission-Timer ab. Dies bewirkt die Umschaltung auf den Congestion-Avoidance-Modus. Dabei werden *ssth* und *cwnd* entsprechend auf die Hälfte des bisherigen *cwnd* bzw. auf eins gesetzt. Da nach dieser Änderung $cwnd < ssth$ gilt, wird erneut auf den Slow-Start-Modus umgeschaltet. Da die letzte Quittung die Nummer 7 hatte, muss erneut ab Segment 8 übertragen werden.

(iii) Ab dem Zeitpunkt $t = 8$ befindet sich der Prozess wieder im Congestion-Avoidance-Modus, denn es gilt $cwnd \geq ssth$. Daher wird *cwnd* linear erhöht.

(iv) Zum Zeitpunkt $t = 10$ werden drei DUPACK empfangen, da hier das Segment 19 verloren ging und die Segmente 20-22 empfangen wurden. Als Folge wird Fast-Retransmit eingesetzt und

das Segment 19 neu übertragen. Anschließend wird Fast-Recovery angewendet, denn das *cwnd* wird nicht auf eins gesetzt, sondern auf *ssth* plus die Anzahl von DUPACK. Dies ermöglicht das Senden der Segmente 23 bis 26.

(v) Die Quittung für neue Data in $t = 11$ beendet Fast-Recovery. Dabei wird das *cwnd* auf den *ssth* vor Fast-Retransmit (2 Segmente) gesetzt. In diesem Fall wird wieder auf den Congestion-Avoidance-Modus umgeschaltet ($cwnd = ssth$).

(vi) Zum Zeitpunkt $t = 12$ befindet sich die Übertragung im Congestion-Avoidance-Modus.

(vii) Zu den Zeitpunkten $t = 13$ und $t = 14$ läuft der Retransmission-Timer ab. Nach $t = 14$ wird erneut auf den Congestion-Avoidance-Modus umgeschaltet, denn es gilt $cwnd \geq ssth$.

4.2.5 Selective-Acknowledgment (SACK)

Der Selective-Acknowledgment Mechanismus [MMFR96] dient zur Signalisierung mehrerer Paketverluste. Die Bereitschaft zur Nutzung der SACK-Option wird mit einer 2-Byte langen TCP-Option in einem SYN-Segment dem Empfänger mitgeteilt. Wird der Verlust von mehr als einem Segment beim Empfänger festgestellt, schreibt der Empfänger in den Paket-Header die Sequenznummern mehrerer bereits empfangener nicht kontinuierlicher Segmentblöcke. Der Sender kann hiernach entscheiden, welche Segmente erneut übertragen werden müssen.

Jeder Segmentblock wird durch seine minimale und maximale Sequenznummer beschrieben. Da eine Sequenznummer 4 Bytes benötigt und die Länge des Options-Felds im Paket-Header insgesamt 40 Bytes beträgt, können bis zu vier Blöcke geschrieben werden[3]. Wird zusätzlich die Timestamp-Option verwendet, dann verringert sich die Anzahl der Segmentblöcke auf 3. Hierbei muss der Sender eine Warteschlange für die Retransmissionen verwalten.

SACK hat den Nachteil, dass das Options-Feld des Paket-Headers durch andere Optionen bereits belegt werden kann. Außerdem muss SACK sowohl bei Sender als auch beim Empfänger implementiert sein.

4.2.6 Newreno

Newreno [Hoe95, Hoe96, FH99] behandelt Recovery von mehreren Paketverlusten, wenn die SACK-Option nicht angewendet werden kann. Er basiert auf dem Empfang von partiellen Quittungen (PACK). Partielle Quittungen sind Quittungen für neue, aber nicht für alle ausstehenden Segmente während Fast-Recovery. Er stellt eine kleine aber wichtige Modifikation zu Fast-Recovery und Fast-Retransmit dar.

Die erste Änderung betrifft den Wert von SSTH: sobald die drei DUPACK empfangen werden, wird SSTH neu berechnet. Sei $cwnd\_fly$ die Anzahl der Segmente in der Leitung, wie in [APS99] definiert. Der neue Wert von SSTH wird nicht wie in Fast-Recovery halbiert, sondern nach folgender Gleichung berechnet:

$$ssth_i = max\left(\frac{cwnd\_fly_i}{2}, 2 \cdot mss\right)$$

Die zweite Modifikation betrifft die Reaktion beim Empfangen einer Quittung für neue Segmente. Die

[3] 2 der 40 Bytes des Options-Felds sind für die eigene Identifikation der Option reserviert.

variable *recover* wird auf die größte gesendete Sequenznummer gesetzt. Falls die Quittungsnummer größer als *recover* ist, wird Fast-Recovery beendet, denn alle Segmente nach der Feststellung des ersten Paketverlusts sind beim Empfänger angekommen. Das neue CWND kann entweder auf das Minimum zwischen dem alten SSTH und $cwnd\_fly+mss$ (Option a) oder auf den alten SSTH (Option b) gesetzt werden. Falls die Quittung eine PACK ist, dann ist ein Segment zwischen dem ersten Segment und dem *recover*-Segment verloren gegangen. Dieses muss dann erneut übertragen werden.

$$cwnd_i = \begin{cases} cwnd_{i-1} - ack + mss & \text{falls PACK} \\ min(ssth_{i-1}, cwnd\_fly_i + mss) & \text{falls ACK - Option a} \\ ssth_{i-1} & \text{falls ACK - Option b} \end{cases}$$

$$rxtmit(seq) \qquad \text{falls PACK}$$

4.2.7 Explicit-Congestion-Notification (ECN)

Explicit-Congestion-Notification [Flo94] ist ein Mechanismus für die Behandlung von Feedback-Signalen, die mit DECBit-ähnlichen Verfahren in den Routern generiert werden. ECN ergänzt die Fast-Retransmit- und Fast-Recovery Mechanismen, denn beim Ankommen eines ECN-Signals wird das CWND halbiert:

$$cwnd_i = cwnd_{i-1}/2$$

Nach einer Reaktion auf ein ECN-Signal werden während einer kurzen Periode weitere ECN-Signale oder DUPACK zunächst ignoriert. Diese Periode endet mit der Quittierung aller ausstehenden Pakete zur Zeit des Ankommens des ersten ECN-Signals. Falls der Sender auf ein ECN-Signal reagierte, und kurz danach drei DUPACK empfangen werden, muss das Paket erneut übertragen werden.

4.2.8 Forward-Acknowledgment (FACK)

Der Forward-Acknowledgment Mechanismus [MM96] setzt auf eine verbesserte Überlastkontrolle während der Recovery auf. Statt jedes DUPACK zu zählen wird die zusätzliche Information von SACK zur expliziten Berechnung der ausstehenden Datenmenge benutzt. FACK benötigt zwei zusätzliche Größen hinzu: snd_fack und *retrans_data*. Die erste ist die höchste Sequenznummer in einer SACK-Option, d.h. die höchste nicht konsekutiv bereits empfangene Sequenznummer. Bei jeder Retransmission wird die Variable *retrans_data* um *mss* erhöht. Die Variable *retrans_data* wird gesenkt, wenn festgestellt wird, dass ein neues Segment empfangen wurde. Auf diese Art kann die ausstehende Datenmenge $awnd_i$ genauer bestimmt werden:

$$awnd_i = snd\_nxt_i - snd\_fack_i + retrans\_data_i$$

Solange $awnd_i < cwnd_i$ gilt, kann ein weiteres Segment gesendet werden.

4.2.9 Rate-Base-Pacing (RBP)

In [VH97] wird RBP als Mechanismus zur Vermeidung von Slow-Start vorgeschlagen, in Situationen, in welchen die Verbindung einige Zeit im Warte-Zustand war. Dies ist z.B. der Fall, wenn mehrere kleine Datenübertragungen erforderlich sind, welche zu einem Web-Zugriff gehören.

Mit RBP wird erreicht, dass das CWND nach einer Pause nicht initialisiert wird. Anstatt dessen werden die neuen Segmente mit einer bestimmten Senderate gesendet bis die Quittierung wieder eintritt. Ein Mechanismus zur Leerlaufzeit-Erkennung setzt einen Prozess in Gang, der ein Segment periodisch sendet.

4.3 Zusammenfassung der TCP-Varianten

Die Zusammenfassung der hier behandelten TCP-Varianten ist in Tabelle 4.1 dargestellt. Es ist wichtig anzumerken, dass SS, CA, CARD, TRI-S, DUAL Fluss- und Überlastkontroll-Mechanismen sind. FR, FC, SACK, Newreno und FACK sind Recovery-Mechanismen, d.h. sie werden erst nach einem Paketverlust aktiviert. VEGAS realisiert beide Mechanismen.

Tabelle 4.1: TCP-Varianten

TCP-Variante	S	R	P	Mechanismen
Tahoe	x	x		SS, CA, FR
Reno	x	x		SS, CA, FR, FC
SACK	x	x		SS, CA, FR, SACK
Newreno	x	x		SS, CA, Newreno
TCP-ECN	x	x		SS, CA, FC, FR, ECN
TCP-RBP		x		SS, CA, FR, RBP
CARD (Abschnitt 3.6.1)			x	FR, CARD
TRI-S (Abschnitt 3.6.1)			x	SS, FR, TRI-S
DUAL (Abschnitt 3.6.1)			x	FR, DUAL
VEGAS (Abschnitt 3.6.1)			x	SS-VEGAS, FR, VEGAS
TFRCP (Abschnitt 3.6.3)			x	TFRCP
TFRC (Abschnitt 3.6.3)			x	TFRC
LEGENDE S=Standard R=Reaktiv P=Präventiv				

4.4 Funktionsmodelle

4.4.1 Aufbauplan der TCP-Datenübertragung

Die im Folgenden verwendete Methodik zur Systemmodellierung basiert auf der am HPI entwickelten Fundamental Modelling Concept (FMC) [FMC] und stellt in Abbildung 4.4 den Aufbauplan der Datenübertragung in TCP dar. Ein Aufbauplan veranschaulicht die statische Struktur des Systems. Die Elemente eines FMC-Aufbauplanes (siehe Notation im Anhang, Abbildung C.3) umfassen im

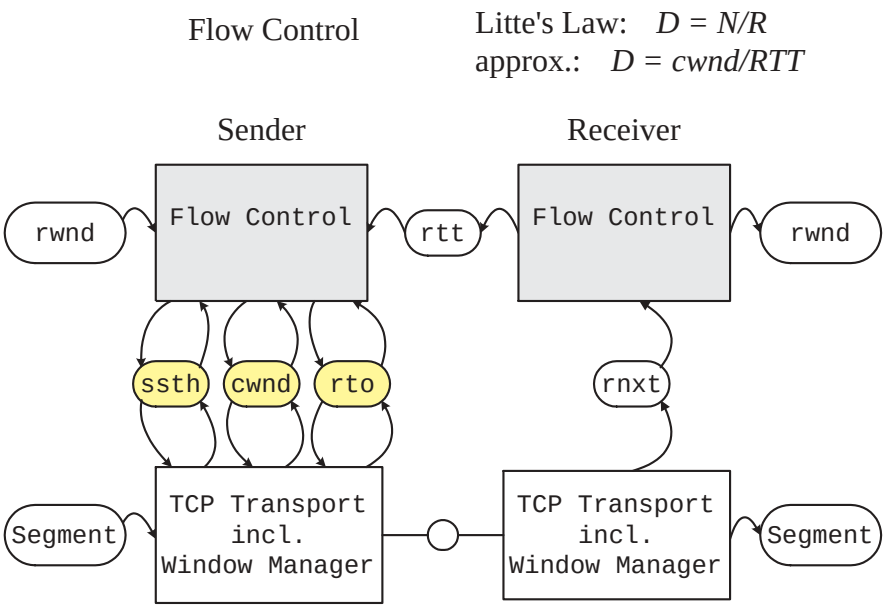


Abbildung 4.4: Aufbauplan eines TCP/IP-Kommunikationssystems [Zor02]

vorliegenden Fall Akteure, Speicher, Kanäle und Anschlüsse. Akteure können entweder über Kanäle bei ungepufferter Kommunikation oder über Speicher bei gepufferter Kommunikation verbunden sein.

Die Funktionalität des TCP wird im vorliegenden Modell unterteilt: Transport und Flusskontrolle. Die Transportinstanzen sind in Abbildung 4.4 über einen Kanal miteinander verbunden, die Flusskontroll-Agenten über einen Speicher. Über den Kanal werden Daten und die Quittungen zwischen den TCP-Akteuren ausgetauscht. Über den Speicher wird die von den Flusskontroll-Agenten gemessene RTT und daraus die Stellgrößen CWND, SSTH und RTO ermittelt.

4.4.2 Ablaufplan der TCP-Datenübertragung

Die Abbildung 4.5 zeigt den TCP-Ablaufplan in Form eines FMC-eCS (extended for communication systems) Petrinetzes mit dem in [Zor02] eingeführten eCS-Erweiterungen. Dabei wird die TCP-Datenübertragung als ein *Producer/Consumer*-System modelliert. Der linke Teil zeigt den Sender (Producer) und der rechte Teil den Empfänger (Consumer). Eine ausführliche Notation befindet sich im Anhang (Abbildung C.2). Dem vorliegenden Modell ist TCP-Reno im Zustand `ESTABLISHED` zugrunde gelegt.

Der in Abbildung 4.5 dargestellte Ablaufplan basiert auf der FMC-eCS Modellvorstellung [Zor02], dass Sender und Empfänger während einer Übertragung ein zusammengehöriges System bilden, dessen Aktionen innerhalb einer gemeinsamen zu aktivierenden Transition ausgeführt werden. In Abbildung 4.5 bedeutet dies, dass die äußeren dunkel grau markierten Stellen beidseits markiert sein müssen, damit die dazwischen liegende Butterfly-förmige Transition schaltet. Die Butterfly-Kontur stellt dabei die Abstraktion des inneren Markenflusses zwischen Sender und Empfänger dar.

Dieser realisiert die Producer/Consumer-Beziehung zwischen Sender und Empfänger dergestalt, dass der Empfänger beim Sender jeweils das nächste erwartete Segment bestellt, beginnend bei Byte Nummer 1. Empfängt der Empfänger das bestellte Segment, so erhöht er die zu erwartende Sequenznum-

mer, andernfalls wiederholt er die nicht erfüllte Bestellung. Die Flusskontrolle von TCP wird dabei durch eine Mehrmarken-Stelle beim Sender modelliert, aus welcher der Sender ohne Quittierung weitere Sendemarken entnehmen kann solange, bis das Fenster bis zum Anschlag gesendet ist.

Hieraus resultiert, dass die Butterfly-förmige Transition von Sender und Empfänger unterschiedlich oft durchlaufen werden kann, sich diese jedoch in jedem Fall in definierten Zuständen synchronisieren und somit das Gesamtsystem immer wieder in einen konsistenten Zustand überführen. Konsistenz wird dabei als Widerspruchfreiheit definiert.

Bei den Übergängen von konsistenten Anfangszuständen in konsistente Endzustände treten innerhalb der zu durchlaufenden Transitionen im Inneren auf beiden Seiten inkonsistente Zustände auf, in Abbildung 4.5 durch die beiden dunkel markierten großen Zuständen repräsentiert. Die Inkonsistenz d.h. die Widersprüchlichkeit von Aussagen über den Systemszustand liegt im vorliegenden Fall darin, dass auf Senderseite unterschiedliche Aussagen darüber möglich sind, ob das Segment korrekt beim Empfänger angekommen ist, während auf Empfängerseite unterschiedliche Aussagen darüber möglich sind, ob der Sender die Quittungen d.h. die Bestellungen korrekt empfangen hat.

Die Mechanismen von Fluss- und Überlastkontrolle werden mit Hilfe der Mehrmarken-Stellen modelliert dergestalt, dass die Fluss- und Überlastkontrolle-Akteure die Anzahl der Marken auf die Mehrmarken-Stellen nach oben und unten variieren können bis hin zum vollständigen Blockieren des Senders durch Entzug sämtlicher Marken. Anschaulich entsprechen die Mehrmarken-Stellen dabei den noch freien Puffern im gemeinsam zwischen Sender und Empfänger jeweils ausgehandelten Fenster. In Abbildung 4.5 sind die Mechanismen der Fluss- und Überlastkontrolle nicht enthalten (siehe dazu [Zor02]).

Die möglichen Aktionen von Sender und Empfänger sind jeweils durch die aus den inkonsistenten Zuständen herausführende Übergänge repräsentiert. Diese folgen dem Zorn'schen *ganz- oder garnicht*-Paradigma und lassen sich demzufolge in Fortschritts- und Rücksetzaktionen unterscheiden, welche im einzelnen durch Kontrollvariablen oder durch Timeouts ausgelöst werden.

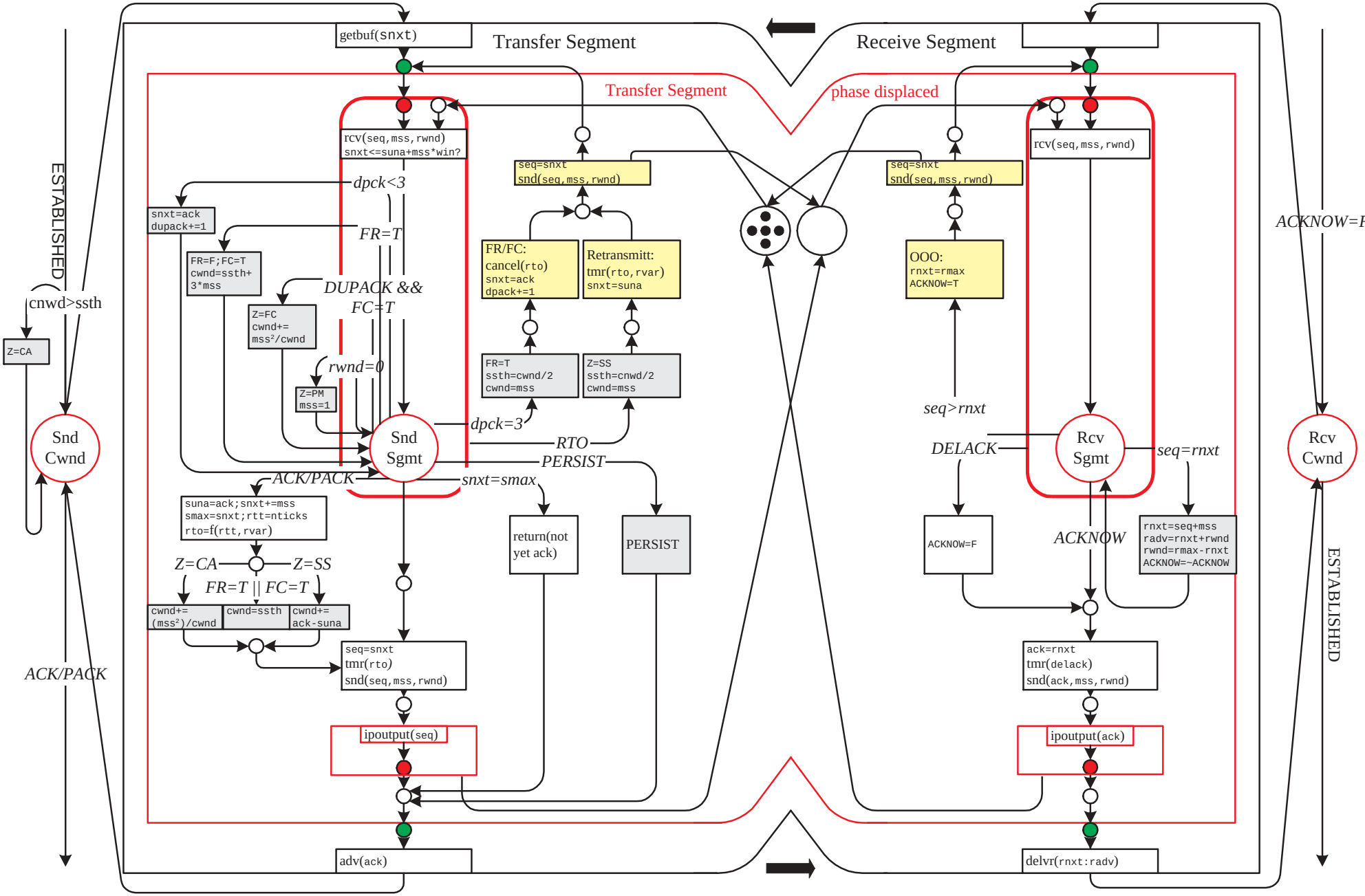


Abbildung 4.5: Zustandsdiagramm während `ESTABLISHED` - TCP-Reno

4.5 Leistungsmessungen als Fallstudie

Im Folgenden wird das Ergebnis einer Fallstudie vorgestellt. Das Ziel war, den Einsatz der TCP-Mechanismen zur Überlastkontrolle zu überprüfen und deren Einfluss auf die Leistungskenngrößen zu bewerten. Vor allem Recovery-Mechanismen sind für das Erreichen von guten Durchsätzen entscheidend. Aus der Spezifikation von TCP geht hervor, dass folgende Faktoren Leistungsverluste verursachen:

- Warten auf den Retransmission-Timer-Ablauf (RT).
- Verlust eines Segments am Anfang einer Slow-Start-Phase. Dadurch muss auf den initialen RTO gewartet werden.
- Verlust eines Segments bei zu kleinem CWND. Dadurch kann Fast-Retransmit nicht eingesetzt werden. Stattdessen kommt der RT-Mechanismus zur Anwendung.
- Verlust von mehreren Segmenten.
- Zu kleines RWND.

4.5.1 Messumgebung und theoretische Leistungskenngrößen

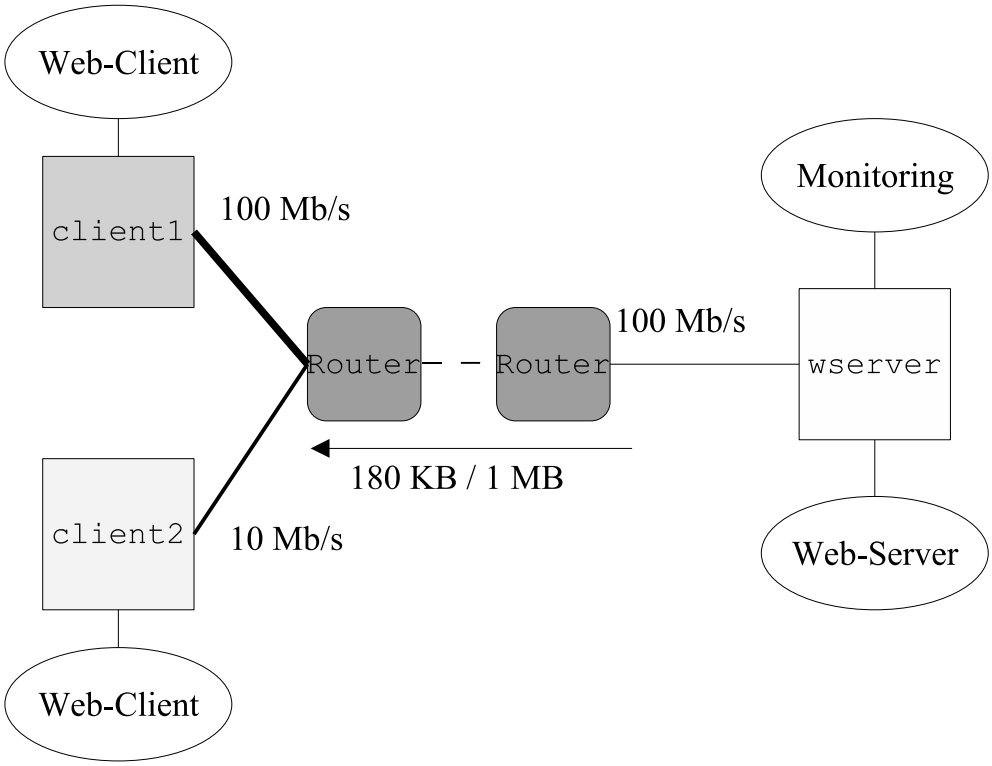


Abbildung 4.6: Schema der HTTP-Messungen

Die Messumgebung ist in Abbildung 4.6 dargestellt. Datenübertragungen von zwei Web-Clients, `client1` und `client2`, zu einem Web-Server `wserver` wurden mit PANiON-Tools[Fri02][4] gestartet und mit dem Programm `tcpdump` [5] protokolliert. Die Web-Clients hatten 100 bzw. 10 Mb/s

[4] `http://www-zorn.ira.uka.de/pan/Welcome.html`
[5] `http://www.tcpdump.org`

Internet-Anbindung, der Web-Server 100 Mb/s. Die Messungen wurden während mehrerer Monate durchgeführt. Es wurden dabei drei Tageszeiten mit unterschiedlichen Lastniveaus ausgewählt. Die Analyse der Messungen wurde mit einer erweiterten Version des Programms `tcptrace` [6] durchgeführt, deren Ausgabe in der Abbildung C.6 im Anhang zu sehen ist.

Jeder Web-Client startet eine passive Verbindung zu `wserver`. Nach den drei Segmenten des Verbindungsaufbaus wird die Datenübertragung im Slow-Start-Modus gestartet mit initialem CWND von zwei Segmenten, MSS von 1460 Bytes und gesetzter SACK-Option. Die Verbindungen werden durch die Web-Clients nach Ankommen von *n* Paketen abgebaut. Die zu übertragende Dateigröße betrug ca. 180 KB und 1 MB.

Aus der TCP-Sicht beträgt die zu erwartende RTT-Differenz ohne andere Faktoren zu berücksichtigen:

$$\begin{aligned}\Delta rtt &= \frac{Paketgroesse \cdot (Datenrate_1 - Datenrate_2)}{Datenrate_1 \times Datenrate_2} \\ &= 1500 \cdot 8 \cdot \frac{10 - 100}{(10 \cdot 100) \cdot 1000}\ \text{ms} \\ &= 1.08\ \text{ms}\end{aligned}$$

Der theoretische Durchsatz in TCP ist in Abbildung 4.7 für kurze Datenübertragungen und in Abbildung 4.8 für den stationären Zustand dargestellt. Die Schritte zur Ermittlung dieser Werte sind:

- Aus den Messungen werden die einzelnen Werte für RTT und Paketverlust ermittelt.
- Es wird der Mittelwert von RTT für jeden Web-Client gebildet.
- Diese RTT-Werte werden mit der Gleichung in 4.1 für kurze Datenübertragungen bzw. mit der Gleichung 4.3 für den stationären Zustand für unterschiedliche Paketverlustraten modelliert.

Modell des TCP-Durchsatzes für kurze Datenübertragungen. Das Modell [CSA98] unterscheidet zwischen der Zeit für die Retransmissionen und der Zeit im Slow-Start-Modus. Da es sich um kurze Datenübertragungen handelt, wird vorausgesetzt, dass der Congestion-Avoidance-Modus nicht erreicht wird und nach dem Ablauf des Retransmission-Timers der Slow-Start-Modus fortgesetzt wird. Diese Periode endet wiederum nach Ablauf des Retransmission-Timers.

$$t = rtt + t_{delack} + t_{rto} + t_{xfer} \tag{4.1}$$

$$\begin{aligned}&= rtt + t_{delack} + l \cdot Q(p) \cdot rto_0 \cdot (1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6) + \\ &+ (l \cdot Q(p) \cdot (1 - p) + 1) \cdot \log_r \left(\frac{e \cdot (r - 1)}{cwnd_0 \cdot mss \cdot k} + 1 \right) \cdot rtt\end{aligned} \tag{4.2}$$

mit

[6] `http://www.tcptrace.org`

t_{delack} Übertragungszeit für die Quittung des ersten Segments

t_{rto} Summe der Übertragungszeit der Retransmissionen

$t_{rto} = u \cdot t_u$

u Anzahl konsekutiver RTO

$u = l \cdot Q(p) \cdot (1 - p)$

t_u Leerlaufzeit für diese konsekutive RTO

$t_u = rto_0 \cdot \frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p}$

l Anzahl der verlorenen Segmente

$l = \frac{data \cdot p}{1-p}$

p Paketverlustrate

$p = \frac{l}{data+l}$

b Anzahl der empfangenen Segmente pro ACK (i.a. 2)

$data$ zu übertragende Datenmenge der Applikationsschicht

$Q(p)$ Wahrscheinlichkeit, dass ein RT-Ereignis vorkommt

$$Q(p) = \begin{cases} 0 & p = 0 \\ \min\left(1, \frac{3}{\sqrt{\frac{8}{3bp}}}\right) & p > 0 \end{cases}$$

rto_0 initialer RTO

t_{xfer} Übertragungszeit im Slow-Start-Modus

$t_{xfer} = v \cdot \log_r \left(\frac{e \cdot (r-1)}{cwnd_0 \cdot mss \cdot k} + 1\right) \cdot rtt$

v Anzahl der aktiven Perioden zwischen den RTO

$v = u + 1$

r Quotient $cwnd_i / cwnd_{i-1} = 1 + 1/b$

e im Durchschnitt gesendeten Daten in einer Slow-Start-Phase

$e = \frac{data+l}{v}$

$cwnd_0$ initiales CWND

k Anzahl der parallelen Verbindungen

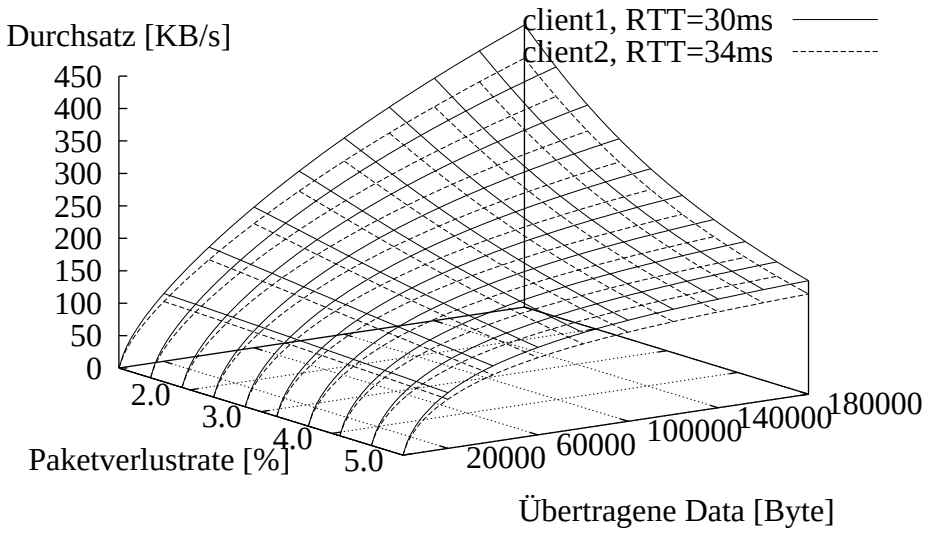


Abbildung 4.7: Modellierungsergebnisse des TCP-Durchsatzes für kurze Datenübertragungen

Der erste Term in Gleichung 4.1 ist die Übertragungszeit für die Verbindungsaufbauphase, der zweite ist die Zeit für die verzögerte Quittung des ersten Segments, der dritte Term ist die Zeit für die Übertragung verlorener Pakete und der vierte Term ist die Übertragungszeit im Slow-Start-Modus.

Es ist aus Gleichung 4.2 ersichtlich, dass der Durchsatz von folgenden Parametern abhängig ist: 1) der zu übertragenen Datenmenge, 2) dem initialen CWND, 3) der Anzahl der TCP-Segmente für den DELACK-Mechanismus (Parameter r), und 4) der Paketverlustrate. Die Abbildung 4.7 zeigt diese Funktion für $cwnd_0 = 2$, $k = 1$, Paketverlustrate zwischen 1.5 und 5 % bei übertragener Datenmenge von 20 bis 180 KB. Die Variation der Paketverlustrate hat eine größere Wirkung auf den Durchsatz als die Variationen der anderen Faktoren.

Modell des TCP-Durchsatzes im stationären Zustand. In Abbildung 4.8 wird ein TCP-Modell als Funktion der Paketverlustrate p für den stationären Zustand dargestellt [PFTK98]. Dieses Modell berücksichtigt die Effekte von Slow-Start, Congestion-Avoidance sowie von Retransmission-Timer und Fast-Retransmit.

$$D(p) \approx \min\left(\frac{rwnd}{rtt}, \frac{mss}{rtt\sqrt{\frac{2bp}{3}} + rto_0 \cdot \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) \cdot p(1+32p^2)}\right) \tag{4.3}$$

mit

D	Durchsatz
p	Paketverlustrate
b	Anzahl der empfangenen Segmente pro Quittung (i.a. 2)
$rwnd$	maximales Empfängerfenster
rto_0	initialer RTO während der Datenübertragung ($\approx 2 \cdot rtt$)

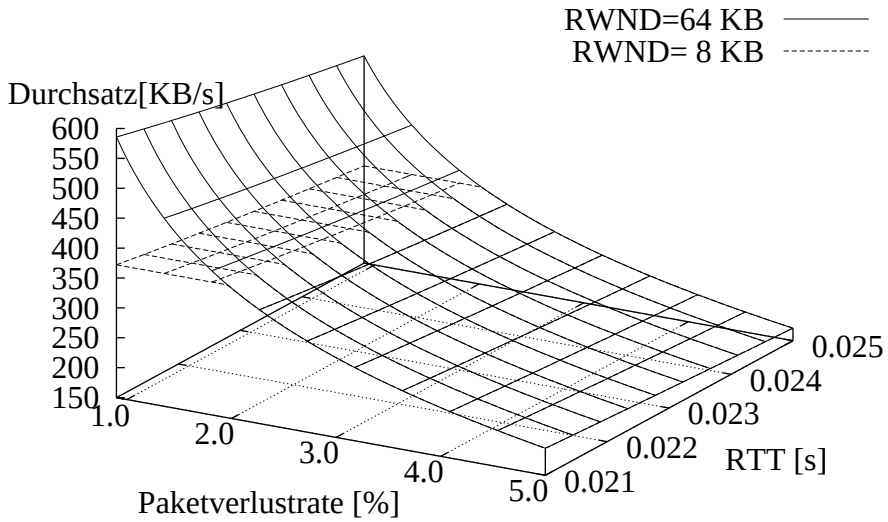


Abbildung 4.8: Modellierungsergebnisse des TCP-Durchsatzes im stationären Zustand

Die Abbildung 4.8 zeigt die Funktion für eine Paketverlustrate zwischen 1 und 5 %, und für eine RTT zwischen 21 und 25 ms. Folgendes ist zu beachten:

- Eine kleine Erhöhung der Paketverlustrate hat dabei einen großen negativen Effekt auf den Durchsatz.
- Der Einfluss der Paketverlustrate ist größer als derjenige der RTT. Bei niedriger Paketverlustrate kann hierdurch der Durchsatz für eine Verbindung mit größerer RTT besser sein, als der für eine Verbindung mit kleinerer RTT. Dies kann der Fall sein, wenn auf Grund einer transienten Netzüberlastung eine Menge von Paketen der Verbindung mit kleinerer RTT verworfen wird. Bei einer Verdoppelung der Paketverlustrate von 1 % auf 2% sind Leistungseinbußen in der Größenordnung von 75 % zu erwarten.

Bei einer Paketverlustrate $p = 0$ reduziert sich die Gleichung 4.3 auf $D = rwnd/rtt$.

4.5.2 Durchschnittliche Leistungskenngrößen

Die vollständigen Ergebnisse der Messungen für die Übertragung von 180 KB und 1 MB werden in Tabellen C.1 und C.2 (a) bis (c) im Anhang gezeigt. Diese Tabellen fassen den Durchsatz (`avg`), die Standardabweichung (`stdv`) und der Variationskoeffizient (`cv`) von 16 wichtigen Kenngrößen aus der Analyse jeder einzelnen Datenübertragung mit `tcptrace` zusammen:

1 Durchsatz inklusive Verbindungsaufbauphase
2 Durchsatz der Datenübertragung ohne Verbindungsaufbauphase
3 Datenübertragungszeit ohne Verbindungsaufbau
4 mittlere RTT
5 Standardabweichung der RTT
6 Anzahl der gesendeten Pakete
7 Anzahl der duplizierten Quittungen (DUPACK)
8 Anzahl der 3-DUPACK (für Fast-Retransmit)
9 maximal Anzahl an Retransmissionen
10 maximale Leerlaufzeit zwischen zwei Paketen
11 maximale Retransmissionszeit. Dies ist die maximale Wartezeit zwischen zwei Paketen mit gleicher Sequenznummer und kann daher auch die maximale Leerlaufzeit darstellen
12 prozentuale maximale Leerlaufzeit bezogen auf die Übertragungszeit
13 prozentuale maximale Retransmissionszeit bezogen auf die Übertragungszeit
14 Paketverlustrate
15 maximale ausstehende Fenstergröße, d.h. Anzahl noch nicht quittierter Segmente
16 mittlere ausstehende Fenstergröße

Bei den Messergebnissen für die Übertragung von 180 KB ist von besonderem Interesse:

- Die Datenübertragung von `wserver` nach `client1` (`Verbindung1`) ist empfindlicher als die von `wserver` nach `client2` (`Verbindung2`). Dies zeigen die Variationskoeffizienten.
- Die RTT (30 ms) der `Verbindung1` ist kleiner als die RTT der `Verbindung2` (34 ms), was zur Folge hat, dass der Durchsatz der `Verbindung1` (179 KB/s) größer als der von `Verbindung2` (102 KB/s) ist.

Kenngroesse	client1			client2		
	avg->	stdv	cv	avg->	stdv	cv
1 Durchsatz [KB/s]	179->	106	0.59	102->	25	0.25
4 RTT-Mittelwert [ms]	29.90->	7.50	0.25	34.20->	3.00	0.09
8 Triple DUPACKs	1.5->	1.4	0.93	1.4->	1.3	0.93
10 Leerlaufzeit max [ms]	712.89->	1090.35	1.53	821.98->	321.79	0.39
12 Leerlaufzeit max [%]	29.78->	16.68	0.56	46.64->	10.61	0.23

Tabelle 4.2: Leistungskenngrößen für die Übertragung von 180 KB

- Der Variationskoeffizient der RTT der `Verbindung1` (0.25) ist größer als der von `Verbindung2` (0.09). Daher ist die Schwankung des Durchsatzes der `Verbindung1` deutlich größer als bei der `Verbindung2`.
- Die RTT erklärt den Durchsatzunterschied nur teilweise. Ein anderer wichtiger Faktor ist die maximale Leerlaufzeit, die in `Verbindung1` (713 ms) kleiner als in `Verbindung2` (822 ms) ist. Die Leerlaufzeit stellt je nur ein Ereignis in der gesamten Datenübertragung dar, das aber zwischen 30 % bzw. 47 % der Übertragungszeit beträgt.
- Fast-Retransmit wird selten angewendet, denn die Anzahl der 3-DUPACK liegt nur bei 1.5. Mit Paketverlustraten von ca. 3 % und Anzahl der Pakete von ca. 130 stellen die Retransmissionen mit dieser Methode nur ein Drittel der Fälle dar.

Diese Ergebnisse sind weiter in zwei Gruppen unterteilt: Datenübertragungen, die parallel verlaufen, und Datenübertragungen, die nicht parallel verlaufen. Dabei ergab sich:

Kenngroesse	client1			client2		
	avg->	stdv	cv	avg->	stdv	cv
Nicht parallel						
1 Durchsatz [KB/s]	239->	81	0.34	108->	22	0.21
4 RTT-Durchsatz [ms]	26.80->	5.00	0.19	33.80->	2.90	0.09
10 Leerlaufzeit max [ms]	198.01->	205.80	1.04	796.95->	164.05	0.21
Parallel						
1 Durchsatz [KB/s]	79->	57	0.72	92->	27	0.29
4 RTT-Durchsatz [ms]	35.10->	8.20	0.23	34.80->	3.10	0.09
10 Leerlaufzeit max [ms]	1584.44->	1386.44	0.88	862.76->	476.33	0.55

Tabelle 4.3: Leistungskenngrößen für nicht parallele und parallele Übertragung von 180 KB

- In den nicht parallelen Datenübertragungen hat nach wie vor die `Verbindung1` einen besseren Durchsatz mit größerer Variation als die `Verbindung2`.
- In den parallelen Datenübertragungen erzielt die `Verbindung2` bessere Leistungskenngrößen als die `Verbindung1`, obwohl letztere eine bessere Internet-Anbindung hat. Der Grund dafür ist die erhöhte RTT, die für die `Verbindung1` von 27 ms (nicht parallel) auf 35 ms (parallel) steigt. Für die `Verbindung2` ist diese Änderung von 34 (nicht parallel) ms auf 35 ms (parallel) vernachlässigbar.
- Die erhöhte RTT hat eine erhöhte Leerlaufzeit zur Folge, die für die `Verbindung1` von 198 ms (nicht parallel) ms auf 1584 ms steigt. Für die `Verbindung2` ist der Unterschied klein, denn die Leerlaufzeit ist 797 ms (nicht parallel) bzw. 863 ms (parallel).

Zu beachten bei den Messergebnissen für die Übertragung von 1 MB ist:

Kenngroesse	client1 avg->	stdv	cv	client2 avg->	stdv	cv
1 Durchsatz [KB/s]	340->	259	0.76	403->	228	0.57
4 RTT-Mitelwert [ms]	21.40->	5.40	0.25	21.00->	6.80	0.32
8 Triple DUPACKs	11.7->	8.4	0.72	10.4->	8.2	0.79
11 Retransmissionszeit max [ms]	2047.95->	7237.84	3.53	1150.96->	4823.41	4.19
13 Retransmissionszeit max [%]	15.41->	16.98	1.10	7.17->	10.75	1.50

Tabelle 4.4: Leistungskenngrößen für die Übertragung von 1 MB

- Der Durchsatz der `Verbindung2` ist besser als der von `Verbindung1`. Der Grund liegt an der höheren Anzahl von Retransmissionen für `Verbindung1` mit entsprechenden Wartezeiten. Sie betragen 15 bzw. 7 % der Übertragungszeit.
- Diese Tendenz zeigt sich sowohl in den parallelen als auch in den nicht parallelen Datenübertragungen.

Im Abschnitt 4.5.3 wird diese Situation noch näher anhand einiger Beispiele erläutert.

4.5.3 Paketverlust am Anfang von Slow-Start

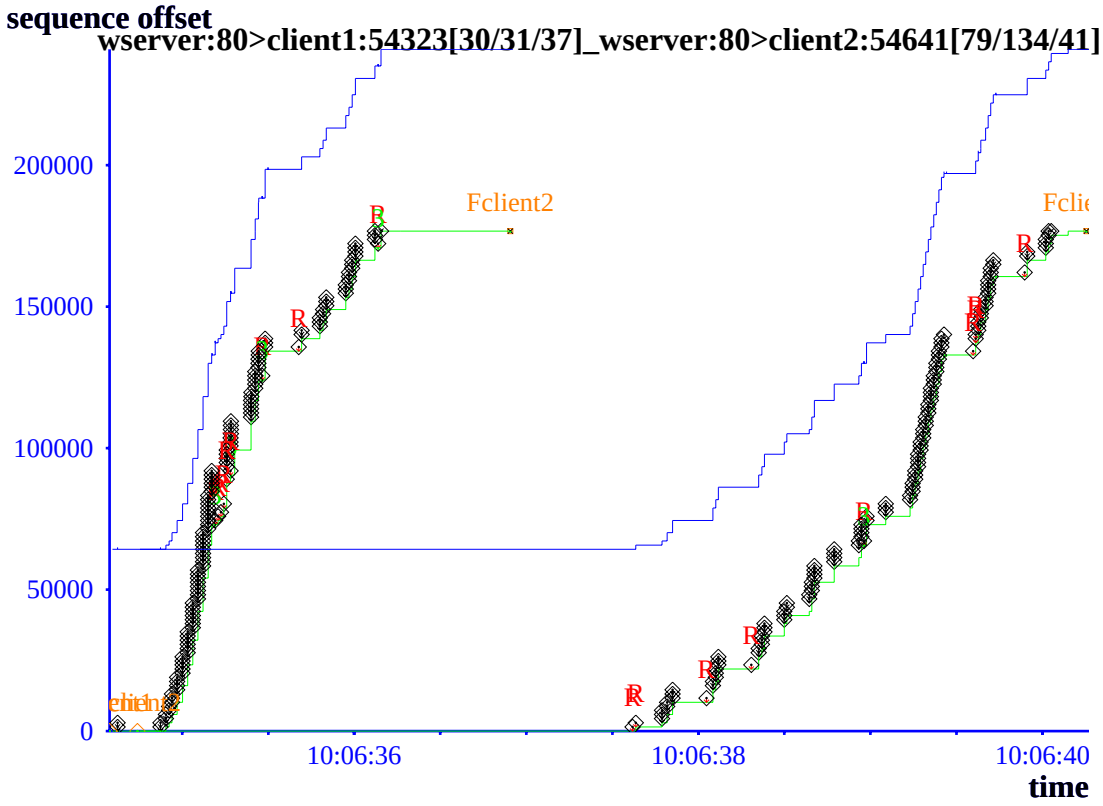


Abbildung 4.9: TCP-Messung - Paketverlust am Anfang von Slow-Start

Die Abbildung 4.9 wurde mit `xplot` [7] erstellt und zeigt die parallele Übertragung von ca. 180 KB zu `client1` über `Verbindung1` und `client2` über `Verbindung2`. Dabei ist Folgendes zu erkennen:

- Im Titel stehen die Quell- und Ziel-Hosts der Datenübertragungen (`host:port`). In eckigen Klammern steht der Durchsatz der Verbindung in KB/s, der Durchsatz der Datenübertragung ohne Verbindungsaufbauphase in KB/s, und die RTT in ms. In der Abbildung steht links die Übertragung von `wserver` auf Port `80` zu `client2` auf Port `54323` und rechts zu `client1` auf Port `54641`
- Die Zeit ist auf der X-, die Sequenznummer (in Bytes) auf der Y-Achse dargestellt.
- Die untere dünne Linie zeigt das maximale quittierte Segment (`snd_una`).
- Die obere dünne Linie zeigt das maximale Segment, das gesendet werden darf (`snd_max`). Die Differenz zwischen unterer und oberer Linie ist das RWND (`rcv_wnd`).
- Die kleinen Rauten zwischen den zwei Linien sind die gesendeten Segmente.
- Beide Linien sind treppenförmig. Die horizontale Ausdehnung der Stufen (X-Achse) ist die RTT; die vertikale Ausdehnung (Y-Achse) ist die Anzahl an quittierten Bytes.

Die Datenübertragung zu `client1` erreicht einen Durchsatz von 30 KB/s mit RTT 37 ms. Der Durchsatz für `client1` liegt bei 179 KB/s mit Standardabweichung 106 KB/s und RTT 30 ms (siehe Tabelle C.1 (a)). In diesem Fall wird ein Durchsatzverlust von 83% erreicht!

```
   1 10:06:34.578283 client1.54323 > wserver.http: S 2503103614:2503103614(0)
   2 10:06:34.578477 wserver.http > client1.54323: S 2443919716:2443919716(0) ack 2503103615
   3 10:06:34.595153 client1.54323 > wserver.http: . 1:1(0) ack 1 win 64240
   4 10:06:34.624141 client1.54323 > wserver.http: P 1:159(158) ack 1 win 64240
   5 10:06:34.624174 wserver.http > client1.54323: . 1:1(0) ack 159 win 31962
  *6 10:06:34.624918 WSERVER.HTTP > CLIENT1.54323: P 1:1461(1460) ACK 159 WIN 32120
  *7 10:06:34.624934 WSERVER.HTTP > CLIENT1.54323: P 1461:2921(1460) ACK 159 WIN 32120
   8 10:06:34.739904 client2.54641 > wserver.http: S 4040330824:4040330824(0) win 64240
   9 10:06:34.739964 wserver.http > client2.54641: S 2454951102:2454951102(0) ack 4040330825
  10 10:06:34.755500 client2.54641 > wserver.http: . 1:1(0) ack 1 win 64240
  11 10:06:34.873211 client2.54641 > wserver.http: P 1:159(158) ack 1 win 64240
  12 10:06:34.873240 wserver.http > client2.54641: . 1:1(0) ack 159 win 31962
  13 10:06:34.874059 wserver.http > client2.54641: P 1:1461(1460) ack 159 win 32120
...
...
...
 202 10:06:36.921948 client2.54641 > wserver.http: . 160:160(0) ack 176698 win 64240
*203 10:06:37.616886 WSERVER.HTTP > CLIENT1.54323: P 1:1461(1460) ACK 159 WIN 32120
 204 10:06:37.635334 client1.54323 > wserver.http: . 159:159(0) ack 1461 win 64240
 205 10:06:37.635354 wserver.http > client1.54323: P 1461:2921(1460) ack 159 win 32120
 206 10:06:37.789007 client1.54323 > wserver.http: . 159:159(0) ack 2921 win 64240
 207 10:06:37.789025 wserver.http > client1.54323: P 2921:4381(1460) ack 159 win 32120
 208 10:06:37.789036 wserver.http > client1.54323: P 4381:5841(1460) ack 159 win 32120
 209 10:06:37.789047 wserver.http > client1.54323: P 5841:7301(1460) ack 159 win 32120
...
...
```

Abbildung 4.10: `tcpdump`-Protokolldatei - Paketverlust am Anfang von Slow-Start

Die genaue Untersuchung der Datenübertragung zeigt, dass der Verlust eines einzelnen Pakets diesen Leistungsverlust verursacht. Dies ist an der großen RTT zu sehen. In Abbildung 4.10 [8] wird gezeigt,

[7] `http://www.xplot.org`

[8] `tcpdump`-Protokolldatei. Für die Erklärung des Formats siehe Anhang C.1

dass das Paket mit der Sequenznummer 1 von `wserver` nach `client1` zweimal übertragen wurde. Entweder ist das Segment 1 oder die Quittung des Empfängers verloren gegangen. Tatsache ist, dass zwischen der Zeit 10:06:34.624918 (Zeile 6) und 10:06:37.616886 (Zeile 203) keine Quittung beim Sender eingetroffen ist. Dies bewirkt die neue Übertragung dieses Segments.

(i) Zeile 1 ist das von `client1` gesendete Segment zum Verbindungsaufbau.
(ii) Zeile 2 ist die Antwort von `wserver`. Zu diesem Zeitpunkt wird der RTO auf den initialen Wert von 3 Sekunden gesetzt.
(iii) Mit den Zeilen 6 und 7 wird das CWND mit dem initialen Wert von 2 Segmenten geöffnet. Der RTO bleibt weiterhin auf 3 Sekunden gesetzt. Solange diese zwei Segmente nicht quittiert sind, kann nichts mehr gesendet werden.
(iv) Nach Ablauf des Retransmission-Timers wird das letzte nicht quittierte Segment erneut gesendet (Zeile 203).

Interessant ist, dass während dieser 3 Sekunden Wartezeit eine andere Datenübertragung stattgefunden hat, die in der Zeile 8 von `client2` aufgebaut wurde. Diese Datenübertragung endet 700 ms vor der Retransmission des Pakets der `Verbindung1`. Der Durchsatz für `Verbindung1` ist 30 KB/s, und 79 KB/s für `Verbindung2`. Das Netz war nicht ausgelastet, lediglich der Sender der `Verbindung1` konnte das CWND nicht in gleichem Umfang wie derjenige von `Verbindung2` vergrößern.

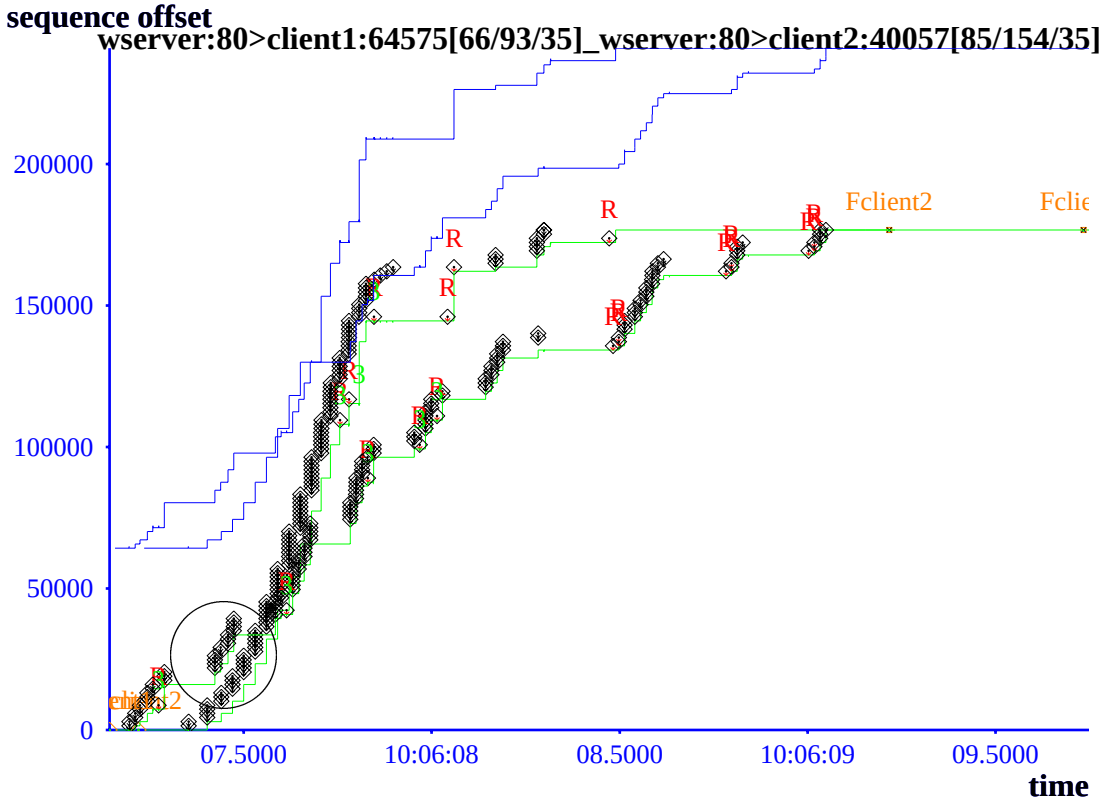
4.5.4 Wachstum des CWND während SS und CA

Abbildung 4.11 (a) zeigt zwei parallele Übertragungen von `wserver` zu `client1` über `Verbindung1` und `client2` über `Verbindung2`. Die `Verbindung1` startet vor der `Verbindung2` aber der Durchsatz für die zweite ist besser (85 KB/s gegenüber 66 KB/s), da das CWND der `Verbindung1` nicht schnell genug wächst.

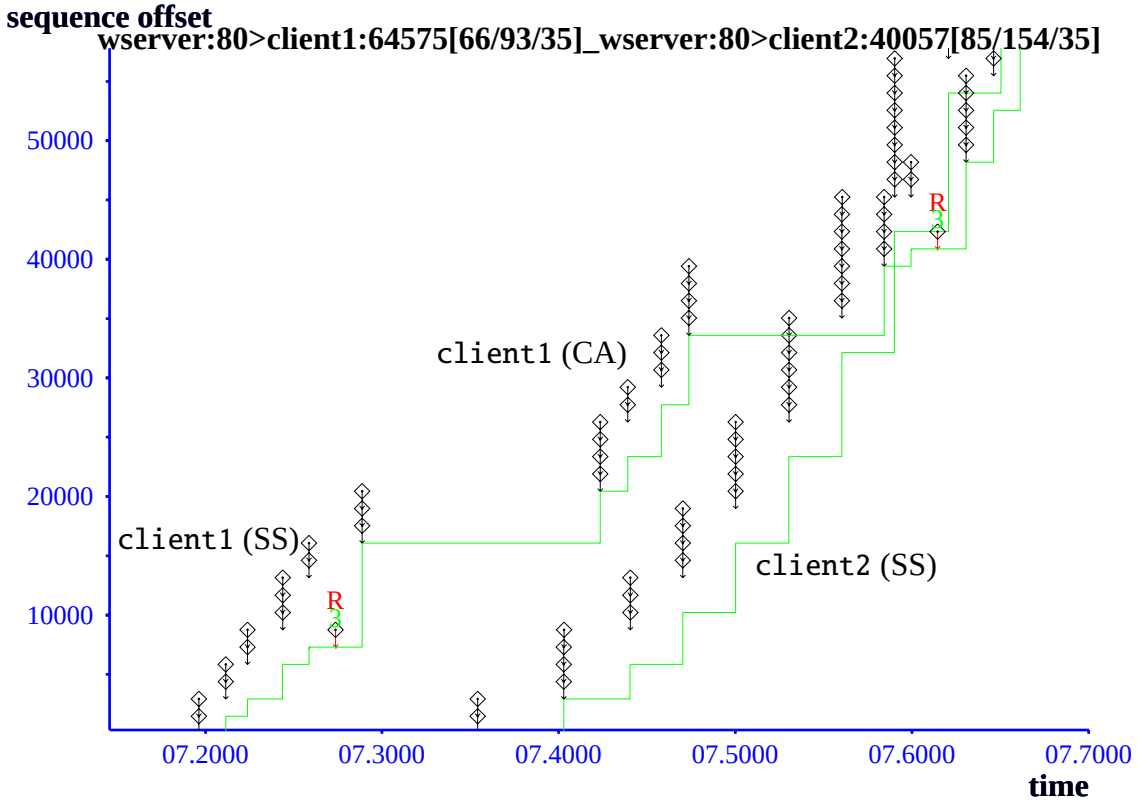
In Abbildung 4.11 (b) wird die Übertragung einiger Fenster beider Verbindungen im Detail gezeigt, um die RTT und das Wachstum des CWND deutlicher zu erläutern. Die RTT der `Verbindung1` beträgt ungefähr die Hälfte derjenigen der `Verbindung2`, aber ihre Abweichung ist größer. Das CWND bleibt bei vier Segmenten, was bedeutet, dass die `Verbindung1` sich im Congestion-Avoidance-Modus befindet. Der Sender der `Verbindung2` kann währenddessen das CWND weiter vergrößern, da sich der Prozess im Slow-Start-Modus befindet.

Diese Details zeigt die Protokolldatei der Abbildung 4.12. Für die `Verbindung1` sind dies:

(i) Zur Zeit 10:06.07 sendet `wserver` ein Segment mit Sequenznummer 18981 (Zeile 32). Dieses Segment wird erst 135 ms später in Zeile 43 quittiert. Die `Verbindung1` befindet sich im Congestion-Avoidance-Modus.
(ii) Zeilen 44-47, 49-50, 56-58 und 65-68 entsprechen den vier Fenstern von Segmenten, die in der Abbildung erkennbar sind. Zeilen 48, 55, 64 und 90 entsprechen ihren Teil-Quittungen, die in der Abbildung durch die treppenförmigen dünnen Linien dargestellt sind. Die Quittungen der Zeilen 48, 55, 64 und 90 ergeben RTT von 16, 19, 16 und 111 ms. Der TCP-Prozess befindet sich im Congestion-Avoidance-Modus, da das CWND während der Übertragungszeit um $4 \times mss$ Bytes erhöht wird. Während der Übertragungszeit der Fenster 1 bis 4 ist das CWND ausgeschöpft und der RTO größer als 100 ms als Folge der längeren RTT der Zeile 43.
(iii) Zeile 48 ist die Quittung der Segmente in Zeilen 44 und 45 des Fensters 1, daher werden in Fenster 2 weitere zwei Pakete gesendet (Zeilen 49 und 50).



(a) Gesamt



(b) Detail

Abbildung 4.11: TCP-Messung - Wachstum des CWND während CA und SS

```
 32 10:06:07.288765 wserver.http > client1.64575: P 18981:20441(1460) ack 159 win 32120
 33 10:06:07.353151 client2.40057 > wserver.http: P 1:159(158) ack 1 win 64240
 34 10:06:07.353183 wserver.http > client2.40057: . 1:1(0) ack 159 win 31962
 35 10:06:07.354038 wserver.http > client2.40057: P 1:1461(1460) ack 159 win 32120
 36 10:06:07.354057 wserver.http > client2.40057: P 1461:2921(1460) ack 159 win 32120
 37 10:06:07.402857 client2.40057 > wserver.http: . 159:159(0) ack 1461 win 64240
 38 10:06:07.402899 wserver.http > client2.40057: P 2921:4381(1460) ack 159 win 32120
 39 10:06:07.402924 wserver.http > client2.40057: P 4381:5841(1460) ack 159 win 32120
 40 10:06:07.402910 client2.40057 > wserver.http: . 159:159(0) ack 2921 win 64240
 41 10:06:07.402946 wserver.http > client2.40057: P 5841:7301(1460) ack 159 win 32120
 42 10:06:07.402973 wserver.http > client2.40057: P 7301:8761(1460) ack 159 win 32120
+43 10:06:07.423480 CLIENT1.64575 > WSERVER.HTTP: . 159:159(0) ACK 20441 WIN 64240
*44 10:06:07.423506 WSERVER.HTTP > CLIENT1.64575: P 20441:21901(1460) ACK 159 WIN 32120
*45 10:06:07.423518 WSERVER.HTTP > CLIENT1.64575: P 21901:23361(1460) ACK 159 WIN 32120
*46 10:06:07.423529 WSERVER.HTTP > CLIENT1.64575: P 23361:24821(1460) ACK 159 WIN 32120
*47 10:06:07.423543 WSERVER.HTTP > CLIENT1.64575: P 24821:26281(1460) ACK 159 WIN 32120
+48 10:06:07.439043 CLIENT1.64575 > WSERVER.HTTP: . 159:159(0) ACK 23361 WIN 64240
*49 10:06:07.439064 WSERVER.HTTP > CLIENT1.64575: P 26281:27741(1460) ACK 159 WIN 32120
*50 10:06:07.439074 WSERVER.HTTP > CLIENT1.64575: P 27741:29201(1460) ACK 159 WIN 32120
 51 10:06:07.440469 client2.40057 > wserver.http: . 159:159(0) ack 5841 win 64240
 52 10:06:07.440490 wserver.http > client2.40057: P 8761:10221(1460) ack 159 win 32120
 53 10:06:07.440500 wserver.http > client2.40057: P 10221:11681(1460) ack 159 win 32120
 54 10:06:07.440512 wserver.http > client2.40057: P 11681:13141(1460) ack 159 win 32120
+55 10:06:07.458180 CLIENT1.64575 > WSERVER.HTTP: . 159:159(0) ACK 27741 WIN 64240
*56 10:06:07.458201 WSERVER.HTTP > CLIENT1.64575: P 29201:30661(1460) ACK 159 WIN 32120
*57 10:06:07.458212 WSERVER.HTTP > CLIENT1.64575: P 30661:32121(1460) ACK 159 WIN 32120
*58 10:06:07.458225 WSERVER.HTTP > CLIENT1.64575: P 32121:33581(1460) ACK 159 WIN 32120
 59 10:06:07.470256 client2.40057 > wserver.http: . 159:159(0) ack 10221 win 64240
 60 10:06:07.470281 wserver.http > client2.40057: P 13141:14601(1460) ack 159 win 32120
 61 10:06:07.470291 wserver.http > client2.40057: P 14601:16061(1460) ack 159 win 32120
 62 10:06:07.470302 wserver.http > client2.40057: P 16061:17521(1460) ack 159 win 32120
 63 10:06:07.470311 wserver.http > client2.40057: P 17521:18981(1460) ack 159 win 32120
+64 10:06:07.473729 CLIENT1.64575 > WSERVER.HTTP: . 159:159(0) ACK 33581 WIN 64240
*65 10:06:07.473752 WSERVER.HTTP > CLIENT1.64575: P 33581:35041(1460) ACK 159 WIN 32120
*66 10:06:07.473764 WSERVER.HTTP > CLIENT1.64575: P 35041:36501(1460) ACK 159 WIN 32120
*67 10:06:07.473776 WSERVER.HTTP > CLIENT1.64575: P 36501:37961(1460) ACK 159 WIN 32120
*68 10:06:07.473788 WSERVER.HTTP > CLIENT1.64575: P 37961:39421(1460) ACK 159 WIN 32120
 69 10:06:07.500146 client2.40057 > wserver.http: . 159:159(0) ack 16061 win 64240
 70 10:06:07.500169 wserver.http > client2.40057: P 18981:20441(1460) ack 159 win 32120
 71 10:06:07.500180 wserver.http > client2.40057: P 20441:21901(1460) ack 159 win 32120
 72 10:06:07.500191 wserver.http > client2.40057: P 21901:23361(1460) ack 159 win 32120
 73 10:06:07.500200 wserver.http > client2.40057: P 23361:24821(1460) ack 159 win 32120
 74 10:06:07.500227 wserver.http > client2.40057: P 24821:26281(1460) ack 159 win 32120
 75 10:06:07.530355 client2.40057 > wserver.http: . 159:159(0) ack 23361 win 64240
 76 10:06:07.530379 wserver.http > client2.40057: P 26281:27741(1460) ack 159 win 32120
 77 10:06:07.530389 wserver.http > client2.40057: P 27741:29201(1460) ack 159 win 32120
 78 10:06:07.530400 wserver.http > client2.40057: P 29201:30661(1460) ack 159 win 32120
 79 10:06:07.530409 wserver.http > client2.40057: P 30661:32121(1460) ack 159 win 32120
 80 10:06:07.530446 wserver.http > client2.40057: P 32121:33581(1460) ack 159 win 32120
 81 10:06:07.530476 wserver.http > client2.40057: P 33581:35041(1460) ack 159 win 32120
 82 10:06:07.560313 client2.40057 > wserver.http: . 159:159(0) ack 32121 win 64240
 83 10:06:07.560338 wserver.http > client2.40057: P 35041:36501(1460) ack 159 win 32120
 84 10:06:07.560348 wserver.http > client2.40057: P 36501:37961(1460) ack 159 win 32120
 85 10:06:07.560359 wserver.http > client2.40057: P 37961:39421(1460) ack 159 win 32120
 86 10:06:07.560369 wserver.http > client2.40057: P 39421:40881(1460) ack 159 win 32120
 87 10:06:07.560401 wserver.http > client2.40057: P 40881:42341(1460) ack 159 win 32120
 88 10:06:07.560432 wserver.http > client2.40057: P 42341:43801(1460) ack 159 win 32120
 89 10:06:07.560450 wserver.http > client2.40057: P 43801:45261(1460) ack 159 win 32120
+90 10:06:07.584200 CLIENT1.64575 > WSERVER.HTTP: . 159:159(0) ACK 39421 WIN 64240
...
```

Abbildung 4.12: `tcpdump`-Protokolldatei - Wachstum des CWND während CA und SS

(iv) Zeile 55 ist die Quittung der Segmente in Zeilen 46 bis 49, daher werden in Fenster 3 weitere drei Pakete gesendet (Zeilen 56 bis 58).
(v) Zeile 64 ist die Quittung der Segmente in Zeilen 56 bis 58, daher werden in Fenster 4 weitere vier Pakete gesendet (Zeilen 65 bis 68).
(vi) Das nächste Fenster wird erst nach der Quittung 111 ms später gesendet (Zeile 90).

Details der `Verbindung2` sind in Abbildung 4.12 zu sehen:

(i) Zeilen 35 und 36 sind die ersten zwei Segmente von `wserver` zu `client2`.
(ii) Die Quittung kommt mit Segment in Zeile 38, und somit ist die RTT 49 ms.
(iii) Das CWND kann damit weiter im Slow-Start-Modus vergrößert werden.

4.5.5 Effekte des RTO

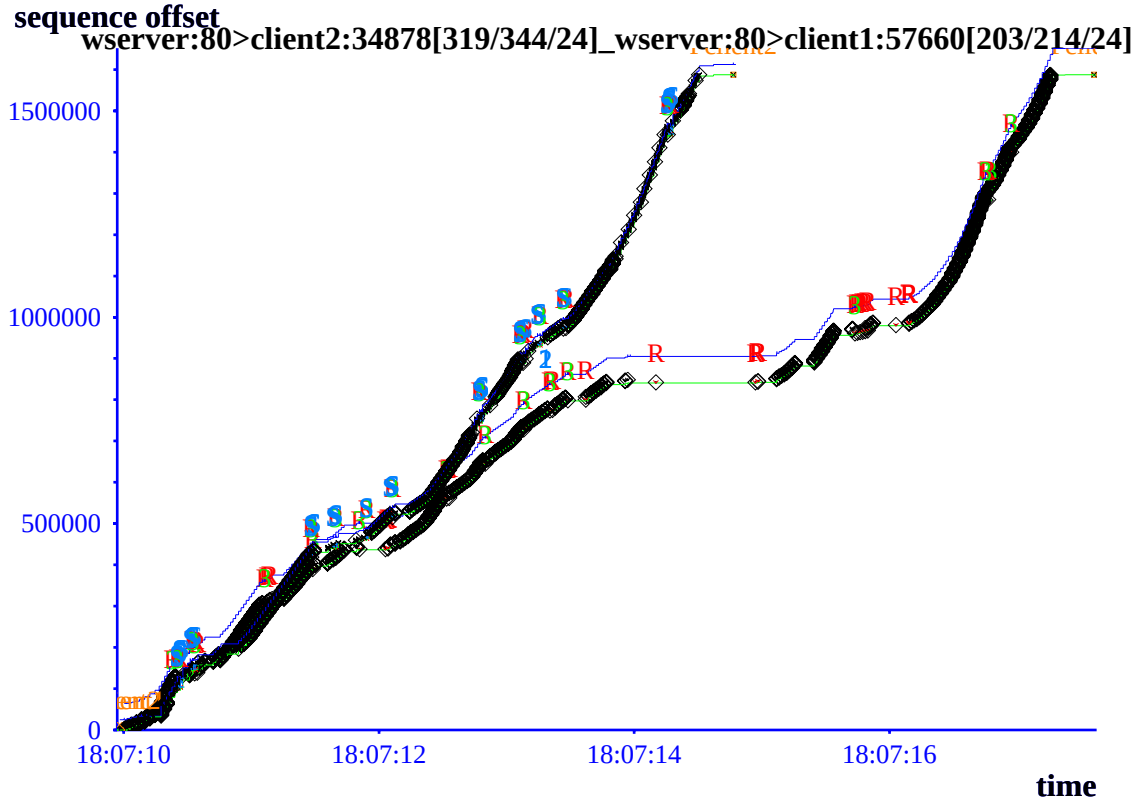
Wenn das CWND klein ist und ein Paket verloren geht, kann Fast-Retransmit als Recovery-Mechanismus nicht eingesetzt werden. In der Mitte einer Übertragung ist dies der Fall, wenn konsekutive RT-Ereignisse bewirken, dass der SSTH mehrmals halbiert und der RTO mehrmals erhöht werden (siehe Abbildung 4.14). In dieser Abbildung sind Datenübertragungen gezeigt, die Durchsätze von 203 und 319 KB/s erreichen. Die zweite ist `Verbindung1`, dauert ca. 7 Sekunden und überträgt ca. 1027 Pakete, wobei mehr als 14 % dieser Zeit für die Retransmission eines einzigen Pakets notwendig ist.

Die Details der `Verbindung1` sind in Abbildung 4.13 dargestellt:

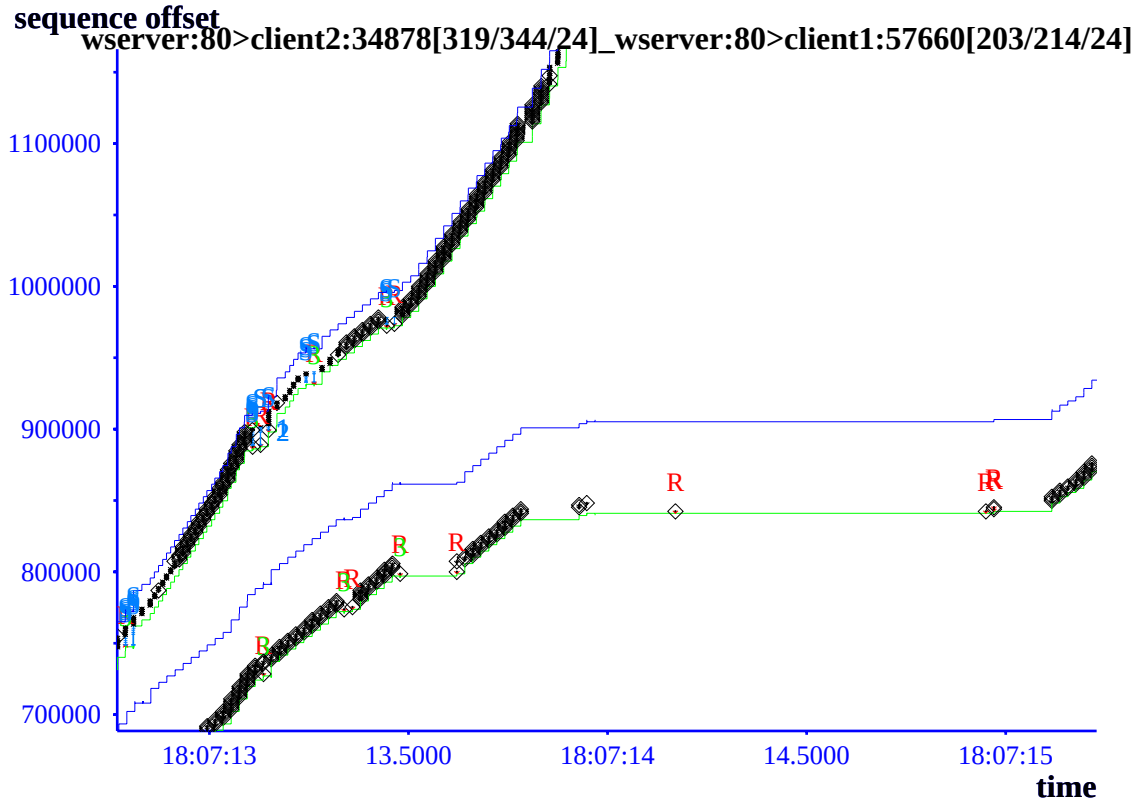
(i) Nach der Quittung in Zeile 1855 werden drei weitere Segmente gesendet, denn $cwnd \leq 5$.

```
 1855 18:07:13.781998 client1.57660 > wserver.http: . 159:159(0) ack 836581 win 64240
 1856 18:07:13.782022 wserver.http > client1.57660: P 839501:840961(1460) ack 159 win 32120
*1857 18:07:13.782032 WSERVER.HTTP > CLIENT1.57660: P 840961:842421(1460) ACK 159 WIN 32120
 1858 18:07:13.782044 wserver.http > client1.57660: P 842421:843881(1460) ack 159 win 32120
...
 1933 18:07:13.927801 client1.57660 > wserver.http: . 159:159(0) ack 839501 win 64240
 1934 18:07:13.927827 wserver.http > client1.57660: P 843881:845341(1460) ack 159 win 32120
 1935 18:07:13.927837 wserver.http > client1.57660: P 845341:846801(1460) ack 159 win 32120
...
+1948 18:07:13.947751 CLIENT1.57660 > WSERVER.HTTP: . 159:159(0) ACK 840961 WIN 64240
 1949 18:07:13.947772 wserver.http > client1.57660: P 846801:848261(1460) ack 159 win 32120
+1950 18:07:13.947808 CLIENT1.57660 > WSERVER.HTTP: . 159:159(0) ACK 840961 WIN 64240
...
+1963 18:07:13.968102 CLIENT1.57660 > WSERVER.HTTP: . 159:159(0) ACK 840961 WIN 64240
...
...
*2107 18:07:14.170021 WSERVER.HTTP > CLIENT1.57660: P 840961:842421(1460) ACK 159 WIN 32120
...
...
...
...
*2320 18:07:14.950034 WSERVER.HTTP > CLIENT1.57660: P 840961:842421(1460) ACK 159 WIN 32120
 2321 18:07:14.969815 client1.57660 > wserver.http: . 159:159(0) ack 842421 win 64240
...
```

Abbildung 4.13: `tcpdump`-Protokolldatei - Effekte des RTO



(a) Gesamte Datenübertragung



(b) Detail der Verdrängung

Abbildung 4.14: TCP-Messung - Effekte des RTO

(ii) Nach der Quittung in Zeile 1933 werden zwei weitere Segmente gesendet.
(iii) Das Segment in Zeile 1857 ging verloren, denn Zeilen 1948 und 1950 sind die Quittungen des Segments in Zeile 1856 bzw. ein DUPACK (Quittung des Segments in Zeile 1858).
(iv) Zeile 1963 ist ein DUPACK (Quittung des Segments in Zeile 1934).
(v) Zeile 2107 ist die Retransmission des Segments in Zeile 1857 nach einem RT-Ereignis von ca. 200 ms. Fast-Retransmit konnte nicht eingesetzt werden!
(vi) Zeile 2320 ist die Retransmission des Segments in Zeile 1857 nach einem RT-Ereignis von ca. 780 ms. Insgesamt werden ca. 990 ms für die Retransmission eines Pakets gebraucht!

4.5.6 Effekte von zyklischem Paketverlust und RTO

```
 1794 05:27:31.931507 client1.36271 > wserver.http: . 159:159(0) ack 908121 win 64240
*1795 05:27:31.931527 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
 1796 05:27:31.931538 wserver.http > client1.36271: P 912501:913961(1460) ack 159 win 32120
 1797 05:27:31.935894 client1.36271 > wserver.http: . 159:159(0) ack 911041 win 64240
 1798 05:27:31.935913 wserver.http > client1.36271: P 913961:915421(1460) ack 159 win 32120
 1799 05:27:31.935923 wserver.http > client1.36271: P 915421:916881(1460) ack 159 win 32120
 1800 05:27:31.965818 client2.59773 > wserver.http: . 159:159(0) ack 772341 win 24820
 1801 05:27:31.965838 wserver.http > client2.59773: P 773801:775261(1460) ack 159 win 32120
 1802 05:27:31.965848 wserver.http > client2.59773: P 775261:776721(1460) ack 159 win 32120
*1803 05:27:32.130015 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*1805 05:27:32.530015 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*1852 05:27:33.330015 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*1891 05:27:34.930028 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*1959 05:27:38.130019 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*2131 05:27:44.530018 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
...
*2615 05:27:57.330018 WSERVER.HTTP > CLIENT1.36271: P 911041:912501(1460) ACK 159 WIN 32120
+2616 05:27:57.369490 CLIENT1.36271 > WSERVER.HTTP: . 159:159(0) ACK 916881 WIN 64240
...
 2713 05:27:59.249392 client2.59773 > wserver.http: . 159:159(0) ack 1578261 win 24820
*2714 05:27:59.249410 WSERVER.HTTP > CLIENT2.59773: P 1581181:1582641(1460) ACK 159 WIN 32120
 2715 05:27:59.249420 wserver.http > client2.59773: P 1582641:1584101(1460) ack 159 win 32120
...
 2720 05:27:59.264454 client2.59773 > wserver.http: . 159:159(0) ack 1581181 win 24820
 2721 05:27:59.264474 wserver.http > client2.59773: P 1584101:1585561(1460) ack 159 win 32120
 2722 05:27:59.264484 wserver.http > client2.59773: P 1585561:1587021(1460) ack 159 win 32120
...
*2724 05:27:59.440015 WSERVER.HTTP > CLIENT2.59773: P 1581181:1582641(1460) ACK 159 WIN 32120
...
*2726 05:27:59.840015 WSERVER.HTTP > CLIENT2.59773: P 1581181:1582641(1460) ACK 159 WIN 32120
...
*2728 05:28:00.640014 WSERVER.HTTP > CLIENT2.59773: P 1581181:1582641(1460) ACK 159 WIN 32120
...
+2732 05:28:00.674406 CLIENT2.59773 > WSERVER.HTTP: . 159:159(0) ACK 1587021 WIN 24820
```

Abbildung 4.15: `tcpdump`-Protokolldatei - Effekte von zyklischem Paketverlust und RTO

Folgendes Beispiel zeigt die Effekte von zyklischen Paketverlusten (siehe Abbildung 4.16): Der Durchsatz sinkt um mehr als 90 %. Die Datenübertragungen werden durch Paketverluste periodisch gestoppt, was jeweils eine Halbierung des SSTH zur Folge hat. Das CWND wird so klein, dass Fast-Retransmit, Fast-Recovery und SACK nicht greifen. Periodisch gibt es in beiden Verbindungen drei Retransmissionen. Nach der Hälfte der Übertragungszeit kommt es bei `Verbindung1` zu weiteren Paketverlusten, was den RT-Mechanismus sieben Mal in Gang setzt, wie in Abbildung 4.15 zu sehen ist. Gegen Ende der Übertragung erfährt der Sender über die `Verbindung2` eine ähnliche Situation mit mehr als 3 Retransmissionen. Daher schneiden beide Verbindungen in etwa gleich schlecht ab.

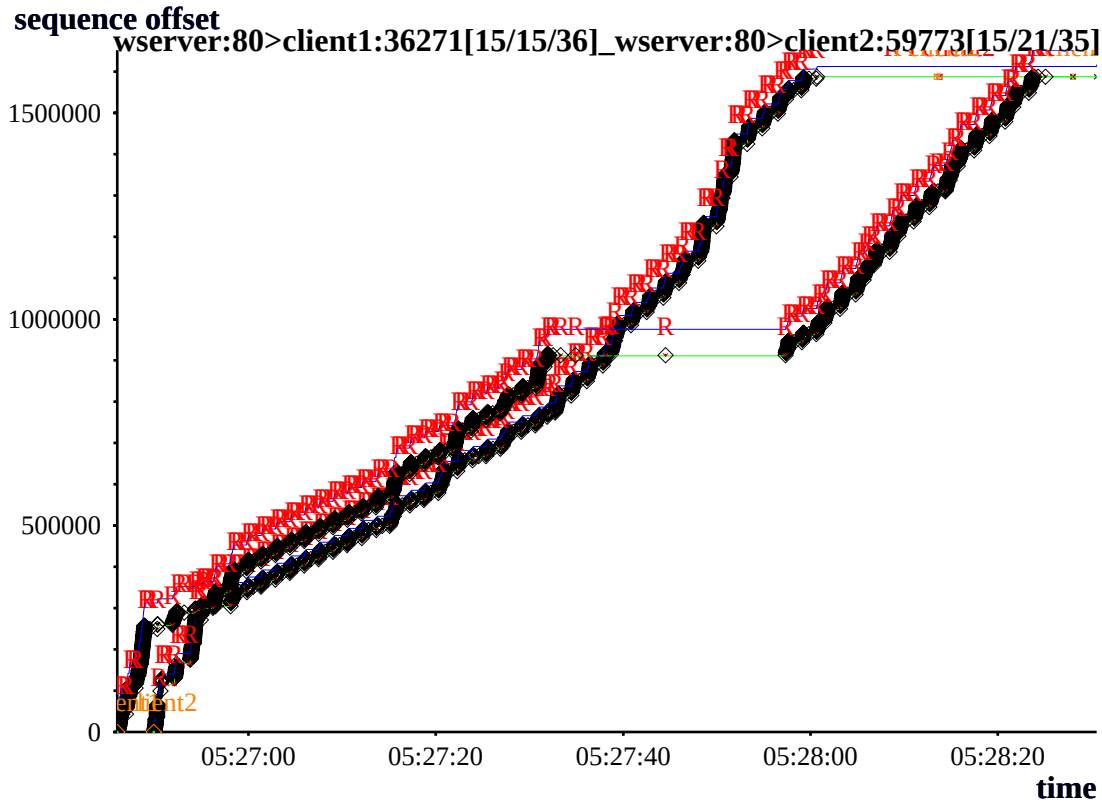
(i) Das Segment in Zeile 1795 geht verloren. Für die `Verbindung1` gilt zu diesem Zeitpunkt $cwnd \approx 4$.
(ii) Zeile 1803 ist die Retransmission des Segments in Zeile 1795 nach einem RT-Ereignis von ca. 199 ms.
(iii) Zeilen 1805 bis 2615 sind die weiteren Retransmissionen des gleichen Segments. Nach jeder Retransmission wird der RTO verdoppelt. Der letzte Wert von RTO ist etwa 13 s.
(iv) Zeile 2616 ist die Quittung für alle ausstehenden Segmente der `Verbindung1` bis Zeile 1799. Ab diesem Punkt wiederholt sich das Muster mit den drei Retransmissionen.
(v) Das Segment in Zeile 2714 der `Verbindung2` geht verloren.
(vi) Zeilen 2724 bis 2728 sind die Retransmission dieses Segments.
(vii) Zeile 2732 ist die Quittung des Segments in Zeile 2714.

4.5.7 Zusammenfassung

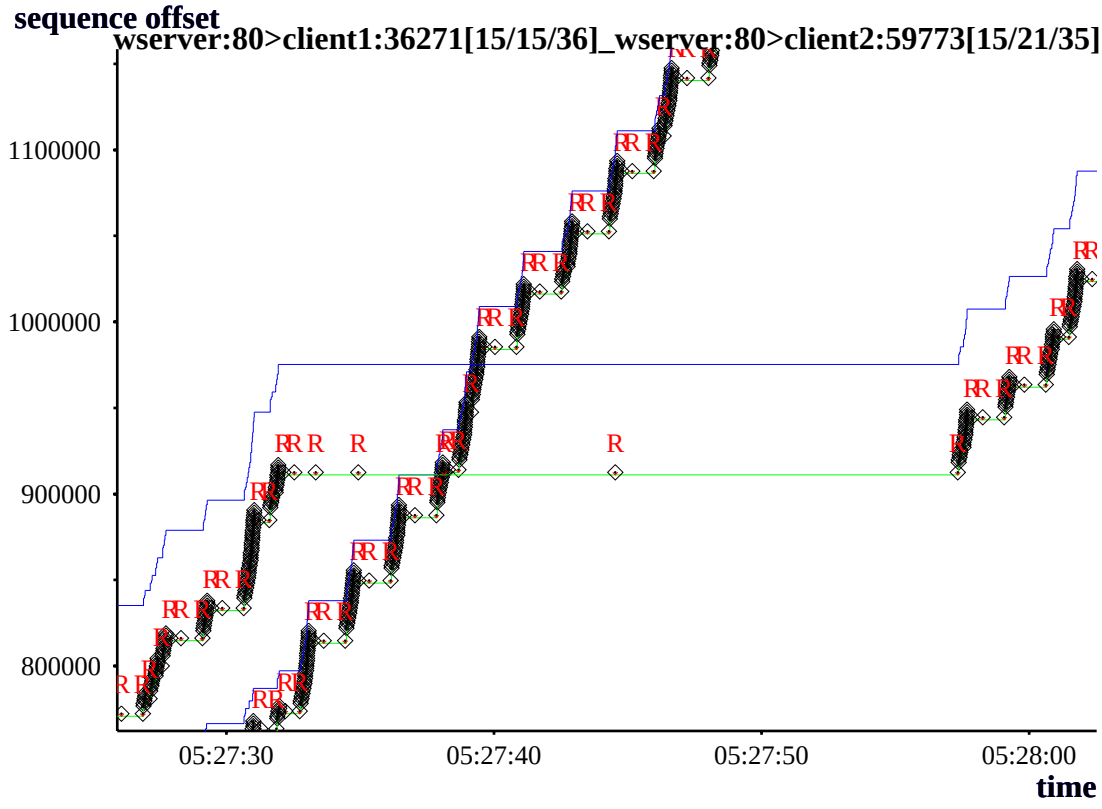
Diese Fallstudie zeigte die Effekte von Paketverlusten in kurzen und langen HTTP-Datenübertragungen. Dabei wurden zwei Clients mit unterschiedlich schneller Internet-Anbindung gewählt.

Es wurde festgestellt, dass ein einziger Paketverlust die Datenübertragung stark negativ beeinflussen kann. Die Effekte sind je nach übertragener Datenmenge unterschiedlich und wirken sich in der Wartezeit zwischen den gesendeten Paketen aus. Die Wartezeit endet nach Ablauf eines Retransmission-Timers oder nach Ankommen einer Quittung. In kurzen Datenübertragungen kann die Wartezeit zwischen den Retransmissionen bis zu 40 % der Übertragungszeit betragen. Noch größer ist die Wartezeit zwischen dem Senden eines Segments und dem Empfangen einer Quittung, die bis zu 50 % der Übertragungszeit betragen kann. In diesen Fällen kann das Fenster nicht vergrößern werden, da der Paketverlust kurz nach dem Anfang der Datenübertragung eintritt und der RTO auf einen großen initialen Vorgabewert gesetzt ist. Als Folge wird der Slow-Start-Modus verlassen und das Fenster nicht vergrößert. Bei langen Datenübertragungen ist dieser Effekt geringer.

Darüber hinaus wurde festgestellt, dass der Client mit schlechterer Internet-Anbindung bessere Leistungskenngrößen bei großen übertragenen Datenmengen aufweist. Bei kurzen Datenübertragungen ist der Durchsatz des Clients mit der besseren Internet-Anbindung etwa 75 % höher, wobei dies nur mit einer großen Variation möglich ist.



(a) Gesamte Datenübertragung



(b) Detail des RTO

Abbildung 4.16: TCP-Messung - Effekte von zyklischem Paketverlust und RTO

Kapitel 5

ANDINA (Adaptive Network Datastream INline Algorithm)

In den vorherigen Kapiteln wurden aktuelle Ansätze zur Überlastkontrolle im Internet diskutiert. Diesbezüglich wurde darauf hingewiesen, dass die Notwendigkeit der Entwicklung neuer dynamischer Verfahren eine ständige Anforderung ist. Neue Techniken und neue Applikationsarten setzten neue Rahmenbedingungen und stellen bis dahin gültige Modelle erneut auf den Prüfstand. Ein Grund dafür ist, dass mit zunehmender Pfadkapazität die Feedback-Signale von Ende-zu-Ende-Überlastkontrolle langsam im Vergleich zu der Geschwindigkeit der Netze sind. Die Anpassungszeit auf die Feedback-Signale hängt dabei von der RTT ab. Außerdem verlangen die Applikationen auf ihre Verkehr-Profile spezifisch zugeschnittene Überlastkontrolle. In diesem Kapitel wird ein Verfahren, genannt ANDINA (**A**daptive **N**etwork **D**atastream **IN**line **A**lgorithm), für die Überlastkontrolle der Best-Effort-Datenübertragung vorgestellt. Mit dem Simulations-Programm `ns`[1] wurde ein Prototyp dieses Modells entwickelt und bewertet.

5.1 Ziele und Anforderungen

ANDINA soll eine Verbesserung des Gesamtdurchsatzes im Netz bei gleichzeitiger Maximierung der Fairness und Minimierung des Paketverlusts erzielen. Die betrachteten Datenübertragungen sind vom Typ Best-Effort sowie in der FTP- oder HTTP-Datenübertragung, d.h. der Sender muss einen Datenstrom von A nach B übertragen. ANDINA ist dann für eine Umgebung entwickelt, in der die unterschiedlichen Verkehrsarten voneinander isoliert werden können, z.B. mittels Warteschlangen.

ANDINA soll dabei Router wie Ende-zu-Ende-Mechanismen einsetzen, jedoch auf den Einsatz vom Paketverlust als Feedback-Signal verzichten. Die vorgesehenen Aufgaben sind vielfältig. Die Router überwachen als Zwischensysteme den Verkehr und generieren Feedback-Signale. Die Sender in den Hosts (Endsysteme) bewerten die Signale und passen ihre Senderaten an.

Daraus resultieren folgende Anforderungen:

[1] `http://www.mash.berkerley.edu/ns`

5.1.1 Weiterleitung präziser Feedback-Signale

Die Router müssen nicht nur qualitative Angaben über ihren Zustand geben können, -ausgelastet oder nicht ausgelastet-, sondern auch präzisere quantitative Feedback-Signale über die geeignete bzw. gewünschte Senderate. Es wird darauf abgezielt, den Sendern die Engpass-Information direkt aus dem Netz mitzuteilen, anstatt sie aus der zur Verfügung stehenden Datenrate mit einem Blackbox-Ansatz abschätzen zu müssen. Dies löst das Problem der Sender, zu stark oder zu vorsichtig auf die Änderung der Last zu reagieren.

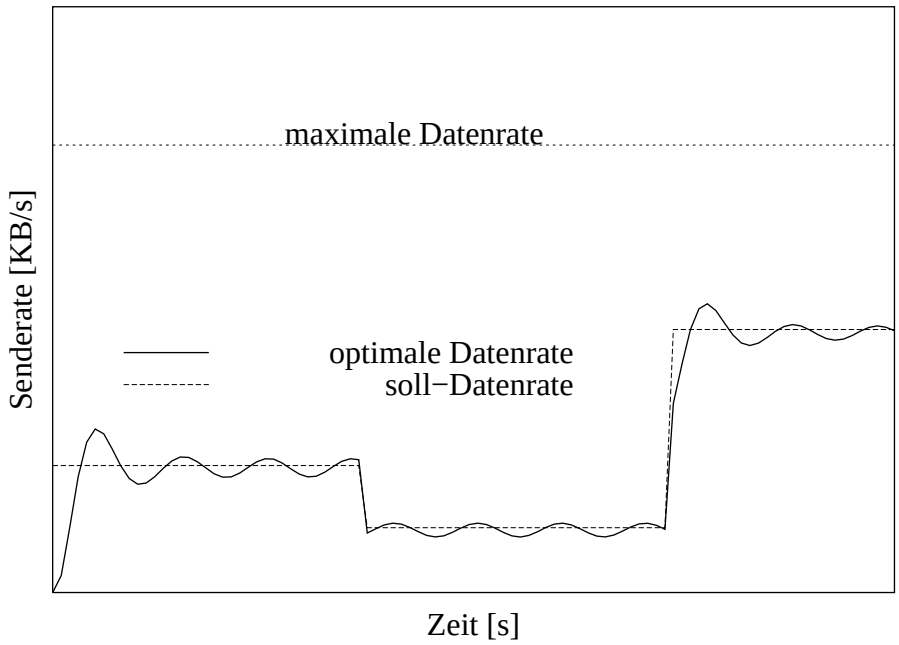


Abbildung 5.1: Anforderung an das ANDINA-Modell

Dies erfolgt, in dem die Router durch Überwachung der Last ein Feedback-Signal mit genug Information generieren und liefern sollen, damit die Sender das optimale Fenster bzw. die optimale Senderate berechnen können. In Abbildung 5.1 ist ein solches Schema für eine einzelne Datenübertragung dargestellt.

5.1.2 Schnelle Anpassung an die Last

Die lastbezogenen Kenngrößen des Netzes müssen dynamisch und schnell ermittelt werden. Hierbei muss sichergestellt werden, dass der Einfluss eines zufälligen Ereignisses sich nicht gravierend auf den Durchsatz auswirkt. Dies kann erreicht werden, in dem die Router mit geringer Pufferauslastung arbeiten.

Eine erste Aufgabe ist die Definition eines Kontrollintervalls zur Generierung der Feedback-Signale. Das Kontrollintervall muss an die Änderung der Last und an die erwartete RTT der Verbindungen gekoppelt sein, denn nur dadurch kann eine sich bildende Überlastung erkannt werden.

Zweitens müssen die Feedback-Signale den Sender schnell und sicher erreichen und sollten keine zusätzliche Last verursachen. Daher werden sie im Paket-Header enthalten sein und sowohl in die gesendeten Pakete als auch in ihre Quittungen geschrieben werden. Die Kosten der Prozess- und Über-

tragungszeit verursacht durch die Erweiterung des Paket-Headers werden durch die geringe Anzahl von zirkulierenden Paketen und die niedrige Paketverlustrate ausgeglichen. Allerdings muss garantiert werden, dass die Erzeugung der Feedback-Signale und ihre Weiterleitung keinen übermäßigen Aufwand verursacht.

5.1.3 Fairness in der Best-Effort-Datenübertragung

Wie in Kapitel 3 erwähnt eignen sich routerbasierte Verfahren dafür, die Fairness zu erhöhen, wenn aggressive Sender mehr als ihren fairen Anteil der Datenrate ausnutzen. Der Mangel an Fairness entsteht vor allem für Verbindungen mit großer RTT, denn die benötigte Reaktionszeit nach einem Paketverlust wirkt sich stark negativ auf den Durchsatz aus.

Die Anzahl der Warteschlangen im Router und ihre Bedienstrategien bestimmen in großem Maß die Fairness. Wird eine einzige Warteschlange für alle Pakete gehalten, kann dieser Zustand von aggressiven Sendern mit kleiner RTT ausgenutzt werden. Werden *n* Warteschlangen der Reihe nach bedient, erreicht ein aggressiver Sender nichts anderes, als seine eigene Warteschlange auszulasten. Für die hier gesuchten Ziele ist dieser Ansatz aus mehreren Gründen jedoch nicht geeignet. Erstens eignet sich eine einzelne Warteschlange besser für transiente Netzüberlastung und burst-artigen Verkehr, wie dies typisch für die hier betrachteten Datenübertragungen ist. Zweitens ist es nicht notwendig für den Best-Effort-Verkehr Ressourcen zu reservieren. Drittens ist das Modell nicht skalierbar, falls die Anzahl von Verbindungen sehr groß ist.

Anstatt dessen sollte die Anzahl der Verbindungen abgeschätzt werden. Dies kann, basierend auf der *Historie* der Puffer, mit einfachen Mechanismen erfolgen. Diese resultierende Kenngröße sollte als Basis für die Generierung der Feedback-Signale eingesetzt werden. Dadurch wird die Leitungskapazität unter den Verbindungen in vorgebbarer Weise aufgeteilt.

Fairness als Ziel kann auch dazu führen, dass alle Datenübertragungen einen gleichen aber geringen Durchsatz erreichen, da die Leitungskapazität begrenzt ist. Wenn die Anzahl der Verbindungen groß ist, ist es schwierig, gleichzeitig einen guten Durchsatz und Fairness zu erhalten. Mit anderen Worten: das Erreichen der Fairness kann zu inakzeptablem Durchsatz führen. In diesem Kontext muss gefragt werden, wie viele Verbindungen das System bedienen kann, damit der Durchsatz noch als akzeptabel bezeichnet werden kann.

5.1.4 Netzgleichgewicht durch stabilere Senderate

Zur Stabilisierung des gesamten Netzes, muss vermieden werden, dass aggressive und durch die gute Antwortzeit ermunterte Sender ihre Senderate zu schnell erhöhen. Instabile Senderaten verursachen große Lastspitzen. Die Signalisierung muss bewirken, dass die Varianz der Senderaten bei Sendern klein wird. Durch stabile Senderaten soll die Aktivierung von RT-Mechanismen vermieden werden.

5.2 Mechanismen zur Überlastkontrolle

Basierend auf diesen Anforderungen wurden die Mechanismen zur Überlastkontrolle in ANDINA entworfen und simulativ untersucht. Nach der Taxonomie in [YR95] kann ANDINA als ein Regelungsverfahren mit expliziter, persistenter, globaler Signalisierung klassifiziert werden.

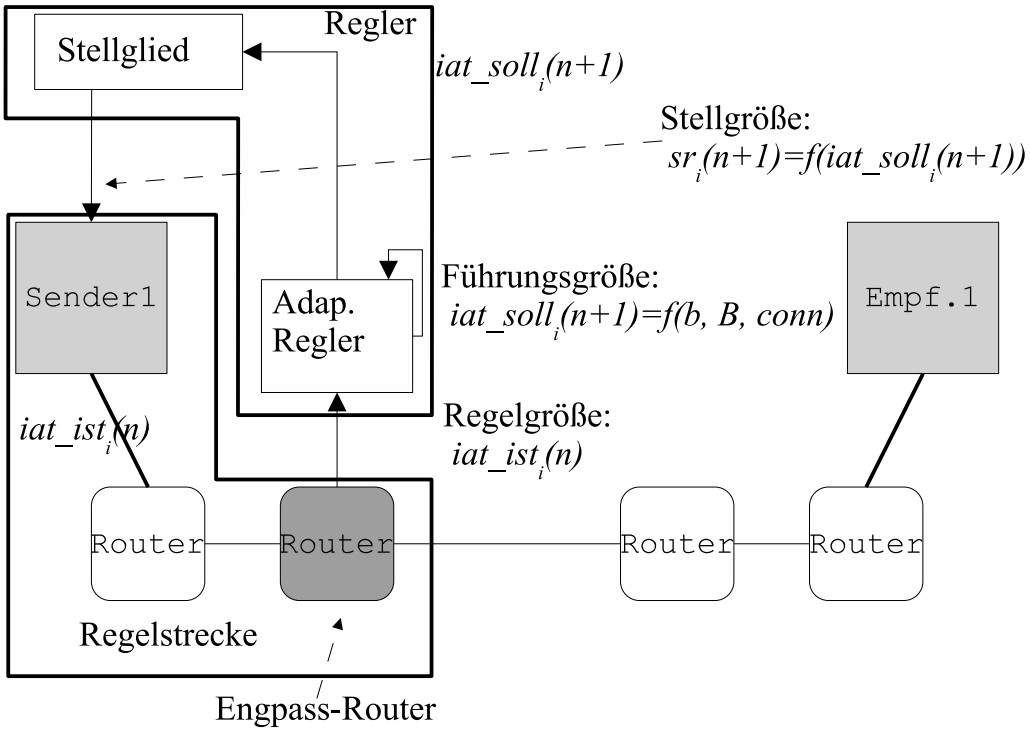


Abbildung 5.2: Modell der ANDINA-Überlastkontrolle

ANDINA besteht im einfachsten Fall aus drei Komponenten: Sender, Router und Empfänger. Die Komponenten kommunizieren mittels eines Paket-Headers.

Als Regelkreis ist das Verfahren in Abbildung 5.2 dargestellt. Router und Hosts übernehmen die Funktionalität des Reglers. Die Regelstrecke geht bis zum Router, der den Engpass der Verbindung darstellt. Die Regelung des Gesamtverkehrs über einen Router wird mittels der Regelung der Zwischenankunftszeit (IAT - *inter arrival time*) jeder einzelnen Datenübertragung *i* erreicht. Zum Zeitpunkt *n* wird die Regelgröße iat_ist_i basierend auf der Führungsgröße iat_soll_i im Router verglichen. Die Führungsgröße wird als Funktion der Paketgröße *b* der Datenübertragung, der zur Verfügung stehenden Datenrate *B* und der Anzahl der aktiven Verbindungen *conn* festgesetzt. Die Router schätzen periodisch die Anzahl von durchquerenden Verbindungen mit statistischen Mechanismen ab. Vorausgesetzt wird die Kenntnis der Router über die Datenrate zu ihren Nachbarn. Die Regelgröße für die nächste Periode wird in den Paket-Header geschrieben. Die Stelltgröße sr_i ist die Senderate der Verbindung und wird in den Hosts als eine Funktion der *iat_soll* eingestellt.

Der Paket-Header enthält die Paketgröße, die IAT und das Kennzeichen des Routers, der die IAT als letzter geändert hat. Die Paketgröße dient zweierlei Zwecken. Erstens kann während der Verbindungsaufbauphase der Router genügend Information über die erwartete Paketgröße der Verbindung erhalten und zweitens kann die IAT in den Quittungen gesetzt werden. Das Kennzeichen dient einem Router zur Feststellung, ob die IAT von ihm selbst gesetzt wurde.

Der Sender kann senderaten- oder fensterbasiert arbeiten.

Im Folgenden werden diese Mechanismen detaillierter beschrieben.

5.2.1 Mechanismen im Router

Abschätzung der Zwischenankunftszeit. Gegeben sei ein Router mit:

B	gesamte Datenrate einer Leitung
$conn$	Anzahl der über dem Router laufenden Verbindungen
$\bar{b}$	mittlere Paketgröße
iat_opt	optimale Zwischenankunftszeit

Um das Stabilitätskriterium in Gleichung 2.5 zu erfüllen, gilt für iat_opt folgende Bedingung:

$$iat\_opt > \frac{\bar{b}}{B} \tag{5.1}$$

Da

$$\mu = \frac{B}{\bar{b}} \tag{5.2}$$

$$\lambda = \frac{1}{iat\_opt} \tag{5.3}$$

Und es gilt auch $\rho = \lambda/\mu < 1$.

Da mehrere Verbindungen sich den Engpass teilen müssen, sollte jede Verbindung nur ein Bruchteil $B_i = B/conn$ benutzen. Daher, wird das optimale Feedback-Signal iat_opt_i der Verbindung folgendermaßen berechnet:

$$iat\_opt_i > \frac{b_i}{B} \cdot conn \tag{5.4}$$

$$B_i = \begin{cases} \frac{B}{conn} & \text{falls} \quad conn <= conn\_lim \\ 0 & \text{sonst} \end{cases} \tag{5.5}$$

$$iat\_opt_i = \begin{cases} \frac{b_i}{B_i} & \text{falls} \quad B_i > 0 \\ 0 & \text{sonst} \end{cases} \tag{5.6}$$

mit

$conn\_lim$	Schwelle für die Anzahl an Verbindungen
B_i	Datenrate für die Verbindung i
iat_opt_i	optimale Zwischenankunftszeit für die Verbindung i
b_i	Paketgröße der Verbindung i

Diese optimale Zwischenankunftszeit iat_opt_i wird samt des Kennzeichens des Routers in den Paket-Header geschrieben (siehe Abbildung 5.3). Jeder Router kann die gesetzte IAT des Pakets-Headers ändern, wenn:

- die iat_opt_i größer als die IAT im Paket-Header ist, oder
- die IAT von ihm gesetzt wurde und sich geändert hat. Der Router erkennt die IAT als seine eigene anhand des Kennzeichens im Paket-Header.

Die Größe *conn* kann nur dynamisch festgestellt werden, was den Vorteil hat, dass die aufzuteilenden Ressourcen bei sinkender Last wieder freigegeben werden können. Es ist zudem wichtig festzustellen, wie und wann dies erfolgen sollte, damit die gewonnenen statistischen Größen zwei Anforderungen erfüllen: sie sollen Information für eine dynamische Anpassung liefern und so wenig wie möglich den Router belasten.

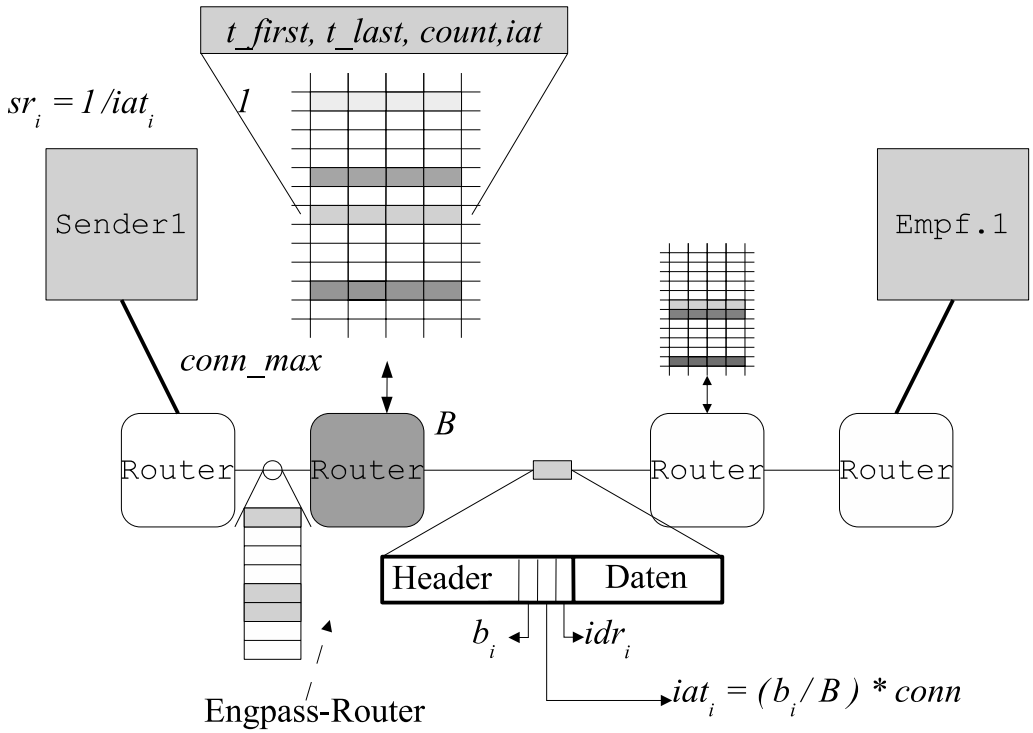


Abbildung 5.3: Mechanismen von ANDINA zur Überlastkontrolle

Abschätzung der Anzahl aktiver Verbindungen. Eine Verbindung kann als nicht aktiv definiert werden, wenn über sie nach einem gewissen Zeitintervall keine Pakete mehr ankommen. Dies hängt davon ab, ob die Verbindung sich in ihrer Aufbauphase oder sich in der Phase der Übertragung von Applikationsdaten befindet. In der Verbindungsaufbauphase werden wenige Pakete ausgetauscht, so dass die Zwischenankunftszeit von der RTT abhängt. Während der Übertragung der Applikationsdaten kann der Sender sein Fenster oder seine Senderate ausschöpfen, so dass die Zwischenankunftszeit vom Engpass der Verbindung abhängt. Hier wird anhand der Anzahl der Pakete einer Verbindung abgeschätzt, ob sie sich in der Verbindungsaufbauphase oder sich in der Phase der Übertragung der Applikationsdaten befindet.

Im ersten Fall muss die erwartete mittlere RTT als Maßstab verwendet werden, bis eine Verbindung als beendet oder unterbrochen erklärt werden kann. Im zweiten Fall wird nach ihrer eigenen Senderate untersucht, ob der Datenfluss beendet ist. Inaktive Verbindungen erfüllen zum Zeitpunkt t diese Bedingungen:

$$iat\_ist_i < \frac{t - t\_last_i}{ti} \qquad \text{falls } count > n_1 \tag{5.7}$$

$$\overline{rtt} < t - t\_last_i \qquad \text{sonst} \tag{5.8}$$

mit

iat_ist_i	Zwischenankunftszeit der Verbindung i
t	aktuelle Zeit
t_last_i	Zeit der letzten Ankunft eines Pakets der Verbindung i
ti	Leerlaufzeit der Verbindungen
$count$	Anzahl der Pakete der Verbindung i
$\overline{rtt}$	erwartete mittlere RTT
n_1	Schwelle für Anzahl der Pakete in der Aufbauphase (i.a. $n_1 = 3$)

Der Faktor ti_i soll die Variation der Ankunftsrate der Verbindungen berücksichtigen. Die Leerlaufzeit einer Verbindung ist schwierig zu definieren, da ihre Änderung auch von der Verlagerung des Engpasses bestimmt wird.

Nach Überprüfung der Bedingungen der Gleichungen 5.7 und 5.8 kann *conn* gesenkt werden. Die Variable *conn* kann bei der Ankunft von Paketen erhöht werden. Dies wird aus dem Verbindungskontext entnommen. Der Verbindungskontext ist eine im Router gehaltener Datenstruktur mit Information über die Verbindung, wie z.B. dem Zeitpunkt der letzten Paketankunft. Wichtig ist, dass wenn die Anzahl der Elemente dieser Datenstruktur groß ist, die Überprüfung der Bedingungen 5.7 und 5.8 nur auf eine Teilmenge ihrer Elemente angewendet wird, denn der Aufwand dieser Operation liegt im Bereich O(*conn*).

In diesem Fall wird statistische Inferenz angewendet, um die Variable *conn* abzuschätzen. Zu jedem Zeitpunkt kann die Menge der Verbindungskontexte als eine Binomialverteilung betrachtet werden, mit der Wahrscheinlichkeit p, dass eine Verbindung inaktiv ist, und der Wahrscheinlichkeit $q = 1 - p$, dass die Verbindung aktiv ist. Der zentrale Grenzwertsatz kann angewendet werden, falls folgende Bedingungen erfüllt sind [Kaz76]:

$$\begin{aligned} conn\_m &> 30 \\ conn\_m \cdot pq &> 5 \end{aligned}$$

mit *conn_m* als die Stichprobenumfang der Verbindungskontexte.

Sei *conn_inactive* die Anzahl der Verbindungskontexte, welche die Bedingungen 5.7 und 5.8 erfüllen, dann wird der Anteil $\tilde{p}$ festgestellt mit:

$$\tilde{p} = \frac{conn\_inactive}{conn\_m}$$

Die Anzahl der aktiven Verbindungen *conn_s* und der Standardfehler $s_{\tilde{p}}$ können anhand von $\tilde{p}$ mit folgender Gleichungen berechnet werden [Kaz76][2]:

$$conn\_s = (1 - \tilde{p}) \cdot conn \tag{5.9}$$

$$s_{\tilde{p}} = \sqrt{\frac{\tilde{p} \cdot (1 - \tilde{p})}{conn\_m}} \cdot \sqrt{\frac{conn - conn\_m}{conn - 1}} \tag{5.10}$$

$$\tilde{p}_t = (1 - \tilde{p}) \pm z \cdot s_{\tilde{p}} \tag{5.11}$$

mit

[2] Der Faktor $\sqrt{(conn - conn\_m)/(conn - 1)}$ ist vernachlässigbar, wenn $conn\_m/conn < 0.05$

$conn\_m$	Stichprobenumfang zur Überprüfung von 5.7 und 5.8
$\tilde{p}$	Anteil inaktiver Verbindungen in $conn\_m$
$s_{\tilde{p}}$	Standardfehler in $\tilde{p}$
$conn\_s$	abgeschätzte Anzahl der aktiven Verbindungen
$\tilde{p}_t$	untere und obere Grenze von $conn\_s$ je nach Konfidenzintervall
z	Quantil der Normalverteilung je nach Konfidenzintervall

Die Variable *conn* in Gleichung 5.5 wird dann mit $conn\_s$ ersetzt. Das Intervall $\tilde{p}_t$ zeigt die untere und obere Grenze für die Korrektur der Abschätzung von $conn\_s$. Das heißt, der Anteil der Population, die die Bedingungen 5.7 und 5.8 nicht erfüllt, liegt in diesem Konfidenzintervall.

Die Anzahl der aktiven Verbindungen kann bei der Ankunft der Pakete erhöht werden. Wenn eine neue Verbindung aufgebaut wird, wird der entsprechende Verbindungskontext im Router erzeugt und die Variable *conn* erhöht. Es dauert dann jedoch eine Round-Trip-Time (RTT), bis ihr Sender anfängt, den Datenstrom zu übertragen. Während dieser Zeit werden die anderen Sender sofort aufgefordert, ihre Senderaten abzusenken, d.h. jedes diesen Router durchquerende Paket wird mit einer geringeren IAT markiert. Daher kann der Sender dieser neuen Verbindung die Datenübertragung mit einer hohen Senderate starten, ohne dass die Warteschlange überläuft.

Das Kontrollintervall. Damit die Regelungsmechanismen im Router stabil wirken, muss das Kontrollintervall an die RTT der Verbindungen angepasst sein. Das Kontrollintervall muss ein Gleichgewicht zwischen Verbindungen mit großen und kurzen RTT darstellen. Wenn das Kontrollintervall zu lang ist, kann der Router nicht schnell auf die Last reagieren. Ist das Kontrollintervall zu kurz, wird das System instabil sein. In ANDINA wird das Kontrollintervall auf den Wert der erwarteten RTT gesetzt. Dieser Parameter ist konfigurierbar oder kann basierend auf den Statistiken der Verbindungen abgeleitet werden.

5.2.2 Mechanismen beim Sender und Empfänger

Die Sender entnehmen die Information über die Zwischenankunftszeit dem Paket-Header und generieren einen Datenstrom, der die entsprechende Zwischenankunftszeit nicht übersteigt. Ein Sender kann entweder die Senderate sr_i oder das Fenster $cwnd_i$ setzen:

$$\begin{cases} sr_i = 1/iat_i & \text{für eine senderatenbasierte ANDINA} \\ cwnd_i = sr_i \times rtt_i & \text{für eine fensterbasierte ANDINA} \end{cases} \tag{5.12}$$

mit

iat_i	Zwischenankunftszeit aus dem Paket-Header entnommen
sr_i	Senderate

Als Recovery-Mechanismen werden weiterhin RT- und Fast-Retransmit-Mechanismen angewendet. Der RT-Mechanismus kann mit den gleichen Kriterien wie in TCP angewendet werden, d.h. der RTO wird exponentiell erhöht und nach Ablauf von n RTT wird die Verbindung abgebrochen. Der Fast-Recovery-Mechanismus von ANDINA besteht aus dem sofortigen Senden eines Segments. Der Sender trennt dadurch die Aufgaben der Recovery von denen der Überlastkontrolle. Diese Trennung ist wichtig, da beide Mechanismen unabhängig voneinander weiter entwickeln werden können.

5.3 Implementierung

Hierbei wird ein Teil der Implementierung von ANDINA mit dem Programm *Network Simulator* (ns) beschrieben. ns ist ein Simulator für diskrete Ereignisse, implementiert in einer objektorientierten Version von Tcl, OTcl, und C++. Die Definition der Topologien, die Modellierung der Last und die Zuweisung von Systemparametern werden in OTcl-Programmen vorgegeben. Die wesentlichen Eigenschaften der Protokolle sind in C++ implementiert. Das Programm ns wird häufig zur Analyse in der Überlastkontrolle angewendet, da die Topologie und die Last einfach nachgebildet werden können. Darüber hinaus stellt ns mehrere Bibliotheken von Protokollimplementierungen zur Verfügung.

Da die größere Komplexität von ANDINA in den Routern liegt, werden die wichtigen Router-Algorithmen im Folgenden genauer beschrieben.

5.3.1 Ankunft der Pakete im Router

Beim Eintreffen eines Pakets (siehe Abbildung 5.4) wird der Verbindungskontext aktualisiert.

```
For each packet arrival at time(t):
    if packet is the same as last packet
       actualise connection context:
          update connection-ctx(packet, time(t));
    else
       generate hash-key for packet;
       access hash-element with hash-key;
       if not exist
          insert hash-element into connection-ctx(packet, time(t));
          conn <- conn + 1;
       else
          update connection-ctx(packet, time(t));
       end if;
    end if;
```

Abbildung 5.4: Algorithmus für die Paketankunft

Dafür gibt es zwei Datenstrukturen: eine Hash-Tabelle und eine doppelt verkettete Liste. Die zweite enthält die Adressen der belegten Plätze in der Hash-Tabelle. Jedes Element der Hash-Tabelle stellt einen Verbindungskontext dar.

Mit der Identifikation der Verbindung wird mittels einer Abbildungsfunktion eine Adresse in der Hash-Tabelle generiert. In ns ist dieser Parameter klar durch ein Identifikations-Feld definiert. Im Prototyp werden die Identifikationen des Senders und Empfängers für den Zugriff auf die Hash-Tabelle zusätzlich miteinbezogen.

Diese Abbildungsfunktion wird nicht bei der Ankunft jedes Pakets aufgerufen, denn frühere Arbeiten [Flo91] haben gezeigt, dass die Pakete einer TCP-ähnlichen Datenübertragung in Clustern ankommen, d.h. einer Gruppe von Paketen einer Verbindung folgen mehrere Gruppen von Paketen anderer Verbindungen. Daher wird beim Ankommen eines Pakets zuerst geprüft, ob dieses Paket und sein Vorgänger der gleichen Verbindung angehören. Ist dies nicht der Fall, wird die Abbildungsfunktion für

die Hash-Tabelle angewendet, um die Adresse zu finden. In dieser Adresse wird der Verbindungskontext aktualisiert. Ist die Adresse frei, wird dort ein neuer Verbindungskontext erzeugt und gleichzeitig diese Adresse am Ende der doppelt verketteten Liste hinzugefügt. Die Abbildung auf die Hash-Tabelle benutzt einen Algorithmus zur linearen Behandlung von Kollisionen.

5.3.2 Aktualisierung des Verbindungskontexts

Der Router sammelt als statistische Größen die Anzahl der Pakete, die erste und die letzte Ankunftszeit der aktiven Verbindungen. Beim Zählen der Pakete kann die Zwischenankunftszeit der Verbindung festgestellt werden. Dabei wird der Paketzähler um eins erhöht, die aktuelle Ankunftszeit aktualisiert und die Zwischenankunftszeit der Verbindung berechnet.

```
actualise connection_ctx(packet, time(t)):
     count <- count + 1;
     t_last  <- time(t);
     iat(t) <- (t_last - t_first) / count;
     siat <- siat * (1 - wq) + iat(t) *  wq;
```

Abbildung 5.5: Algorithmus für die Aktualisierung des Verbindungskontexts

5.3.3 Weiterleitung der Pakete

Bei der Weiterleitung der Pakete wird die optimale IAT der Verbindung berechnet und ggf. in den Paket-Header geschrieben. Diese Aktion ist hier notwendig, wenn große Puffer benutzt werden. Dabei kann das Feedback-Signal mit der aktuellen Anzahl der abgeschätzten Verbindungen aktualisiert werden.

```
 calculate iat_soll:
    iat_soll <- packet(datalen) * conn / bw;

 update packet-header if necessary:
   if iat_soll > packet(iat)
      write iat_soll, idr into packet-header;
```

Abbildung 5.6: Algorithmus für die Weiterleitung der Pakete

5.3.4 Abschätzung der Anzahl aktiver Verbindungen

Periodisch wird die Tabelle der Verbindungskontexte nach inaktiven Verbindungen abgesucht. Zuerst wird eine Anzahl *conn_m* der Elemente der doppelt verketteten Liste ausgewählt. Somit kann auf die Information der Verbindungskontexte zugegriffen werden. Falls in der Ausfallperiode keine Pakete mehr angekommen sind, wird dieses Element aus der verketteten Liste ebenso wie aus der Hash-Tabelle entfernt und die Anzahl an inaktiven Verbindungen *conn_inactive* der Stichprobe erhöht. Diese Anzahl wird dann für die Berechnung der abgeschätzten aktiven Verbindungen angewendet, und anschließend an die Pufferauslastung angepasst.

```
Initialise:
  conn_inactive <- 0;

  check for conn_m elements in connection-ctx:
      select element;
      test if packet has not come in ti periods:
          if ((count > n1  && (t_last + siat*ti < time(t)))
              (count <= n1 && (t_last + rtt < time(t))))
                delete element from connection-ctx;
                increment propor_m:
                   conn_inactive <- conn_inactive + 1;
          end
    end

  compute conn_s:
    propor <- conn_inactive / conn_m;
    conn_s <- (1 - propor) * conn;

  compute error with 99 % (z = 2.58);
    s <- sqrt((propor * (1 - propor)) / conn_m) * z
    sp_min <- (1 - propor) - s
    sp_max <- (1 - propor) + s

  adjust conn_s if load is too high of too low:
     if rho > rho_max
        conn_s <- sp_max * conn;
     if rho < rho_min
        conn_s <- sp_min * conn;

  compute conn:
     conn <- conn_s;
```

Abbildung 5.7: Algorithmus für die Feststellung von aktiven Verbindungen

Die Variable *conn_max* gibt an, wie viele Elemente der Hash-Tabelle untersucht werden können. Bei geringer Last, d.h. wenn die Anzahl der Verbindungen klein ist, kann die ganze Liste der Verbindungskontexte untersucht werden. Bei hoher Last, d.h. wenn *conn* > *conn_max*, werden nur *conn_m* Elemente der Liste der Verbindungskontexte untersucht. Die Variable *conn_max* sollte an die Prozessorleistung des Routers angepasst werden.

5.4 Vergleich zu anderen Verfahren

Verfahren mit ähnlichem Feedback-Signal sind u.a. BBN, DTW, ERICA und XCP. Im Gegensatz zu BBN und DTW werden jedoch in ANDINA keine Kontrollpakete mit Feedback-Signalen gesendet. In ERICA und XCP wird das Feedback-Signal im Paket-Header gesendet.

ANDINA unterscheidet sich von ERICA in der Abschätzung der Anzahl von aktiven Verbindungen und der Länge des Kontrollintervalls. Im Gegensatz zu ERICA verlangt ANDINA keine genaue Kalkulation der Anzahl der aktiven Verbindungen, da keine Leitungskapazität pro Verbindung zugewiesen wird.

Im Gegensatz zu XCP muss ANDINA Statistiken über die Historie des Puffers halten. Das Feedback-Signal in XCP ist der Wert der erforderlichen Änderung des Fensters. Da diese Änderung von den aktuellen Werten der RTT und des Fensters abhängig ist, konvergieren alle Datenübertragungen zum gleichen Durchsatzwert. Für extrem kurze Datenübertragungen mit sehr unterschiedlichen Pfadkapazitäten kann die Zeit zur Berechnung des jeweiligen Soll-Fensters nicht ausreichend sein. Im Gegensatz zu XCP empfangen daher alle ANDINA-Sender, unabhängig von ihren aktuellen Senderaten, innerhalb einer RTT die gleichen Feedback-Signale zur Anpassung der Senderate.

METHODE	**RED [FJ93]**	**FRED [LM97]**	**ERICA [KJF+97]**	**SRED [OLW99]**	**REM [Ath01]**	**PI [HMTG01]**	**ARED [FGS01]**	**XCP-Router[KHR02]**	**ANDINA-Router**
Eigenschaften									
Leitungskapazität als Parameter				x	x	x	x	x	x
Zuweisung der Kapazität pro Verbindung			x						
Berechnung der mittleren WS-Länge	x	x	x				x		
WS-Länge als Sollwert					x	x			
Abschätzung der aktiven Verbindungen		x	x	x					x
Verbindungszustände			x						
Verbindungskontexte		x							x
Andere Statistiken				x		x			
Feedback-Signal									
Gezieltes Verwerfen von Paketen	x	x		x	x	x	x		
Verbindungsauswahl zum Verwerfen von Paketen		x		x					
Bit im Paket-Header als Feedback-Signal	x				x	x	x		
Quantitative Feedback-Signal			x					x	x
Gleiches Feedback-Signal für alle Sender									x
Einsetzbare Transportprotokolle									
UDP, TCP, TCP-ECN	x	x	x	x	x	x	x		
XCP Transportprotokoll								x	
ANDINA Transportprotokoll									x

Tabelle 5.1: AQM-Verfahren und ANDINA

Gegenüber den herkömmlichen AQM-Verfahren wie die RED-Varianten hat ein ANDINA-Sender den Vorteil, dass Paketverlust nicht als Feedback-Signal eingesetzt wird. Er muss somit nicht warten, bis die Schwelle der Warteschlangenlänge erreicht ist, bevor das Feedback-Signal gesendet wird. Eine andere Alternative in AQM-Verfahren ist das Setzen eines Bits im Paket-Header. Damit kann höchstens die Aussage erfolgen, dass Netzüberlastung vorliegt, damit ist es jedoch nicht möglich

den Grad der Überlastung zu wissen. Es besteht weiterhin das Problem, um wie viel das Fenster verkleinert oder vergrößert werden muss. Außerdem stellt sich die Frage, ob für Verbindungen mit großer Pfadkapazität der Paketverlust ein geeignetes Feedback-Signal ist.

5.5 Funktionsmodelle

5.5.1 Aufbauplan der Datenübertragung

Analog zu TCP soll hier das entsprechende statische Modell des fensterbasierten ANDINA erläutert werden (siehe Abbildung 5.8). Die Router messen die adäquate IAT und leiten sie weiter durch den Pfad an die Transport-Akteure. Die Flusskontroll-Agenten berechnen dann das CWND auf Basis der IAT.

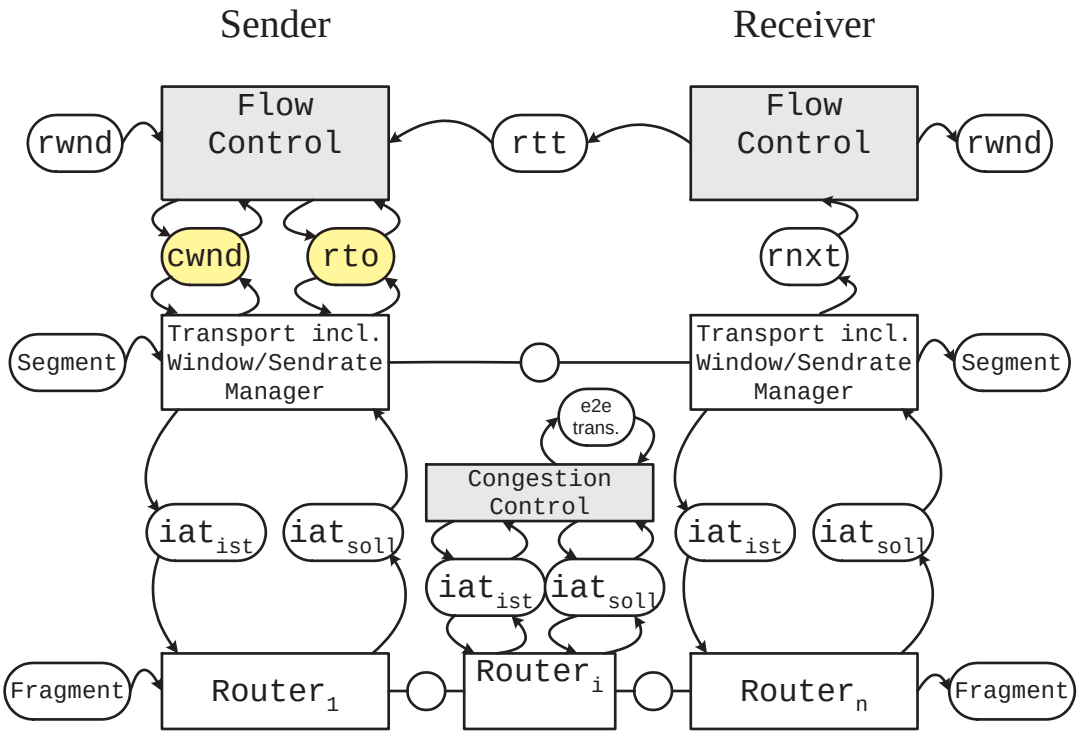


Abbildung 5.8: Aufbauplan eines ANDINA-Kommunikationssystems

5.5.2 Ablaufplan der Datenübertragung

Der Ablaufplan der Datenübertragung ist in Abbildung 5.9 in Form eines FMC-eCS Petrinetzes dargestellt. Dabei wird ANDINA im Zustand `ESTABLISHED` als Producer/Consumer-System modelliert.

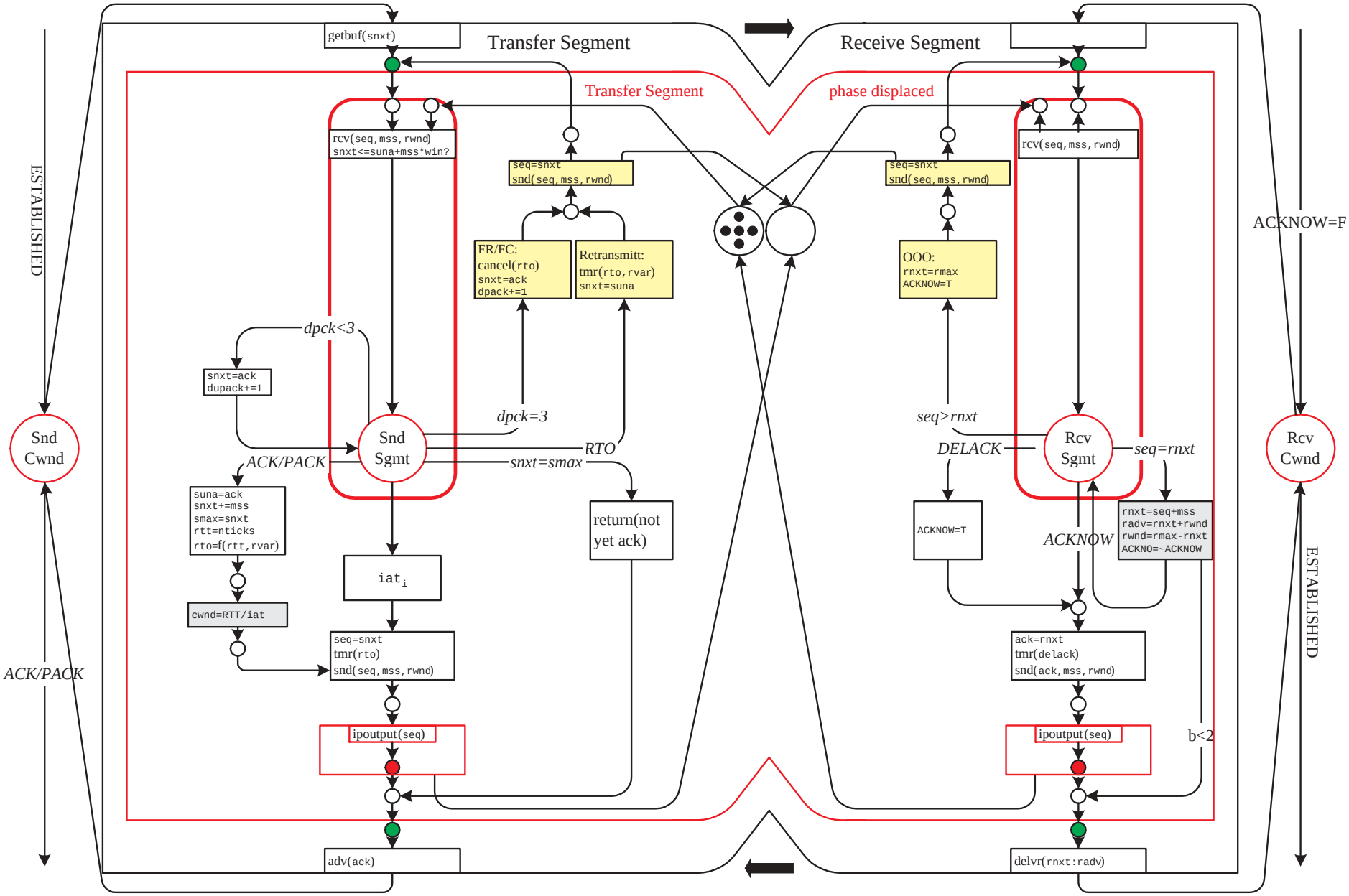


Abbildung 5.9: Zustandsdiagramm während `ESTABLISHED`

5.6 Analyse und Simulation

Im Folgenden werden die Leistung und Stabilität von ANDINA anhand von Simulationen mit `ns` gezeigt.

Die hier benutzten Topologien sind in ähnlichen Studien angewendet worden (siehe z.B. [FJ93, FF96, FGS01]). Dabei werden die Leistungskenngrößen (Durchsatz, Paketverlust, Fairness) konkurrierender Datenübertragungen und ihr gegenseitiger Einfluss unter Berücksichtigung folgender Faktoren untersucht:

- Unterschiedliche Datenraten
- Pfadlänge, d.h. die Größe der RTT
- Gesendete Datenmenge
- Auslastung der Leitung
- Paketverlust

Mit `ns` können TCP-Akteure in den Standard-Implementierungen und IP-Router simuliert werden. Die ausgewählten Protokolle waren in der `ns`-Version bereits vorhanden. Die TCP-Akteure implementieren ein Vollduplex-TCP. ANDINA wurde mit den aktuellen TCP-Varianten Reno, Newreno und SACK verglichen. Router mit Warteschlangen vom Typ PD, ARED, REM und PI wurden eingesetzt.

Es werden die Ergebnisse von folgenden vier typischen Tests gezeigt:

- Test 1) Zwei lange Datenübertragungen
- Test 2) Einfluss durch eine kurze Datenübertragung
- Test 3) Vier lange Datenübertragungen, sowohl bei niedriger als auch bei hoher Last
- Test 4) Mehrere lange und kurze Datenübertragungen

Die ersten zwei Tests wurden mit einer kleinen Anzahl an Datenübertragungen durchgeführt, um die Effekte auf die einzelnen Verbindungen unter optimalen Bedingungen zu zeigen. Sie wurden mit den TCP-Varianten gegenüber ANDINA bewertet. Dabei wurden PD-Warteschlangen benutzt. Der dritte Test veranschaulicht die Effekte auf wenige Datenübertragungen bei niedriger und bei hoher Last. Hierbei wurden Reno, Newreno und SACK mit ARED-Warteschlangen gegenüber ANDINA bewertet. Der vierte Test zeigt durch die Simulation von mehreren Hundert Verbindungen die Eigenschaften bei stationärem Verkehr. Dabei wurden einerseits mit SACK in Kombination mit ARED, REM und PI und andererseits mit ANDINA sowohl lange als auch kurze Datenübertragungen betrachtet.

Falls nicht anders angegeben, wurden die Vorgabewerte von `ns` angewendet. Damit die Übertragungen nicht vom Empfängerfenster begrenzt werden, wurde RWND auf 1000 Segmente gesetzt. Andere wichtige Parameter wurden auf die folgenden Werte gesetzt: das initiale CWND auf 1 Segment, eine Quittung pro zwei Segmente, Segmentgröße auf 536 Bytes, usw. Die Vorgabewerte für ANDINA wurden folgendermaßen gesetzt: das Kontrollintervall auf 0.1 Sekunden, die Anzahl der Verbindungskontexte auf 60, *ti* auf 4, und n_1 auf 3.

5.6.1 Test 1) Zwei lange Datenübertragungen

Diesem Test liegt die Topologie 1 (siehe Abbildung 5.10) zugrunde. Es werden dabei die Messungen in 4.5 nachvollzogen, bei denen zwei TCP-Verbindungen den gleichen Engpass haben. Im Test

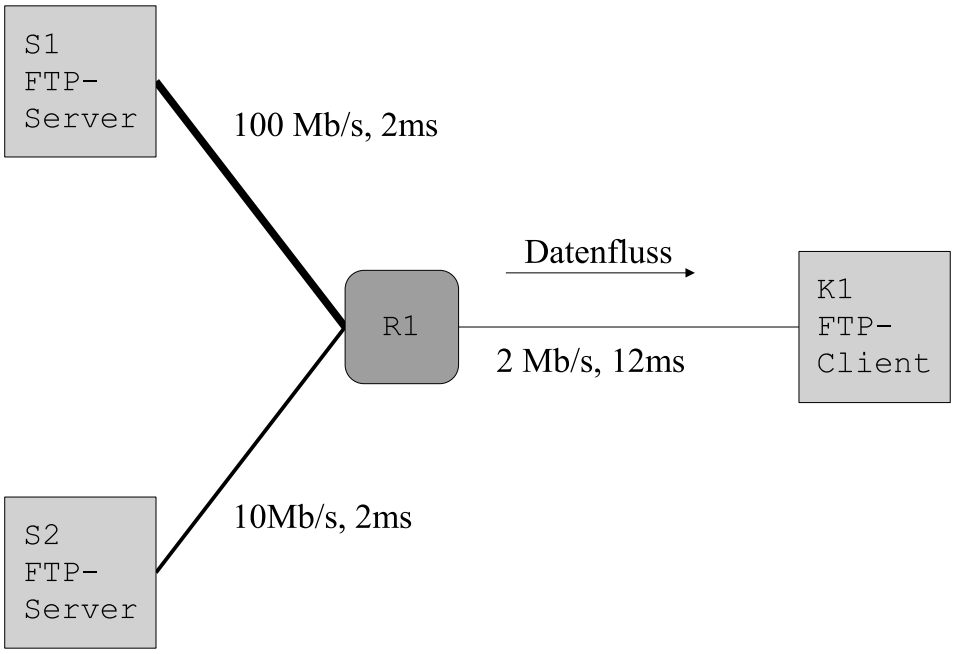


Abbildung 5.10: Topologie 1

werden gleichzeitig zwei lange Datenübertragungen (ca. 1.6 MB) von den Knoten S1 und S2 nach Knoten K1 mit jeweils Reno, Newreno, SACK und ANDINA gestartet. PD-Warteschlangen mit gesetzter Puffergröße auf 8 Pakete werden in Kombination mit den TCP-Varianten eingesetzt.

Das Ziel dieses Tests ist, durch die Isolierung der zwei Datenübertragungen, die Protokolle unter optimalen Bedingungen zu untersuchen. Die Überlast entsteht dann nicht nur durch andere Datenübertragungen, sondern kommt auch durch eine aggressive Senderate des eigenen TCP zu Stande.

Die divergente Entwicklung des Durchsatzes (siehe Abbildung 5.11) und der Übertragungsrate über die Zeit (siehe Abbildung 5.12) für beide Verbindungen wird je nach Protokoll gezeigt. Interessant sind vor allem die Schwankungen bei Reno und Newreno.

- Bei ANDINA werden höhere Durchsätze erreicht, da die Sender auf die optimale Datenrate eingestellt werden. Dies erfolgt unabhängig von der Dateigröße, denn nach der Verbindungsaufbauphase stellt der Sender seine Senderate entsprechend dem Engpass des Pfades ein. Da kein zusätzlicher Verkehr vorhanden ist, bleibt der Durchsatz stabil auf etwa 110 KB/s. Bei ANDINA erreicht kein Sender mehr als den mittleren Durchsatz. Aus diesen Abbildungen geht hervor, dass die Durchsätze der zwei Verbindungen mit den TCP-Varianten stark schwanken. Die Sender der Verbindungen verdrängen sich ständig: In dem der eine Sender eine hohe Datenrate beansprucht, verschlechtert sich temporär der Durchsatz des anderen, bis der erste erneut einen Paketverlust erfährt. In Reno und Newreno hat eine Datenübertragung einen guten Durchsatz, während die andere einen schlechten Durchsatz erreicht. Der Unterschied kann bis zu 90 % betragen. Siehe z.B. Newreno: nach 8 Sekunden hat die eine Datenübertragung einen Durchsatz von 180 KB/s, die andere nur 20 KB/s. Dies impliziert, dass der eine Sender die Übertragung von 1.6 MB beendet hat, während der andere nur ein Zehntel davon übertragen hat. In SACK

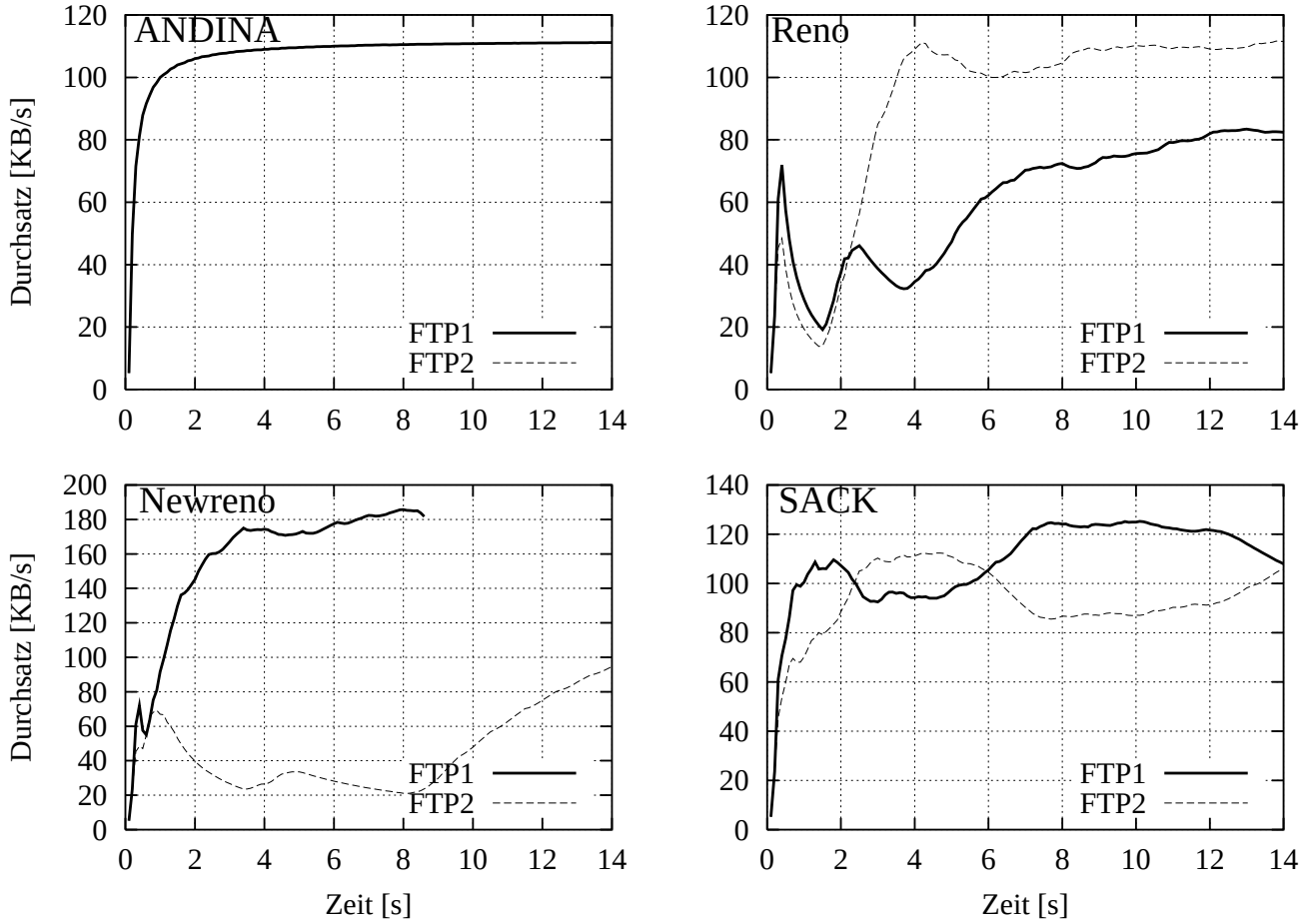


Abbildung 5.11: Test 1) Durchsatz über die Zeit

kommen diese Schwankungen periodisch vor.

- Wird die Senderate in Abbildung 5.12 betrachtet, wird festgestellt, dass die Sender etwa 45 Pakete/s senden können. Bei ANDINA werden höchstens ca. 42 Pakete/s übertragen aber diese Senderate wird gleich zwischen den Sendern geteilt. Bei Reno erreichen beide Sender schnell die Pfadkapazität der Verbindung und sie müssen auf dem Retransmission-Timer warten. Erst nach 4 Sekunden können beide Sender die mittlere Senderate erreichen. Bei Newreno erreicht nur der eine Sender ca. 45 Pakete/s während der andere nach mehreren RT-Ereignissen seine Senderate auf Null setzt. Erst nachdem der erste Sender seine Übertragung nach 8 Sekunden beendet hat, kann der andere Sender seine Senderate erhöhen. Bei SACK fängt die Senderate erst an zu schwanken, nachdem mehrere KB übertragen werden. Es ist daher zu erwarten, dass SACK bessere Durchsätze für kurze Datenübertragungen als Reno und Newreno erreicht.

Die Abbildung 5.13 zeigt die Auslastung des Engpasses. Reno erreicht die niedrigste Auslastung (ca. 85 %). Die absinkende Auslastung nach 2 Sekunden Übertragungszeit liegt an der Reduzierung des CWND auf 1 Segment nach einem RT-Ereignis. Newreno und SACK haben eine Auslastung von ca. 93 %. Die höhere Auslastung liegt in den Recovery-Mechanismen begründet. ANDINA erreicht eine Auslastung von knapp 100 %. Da die Sender sich an die avisierte Senderate halten und keine weiteren Störungen vorhanden sind, kommen keine Recovery-Mechanismen zum Einsatz. Es gibt keinen Paketverlust und die Datenrate der Leitung kann nahezu vollständig ausgenutzt werden.

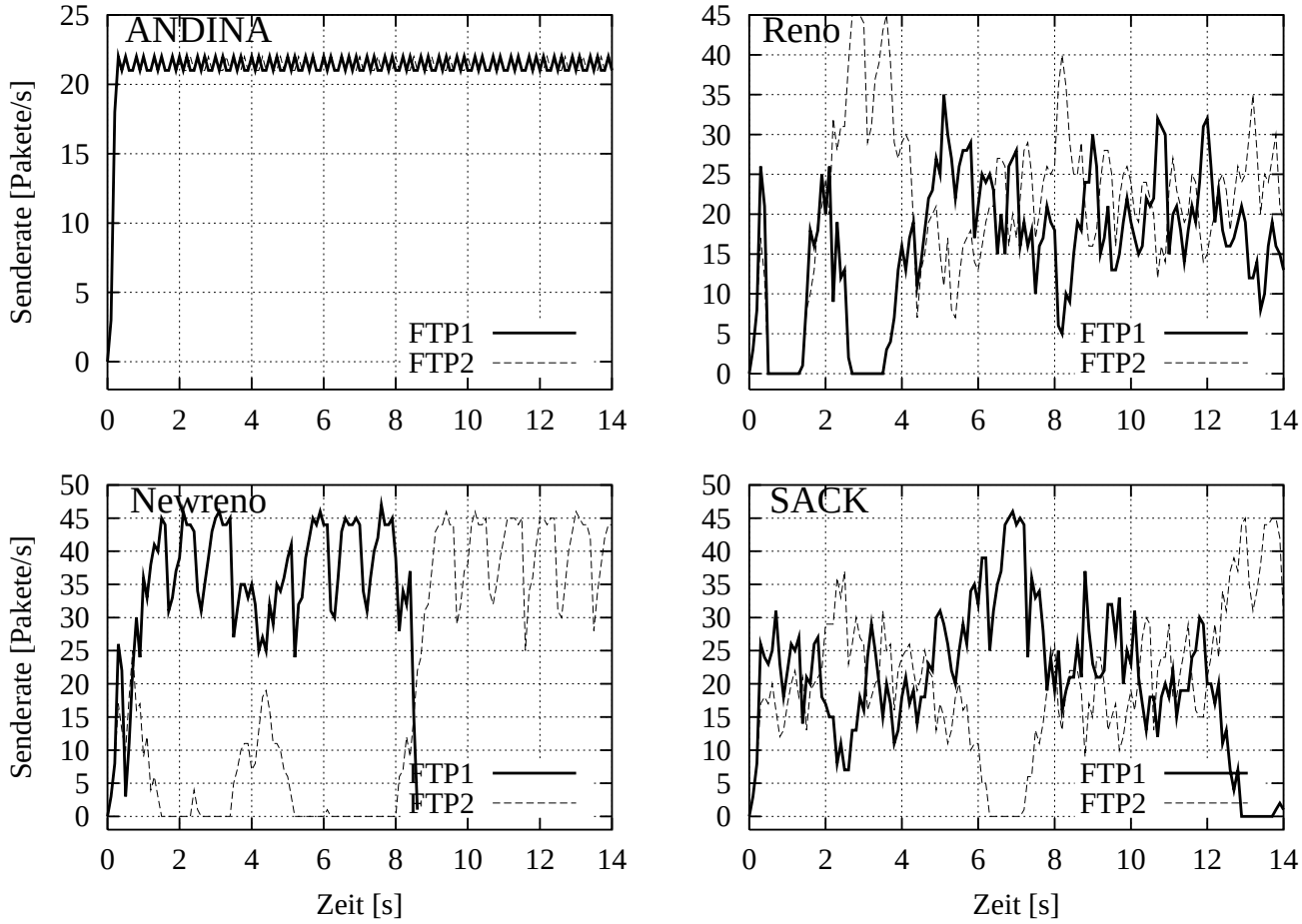


Abbildung 5.12: Test 1) Senderate über die Zeit

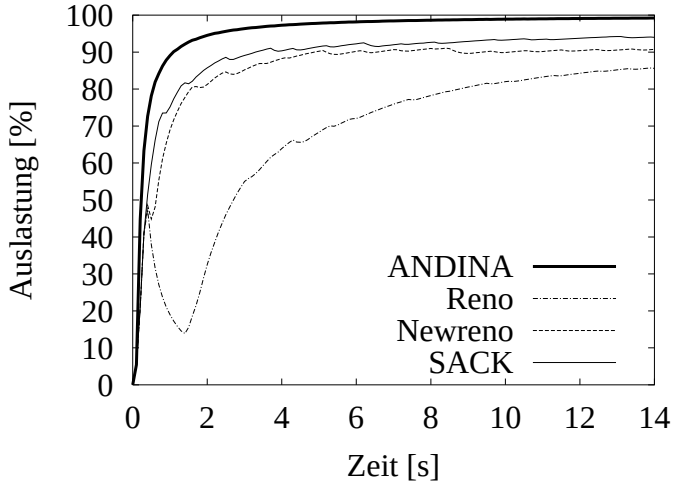


Abbildung 5.13: Test 1) Auslastung des Engpasses

Der Test wurde dann mit mehreren unterschiedlichen Datenraten, RTT und Puffergrößen im Engpass wiederholt. Die Datenrate variiert von 1 bis 10 Mb/s, die RTT von 5 bis 20 ms und die Puffergröße von 8 bis 98 Pakete. Eine kleinere Datenrate impliziert, dass TCP den Slow-Start-Modus schneller

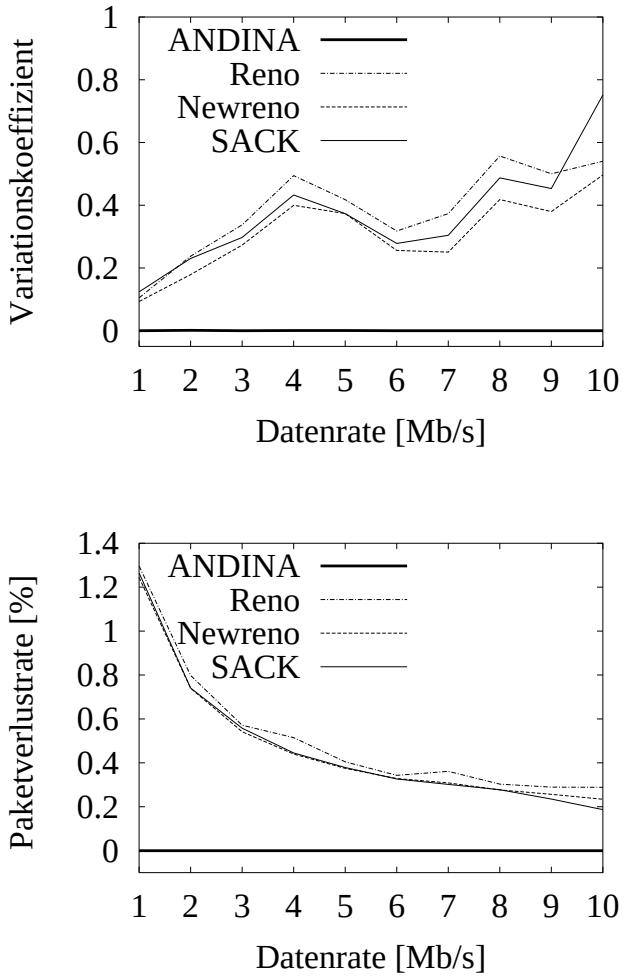


Abbildung 5.14: Test 1) Leistungskenngrößen in Abhängigkeit der Datenrate des Engpasses

verlässt. Die Simulation dauert 16 Sekunden. Ein Messwert dieses Tests, z.B. Datenrate von 2 Mb/s, RTT von 12 ms und Puffergröße von 8 Paketen entspricht dem vorher beschriebenen Test.

Die Ergebnisse sind in Abbildung 5.14 dargestellt. Die RTT und Puffergröße wurden gemittelt und sowohl der Variationskoeffizient des Durchsatzes als auch die Paketverlustrate in Abhängigkeit der Datenrate des Engpasses aufgetragen.

Je größer die Datenrate des Engpasses, desto größer wird die Variation des Durchsatzes in den TCP-Varianten, obwohl die Paketverlustrate sinkt. Die Paketverlustrate sinkt in Abhängigkeit der Datenrate von 1.3 auf 0.2 %. Wenn die Datenrate 10 Mb/s beträgt, sind die mittleren Durchsätze je ca. 500 KB/s. Bei ANDINA ist der Variationskoeffizient des Durchsatzes von der Datenrate unabhängig und tendiert gegen Null. In den TCP-Varianten steigt der Variationskoeffizient bis auf ca. 80 %.

Es ist interessant, dass für einige Datenraten die Übertragung mit allen TCP-Varianten entsprechend der in Abbildung 5.11 gezeigten Newreno ist, d.h. der Sender der zweiten Verbindung kann erst seine Übertragung zu Ende bringen, wenn der erste beendet hat. Diese Datenraten sind z.B. 4 Mb/s oder 8 Mb/s. Der Grund ist, dass unter gewissen Umständen ein Paketverlust zum ungünstigen Moment das Vergrößern des CWND verhindert. In diesen Fällen bestimmt ein RT-Ereignis den Durchsatz.

5.6.2 Test 2) Einfluss durch eine kurze Datenübertragung

In diesem Test werden die Effekte einer kurzen Datenübertragung wie z.B. in HTTP auf die Datenübertragung einer großen Datei anhand der Topologie 1 untersucht. Zuerst wird eine lange Datenübertragung von ca. 1 MB von Knoten `S1` zu Knoten `K1` gestartet. Nach drei Sekunden wird eine HTTP-Datenübertragung von ca. 55 KB von Knoten `S2` ebenfalls zu Knoten `K1` gestartet.

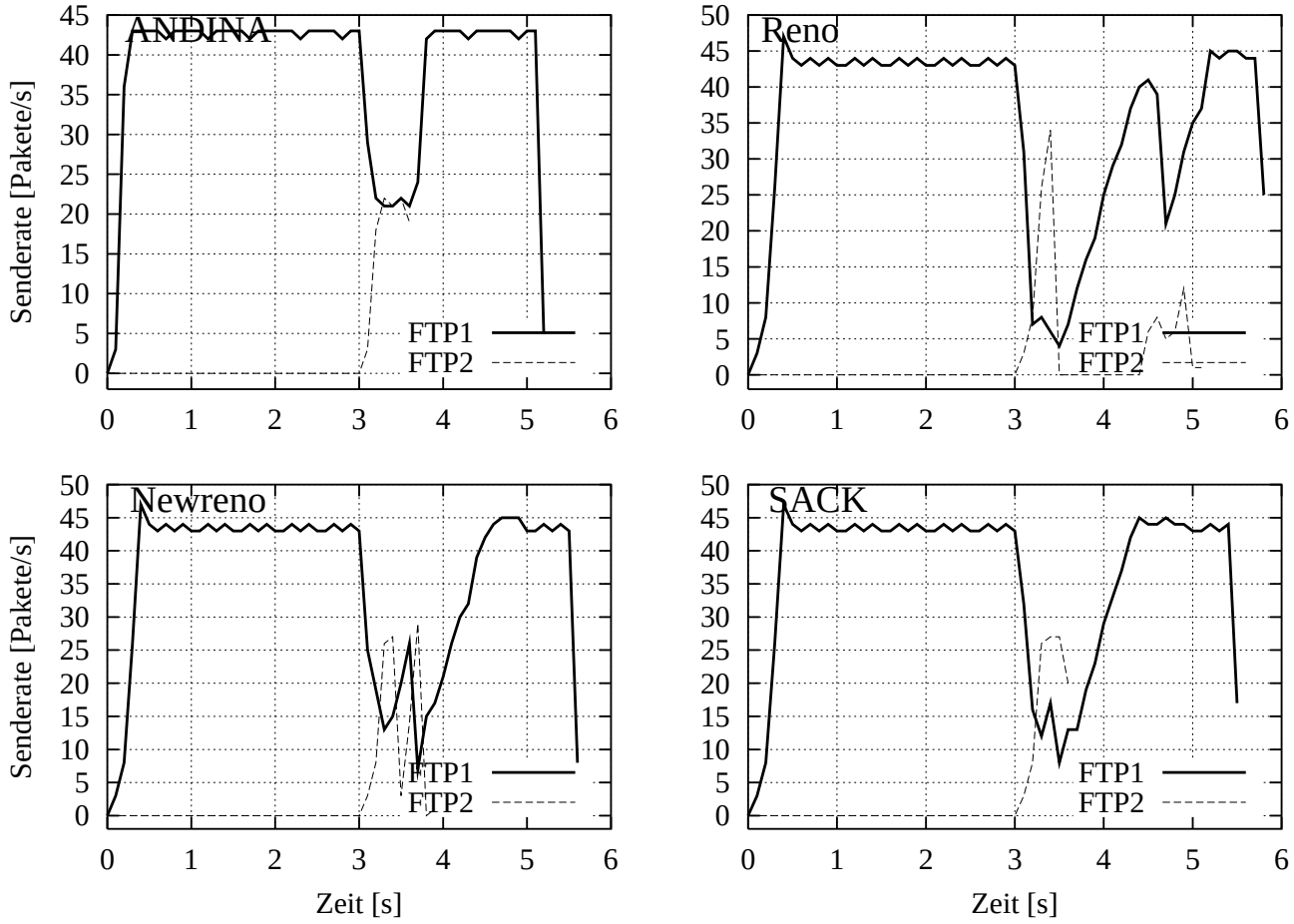


Abbildung 5.15: Test 2) Senderate über die Zeit

In diesem Test wird die Reaktionszeit des Senders der langen Datenübertragung untersucht. Unter Reaktionszeit ist hier die Zeit zu verstehen, die verstreicht, bis die Senderate wieder ihren optimalen Wert erreicht, nachdem die HTTP-Datenübertragung beendet wurde.

Da die ersten drei Sekunden nur die lange Datenübertragung das Netz beansprucht, ist es für den Sender möglich, die Datenrate des Engpasses zu benutzen.

Würde die Datenrate halbiert, könnte die HTTP-Datenübertragung in etwa einer Sekunden beendet sein. Bei ANDINA bewirkt die HTTP-Datenübertragung, dass die Senderate der langen Datenübertragung quasi sofort halbiert wird. Dadurch entstehen geringere Paketverlustraten als bei den TCP-Varianten. Dies bewirkt, dass die HTTP-Datenübertragung schneller als bei den TCP-Varianten beendet wird. Nach Ende der HTTP-Datenübertragung wird, bei ANDINA, dem Sender die entsprechende Datenrate erneut mitgeteilt. Innerhalb einer Drittelsekunde steigt die Senderate wieder auf ihren ursprünglichen Wert an.

Bei den TCP-Varianten dagegen beginnt der Sender der HTTP-Datenübertragung Pakete zu senden, bevor der konkurrierende Sender der langen Datenübertragung Zeit hat, sein eigenes CWND zu senken. Erst nach dem ersten Paketverlust ist der Sender gezwungen, dies auszuführen. Nachdem die HTTP-Datenübertragung beendet wird, braucht der Sender der langen Datenübertragung ca. 1 bis 2 Sekunden bis die Senderate auf den optimalen Punkt erhöht wird.

5.6.3 Test 3) Vier lange Datenübertragungen bei niedriger und bei hoher Last

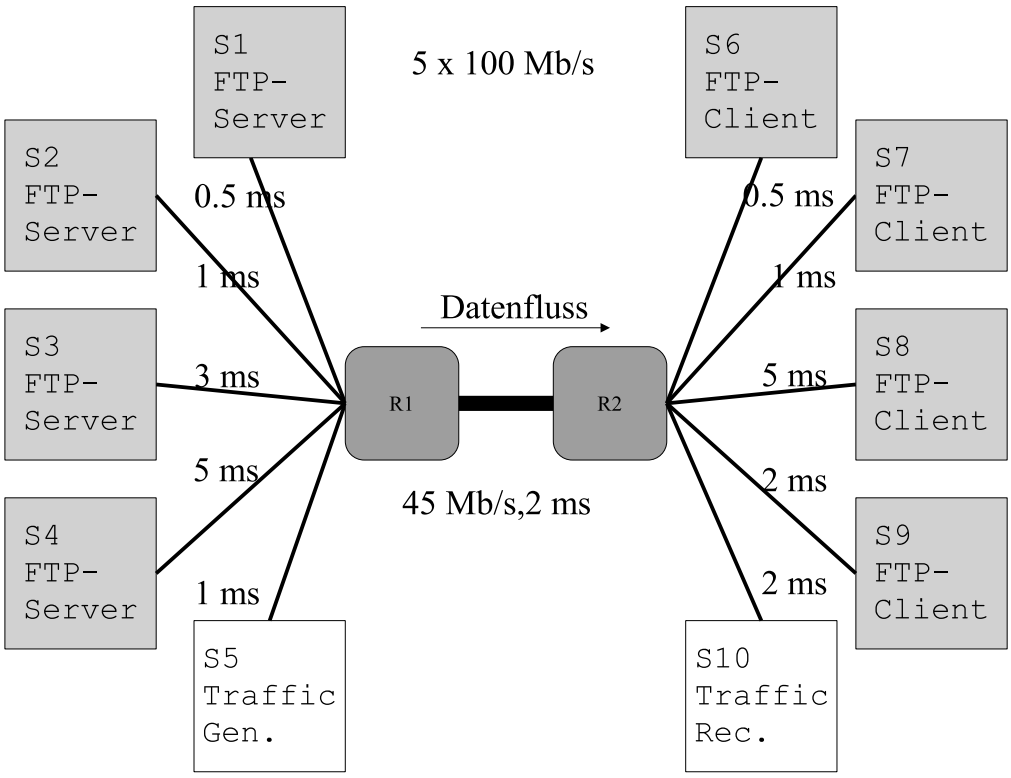


Abbildung 5.16: Topologie 2

In diesem Szenario wird der Einfluss der RTT auf den Durchsatz unter Einsatz einer großen nicht ausgelasteten Leitung untersucht. Anhand der Topologie in Abbildung 5.16 kann gezeigt werden, dass die RTT um Faktor 4 variiert. Die übertragenen Datenmengen betragen je 500 KB und simulieren dabei FTP-Datenübertragungen. Sie starten jede 0.10 Sekunden aus den Knoten `S1` bis `S4` zu den Knoten `S6` bis `S9`. ARED-Warteschlangen mit Puffergröße von 12 Paketen werden bei den TCP-Varianten eingesetzt.

Zuerst wird eine niedrige Lastumgebung generiert. Diese vier Datenübertragungen konkurrieren mit dem zusätzlichen Verkehr, der zwischen den Knoten `S5` und `S10` durch einen *Traffic-Generator* erzeugt wird. Dieser Verkehr wird als Störung oder transiente Netzüberlastung aufgefasst.

Die Durchsätze über die Zeit zeigt die Abbildung 5.17. Der Abbildung kann entnommen werden, dass es bei den TCP-Varianten jeweils einen Sender gibt, der die anderen verdrängt. Dies ist der Sender der Datenübertragung mit der kleinsten RTT. Anhand des TCP-Modells in Gleichung 4.1 wird für diese Verbindung auf Grund seiner RTT ein besserer Durchsatz als von den anderen erwartet. In der Simulation ist dieser Unterschied noch größer als im Modell, da deren CWND höher wird. Bei

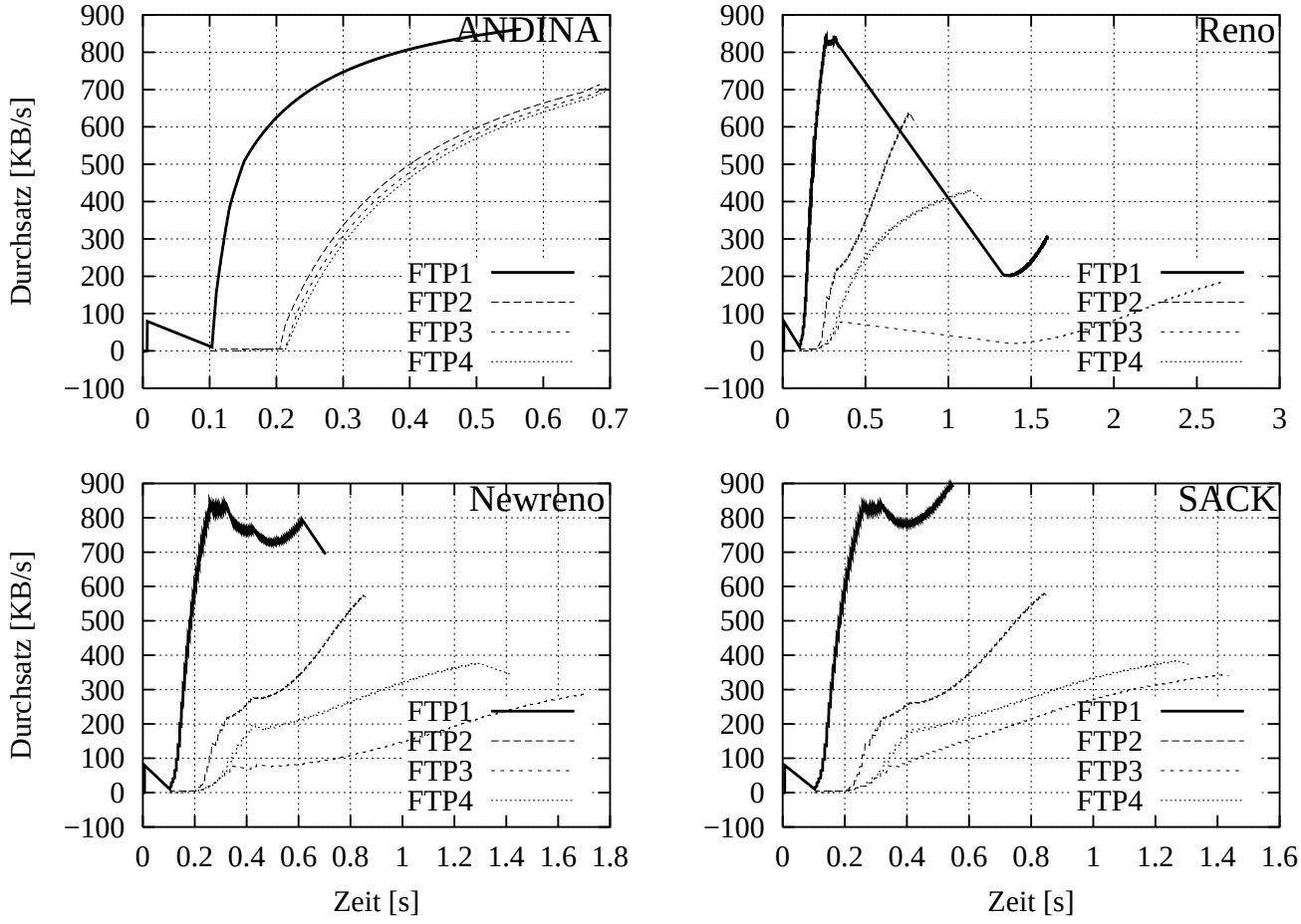


Abbildung 5.17: Test 3) Durchsatz über die Zeit von vier Datenübertragungen bei niedriger Last

ANDINA erhalten alle Sender den gleichen Anteil der Datenrate und alle vier Verbindungen sind nach 0.7 Sekunden beendet. Bei Newreno und SACK liegt die Zeit zwischen 0.5 und 1.7 Sekunden, bei Reno zwischen 0.8 und 2.5 Sekunden.

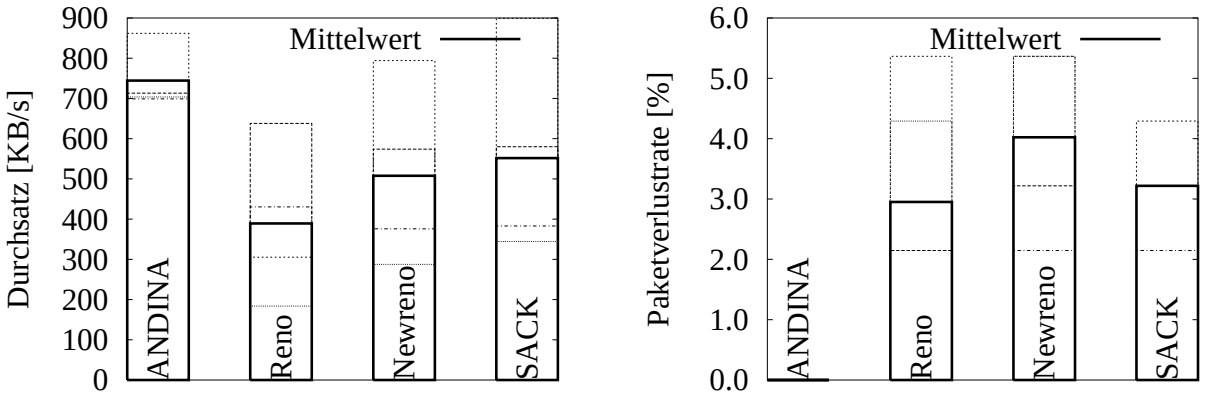


Abbildung 5.18: Test 3) Leistungskenngrößen von vier Datenübertragungen bei niedriger Last

Abbildung 5.18 zeigt die Durchsätze und Paketverlustraten für die Übertragung von 500 KB.

- Bei ANDINA wird der bessere mittlere Durchsatz erreicht. Er liegt bei ca. 700 KB/s. Bei Reno ist der Durchsatz ca. 400 KB/s und ca. 500 KB/s bei Newreno und SACK. Der bessere Durchsatz bei ANDINA impliziert in diesem Fall, dass die Datenübertragung mit der kleinsten RTT die anderen nicht verdrängt. Bei Newreno und SACK erreicht nur eine der Datenübertragungen einen Durchsatz in Höhe des Durchsatzes von ANDINA, während die schlechteste nur die Hälfte davon erreicht. Es ist auch zu sehen, dass je höher der Durchsatz bei den TCP-Varianten ist, desto größer die Variation ist.
- Die Paketverlustrate ist bei ANDINA am geringsten. Die Recovery-Mechanismen von SACK wirken besser als die von Newreno, denn seine mittlere Paketverlustrate ist geringer als Newreno. Die Paketverlustrate ist relativ gering, ca. 3 %. Dies ist jedoch ausreichend um, Unterschiede der Durchsätze in Höhe von 50 % zu bewirken.

Der Test wurde dann mit hoher Last wiederholt. Zur Simulation von HTTP werden dabei in der ersten Sekunde 100 kurze Datenübertragungen gestartet. Es wird dabei jeweils ca. 10 KB zwischen den Knoten `S5` und `s10` gesendet.

Die Details der FTP-Datenübertragungen sind in Abbildung 5.19 dargestellt. Bei SACK werden die Effekte der Verdrängung noch verschärft, da nur eine Datenübertragung einen guten Durchsatz er-

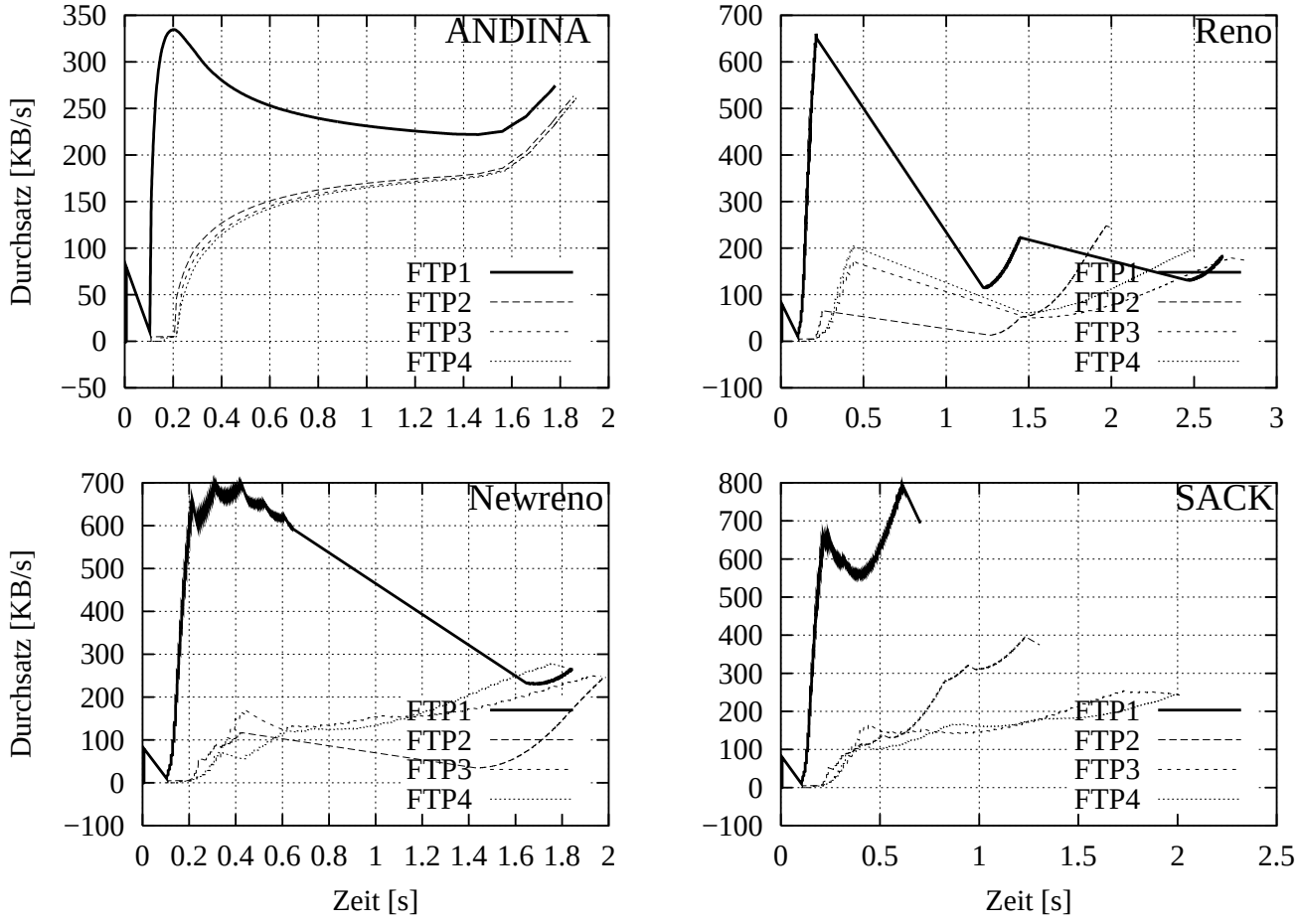


Abbildung 5.19: Test 3) Durchsatz über die Zeit von vier Datenübertragungen bei hoher Last

reicht. Der Durchsatz dieser Datenübertragung ist etwas gleich bei niedriger und hoher Last, was als Folge hat, dass die anderen Datenübertragungen entsprechende geringe Durchsätze aufweisen.

Die Durchsätze und Paketverlustraten für die Übertragung von 500 MB werden in Abbildung 5.20 und Tabelle 5.2 gezeigt. Während die Durchsätze der FTP-Datenübertragungen bei Newreno und bei SACK etwas größer als bei ANDINA sind, ist der Durchsatz der HTTP-Datenübertragungen bei ANDINA viel besser als bei den TCP-Varianten. Im ersten Fall erreicht ANDINA ca. 265 KB/s während Reno 201 KB/s, Newreno 260 KB/s und SACK 417 KB/s erreichen. Der Durchsatz der HTTP-Datenübertragungen ist ca. 60 KB/s bei ANDINA, 22 KB/s bei Reno, 40 KB/s bei Newreno und 33 KB/s bei SACK. Noch interessanter sind die Standardabweichungen und die Variationskoeffizienten der HTTP-Datenübertragungen. Bei ANDINA ist die Variation ca. 34 %, bei Newreno und SACK über 67 % und bei Reno über 90 %. Der Grund ist, dass bei den TCP-Varianten viele Verbindungen einen Durchsatz von Null haben!

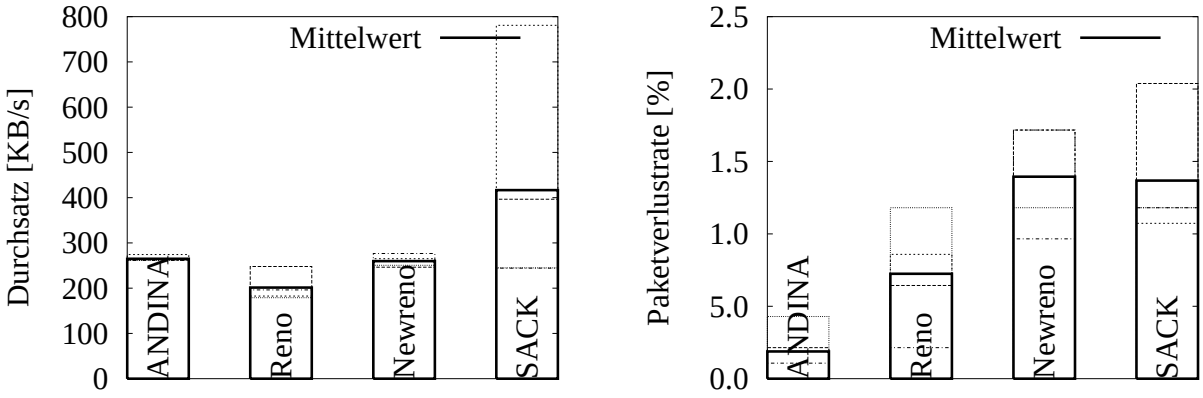


Abbildung 5.20: Test 3) Leistungskenngrößen von vier Datenübertragungen bei hoher Last

	ANDINA			Reno			Newreno			SACK		
Nr. Verb.	Durchsatz [KB/s]	Std.	cv	Durchsatz [KB/s]	Std.	cv	Durchsatz [KB/s]	Std.	cv	Durchsatz [KB/s]	Std.	cv
	L A N G E			D A T E N U E B E R T R A G U N G E N			(F T P)					
	265			201			260			417		
	K U R Z E			D A T E N U E B E R T R A G U N G E N			(H T T P)					
	60.1	20.6	0.34	22.4	20.6	0.92	40.3	26.9	0.67	32.9	25.7	0.78

Tabelle 5.2: Test 3) Durchsatz von langen und kurzen Datenübertragungen bei hoher Last

5.6.4 Test 4) Mehrere lange und kurze Datenübertragungen

In diesem Szenario wird der stationäre Zustand der Warteschlange bei ANDINA, ARED, REM und PI getestet. Die automatisch generierte Topologie hat einen Engpass von 15 Mb/s und RTT zwischen 100 und 160 ms. Die Puffergröße variiert von 20 bis 80 Pakete. Während 100 Sekunden werden jeweils

5 bis 100 lange und viele kurze TCP-Datenübertragungen gestartet. Die langen Datenübertragungen simulieren FTP-Verbindungen, die jede 0.1 Sekunde starten. Zur Simulation von HTTP-Verkehr werden pro Sekunden 5 kurze Datenübertragungen gestartet, die jeweils 5 KB senden. In einer anderen Variante dieses Tests dauern die HTTP-Datenübertragungen jeweils 0.2 Sekunden. Als TCP-Variante wird SACK eingesetzt. Das initiale CWND wird auf 2 Segmente gesetzt.

Mit ARED kann w_q automatisch konfiguriert werden. Jedoch wird $w_q = 0.00027$ angewendet, da dies der optimale Wert für eine Leitung der Datenrate 15 Mb/s ist (siehe [FGS01]). Die Parameter min_{th} und max_{th} werden auf ein Viertel bzw. auf die Hälfte der Puffergröße gesetzt. Die Werte von min_{th} und max_{th} folgen den Richtlinien von [OLW99].

Für REM und PI wird der Parameter für den Sollwert der Warteschlange auf ein Viertel der Puffergröße gesetzt. Die Pakete werden in ARED, REM und PI nicht verworfen sondern markiert.

Die Ergebnisse dieses Tests zeigen die Abbildungen 5.21 und 5.22 sowie die Tabelle 5.3.

- Die Auslastung des Engpasses (siehe Abbildung 5.21) ist bei ARED und bei PI höher als bei ANDINA. Mit 100 parallelen Verbindungen erreicht sie 96 % bei ARED, 98 % bei PI und nur 95 % bei ANDINA. Bei REM wird sogar nur ca. 70 % erreicht. Die höhere Auslastung bei

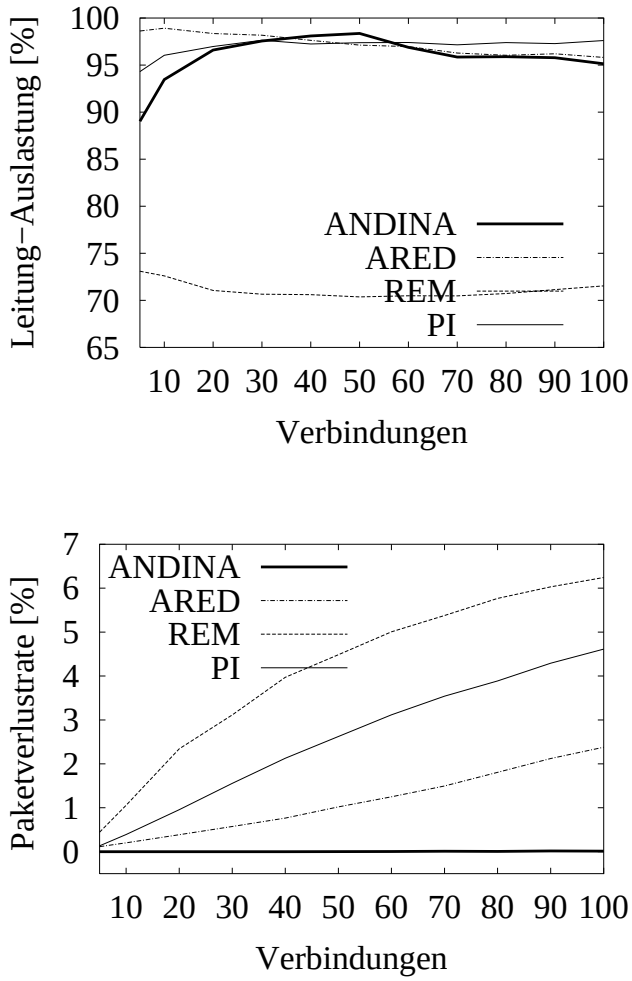


Abbildung 5.21: Test 4) Leistungs-Auslastung und Paketverlustrate im stationären Zustand

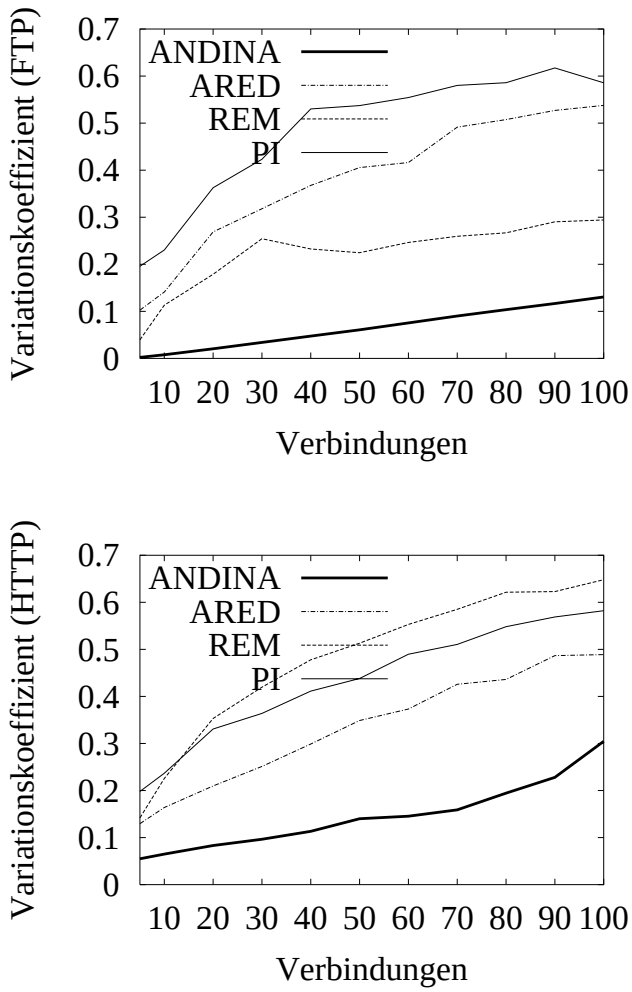


Abbildung 5.22: Test 4) Variation des Durchsatzes im stationären Zustand

ARED und PI wird jedoch mit höherem Paketverlust und geringerer Fairness erkauft.

- Die Paketverlustrate (siehe Abbildung 5.21) mit 100 parallelen Verbindungen ist ca. 2.6 % bei ARED, 6.2 % bei REM, 4.7 % bei PI und. ca. 0 % bei ANDINA. Außerdem ist bei ARED, REM und PI die Paketverlustrate mit zunehmender Last positiv korreliert.
- Die Tabelle 5.3 zeigt für FTP- und HTTP-Datenübertragungen den Durchsatz, die Standardabweichung und deren Variationskoeffizienten. Bei höherer Auslastung ist der Durchsatz der FTP-Datenübertragungen bei ARED und PI minimal höher als bei ANDINA (17.4 KB/s). Der Durchsatz der FTP-Datenübertragungen stellt den Durchsatz für den stationären Zustand dar. Im Fall der HTTP-Datenübertragungen ist dagegen der Durchsatz bei ARED und REM etwas höher als bei ANDINA (13.2 KB/s).
- Gravierend ist der Mangel an Fairness bei ARED, REM und PI. Der Variationskoeffizient des Durchsatzes für die FTP-Datenübertragungen (siehe Abbildung 5.22) ist bei ARED und PI höher als 50 %. Als Folge davon erreichen viele Datenübertragungen einen viel niedrigeren Durchsatz als 17 KB/s. In REM beträgt der Variationskoeffizient ca. 29 %, aber der Durchsatz ist kleiner als bei RED, PI oder ANDINA. Im Gegensatz dazu beträgt der Variationskoeffizient des Durchsatzes bei ANDINA ca. 13 %. Die HTTP-Datenübertragungen in ARED und PI haben

Variationskoeffizienten von jeweils 49 % und 58 %, in ANDINA dagegen nur 30 %. Der Grund dafür ist, dass es bei ARED und PI eine große Anzahl an Datenübertragungen gibt, die nach wiederholtem Ablauf des Retransmission-Timers abgebrochen werden. In diesem Fall erreicht REM einen niedrigeren Durchsatz bei höherer Variation.
In der Variante mit HTTP-Übertragungszeiten von 0.20 Sekunden ist diese Tendenz noch stärker, da weniger Pakete gesendet werden. Die HTTP-Datenübertragungen in ARED und REM haben Variationskoeffizienten von mehr als 90 %, in ANDINA dagegen nur 66 %.

	ANDINA			ARED			REM			PI		
Nr. Verb.	Durchsatz/Std./cv [KB/s]			Durchsatz/Std./cv [KB/s]			Durchsatz/Std./cv [KB/s]			Durchsatz/Std./cv [KB/s]		
	L A N G E			D A T E N U E B E R T R A G U N G E N			(F T P)					
5	293.3	0.6	0.00	329.9	33.8	0.10	245.2	9.7	0.04	315.7	61.8	0.20
10	154.9	1.2	0.01	166.2	23.5	0.14	124.6	14.1	0.11	162.2	37.4	0.23
20	81.1	1.7	0.02	83.5	22.4	0.27	63.9	11.4	0.18	82.8	30.0	0.36
30	55.2	1.9	0.03	56.0	17.8	0.32	43.4	11.1	0.25	55.8	23.6	0.42
40	42.0	2.0	0.05	42.1	15.5	0.37	33.1	7.7	0.23	42.1	22.3	0.53
50	34.0	2.1	0.06	33.8	13.7	0.41	26.9	6.0	0.22	34.0	18.3	0.54
60	28.3	2.1	0.08	28.4	11.8	0.42	22.8	5.6	0.25	28.5	15.8	0.55
70	24.2	2.2	0.09	24.4	12.0	0.49	19.8	5.1	0.26	24.6	14.3	0.58
80	21.4	2.2	0.10	21.5	10.9	0.51	17.6	4.7	0.27	21.7	12.7	0.59
90	19.2	2.2	0.12	19.2	10.1	0.53	15.9	4.6	0.29	19.4	12.0	0.62
100	17.4	2.3	0.13	17.4	9.4	0.54	14.5	4.3	0.29	17.7	10.3	0.59
	K U R Z E			D A T E N U E B E R T R A G U N G E N			(H T T P)					
5	41.5	2.3	0.06	28.7	3.7	0.13	31.9	4.5	0.14	29.2	5.8	0.20
10	36.7	2.4	0.06	27.0	4.4	0.16	29.5	6.7	0.23	26.9	6.4	0.24
20	30.1	2.5	0.08	25.1	5.3	0.21	25.3	8.9	0.35	23.2	7.7	0.33
30	25.1	2.4	0.10	23.0	5.8	0.25	22.1	9.3	0.42	20.8	7.6	0.36
40	22.0	2.5	0.11	21.5	6.4	0.30	20.0	9.6	0.48	19.3	7.9	0.41
50	19.4	2.7	0.14	19.8	6.9	0.35	18.2	9.3	0.51	17.8	7.8	0.44
60	17.5	2.5	0.15	18.9	7.1	0.37	17.0	9.4	0.55	16.5	8.1	0.49
70	15.8	2.5	0.16	17.5	7.5	0.43	15.8	9.2	0.59	15.7	8.0	0.51
80	14.7	2.9	0.19	16.8	7.3	0.44	14.7	9.1	0.62	14.6	8.0	0.55
90	13.4	3.1	0.23	15.3	7.4	0.49	13.9	8.7	0.62	13.6	7.7	0.57
100	13.2	4.0	0.30	15.4	7.5	0.49	13.4	8.7	0.65	13.2	7.7	0.58

Tabelle 5.3: Test 4) Stationärer Zustand (ANDINA/AQM) - Durchsatz und Std.-Abweichung

5.6.5 Zusammenfassung

Die Test-Szenarios zeigen, dass herkömmliche TCP-Varianten es nicht erlauben, die Leitungskapazität des Engpasses fair aufzuteilen. Je höher die Last, desto stärker variieren die Leistungskenngrößen bei den TCP-Varianten. ANDINA hingegen erreicht mit zunehmender Last eine bessere Aufteilung der Leitungskapazität bei gleichzeitig geringerem Paketverlust und etwas kleinerer Auslastung der Leitung. Vor allem in kurzen Datenübertragungen ist der Vorteil von ANDINA gegenüber anderen AQM-Verfahren spürbar besser, da hierbei Mechanismen zur schnellen Weiterleitung der Feedback-Signale vorhanden sind und diese nicht auf Paketverlust basieren.

Kapitel 6

Zusammenfassung und Ausblick

Gegenstand dieser Arbeit ist die Analyse und Optimierung der Überlastkontrolle von TCP-Datenübertragungen mit Hilfe eines netzweit verteilten Verfahrens, welches insbesondere die Router als Überwachungsinstanzen miteinbezieht.

Ausgangspunkt ist die Feststellung der Schwachstellen der derzeitig gebräuchlichsten TCP-Implementierungen. Hierzu wurden die TCP-Abläufe innerhalb des Zustands `ESTABLISHED` sowohl messtechnisch als auch funktional im Detail analysiert. Das dabei entwickelte Zustandsdiagramm veranschaulicht die Einbettung der Überlastkontrolle in TCP innerhalb des Protokollstacks. Die Abhängigkeit der Leistungskenngrößen von zufällig auftretenden Störgrößen wird anhand einer ausgewählten Fallstudie aufgezeigt. Anhand weiterer exemplarischer Fälle wird gezeigt, wie sich ein einzelner Paketverlust völlig unterschiedlich auf zwei konkurrierende TCP-Prozesse auswirken kann. Auf Grund der hohen Variation der Last wird die Leitungskapazität nicht fair aufgeteilt.

Das im Rahmen der vorliegenden Arbeit vorgeschlagene Optimierungsverfahren ANDINA (**A**daptive **N**etwork **D**atastream **IN**line **A**lgorithm) löst das bestehende Problem der Aufteilung der Leitungskapazität für den Best-Effort-Fall innerhalb eines Mehrklassen-Modells, in welchem einzelnen Verbindungen einer Klasse eine vorgebbare Dienstgüte dynamisch zugeteilt wird.

ANDINA besteht aus Mechanismen, die sowohl in den Routern als auch in den Hosts agieren, denn diese Verfahren zeigen auf Grund der durchgeführten Untersuchungen die besseren Leistungsverhalten. Es unterscheidet sich von anderen Verfahren vor allem durch die eingesetzten Router-Mechanismen. Statt das Verwerfen von Paketen oder das Setzen eines Bits in dem Paket-Header baut ANDINA auf quantitative Feedback-Signale zur genauen Feststellung der Senderate in den Hosts auf. Die Router-Mechanismen verwenden keine Ressourcen-Reservierung, sondern schätzen die Last und die optimale Senderate ab. Diese Information wird im Paket-Header weitergeleitet. Dazu speichert der Router Zustandsvariablen der ankommenden Pakete ab. Der Aufwand zur Abschätzung der optimalen Senderate steigt dabei linear mit der Anzahl der Verbindungen. Die Host-Mechanismen von ANDINA entkoppeln die Aufgaben der Überlastkontrolle von denen der Recovery–Mechanismen. Während der Empfänger weiterhin dem Sender die Information zur Recovery im Paket-Header mitteilt, kontrolliert der Sender die Senderate mit Hilfe der Feedback-Signale, die im Pfad von den einzelnen Routern gesetzt werden. Die Trennung dieser Aufgaben ermöglicht gezieltere Optimierungen, die in den aktuellen TCP-Implementierungen nicht möglich sind.

Ein Prototyp von ANDINA wurde dabei in das Simulations-Programm `ns` implementiert und anhand einer Vielzahl von Szenarios auf seine Leistungsfähigkeit und Robustheit getestet. ANDINA wurde

mit den in `ns` vorhandenen TCP-Varianten (Reno, Newreno, SACK) mit den Warteschlangen PD, RED, ARED, REM und PI verglichen.

Die Ergebnisse zeigen deutliche Vorteile gegenüber den herkömmlichen TCP-Varianten auf Grund geringeren Paketverlusts, höheren Durchsatzes und mehr Fairness. Vor allem mit zunehmender Last und kurzen Datenübertragungen, wie es für HTTP-Dienste typisch ist, zeigte sich eine bessere Aufteilung der Leitungskapazität bei gleichzeitig höherem Durchsatz und geringerer Paketverlustrate.

Aufgrund dieser positiven Ergebnisse bietet sich als nächster Schritt eine Implementierung und Bewertung von ANDINA in einem Testnetz an. Um die Leistung von ANDINA sicherzustellen, sind mehrere Aspekte zu berücksichtigen, z.B. die Definition des Paket-Headers, die Trennung der Algorithmen zur Fluss- und Überlastkontrolle bei den Sendern und die Parametrisierung der Router-Mechanismen. Neben Verkehrs-Profilen muss gleichzeitig die Leistungsfähigkeit der Rechner berücksichtigt werden, um ein stabiles Betriebsverhalten zu gewährleisten.

Literaturverzeichnis

[AKS91] A. Agrawala, S. Keshav, and S. Sing. Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers. In I. Press, editor, *Protocols for High Speed Networks II*. April 1991.

[APS99] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC2581, April 1999.

[Arm00] G. Armitage. *Quality of Service in IP-Networks*. MTP Verlag, Indianapolis, April 2000.

[Ath01] S. Athuraliyy. REM: Active Queue Management. *IEEE Network*, June 2001.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC2475, December 1998.

[BCS94] R. T. Branden, D. D. Clark, and S. Schenker. Integrated Services in the Internet Architecture: an Overview. RFC1633, July 1994.

[Bol89] G. Bolch. *Leistungsbewertung von Rechensystemen*. B.G. Teubner, 1989.

[BOP94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. ACM SIGCOMM'94*, pages 24–35. 1994.

[Bra89] R. T. Branden. Requirements for Internet Host – Communication Layers. RFC1122, October 1989.

[BRS99] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM'99*. Cambridge, MA, September 1999.

[CJ89] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[Com95] D. E. Comer. *Internetworking with TCP/IP*. Prentice-Hall, Englewood Cliffs, New Jersey 07632, third edition edition, 1995.

[CSA98] N. Cardwell, S. Savage, and T. Anderson. Modeling the Performance of Short TCP Connections. Technical report, University of Washington, Department of Computer Science and Engineering, October 1998. 1-26.

[Dav72] D. Davies. The Control of Congestion in Packet Switch Networks. *IEEE Transaction on Communications*, June 1972.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queuing Algorithm. In *Proc. ACM SIGCOMM'89*, pages 118–130. 1989.

[ENW96] A. Erramilli, O. Narayan, and W. Willinger. Experimental Queuing Analysis with Long-Range Dependent Packet Traffic. *IEEE/ACM Transactions on Networking*, 1996.

[FF96] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *ACM Computer Communication Review*, 26(3), July 1996.

[FGS01] S. Floyd, R. Gummadi, and S. Schenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Technical report, AT & T Center of Internet Research at ICSI, http://www.icir.org/floyd/papers/adaptiveRed.pdf, August

2001.

[FH99] S. Floyd and T. R. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC2582, April 1999.

[FHPW00] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM'00*. 2000.

[FJ92] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–116, September 1992.

[FJ93] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo91] S. Floyd. Connections with Multiple Congested Gateway in Packet-Switched Networks, Part 1: One-way Traffic. *ACM Computer Communication Review*, 21(5):30–47, October 1991.

[Flo94] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, October 1994.

[FMC] Fundamental Modeling Concepts (FMC). http://fmc.hpi.uni-potsdam.de.

[FP01] S. Floyd and V. Parson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 2001.

[Fri02] E. Fridrich. *Engpassanalyse in offenen Rechnernetzen*. Ph.D. thesis, Universität Karlsruhe, Oktober 2002.

[Her95] B. Hertzberger. Obtaining high performance data transmission in the Internet Characterizing the resource demands of TCP/IP. In *Lecture Notes in Computer Science. N 919. High-Performance Computing and Networking. International Conference and Exhibition. Milan, Italy, May 1995 Proceedings*. 1995.

[HL97] B. A. Huberman and R. M. Lukose. Social Dilemmas and Internet Congestion. *Science*, 1997.

[HMTG01] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. *IEEE Infocom 2001*, 2001.

[Hoe95] J. C. Hoe. *Startup Dynamics of TCP's Congestion Control and Avoidance Schemes*. Master's thesis, MIT, 1995.

[Hoe96] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proc. ACM SIGCOMM'96*, pages 270–280. 1996.

[HTM02] G. Hasegawa, K. Tokuda, and M. Murata. Analysis and Improvement of fairness among many TCP Connections sharing Tail-Drop and RED Router. Technical report, Cybermedia Center, 2002.

[HZ95] M. Haas and W. Zorn. *Methodische Leistungsanalyse von DV-Systemen*. Oldenburg Verlag GmbH, München, 1995.

[Jac57] J. R. Jackson. *Network of Waiting Lines, Operations Research*. J. R. Jackson, 1957.

[Jac88] V. Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review*, 18(4):314–329, 1988. Ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[Jac90] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. *end2end-interest mailing list*, April 1990. Ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[JBB92] V. Jacobson, R. T. Branden, and D. A. Borman. TCP Extensions for High Performance. RFC1323, May 1992.

[JCH84] R. Jain, D. M. Chin, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems. Technical report, DEC Research Report TR-301, 1984.

[JRC87] R. Jain, K. K. Ramakrishnan, and D.-M. Chiu. Congestion Avoidance in Computer

Networks With a Connectionless Network Layer, Part II: An Explicit Binary Feedback Scheme. Technical report, DEC Research Report TR-508, April 1987.

[Kaz76] L. J. Kazmier. *Estadística Aplicada a la Administración y la Economía*. McGraw-Hill, 1976.

[KHR02] D. Katabi, M. Handley, and C. Rohrs. Internet Congestion Control for Future High Bandwidth-Delay Product Environments. In *Proc. ACM SIGCOMM'02*. 2002.

[KJF+97] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore. The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks, Part I: Description. *IEEE/ACM Transactions on Networking*, January 1997.

[Kle75] L. Kleinrock. *Queuing Systems*, volume I: Theory. John Wiley & Sons, 1975.

[Kle76] L. Kleinrock. *Queuing Systems*, volume II: Computer Applications. John Wiley & Sons, 1976.

[Kle79] L. Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communication. In *Proc. Int. Conf. Commun*. 1979.

[KP91] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems*, 9(4), 1991.

[Lai02] K. Lai. CS 268: Congestion Control and Avoidance, February 2002. Lecture Schedule & Notes.

[Lit61] J. D. C. Little. A Proof of the Queuing Formula $L = \lambda * W$. *Operations Research 9*, 1961.

[LM97] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proc. ACM SIGCOMM'97*, pages 127–137. Cannes, France, September 1997.

[LR79] S. Lam and M. Reiser. Congestion Control of Store-and-Forward Network by Input Buffer Limits Analysis. *IEEE Transactions on Communication*, 1, January 1979.

[McK90] P. McKenney. Stochastic Fairness Queuing. In *Proc. IEEE INFOCOM'90*. March 1990.

[Med92] A. Medio. *Chaotic Dynamics*. Cambridge University Press, 1992. Pag. 4-6.

[MK92] P. P. Mishra and H. Kanakia. A Hop by Hop Rate-based Congestion Control Scheme. In *Proc. ACM SIGCOMM'92*. 1992.

[MLF92a] A. Mukherjee, L. H. Landweber, and T. Faber. Dynamic Time Windows and Generalized Virtual clock Combined Closed Loop/Open Loop Congestion Control. In *Proc. IEEE INFOCOM'92*. 1992.

[MLF92b] A. Mukherjee, L. H. Landweber, and T. Faber. Dynamic Time Windows: Packet Admission Control with Feedback. In *Proc. ACM SIGCOMM'92*. October 1992.

[MM96] M. Mathis and J. Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *Proc. ACM SIGCOMM'96*. 1996.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC2018, October 1996.

[Mog93] J. C. Mogul. IP Network Performance. In D. Lynch and M.T.Rose, editors, *Internet System Handbook*, pages 575–675. Addison-Wesley, 1993.

[Nag85] J. Nagle. On Packet Switches With Infinite Storage. RFC970, 1985.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC2474, December 1998.

[OLW99] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proc. IEEE INFOCOM'99*. March 1999.

[Par93] K. Park. Warp Control: a Dynamically Stable Congestion Protocol and Analysis. In *Proc. ACM SIGCOMM'93*. 1993.

[Pax97a] V. Paxon. *Measurements and Analysis of End-to-End Internet Dynamics*. Ph.D. thesis,

University of California, Berkeley, April 1997.

[Pax97b] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proc. ACM SIGCOMM'97*. 1997.

[Pax97c] V. Paxson. End-to-End Routing Bahavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–605, October 1997.

[Pax99] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[PF95] V. Paxson and S. Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 1995.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM'98*. 1998.

[PKTK98] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A Model Based TCP-Friendly Rate Control Protocol. *UMass-CMPSCI Technical Report TR 98-04*, 1998.

[Pos81] J. Postel. Internet Control Message Protocol. RFC792, September 1981.

[Pos85] J. Postel. Transmission Control Protocol. RFC793, September 1985.

[PP87] W. Prue and J. Postel. Something a Host Could Do with Source Quench. RFC1016, July 1987.

[PP93] C. Papadopoules and G. M. Parulkar. Experimental Evaluation of SunOS IPC and TCP/IP Protocol Implementation. *IEEE/ACM Transactions on Networking*, 1(2):199–216, April 1993.

[Rai85] R. Rain. CUTE: A Timeout-Based Congestion Control Scheme for Digital Networking architecture. Technical Report Research Report TR-353, DEC, 1985.

[Rai86] R. Rain. A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks. *IEEE Journal on Selected Areas in Comm.*, pages 1162–1167, 1986.

[Rai89] R. Rain. A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. *ACM Computer Communication Review*, 19:56–71, October 1989.

[Rai90] R. Rain. Congestion Control in Computer Networks: Issue and Trends. *IEEE Network Magazine*, pages 24–30, May 1990.

[RFS90] J. Robinson, D. Friedman, and M. Steenstrup. Congestion Control in BBN Packet-Switched Networks. *ACM Computer Communication Review*, 1990.

[RJC91] K. K. Ramakrishnan, R. Jain, and D.-M. Chiu. A Selective Binary Feedback Scheme for General Topologies. Technical report, Digital Equipment Corporation, 1991.

[Ros93] O. Rose. *Lastkontrolle in Hochgeschwindigkeitsnetzen*. Ph.D. thesis, Universität Karlsruhe, 1993.

[RR88] K. K. Ramakrishnan and R. Rain. A Binary Feedback Scheme for Congestion Avoidance in Computer Network With a Connectionsless Network Layer. In *Proc. ACM SIGCOMM'88*, pages 303–313. 1988.

[SC01] A. Shaikh and K. J. Christensen. Traffic Characteristics of Bulk Data Transfer using TCP/IP over Gigabit Ethernet. In *Proc. IEEE 2001 International Performance, Computing, and Communications Conference*, pages 103–102. 2001.

[Sie03] Das Siemens Online Lexikon. http://w3.siemens.de/solutionprovider/_online_lexikon/, 2003. Quelle: Klaus Lipinski - Lexikon der Datenkommunikation: DATACOM Buchverlag GmbH.

[ST01] S. Schalunov and B. Teitelbaum. Bulk TCP Use and Performance on Internet2, 2001.

[Ste95] W. R. Stevens. *TCP/IP Illustrated*, volume I: The Protocols. Addison-Wesley, 1995.

[Ste97] W. R. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Reco-

very Algorithms. *RFC2001*, January 1997.

[Tan81] A. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Tan96] A. Tanenbaum. *Computernetzwerke*. Prentice-Hall PTR, 3. edition, 1996.

[TMW97] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11:10–23, November 1997.

[VH97] V. Visweswaraiah and J. Heidemann. Improving Restart of Idle TCP Connections. Technical Report 97-661, USC/Information Sciences Institute, July 1997.

[VSN86] R. Varakulsiripunth, N. Shiratori, and S. Naguchi. A Congestion Control Policy on the Internetwork Gateway. *Computer Networks and ISDN Systems*, pages 43–58, 1986.

[WC91] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review*, 21:32–43, January 1991.

[WC92] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. *ACM Computer Communication Review*, 22:9–16, April 1992.

[Wil93] C. L. Williamson. Optimizing File Transfer Response Time Using the Loss-Load Curve, Congestion Control Mechanism. In *Proc. ACM SIGCOMM'93*. September 1993.

[WP98] W. Willinger and V. Paxon. When Mathematics meets the Internet. *Notices of the AMS*, 45(8):961–970, 1998.

[WS95] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated*, volume II: The Implementation. Addison-Wesley, 1995.

[WTSW97] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. *IEEE/ACM Transactions on Networking*, 1997.

[YR95] C.-Q. Yang and A. V. S. Reddy. A Taxonomy for Congestion Control Algorithms in Packet Switching Networks. *IEEE Network*, July/August 1995.

[Zha90] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proc. ACM SIGCOMM'90*, pages 19–29. September 1990.

[Zit00] M. Zitterbart. Unterlagen zur Vorlesung Telematik, 2000. Universität Karlsruhe.

[ZL98] W. Zorn and G. Liefländer. Vorlesung: Systemarchitektur WS 98/99, 1998. Hasso Plattner Institut. Univeristät Potsdam.

[Zor02] W. Zorn. Antrittsvorlesung an der Universität Potsdam: 'Kommunikationssysteme - durch Abstraktion zum Durchblick - am Beispiel eines der meistgenutzten Protokolle', 04.10.2002, Oktober 2002.

[Zor03] W. Zorn. Vorlesung Kommunikationssysteme I, Oktober 2003.

Anhang A

Glossar

ACK:	Eine Quittung in TCP.
AQM:	Active Queue Management.
ARED:	Adaptive Random Early Detection [FGS01].
BBN:	Ein Verfahren zur Überlastkontrolle [RFS90],
Bit-Round FQ:	FQ-Variante [DKS89].
Best-Effort:	Übertragung von Daten ohne Garantie für Dienstgüte.
CA:	Congestion Avoidance [Jac88].
CARD:	Ein Verfahren zur Ende-zu-Ende-Überlastkontrolle [Rai89].
Choke-Packet:	Überlastkontrolle mit Kontrollpaketen [VSN86].
Cliff-Punkt:	In Überlastkontrolle bezeichnet man den Auslastungsgrad, ab dem die RTT zu Unendlich grenzt, der Durchsatz schnell zum Null sinkt, und die Power Null ist.
Congestion-Collapse:	Bezeichnet den Zustand, in dem alle Netz-Ressourcen ausgeschöpft sind, RTT Unendlich ist, und Durchsatz Null ist [Jac88].
CUTE:	Congestion-Control using Timeouts [Rai85].
CWND:	Congestion-Window. Das Fenster zur Überlastkontrolle in TCP.
Datagramm:	Transporteinheit in IP.
DECBit:	Ein Verfahren zur Überlastkontrolle [JRC87].
DiffServ:	Differentiated Services [BBC+98].
DTW:	Dynamic Time Windows [MLF92b].
DUAL:	Ein Verfahren zur Ende-zu-Ende-Überlastkontrolle [WC92].
DUPACK:	Eine duplizierte Quittung in TCP.
Echtzeit-Verkehr:	Zeitkritische Übertragung wie Video und Audio.
E2E:	End-to-End. Ende-zu-Ende, z.B. Ende-zu-Ende-Überlastkontrolle.
ERICA:	Explicit Rate Indication for Congestion Avoidance [KJF+97].
FC:	Fast-Recovery. TCP-Mechanismus zur Behandlung von Paketverlust [Jac90].

Feedback-Signal:	Rückmeldung des Netzes zur Signalisierung des Zustandes und Regelung der Last.
FIN:	Finalise-Flag in TCP.
Flusskontrolle:	Mechanismus zur Steuerung des Datenflusses, damit ein schneller Host einen langsamen nicht überholen kann [Tan96]. Auch Flusssteuerung (engl. Flow-Control) genannt.
FQ:	Fair-Queuing [DKS89], siehe auch Bit-Round FQ [DKS89].
FMC:	Fundamental Modeling Concepts. Siehe `http://fmc.hpi.uni-potsdam.de`.
FR:	Fast-Retransmit. TCP-Mechanismus zur Behandlung vom Paketverlust beim Eintreffen von duplizierten Quittungen (DUPACK).
FRED:	Flow Random Early Drop [LM97].
FTP:	File Transfer Protocol.
Fragment:	Teilstück eines Datagramms.
Fragmentierung:	Teilung eines IP-Datagramms in mehrere Fragmente.
Hop-by-Hop:	Ein Verfahren zur Überlastkontrolle [MK92].
Host:	Endsysteme der Rechnernetze.
HTTP:	Hyper Text Transport Protocol.
ICMP:	Internet Control Message Protocol.
Input-Buffer-Limit:	Überlastkontrolle mit Pufferbehandlung [LR79].
Internet:	Die Sammlung von Netzen und Gateways, einschließlich die ARPANET, MILNET und NSFnet, die die TCP/IP Familien-Protokolle benutzt, und die wie ein einziges, virtuelles kooperatives Netz funktioniert.
IntServ:	Integrated Services [BCS94].
IP:	Internet Protocol. Die Transporteinheit ist ein IP-Datagramm.
Isarithmatic:	Verfahren zur Überlastkontrolle [Dav72].
Knee-Punkt:	In Überlastkontrolle bezeichnet den Auslastungsgrad ab dem, die RTT schnell steigt, der Durchsatz langsam steigt, und die Power maximiert wird.
LAN:	Local Area Network.
Load-Loss-Curve:	Ein Verfahren zur Überlastkontrolle [Wil93].
MSS:	Maximum Segment Size. In TCP ist die größte Segmentlänge.
MTU:	Maximum Transmission Unit.
Newreno:	TCP-Variante [Hoe95, Hoe96, FH99].
Netzüberlastung:	Zustand, wenn die Leistungskenngrößen als Folge der erhöhten Last sinken. (engl. Congestion).
`ns`:	Network Simulator Program. Siehe `http://www.mash.berkerley.edu/ns` Version `ns-2.226`.
OOO-Segmente:	Out-of-Order Segment. Ungeordnete Pakete.

PACK:	Eine partielle Quittung in TCP.
Paket:	Die Dateneinheit, die durch ein paketvermitteltes Netz übertragen wird.
Packet-Pair:	Ein Verfahren zur Ende-zu-Ende-Überlastkontrolle [AKS91].
PD:	Packet-Discarding [Tan81].
PANiON:	Performance Analysis in Open Networks. Siehe `http://www-zorn.ira.uka.de/pan/Welcome.html`.
PI:	Proportional Integrator [HMTG01].
QoS:	Quality-Of-Service. Dienstgüte.
Q-bit:	DECBit-Variante [Ros93].
RBP:	Rate Base Pacing. TCP-Variante [VH97].
Recovery:	Der Prozess zur Wiederherstellung von Paketverlusten.
RED:	Random Early Detection [FJ93].
REM:	Random Early Mark [Ath01].
Reno:	TCP-Variante, die SS, CC und FC implementiert.
RFC:	Request For Comments.
Router:	Zwischensysteme der Rechnernetze.
RT:	Retransmission-Timer.
RTO:	Retransmission-Timeout.
RTT:	Round Trip Time. Zeitverzögerung der Rückmeldung eines Pakets, d.h. benötigte Zeit eines Pakets zum Durchqueren des Pfades.
RWND:	Receiver-Window. Empfängerfenster in TCP, d.h. das Fenster zur Flusskontrolle.
SACK:	Selective Acknowledgment. TCP-Variante [MMFR96] .
Segment:	Transporteinheit in TCP.
SFQ:	Stochastic Fair-Queuing [McK90].
Signallaufzeit:	Zeit zwischen Absenden und Empfangen einer Übertragungseinheit [Bit].
SS:	Slow-Start [Jac88].
Source-Quench:	Überlastkontrolle mit Kontrollpaketen [PP87].
SRED:	Stabilized Random Early Drop [OLW99].
SSTH:	Congestion-Window-Threshold. In TCP ist der obere Schwellenwert des CWND.
SYN:	Synchronise-Flag in TCP.
Tahoe:	TCP-Variante, die SS, CA und FR implementiert.
TCP:	Transfer Control Protocol [Pos85].
`tcpdump`:	Programm zum Protokollieren des Datenverkehrs. Siehe `ftp://ftp.ee.lbl.gov/tcpdump.tar.Z`.
`tcptrace`:	Programm zum Analysieren des TCP-Verkehrs. Siehe `http://irg.cs.ohiou.edu/software/tcptrace/index.html`.

TCP/CM:	TCP-Congestion-Manager [BRS99].
TFRC:	TCP Friendly Rate Control [FHPW00].
TFRCP:	TCP Friendly Rate Control Protocol [PKTK98].
TRI-S:	Ein Verfahren zur Ende-zu-Ende-Überlastkontrolle [WC91].
UDP:	User Datagramm Protocol.
Virtual-Clock:	Überlastkontrolle mit Virtual-Circuits [Zha90].
VEGAS:	Ein Verfahren zur Ende-zu-Ende-Überlastkontrolle [BOP94].
WAN:	Wide Area Network.
Warp:	Ende-zu-Ende-Überlastkontrolle [Par93].
Warteschlangesystem:	Ein aus einem Bediener und einer Warteschlange bestehendes System.
Warteschlangennetz:	Ein aus mehreren Warteschlangensystemen bestehendes System.
WFQ:	Weighted-Fair-Queuing [DKS89].
`xplot`:	Mal-Programm. Unterstützung der Darstellung des TCP-Verkehrs. Siehe `http://www.xplot.org`.
XCP:	Explicit Control Protocol [KHR02].

Anhang B

Mathematisches Glossar

A:	Ankunftsrate der Aufträge
B:	Übertragungsrate
conn:	Anzahl der durchquerenden Verbindungen in ANDINA
conn_lim:	Schwelle für die Anzahl an Verbindungen in ANDINA
conn_m:	Größe der Stichprobe der Verbindungskontexte in ANDINA
conn_s:	Abgeschätzte Anzahl der Verbindungen in ANDINA
cwnd:	CWND
D:	Durchsatz des Systems
D_{max}:	Maximaldurchsatz
iat_opt_i:	optimale Zwischenankunftszeit der Verbindung *i* in ANDINA
iat_ist_i:	Zwischenankunftszeit der Verbindung *i* in ANDINA
K:	Pfadkapazität einer Verbindung
L:	Last
M:	Paketierungszeit
N:	Anzahl der Aufträge im System
N_q:	Anzahl der Aufträge in der Warteschlange
N_s:	Anzahl der Aufträge beim Bediener
P:	Power
P_d:	Signallaufzeit
R:	Reaktionszeit
rto:	RTO
rtt:	RTT
sr:	Senderate
t_first_i:	Zeit der ersten Ankunft eines Pakets der Verbindung *i* in ANDINA
t_last_i:	Zeit der letzten Ankunft eines Pakets der Verbindung *i* in ANDINA

U: Auslastung

v: Signalausbreitungsgeschwindigkeit

X: Bedienzeit beim Bediener

λ: Ankunftsrate der Aufträge (stochastisches System)

μ: Bedienzeit beim Bediener (stochastisches System)

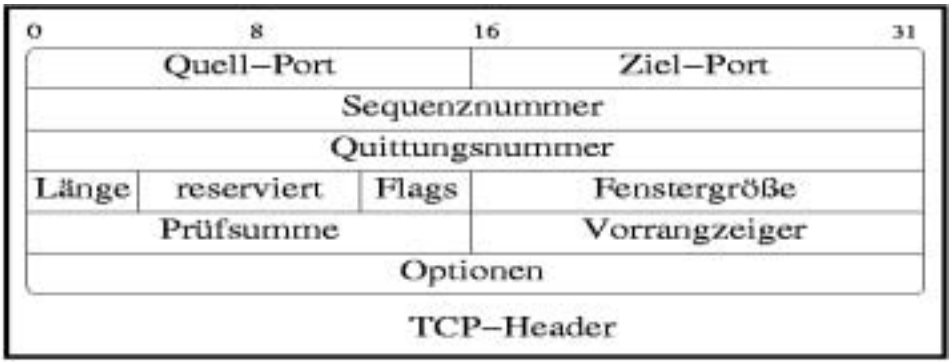
ρ: Auslastung (stochastisches System)

Anhang C

Tools und Tabellen

`tcpdump`-Format:

```
 6 10:06:34.624918 wserver.http > client1.54323:
P 1:1461(1460) ack 159 win 32120
```



`10:06:34.624918`	Uhrzeit
`wserver.http`	Quell-Host.Quell-Port
`client1.54323`	Ziel-Host.Ziel-Port
`P`	URG-Flag
`1:1461(1460)`	Sequenznummer --> Start:Ende(MSS -Bytes)
`159`	Quittungsnummer
`32120`	Fenstergröße

Abbildung C.1: Korrespondenz des `tcpdump`-Formats zu IP- und TCP-Header

Petri-Network and TCP-Variable (4.4BSD-LIte, [WS95]):

	PETRI-NETWORK	TCP	DESCRIPTION
In TCP control block (* not in TCP)			
Sender:	cwnd	snd_cwnd	Congestion Window
	ssth	ssthresh	Congestion Window's threshold
	wnd	snd_wnd	Flow Control Window
	rtt	t_srtt	RTT
	rvar	t_srttvar	RTT's variance approximation
	dpck	t_dupacks	Duplicated acknowledgments
	RTO	t_timer[REXMT]	Retransmission timer
	PERSIST	t_timer[PERSIST]	Persist timer
	rto	rto*	RTO
	suna	snd_una	Segment sent but not acknowledged
	snxt	snd_nxt	Next sequence to be send
	smax	snd_max	Max sequence sent
Receiver:	rwnd	rcv_wnd	Receiver Window
	rnxt	rnd_nxt	Max sequence to receive
	radv	rcv_adv	Advertised window by the other end
	DELACK	TF_DELACK	Delayed acknowledgment timer
	ACKNOW	TF_ACKNO	Constant to send ACK ASAP
In TCP-Header:	seq	th_seq	Sequence number
	ack	th_ack	Acknowledgment number
	rwnd	th_win	Flow Control Window
In TCP/IP/socket functions:			
	rcv(...)	f(tcp_input(...))	Receives IP-buffer
	snd(...)	f(tcp_output(...))	Sends buffer to IP
	getbuf(snxt)	f(sendbuffer,snd_nxt)	Gets next segment from send buffer
	adv(ack)	f(ack, snd_una)	Advances acknowledged data in send buffer
	delvr(rnxt,radv)	f(tcp_reass(...))	Delivers queued data to process
	tmr(...)	f(t_timer[...]))	Sets timer if not already set
	ipoutput(...)	f(ip_output(...))	Sends IP-Packet
"States" Z:	SS	cwnd<ssth	Slow-Start
	CA	cwnd>=ssth	Congestion Avoidance
	FR=T	dpck=3	Fast-Recovery
	FC=T	dpck>3 && FR=T	Fast-Retransmit
	ACKNOW=T	TF_ACKNO = true	Sends ACK ASAP

Data transfer conditions:

	NAME	CONDITION	DESCRIPTION
Sender:	*ACK*	ack=smax+1	Received ACK for all pending segments
	PACK	suna<ack<smax	Received ACK for some pending segments
	DUPACK	ack<=suna	Received duplicated ACK
	RTO	RTO expires	Retransmission Timeout
	PERSIST	PERSIST expires	Persist Mode
Receiver:	*ACKNOW*	ACKNOW=T	Sets Flag to send ACK ASAP
	DELACK	DELACK expires	Delayed timer expired
	OOO	rmax<seq	Received Out-of-Order segment

Connection initial conditions:

NAME	CONDITION
CONN	mss=x
INI:	ssth=32 cwnd=mss suna=snxt=seq=1 smax=mss win=min(wnd,cwnd) rtt=0 rto = 3 . ntics

Abbildung C.2: Erklärung zum Zustandsdiagramm für TCP-Datenübertragung

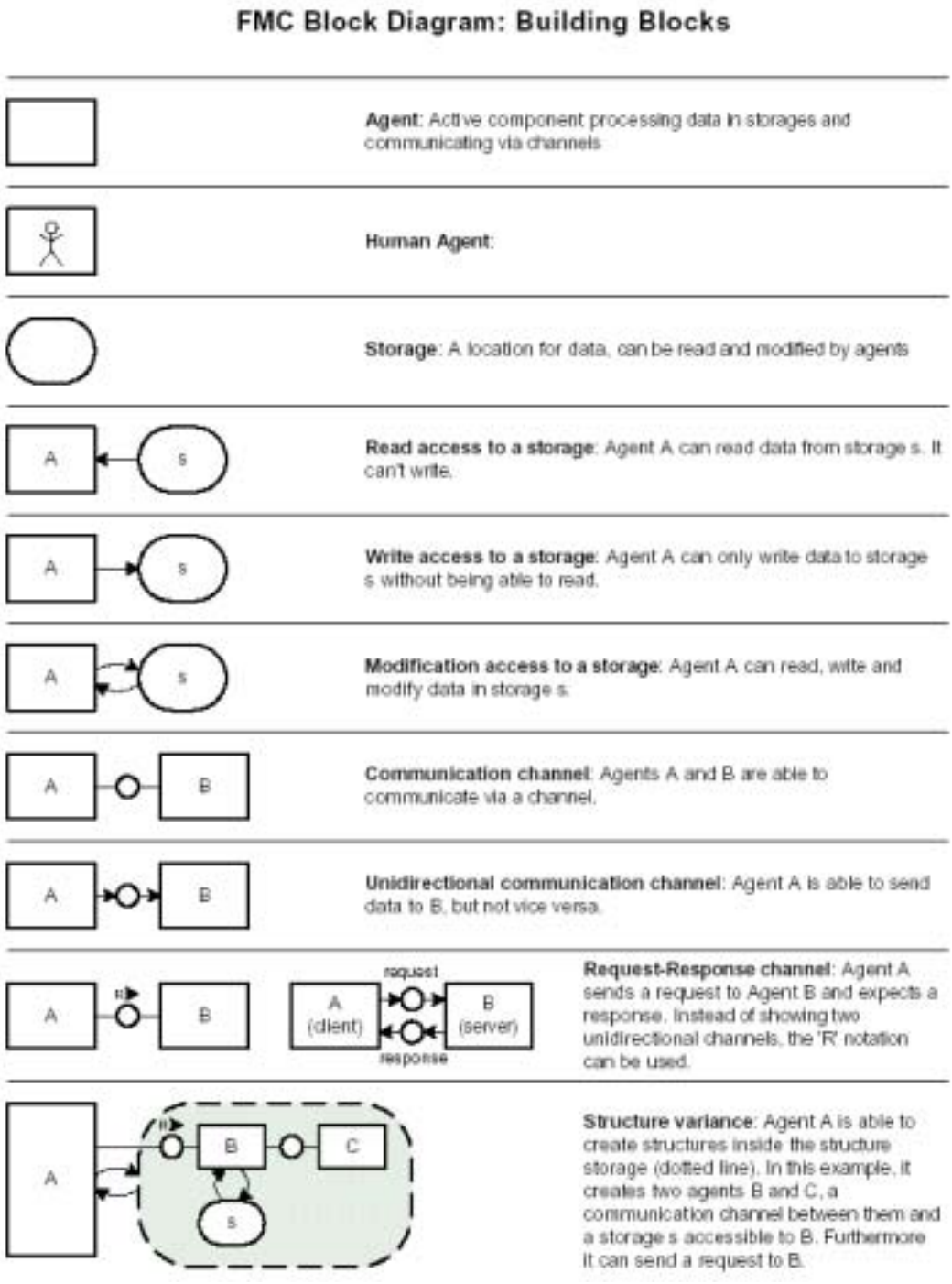


Abbildung C.3: Notation für FMC-Blöckdiagramme [FMC]

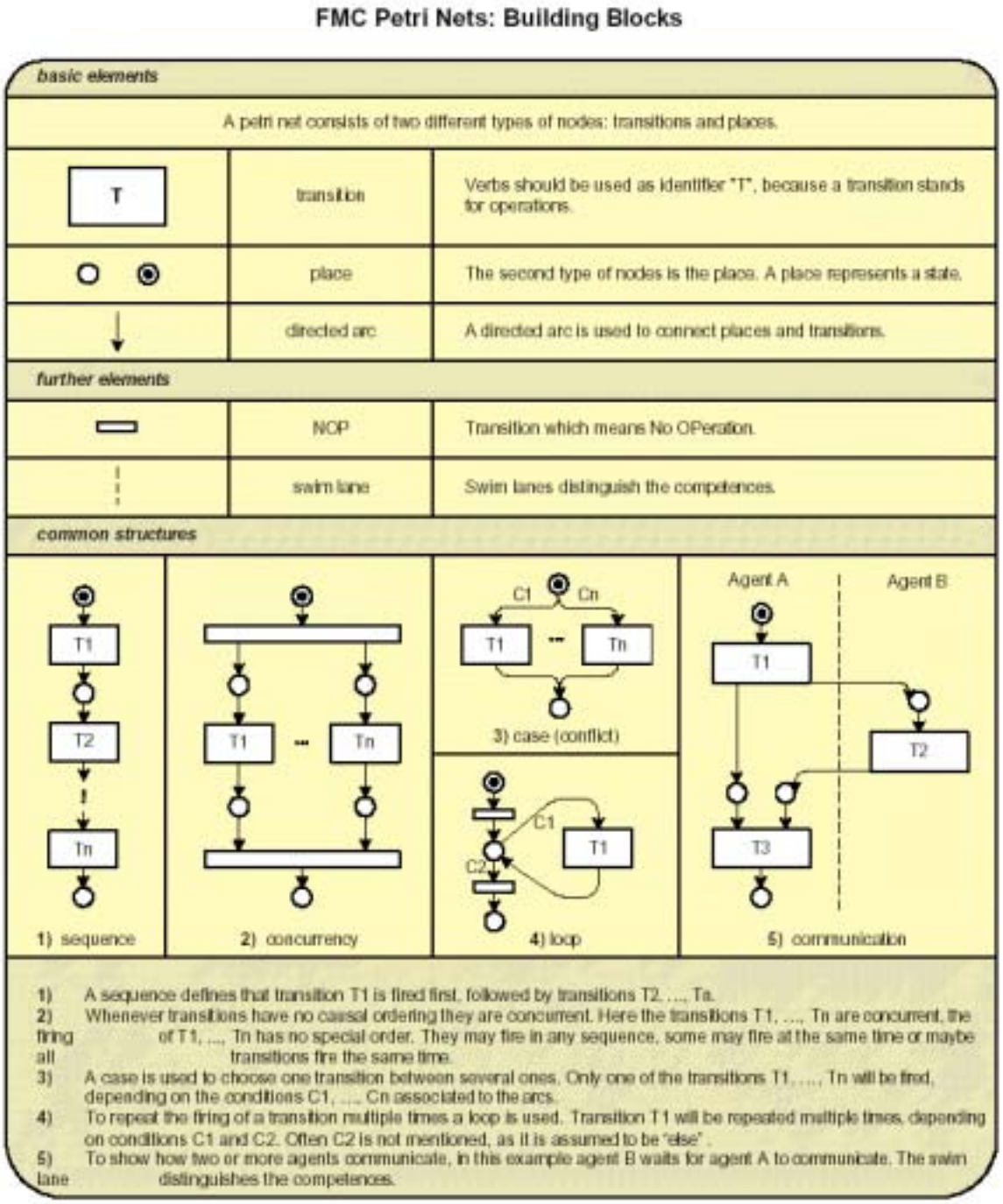


Abbildung C.4: Notation für FMC-Petrinetze [FMC]

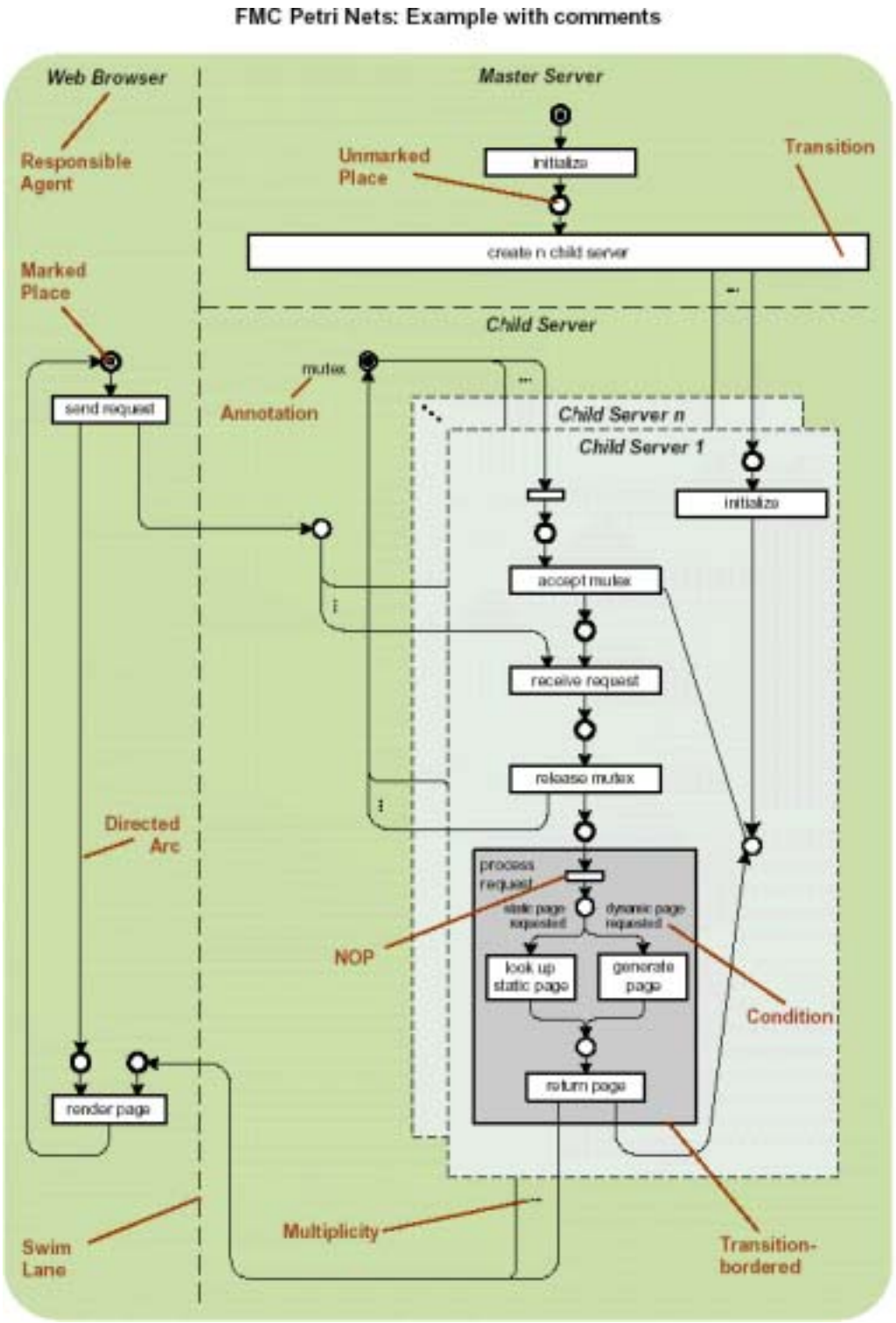


Abbildung C.5: Beispiel eines FMC-Petrineztes [FMC]

```
TCP connection 2:
host c:        client1:57660
host d:        wserver:80
complete conn: yes
first packet:  Fri Apr 19 18:07:09.986725 2002
last packet:   Fri Apr 19 18:07:17.621583 2002
elapsed time:  0:00:07.634858
total packets: 1629
filename:      trace_file_2002.04.19_18:00
   c->d:       d->c:
     total packets:           501          total packets:          1128
     ack pkts sent:           500          ack pkts sent:          1128
     pure acks sent:          498          pure acks sent:            2
     sack pks sent:             0          sack pks sent:             0
     max sack blks/ack:         0          max sack blks/ack:         0
     unique bytes sent:       158          unique bytes sent:   1587602
     actual data pkts:          1          actual data pkts:       1124
     actual data bytes:       158          actual data bytes:   1640162
     rexmt data pkts:           0          rexmt data pkts:          36
     rexmt data bytes:          0          rexmt data bytes:      52560
*****PKT LOST %:             0.00 %       PKT LOST %:             3.19 %
     zwnd probe pkts:           0          zwnd probe pkts:           0
     zwnd probe bytes:          0          zwnd probe bytes:          0
     outoforder pkts:           0          outoforder pkts:           0
     pushed data pkts:          1          pushed data pkts:       1124
     SYN/FIN pkts sent:       1/1          SYN/FIN pkts sent:       1/1
     urgent data pkts:          0 pkts     urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes    urgent data bytes:         0 bytes
     mss requested:          1460 bytes    mss requested:          1460 bytes
     max segm size:           158 bytes    max segm size:          1460 bytes
     min segm size:           158 bytes    min segm size:           582 bytes
     avg segm size:           157 bytes    avg segm size:          1459 bytes
     max win adv:           64240 bytes    max win adv:           32120 bytes
     min win adv:           64240 bytes    min win adv:           31962 bytes
     zero win adv:              0 times    zero win adv:              0 times
     avg win adv:           64240 bytes    avg win adv:           32119 bytes
     initial window:          158 bytes    initial window:         2920 bytes
     initial window:            1 pkts     initial window:            2 pkts
     ttl stream length:       158 bytes    ttl stream length:   1587602 bytes
     missed data:               0 bytes    missed data:               0 bytes
     truncated data:           68 bytes    truncated data:      1539002 bytes
     truncated packets:         1 pkts     truncated packets:      1124 pkts
     data xmit time:        0.000 secs     data xmit time:        7.230 secs
*****IDLETIME SAMPLES:        476          IDLETIME SAMPLES:         11
*****IDLETIME MIN:           19.5 ms       IDLETIME MIN:           343.7 ms
     idletime max:         1001.7 ms       idletime max:           780.0 ms
*****IDLETIME AVG:           16.0 ms       IDLETIME AVG:           200.4 ms
*****IDLETIME STDEV:         52.8 ms       IDLETIME STDEV:         214.2 ms
*****IDLETIME TOTAL:       7634.4 ms       IDLETIME TOTAL:        2204.0 ms
*****IDLETIME MAX %:       13.12 %         IDLETIME MAX %:         10.22 %
     throughput:              21 Bps       throughput:            207941 Bps
*****THROUGHPUT XMIT:          0 Bps       THROUGHPUT XMIT:       219579 Bps

     RTT samples:              3           RTT samples:             373
     RTT min:                0.0 ms        RTT min:                19.2 ms
     RTT max:                0.0 ms        RTT max:               173.0 ms
     RTT avg:                0.0 ms        RTT avg:                24.6 ms
     RTT stdev:              0.0 ms        RTT stdev:              20.9 ms

     RTT from 3WHS:          0.0 ms        RTT from 3WHS:          19.2 ms

     RTT full_sz smpls:        1           RTT full_sz smpls:       370
     RTT full_sz min:        0.0 ms        RTT full_sz min:        19.7 ms
     RTT full_sz max:        0.0 ms        RTT full_sz max:       173.0 ms
     RTT full_sz avg:        0.0 ms        RTT full_sz avg:        24.3 ms
     RTT full_sz stdev:      0.0 ms        RTT full_sz stdev:      20.4 ms

     post-loss acks:           0           post-loss acks:          16
  For the following 5 RTT statistics, only ACKs for
  multiply-transmitted segments (ambiguous ACKs) were
  considered.  Times are taken from the last instance
  of a segment.
     ambiguous acks:           0           ambiguous acks:          16
     RTT min (last):         0.0 ms        RTT min (last):          0.3 ms
     RTT max (last):         0.0 ms        RTT max (last):        142.1 ms
     RTT avg (last):         0.0 ms        RTT avg (last):         25.2 ms
     RTT sdv (last):         0.0 ms        RTT sdv (last):         31.8 ms
     segs cum acked:           0           segs cum acked:          685
     duplicate acks:        1125           duplicate acks:           95
     triple dupacks:           0           triple dupacks:           14
     max # retrans:            0           max # retrans:             2
     min retr time:          0.0 ms        min retr time:          20.5 ms
     max retr time:          0.0 ms        max retr time:        1187.8 ms
     avg retr time:          0.0 ms        avg retr time:         211.7 ms
     sdv retr time:          0.0 ms        sdv retr time:         269.0 ms
*****MAX RETR TIME %:       0.00 %        MAX RETR TIME %:        15.56 %
```

Abbildung C.6: `tcptrace`-Format (Erweiterungen markiert mit *****)

Kenngroesse	client1 avg->	stdv	cv	client2 avg->	stdv	cv
1 Durchsatz [KB/s]	179->	106	0.59	102->	25	0.25
2 Durchsatz-Data [KB/s]	281->	201	0.72	262->	113	0.43
3 Uebertragungszeit-Data [s]	1.5169->	1.7847	1.18	0.9010->	0.7420	0.82
4 RTT-Mittelwert [ms]	29.90->	7.50	0.25	34.20->	3.00	0.09
5 RTT-Standardabweichung [ms]	30.20->	12.00	0.40	25.00->	7.90	0.32
6 Total Pakete	130.6->	4.7	0.04	129.5->	4.4	0.03
7 DUPACKs	11.8->	8.2	0.69	11.9->	9.2	0.77
8 Triple DUPACKs	1.5->	1.4	0.93	1.4->	1.3	0.93
9 Retransmission max	0.9->	0.6	0.67	0.7->	0.6	0.86
10 Leerlaufzeit max [ms]	712.89->	1090.35	1.53	821.98->	321.79	0.39
11 Retransmissionszeit max [ms]	653.94->	1220.99	1.87	146.14->	399.80	2.74
12 Leerlaufzeit max [%]	29.78->	16.68	0.56	46.64->	10.61	0.23
13 Retransmissionszeit max [%]	17.69->	22.02	1.24	5.18->	7.51	1.45
14 Paketverlustrate [%]	3.4->	3.3	0.97	2.6->	3.1	1.19
15 Fenster max [Byte]	20140.0->	9674.0	0.48	22912.0->	8553.0	0.37
16 Mittleres Fenster [Byte]	8651.0->	4332.0	0.50	10160.0->	5578.0	0.55

(a) Gesamt

1 Durchsatz [KB/s]	239->	81	0.34	108->	22	0.21
2 Durchsatz-Data [KB/s]	384->	171	0.45	290->	105	0.36
3 Uebertragungszeit-Data [s]	0.5973->	0.4113	0.69	0.7475->	0.4982	0.67
4 RTT-Mittelwert [ms]	26.80->	5.00	0.19	33.80->	2.90	0.09
5 RTT-Standardabweichung [ms]	24.60->	8.40	0.34	23.70->	7.30	0.31
6 Total Pakete	129.0->	3.4	0.03	128.6->	3.8	0.03
7 DUPACKs	11.8->	8.9	0.75	10.4->	8.9	0.86
8 Triple DUPACKs	1.4->	1.4	1.00	1.1->	1.1	1.00
9 Retransmission max	0.7->	0.5	0.71	0.6->	0.6	1.00
10 Leerlaufzeit max [ms]	198.01->	205.80	1.04	796.95->	164.05	0.21
11 Retransmissionszeit max [ms]	87.56->	226.92	2.59	77.44->	129.43	1.67
12 Leerlaufzeit max [%]	24.88->	8.39	0.34	49.10->	9.85	0.20
13 Retransmissionszeit max [%]	7.30->	9.02	1.24	3.59->	4.65	1.30
14 Paketverlustrate [%]	2.3->	2.5	1.10	1.9->	2.8	1.43
15 Fenster max [Byte]	24985.0->	8290.0	0.33	24496.0->	8215.0	0.34
16 Mittleres Fenster [Byte]	10641.0->	4024.0	0.38	11100.0->	5574.0	0.50

(b) Nicht Parallel

1 Durchsatz [KB/s]	79->	57	0.72	92->	27	0.29
2 Durchsatz-Data [KB/s]	109->	112	1.03	215->	111	0.52
3 Uebertragungszeit-Data [s]	3.0654->	2.1056	0.69	1.1551->	0.9709	0.84
4 RTT-Mittelwert [ms]	35.10->	8.20	0.23	34.80->	3.10	0.09
5 RTT-Standardabweichung [ms]	39.70->	11.50	0.29	27.30->	8.20	0.30
6 Total Pakete	133.3->	5.3	0.04	131.1->	4.8	0.04
7 DUPACKs	11.8->	6.9	0.58	14.3->	9.0	0.63
8 Triple DUPACKs	1.6->	1.4	0.87	1.9->	1.4	0.74
9 Retransmission max	1.2->	0.5	0.42	0.9->	0.5	0.56
10 Leerlaufzeit max [ms]	1584.44->	1386.44	0.88	862.76->	476.33	0.55
11 Retransmissionszeit max [ms]	1611.36->	1568.45	0.97	259.06->	612.28	2.36
12 Leerlaufzeit max [%]	38.15->	22.75	0.60	42.53->	10.52	0.25
13 Retransmissionszeit max [%]	35.26->	26.02	0.74	7.78->	10.14	1.30
14 Paketverlustrate [%]	5.3->	3.6	0.69	3.8->	3.4	0.90
15 Fenster max [Byte]	12024.0->	5507.0	0.46	20208.0->	8443.0	0.42
16 Mittleres Fenster [Byte]	5317.0->	2324.0	0.44	8554.0->	5213.0	0.61

(c) Parallel

Tabelle C.1: TCP-Leistungskenngrößen für die Übertragung von 180 KB

Kenngroesse	client1 avg->	stdv	cv	client2 avg->	stdv	cv
1 Durchsatz [KB/s]	340->	259	0.76	403->	228	0.57
2 Durchsatz-Data [KB/s]	374->	320	0.86	465->	286	0.61
3 Uebertragungszeit-Data [s]	10.3708->	29.9081	2.88	8.2702->	21.4749	2.60
4 RTT-Mittelwert [ms]	21.40->	5.40	0.25	21.00->	6.80	0.32
5 RTT-Standardabweichung [ms]	16.20->	6.00	0.37	10.80->	4.50	0.42
6 Total Pakete	1125.4->	58.6	0.05	1121.9->	33.7	0.03
7 DUPACKs	81.8->	45.6	0.56	76.3->	46.7	0.61
8 Triple DUPACKs	11.7->	8.4	0.72	10.4->	8.2	0.79
9 Retransmission max	1.6->	1.2	0.75	1.5->	1.2	0.80
10 Leerlaufzeit max [ms]	1771.07->	6469.73	3.65	1235.54->	4547.94	3.68
11 Retransmissionszeit max [ms]	2047.95->	7237.84	3.53	1150.96->	4823.41	4.19
12 Leerlaufzeit max [%]	15.19->	15.52	1.02	11.15->	8.66	0.78
13 Retransmissionszeit max [%]	15.41->	16.98	1.10	7.17->	10.75	1.50
14 Paketverlustrate [%]	3.0->	3.0	0.98	2.6->	2.8	1.07
15 Fenster max [Byte]	35755.0->	14386.0	0.40	25481.0->	7180.0	0.28
16 Mittleres Fenster [Byte]	11308.0->	6743.0	0.60	12242.0->	4655.0	0.38

(a) Gesamt

1 Durchsatz [KB/s]	228->	174	0.76	304->	196	0.65
2 Durchsatz-Data [KB/s]	245->	210	0.86	349->	236	0.68
3 Uebertragungszeit-Data [s]	16.9479->	34.8858	2.06	14.5709->	29.9033	2.05
4 RTT-Mittelwert [ms]	25.20->	5.10	0.20	26.10->	7.30	0.28
5 RTT-Standardabweichung [ms]	19.30->	6.10	0.32	12.00->	4.20	0.35
6 Total Pakete	1140.9->	43.0	0.04	1132.1->	44.7	0.04
7 DUPACKs	81.0->	48.5	0.60	76.4->	47.9	0.63
8 Triple DUPACKs	12.2->	9.4	0.77	10.3->	8.0	0.78
9 Retransmission max	2.0->	1.6	0.80	1.9->	1.6	0.84
10 Leerlaufzeit max [ms]	2527.85->	5495.19	2.17	2251.49->	6048.50	2.69
11 Retransmissionszeit max [ms]	3230.55->	7597.20	2.35	2165.37->	6573.99	3.04
12 Leerlaufzeit max [%]	14.85->	13.52	0.91	11.51->	8.09	0.70
13 Retransmissionszeit max [%]	16.85->	15.55	0.92	7.27->	11.89	1.64
14 Paketverlustrate [%]	4.1->	3.4	0.83	3.4->	3.6	1.06
15 Fenster max [Byte]	33662.0->	13806.0	0.41	27551.0->	10818.0	0.39
16 Mittleres Fenster [Byte]	10400.0->	5722.0	0.55	12296.0->	5268.0	0.43

(b) Nicht Parallel

1 Durchsatz [KB/s]	403->	278	0.69	458->	225	0.49
2 Durchsatz-Data [KB/s]	447->	348	0.78	531->	290	0.55
3 Uebertragungszeit-Data [s]	6.6859->	25.9996	3.89	4.7402->	13.5594	2.86
4 RTT-Mittelwert [ms]	19.20->	4.20	0.22	18.20->	4.40	0.24
5 RTT-Standardabweichung [ms]	14.50->	5.10	0.35	10.20->	4.50	0.44
6 Total Pakete	1116.7->	64.1	0.06	1116.3->	23.7	0.02
7 DUPACKs	82.2->	43.8	0.53	76.3->	46.1	0.60
8 Triple DUPACKs	11.4->	7.7	0.68	10.5->	8.3	0.79
9 Retransmission max	1.3->	0.8	0.62	1.3->	0.8	0.62
10 Leerlaufzeit max [ms]	1347.08->	6920.23	5.14	666.35->	3297.43	4.95
11 Retransmissionszeit max [ms]	1385.39->	6940.96	5.01	582.62->	3344.76	5.74
12 Leerlaufzeit max [%]	15.38->	16.53	1.07	10.95->	8.96	0.82
13 Retransmissionszeit max [%]	14.60->	17.68	1.21	7.12->	10.06	1.41
14 Paketverlustrate [%]	2.4->	2.5	1.03	2.1->	2.0	0.94
15 Fenster max [Byte]	36927.0->	14570.0	0.39	24321.0->	3334.0	0.14
16 Mittleres Fenster [Byte]	11816.0->	7203.0	0.61	12212.0->	4274.0	0.35

(c) Parallel

Tabelle C.2: TCP-Leistungskenngrößen für die Übertragung von 1 MB