Ein Simulations-Framework für Endpoint Admission Control

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von:
Marco Studer

2002

Leiter der Arbeit:
Prof. Dr. Torsten Braun

Forschungsgruppe Rechnernetze und Verteilte Systeme (RVS)
Institut für Informatik und angewandte Mathematik

Zusammenfassung

Die Differentiated Services Architektur gilt als vielversprechender Ansatz, um Quality of Service im Internet unterstützen zu können. Die Zugangskontrolle, die so genannte Admission Control, ist dabei ein wichtiges Thema. In letzter Zeit fand vor allem die auf Messungen zwischen Endpunkten beruhende Endpoint Admission Control Beachtung, wobei viele unterschiedliche Messtechniken vorgeschlagen wurden.
Im Rahmen dieser Arbeit wurde ein Simulations-Framework entwickelt, mit dem sich die verschiedenen Architekturen auf einfache Art und Weise evaluieren lassen. Mit Hilfe dieses Frameworks wurden in einem weiteren Teil der Arbeit bestehende Ansätze miteinander verglichen und ein bisher noch nicht untersuchter Ansatz evaluiert.

Inhaltsverzeichnis

1 Einführung

Das Internet stellt im Moment einen einfachen Best-Effort-Service zur Verfügung, bei dem das Netzwerk alle Datenpakete gleich behandelt. Dieses Modell reicht für traditionelle und relativ anspruchslose Internetanwendungen wie Email, Dateitransfer oder Telnet aus. Seit dem Aufkommen von Sprach- und Videoübertragung gibt es Qualitätsansprüche an das Internet, die über die Anforderung "so schnell wie möglich" hinausgehen. Um diese Bedürfnisse zu erfüllen, müssen neue Funktionalitäten eingeführt werden. Die IETF (Internet Engineering Task Force) hat deshalb mit Integrated Services (IntServ) und Differentiated Services (DiffServ) zwei Architekturen für Quality of Service (QoS) entwickelt, auf deren Basis zeitkritische Applikationen in IP-Netzen ohne nennenswerte Qualitätsverluste unterstützt werden.

Die letztere der beiden Architekturen gilt vor allem wegen ihrer Skalierbarkeit als vielversprechender Ansatz. Damit aber in einem DiffServ-Netz die vom Benutzer gestellten Ansprüche auch garantiert werden können, braucht es eine Komponente, die den Zugang des eintreffenden Verkehrs kontrolliert. Besonders Anklang fand in letzter Zeit die auf Messungen zwischen Endpunkten basierende *Endpoint Admission Control*. Endpoint Admission Control beabsichtigt hauptsächlich, einen Soft-Echtzeitdienst zu unterstützen, indem die aggregierte Last auf einem angemessenen Niveau gehalten werden kann, aber keine harten oder präzisen Garantien an individuelle Flüsse gemacht werden können. Die Dienstqualität wird dabei mittels Paketverlust gemessen, und es wird das Ziel verfolgt, diese Verlustrate möglichst klein zu halten.

Endpoint Admission Control ist ein Gebiet, in dem im Moment intensive Forschung betrieben wird. Allein während der Erstellung dieser Arbeit wurden mehrere neue Ansätze vorgeschlagen. Neue Ansätze sind aber nur dann brauchbar, wenn sie gleichzeitig analysiert und evaluiert werden. Im Bereich Computernetze können dazu formale Methoden, Experimente in realen Netzwerken oder Netzwerksimulatoren verwendet werden. Formale Methoden sind für Endpoint Admission Control nicht geeignet, da die verwendeten Algorithmen auf Messungen des aktiven Verkehrs basieren. Da Experimente in Testumgebungen und Labors sehr teuer und zu unflexibel sind, werden zur Analyse von Endpoint Admission Control Konzepten hauptsächlich Netzwerksimulatoren verwendet. Das Ziel dieser Diplomarbeit war, ein Simulations-Framework zur Verfügung zu stellen, mit dem sich Endpoint Admission Control Konzepte auf einfache Art und Weise testen lassen und somit die weiteren Forschungsarbeiten in diesem Bereich zu unterstützen.

Die Struktur dieser Arbeit ist wie folgt: Im zweiten Kapitel werden die bereits erwähnten QoS-Architekturen vorgestellt und weitere Grundlagen zu Endpoint Admission Control eingeführt. Das dritte Kapitel beschreibt die grundlegende Architektur von Endpoint Admission Control, diskutiert dessen Nachteile und Probleme und stellt spezifische Konzepte aus der Literatur vor. Kapitel 4 liefert eine kurze Einführung in den Netzwerksimulator ns-2, während Kapitel 5 die Dokumentation zum entwickelten Simulations-Framework enthält. Das folgende Kapitel präsentiert und analysiert Simulationsergebnisse, die mit Hilfe des Frameworks erzeugt wurden. Bestehende Endpoint Admission Control Konzepte werden in diesem Kapitel miteinander verglichen und ein bisher noch nicht untersuchter Ansatz evaluiert. Kapitel 7 schliesst diese Diplomarbeit ab.

2 Quality of Service und Queueing im Internet

Die Internet-Technologie basiert auf der Übertragung von Datenpaketen. Die Behandlung dieser Datenpakete in den einzelnen Knoten (z. B. Router) erfolgt jedoch in der Praxis ohne die Möglichkeit zur Vergabe von Prioritäten. Aus diesem Grund wird oft vom *Best-Effort-Service* gesprochen, da das Internet versucht, die Datenpakete so schnell wie möglich und mit geringstem Aufwand zuzustellen.

Wenn die Internet-Technologie als Basis professioneller Netze dienen soll, müssen gewisse Garantien in Bezug auf die Qualität der Übertragung möglich sein. Diese Forderung wird umso wichtiger, je mehr zeitkritische Datenpakete übertragen werden sollen (z. B. für die Übertragung von Audio und Video).

Methoden für eine hohe Dienstgüte sind bereits wesentlicher Bestandteil von ATM (Asynchronous Transfer Mode). ATM wurde vor allem im Kern von IP-Netzen eingebettet. Dienstgüte muss jedoch über alle Einrichtungen hinweg bis zum jeweiligen Endteilnehmer sichergestellt werden, um gewisse Garantien abgeben zu können.

Allgemein anerkannte Methoden für die Bereitstellung von Dienstgüte im Internet sind u.a. Paketklassifikation, Staukontrolle und Queue-Management. Diese Methoden bilden die Grundlage für DiffServ und IntServ und werden auch von Endpoint Admission Control genutzt.

2.1 Integrated Services

Durch die Integrated Services Architektur [3] kann eine Echtzeitanwendung einen Ende-zu-Ende-Service mit fester Kapazität und Laufzeit reservieren. Diese Reservierung wird durch ein spezielles Protokoll realisiert, dem Resource Reservation Setup Protocol (RSVP, [2]). Die Grundidee von RSVP ist es, den Pfad festzusetzen, den die Pakete zwischen den zwei beteiligten Endpunkten zurücklegen. Entlang dieses Pfades wird dann die Bandbreite für die Verbindung reserviert (siehe Abb. 2.1). Durch die Reservierungen werden Pakete einer Anwendung vor den Auswirkungen des unkontrollierten Verkehrs geschützt. Pakete von verschiedenen Datenflüssen werden verschiedenartig behandelt. Dies geschieht mit einem Scheduling-Mechanismus wie WFQ (siehe Abschnitt 2.3.2), mit dem die Reihenfolge von ausgehenden Paketen in einer router-internen Queue bestimmt wird.

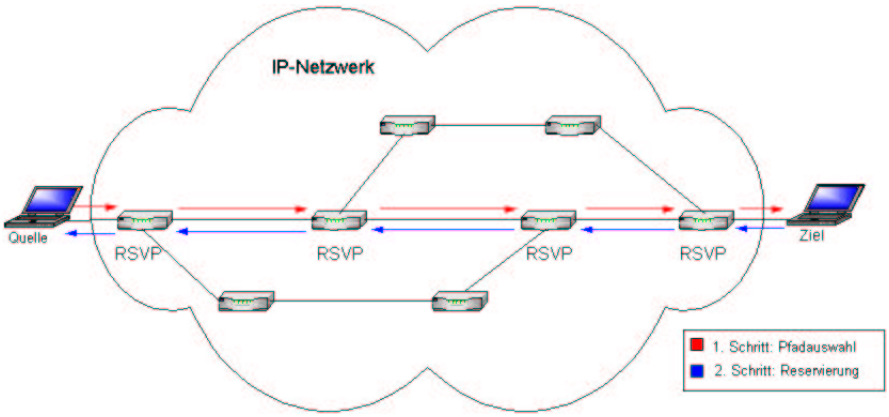


Abbildung 2.1: IntServ-Verbindungsaufbau mit Pfadauswahl und Reservierung

Damit ein Router entscheiden kann, ob er genügend Ressourcen zur Verfügung hat, um die QoS-Anforderungen einer Verbindung zu erfüllen, muss die Verbindung zuerst ihre QoS-Anforderungen spezifizieren und den zu sendenen Verkehr charakterisieren. IntServ fordert daher von der Quelle eine Flussspezifikation (Flowspec), welche zwei Teile umfasst.

Verkehrsspezifikation Die Verkehrsspezifikation (Traffic Specification, TSpec) beschreibt die Variation der angeforderten Bandbreite über der Zeit. Bei den meisten Applikationen ist diese Bandbreite nicht konstant. Eine Videoanwendung generiert z. B. mehr Bit pro Sekunde, wenn sich die Szene schnell bewegt. Eine Möglichkeit die Bandbreite einer Quelle zu beschreiben, ist der Token-Bucket-Filter. Man stellt sich hierbei einen Behälter (Bucket) mit einem bestimmten Volumen vor. Von einem Tokengenerator werden Token mit der konstanten Rate r erzeugt. Damit wird der Behälter kontinuierlich gefüllt bis er voll ist und die Tiefe B erreicht ist. Für jedes gesendete Byte wird ein Token verbraucht. Dies hat zur Folge, dass zu Burst-Zeiten bis zu B Bytes ins Netzwerk gesendet werden können, in einem längeren Intervall können aber nicht mehr als r Bytes pro Sekunde gesendet werden.

Anforderungsspezifikation Die Anforderungsspezifikation (Request Specification, RSpec) beschreibt die vom Netzwerk verlangte Dienstgüte. In der RSpec können Parameter, wie z. B. eine obere Grenze für die Verzögerung, spezifiziert werden, die vom Netzwerk garantiert werden sollen.

2.1.1 Zugangskontrolle

Die Zugangskontrolle ist in IntServ eine Routine zur Überprüfung der Gewährbarkeit einer Reservierungsanforderung. Der Zugang wird jedesmal überprüft, bevor eine neue Verbindung akzeptiert wird. Es wird entschieden, ob

der gewünschte Dienst mit den im Moment verfügbaren Ressourcen und dem aktuellen Verkehr zur Verfügung gestellt werden kann, ohne dass existierende Verbindungen schlechtere Dienste erhalten würden. Kann der Dienst zur Verfügung gestellt werden, wird der Fluss zugelassen, anderenfalls wird er zurückgewiesen. Genauer betrachtet hat die Zugangskontrolle mehrere Zwecke:

- Reservierung von Netzwerkressourcen
- Optimierung der Verwendung der Ressourcen
- Maximierung der Anzahl akzeptierter Dienstanforderungen
- Garantie der angeforderten Dienstgüte
- Führen von aktualisierten Informationen über die Verfügbarkeit der Ressourcen

Es gibt zwei grundlegende Ansätze für die Zugangskontrolle: Der parameterbasierte und der messbasierte Ansatz. Sie unterscheiden sich darin, ob die Entscheidung zur Gewährung mit Hilfe von gegebenen Flussbeschreibungen oder gemessenen Daten erfolgt. Der parameterbasierte Ansatz berechnet anhand einer *a priori* Flusscharakterisierung (z. B. Leaky-Bucket) die Menge der Netzwerkressourcen, die notwendig sind, um eine Reihe von Flüssen zu unterstützen [11]. Ein Zugangstest könnte z. B. so erfolgen, dass die Summe der belegten Bandbreite und der angeforderten Bandbreite mit der insgesamt verfügbaren Bandbreite verglichen wird. Ist diese Summe kleiner als die insgesamt verfügbare Bandbreite, würde in diesem Fall die Anforderung gewährt werden. Meistens ist es aber schwierig, im Voraus eine genaue Beschreibung des Verkehrs zu geben, da der Inhalt einer Verbindung dynamisch generiert wird (z. B. von einem Kompressionsverfahren) und der Paketfluss durch das Zwischenspeichern in Routern beeinflusst wird.

Der messbasierte Ansatz benutzt primär Messungen des aktuellen Verkehrs, um Entscheidungen zu fällen. Verschiedene Messverfahren können dazu verwendet werden, die aktuelle Last des Netzwerks zu schätzen. Dieser Ansatz hat den Vorteil, dass kaum *a priori*-Details bekannt sein müssen.

2.1.2 Nachteile

IntServ hat leider einige Nachteile: So müssen alle Router im Netz und die Endsysteme einschliesslich Applikation RSVP unterstützen. Für einen flächendeckenden Einsatz ist dies angesichts der Vielfalt der Systeme im Internet illusorisch. Ein nicht RSVP-fähiger Router verhindert zwar nicht den Einsatz von RSVP, da die Signalisierungsnachrichten dennoch weitergeleitet werden, die Verlässlichkeit einer Reservierung wird aber stark eingeschränkt. Ein weiterer Nachteil ist der Scheduling-Overhead, der in Routern entsteht, wenn viele Flüsse aktiv sind. Flüsse müssen in den Routern identifiziert und deren Pakete von einem Paket-Scheduling-Mechanismus weitergeleitet werden. Bei einer

grossen Anzahl Flüsse kann das Scheduling sehr aufwändig werden. Zusätzlich entsteht bei kurzlebigen Verbindungen, wie z. B. HTTP-Verbindungen, das Problem, dass Ressourcen reserviert werden, die dann nur kurz gebraucht werden. Da RSVP ein Soft-State-Protokoll ist, müssen die Reservierungen zwar nicht explizit abgebaut werden, dennoch bleiben sie bis zum Ablauf des Reservierungs-Zeitgebers bestehen.

Das Hauptproblem ist jedoch die mangelnde Skalierbarkeit, da jeder Router, der sich auf dem reservierten Pfad zwischen Sender und Empfänger befindet, Zustandsinformationen über einzelne Datenströme speichern muss. Dies funktioniert sehr gut in lokalen Netzwerken mit einer geringen Anzahl von Endsystemen. In einem Backbone-Router werden die Tabellen zur Verwaltung der Flüsse aber sehr gross. Trotz leistungsstarken Prozessoren kann die Verzögerung, die durch das Suchen eines bestimmten Tabelleneintrags entsteht, unakzeptable Ausmasse annehmen.

2.2 Differentiated Services

Motiviert durch die Nachfrage nach einer besser skalierenden QoS-Architektur, wurde die Differentiated Services (DiffServ, [1]) Architektur entwickelt. Anstelle einer Ressourcenverwaltung auf Mikrofluss-Ebene, definiert sie eine kleine Anzahl an Dienstgüteklassen. Pakete werden beim Eintritt ins IP-Netz (dies kann entweder ein DiffServ-fähiger Host oder der erste DiffServ-fähige Router sein, durch den der Verkehr geht) markiert. Die Markierung, die ein Paket erhält, identifiziert die Verkehrsklasse[1], zu der es gehört. Unterschiedliche Klassen erhalten innerhalb des Netzes eine spezifische Dienstgüte. Die einzelnen Router behandeln dazu die Pakete nach einem ihrer Klasse zugeordneten Muster, dem Per-Hop-Behavior (PHB). Bei DiffServ steht also nicht der ganze Pfad im Mittelpunkt, sondern die Art, wie Pakete in den einzelnen Knoten behandelt werden. Der Verwaltungsaufwand ist nicht von der Anzahl Mikroflüsse abhängig, wodurch beliebig grosse Netze verwaltet werden können. Desweitern wird keine Signalisierung verwendet, was die Komplexität erheblich reduziert.

2.2.1 Struktur eines DiffServ-Netzes

Das DiffServ-Modell unterteilt das Internet in DS-Domänen und Domänen ohne DiffServ-Unterstützung, die von unabhängigen Parteien verwaltet werden. Als DS-Domäne wird ein zusammenhängendes Netz von Knoten mit einer gemeinsamen DiffServ-Konfiguration bezeichnet. Die Struktur einer DS-Domäne ist in Abbildung 2.2 dargestellt. Datenströme treten über Ingress-Knoten in die Domäne ein, wo sie einer bestimmten Verkehrsklasse zugeordnet werden. Dies geschieht durch einen *Classifier* und einen *Marker*. Der Classifier wählt Pakete abhängig von bestimmten Werten im Paket-Header wie

[1] RFC 2475 benutzt den Begriff *Behavior Aggregate* anstatt Verkehrsklasse. Im Weiteren wird aus Verständlichkeitsgründen letzterer der beiden Begriffe verwendet.

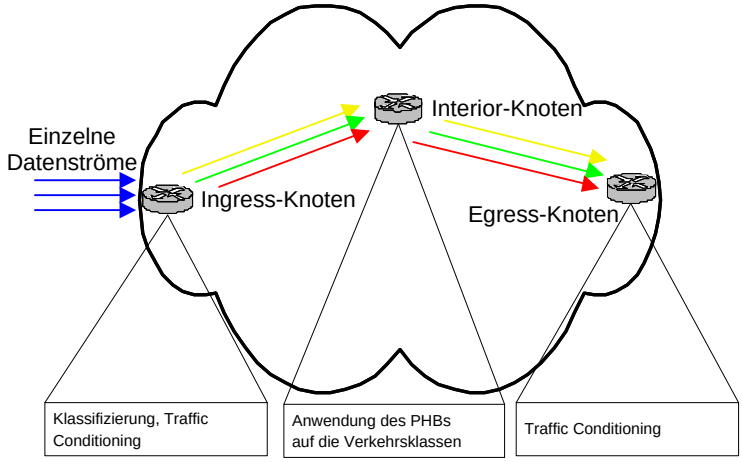


Abbildung 2.2: Struktur eines DiffServ-Netzes mit Ingress-, Interior und Egress-Knoten und deren wichtigsten Aufgaben

Quelladresse oder Zieladresse aus und leitet sie zur passenden Markierungsfunktion weiter. Im Marker wird das Paket im IP-Header mit einem Codepunkt versehen. Da der IPv4-Header aus Kompatibilitätsgründen nicht geändert werden konnte, wurde das 8 Bit breite Type of Service (TOS) Feld dazu verwendet. Das TOS-Byte sollte ursprünglich zur Priorisierung von Paketen nach unterschiedlichen Kriterien dienen. Diese Verwendung hat sich jedoch nicht durchgesetzt. Die IETF Differentiated Services Working Group definierte deshalb 6 Bit dieses Feldes zum Differentiated Service Code Point (DSCP). Anstatt, dass nicht-konforme Pakete mit einem speziellen DSCP markiert werden, können sie von einem *Shaper* verzögert oder einem *Dropper* verworfen werden. Abbildung 2.3 zeigt die Klassifikation und die damit verbundenen Komponenten.

Innerhalb der DS-Domäne werden die Verkehrsklassen von den Interior-Knoten identifiziert und gemäss dem vordefinierten PHB weitergeleitet, bis sie einen Egress-Knoten erreichen. Dieser kann die Ströme normalerweise an die nächste Domäne weiterleiten, ohne sie zu verändern. Werden die Regeln bezüglich eines Datenstroms, die im Traffic Conditioning Agreement (TCA) festgelegt sind, nicht eingehalten, muss der Egress-Knoten die Datenströme zuerst glätten. Erst danach kann er sie weiterschicken.

2.2.2 Per-Hop-Behaviors

Innerhalb eines DiffServ-fähigen Routers erfolgt eine Abbildung zwischen dem Wert des DSCPs und dem PHB. Ein PHB beschreibt nur den von aussen sichtbaren Effekt auf die unterschiedlichen Klassen. Innerhalb eines Knotens werden Mechanismen eingesetzt, um ein erwünschtes Verhalten zu erreichen. Die Mechanismen sind in diesem Sinne austauschbar. Sie können Klassen einen

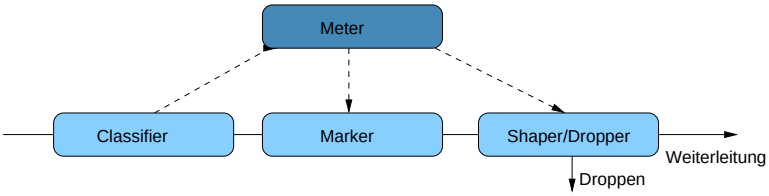


Abbildung 2.3: Klassifikation und Traffic Conditioning in einem DiffServ-Edge-Router

bestimmten Interface-Durchsatz zuordnen (Weighted Fair Queueing), absoluten Vorrang bei der Übertragung einräumen (Priority Queueing) und festlegen, welche Pakete oder Klassen bei Überlast zuerst verworfen werden sollen. Solange der gewünschte Effekt erzielt wird, ist jede Kombination von Mechanismen erlaubt.

Die innerhalb einer DS-Domäne implementierten PHBs haben eine lokale Bedeutung. Die Interoperation zwischen mehreren DiffServ-Domänen beruht auf den gegenseitig vereinbarten Verträgen, den Service Level Agreements (SLA). Somit ist es für die DiffServ-Realisierung nicht zwingend notwendig, spezielle PHBs Internet-weit zu standardisieren. Trotzdem wurden vorläufig zwei Dienste als Standard verabschiedet:

Expedited Forwarding: Mit Expedited Forwarding (EF) wird zugesichert, dass Pakete mit fixer Datenrate und begrenzter Verzögerung weitergeleitet werden. Das so erreichte Verhalten kommt einer Standleitung sehr nahe. Der Durchsatz muss für EF-Daten unabhängig von der Belastung auf dem Link gewährleistet sein. Wenn der Verkehr anderer Klassen Link- und Router-Ressourcen beansprucht, muss immer noch genügend dieser Ressourcen den EF-Strömen zur Verfügung stehen, damit ihre geforderte Bandbreite garantiert werden kann.

Assured Forwarding Assured Forwarding (AF) ist komplexer als EF. AF unterteilt den Verkehr in vier unabhängige Klassen, wobei jeder AF-Klasse eine minimale Bandbreite garantiert wird. Innerhalb jeder Klasse wird jedem Datenpaket eine (relative) Wegwerfpriorität (Drop Precedence) zugewiesen. Überschreitet eine Klasse ihre Ressourcen, werden zuerst Pakete mit hoher Priorität, danach solche mit mittlerer Priorität verworfen. Ein ISP kann den Verkehrsklassen unterschiedliche Leistungsniveaus zuordnen, indem er die Menge an Ressourcen variiert, die jeder Klasse zugewiesen ist.

2.2.3 Vergleich mit IntServ

Durch die Reduktion auf überschaubare Klassen erreicht der DiffServ-Ansatz eine bessere Skalierbarkeit als IntServ, was vor allem beim Einsatz auf Fernnetzen (Wide Area Networks) grosse Vorteile bringt. Andererseits ist es aber

schwerer, ein spezifisches Verhalten für individuelle Flüsse zu garantieren, da die Verzögerung oder die Rate, die ein Fluss erfährt, von der aggregierten Klasse und der Menge an Ressourcen, die für sie reserviert wurde, bestimmt wird. IntServ kann dagegen eine bestimmte Dienstgüte garantieren, was es für Echtzeitanwendungen in einem lokalen Netz sehr attraktiv macht.

Neuere Ansätze verbinden deshalb die Vorteile von DiffServ und IntServ. Nutzt man, wie in [4] beschrieben, IntServ mit RSVP in den Randbereichen und DiffServ im Kernbereich eines Netzwerks, können daraus sowohl Leistungs- als auch Management-Vorteile entstehen.

2.3 Queueing-Komponenten für Quality of Service

Die vorgestellten QoS-Architekturen setzen voraus, dass Netzwerkkomponenten Pakete differenziert behandeln, um eine bestimmte Dienstgüte zu erreichen. Diese Unterscheidung zwischen Paketen kann pro Fluss erfolgen, wie in IntServ, oder für eine Aggregation von Paketen, wie es in DiffServ der Fall ist. Gäbe es im Internet überall schnelle und wenig belastete Verbindungen, würden die Daten so fliessen, wie sie generiert wurden, und es müsste nichts getan werden. Die Realität ist aber so, dass Router oft mehr Pakete über einen Link transferieren müssen, als es die Kapazität des Links erlaubt. Dies tritt insbesondere dann ein, wenn an den Eingangs-Ports eines Routers verschiedene Ströme eintreffen, die auf dem gleichen Ausgangs-Port weitergeleitet werden sollen. Es entsteht Stau und der Router muss Pakete fallen lassen, falls seine Pufferspeicher voll sind. Alle Ansätze, die in irgendeiner Weise Dienstgüte realisieren, basieren darauf, die "richtigen" Pakete fallen zu lassen.

Ein Internet-Router verwendet üblicherweise für jedes ausgehende Interface einen Queueing-Mechanismus, um Pakete differenziert zu behandeln. Einfache Mechanismen arbeiten nach dem FIFO-Prinzip. Pakete werden in der Reihenfolge in der Queue zwischengespeichert wie sie ankommen. Ist der Pufferspeicher voll, werden Pakete am Anfang oder am Ende der Queue verworfen. Komplexere Mechanismen können bestimmte Pakete schneller weiterleiten als andere oder sie vom Verlust bewahren. Mehrere Mechanismen wurden für verschiedene Zwecke entwickelt.

2.3.1 Priority Queueing

Im *Priority Queueing* werden mehrere Queues verwendet, die von einem Scheduler bedient werden. Jede Queue hat eine Priorität, mit der ihre Pakete weitergeleitet werden. Der Scheduler sucht die Queues von der höchsten zur niedrigsten Priorität ab und leitet vorhandene Pakete weiter. Wurde ein Paket gesendet, beginnt der Scheduler wieder mit der Queue mit der höchsten Priorität. Pakete in dieser Queue erhalten somit den besten Service.

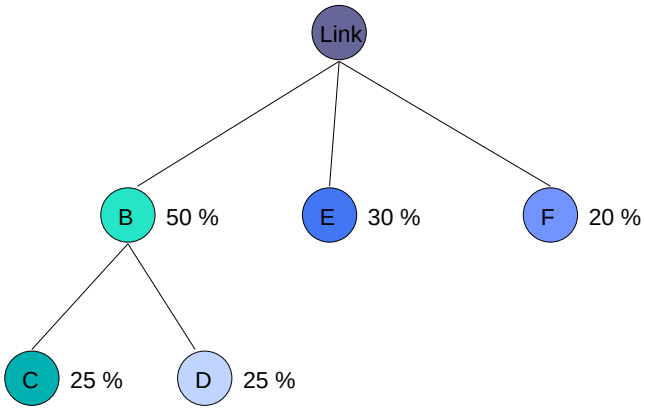


Abbildung 2.4: Link Sharing Modell von CBQ

2.3.2 Fair Queueing

Fair Queueing sieht im Wesentlichen vor, dass für jeden Fluss, den ein Router behandelt, eine Queue verwaltet wird. Ist eine Ausgangsleitung frei, tastet der Router die Queues nacheinander ab und nimmt das erste Paket heraus. Auf diese Weise erhalten alle Hosts (oder Flüsse), die um eine bestimmte Ausgangsleitung konkurrieren, die Gelegenheit, eines ihrer Pakete zu senden. Die Allokation der Ressourcen erfolgt nach der sogenannten Max-Min Fair Share Allokation. Das bedeutet, dass kein Fluss mehr Bandbreite erhält als er verlangt und dass Flüsse mit nicht befriedigten Anforderungen gleiche Anteile erhalten.

Eine Variation von Fair Qeueuing ist *Weighted Fair Queueing* (WFQ). WFQ ermöglicht es, jeder Queue ein Gewicht zuzuordnen. Dieses Gewicht gibt an, wie viele Bit transferiert werden können, wenn die Queue bedient wird. Die Konfiguration von WFQ erfolgt dynamisch, da es Veränderungen in den Netzwerkbedingungen automatisch adaptiert. Die TCP-Staukontrolle und der Slow-Start-Mechanismus werden durch WFQ auch verbessert. Das Resultat ist ein vorhersehbarer Durchsatz für jeden aktiven Fluss.

Beide Scheduling-Mechanismen können nicht nur auf einzelne Flüsse, sondern auch auf eine Aggregation von Flüssen angewendet werden. WFQ kann somit dazu verwendet werden, einen Anteil der Bandbreite für eine priorisierte DiffServ-Klasse zu reservieren, während der andere Teil für den Best-Effort-Verkehr übrig bleibt.

2.3.3 Class Based Queueing

Class Based Queueing (CBQ) ist ein Modell, das die Verteilung der Übertragungsraten zwischen unterschiedlichen Verkehrsklassen festlegt. Kernstück dieses Mechanismus sind verschiedene Queues für verschiedene Verkehrsklassen. Zum Senden eines Pakets wird mit einem gewichteten Round-Robin-

Verfahren eine der Queues ausgewählt. Das gesendete Paket durchläuft dann eine Messeinrichtung (Estimator), welche die Gewichtung der verschiedenen Klassen entsprechend beeinflusst. Den verschiedenen Klassen können sowohl Bandbreiten-Anteile als auch Prioritäten zugewiesen werden. Zudem können die Klassen hierarchisch strukturiert werden. Nicht genutzte Bandbreite der übergeordneten Klasse kann dann an die Unterklassen übergeben werden. Abbildung 2.4 zeigt die Verteilung eines Links auf 5 Klassen. An der Wurzel der Hierarchie steht der Link selber. Die Knoten repräsentieren verschiedene Klassen, die wiederum Unterklassen enthalten können.

2.3.4 Active Queue Management

Infolge der enormen Zunahme der Anzahl Benutzer und Anwendungen, die das Internet nutzen, ist Überlastung (Congestion) eine der Hauptschwierigkeiten im heutigen Internet geworden, was zu übermässigem Paketverlust und Verzögerungen im Netz führte. Lösungen dieses Problems basieren hauptsächlich darauf, dass Benutzer auf Änderungen in den Netzbedingungen mit minimalen Informationen vom Netz reagieren. Aufgabe des Active Queue Managements (AQM) ist dabei, die durchschnittliche Queueing-Verzögerung zu kontrollieren, während gleichzeitig vorübergehende Fluktuationen in der Queue-Größe erlaubt werden. AQM kann verschiedene Methoden anwenden, um dem Endsystem Stau anzuzeigen. Beispielsweise können Pakete verworfen (siehe Random Early Detection) oder markiert (siehe Explicit Congestion Notification) werden.

2.3.4.1 Random Early Detection

Ein wichtiger Einflussfaktor für die Datenübertragung mit hoher Übertragungsrate ist die Länge der Queue innerhalb von Routern. In Zeiten niedriger Anforderungen kann der Router seine Queue entleeren. Daher ist es vorteilhaft, wenn die Queue möglichst gross ist. Da sich diese aber nicht beliebig vergrössern lässt, ist man daran interessiert, sie so effektiv wie möglich zu nutzen. TCP-Datenströme haben allerdings die unangenehme Eigenschaft sich zu synchronisieren. Das bedeutet, dass zwei Datenströme, die ursprünglich zu unterschiedlichen Zeitpunkten hohe Datenraten lieferten, nach einiger Zeit zum selben Zeitpunkt hohe Datenraten liefern. Im zeitlichen Mittel wird die Queue des Routers hierdurch sehr schlecht genutzt [6].

Man versucht, diesem Effekt mit *Random Early Detection* (RED) entgegenzuwirken. Dabei verwirft der Router gezielt Pakete, noch bevor die Kapazität seiner Queue überschritten wurde. Auf diese Weise kann er die einzelnen Datenströme zu unterschiedlichen Zeitpunkten zu einer Reduzierung der Übertragungsrate bewegen und damit eine Synchronisation der Datenströme verhindern. RED versucht, die durchschnittliche Queue-Länge dadurch zu kontrollieren, dass es den End-Hosts anzeigt, wann sie temporär ihre Übertragung

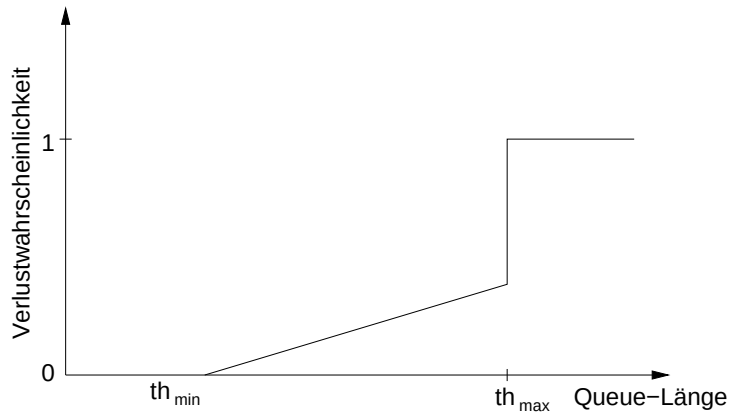


Abbildung 2.5: Verlustwahrscheinlichkeit für ein Paket in einer RED-Queue

von Paketen verringern sollen. Die Quelle wird somit durch Timeouts oder Duplikate implizit benachrichtigt.

Der RED-Algorithmus besteht im Wesentlichen aus zwei Teilen:

- Bewertung der durchschnittlichen Queue-Länge
- Entscheidung ob ein Paket verworfen werden soll

Die durchschnittlichen Queue-Länge wird durch

$$avg = (1 - w_q) \cdot avg_{old} + v \cdot w_q$$

berechnet, wobei $0 < w_q < 1$, v die aktuelle Queue-Länge und avg_{old} der alte Durchschnittswert ist.

Abbildung 2.5 zeigt die Verlustwahrscheinlichtkeit p_b für eine RED-Queue. Zwischen den beiden Parametern th_{min} und th_{max} wird der RED-Algorithmus verwendet. Die Queue-Länge kann dabei entweder in Paketeinheiten oder in Bytes gemessen werden. Die durchschnittliche Queue-Länge wird mit einem minimalen (th_{min}) und einem maximalen (th_{max}) Schwellenwert verglichen. Wenn der Durchschnittswert unter dem minimalen Schwellenwert liegt, werden keine Pakete verworfen. Liegt der Durchschnittswert über dem maximalen Schwellenwert, so werden alle eintreffenden Pakete gelöscht. Liegt der Durchschnittswert zwischen dem minimalen und maximalen Schwellenwert, so steigt die Wahrscheinlichkeit, dass ein eintreffendes Paket verworfen wird linear mit der durchschnittlichen Queue-Länge, bis der maximale Schwellenwert erreicht wird.

Die Wahrscheinlichkeit, dass ein Paket verworfen wird, ist auch von der Grösse des Pakets abhängig. Bei diesem Algorithmus werden mehr Pakete verworfen, sobald die Überlastung wächst. Grössere Pakete werden eher verworfen als kleinere Pakete, die weniger Ressourcen benötigen.

2.3.4.2 Weighted RED und RIO

Weighted RED (WRED) kombiniert die Möglichkeiten des RED-Algorithmus mit den IP-Prioritäten, um eine bevorzugte Behandlung von Paketen mit höherer Priorität zu erreichen. WRED kann selektiv Verkehr mit niedriger Priorität verwerfen, sobald ein Stau bevorsteht und bietet verschiedene Leistungseigenschaften für die unterschiedlichen Dienstklassen. RIO (RED with In and Out) verwendet wie WRED einen RED-Algorithmus pro Prioritätsstufe. Es können aber nur zwei Prioritäten unterschieden werden. Im Gegensatz zu WRED berechnet RIO eine durchschnittliche Queue-Länge für IN-Pakete (in-profile) und benutzt die durchschnittliche Queue-Länge aller Pakete in der Queue (avg_{all}) für OUT-Pakete (out-of-profile). Beide Paketklassen benutzen die gleichen Parameter für den Schwellenwert, wobei die Verlustwahrscheinlichkeit für IN-Pakete von avg_{in} abhängt und die für OUT-Pakete von avg_{all}.

Mit beiden Algorithmen kann z. B. in einem IntServ-Netz erreicht werden, dass bestimmte Flüsse bevorzugt behandelt werden und Pakete anderer Datenflüsse eher verworfen werden.

2.3.4.3 Explicit Congestion Notification

ECN [10] ist ein Verfahren, bei dem ein Router durch eine einfache Markierung eines Bits im IP-Header eine drohende Überlastung mitteilen kann. ECN ist sowohl auf der IP-Ebene als auch auf der TCP-Ebene des Netzwerk-Schichtenmodells definiert und benötigt jeweils zwei Bit im IP- und im TCP-Header.

Es werden die umdefinierten Bits 6 und 7 des IPv4-TOS-Feldes herangezogen.

- Das ECN-Capable-Transport(ECT)-Bit (Bit 6 im IPv4-TOS-Feld) wird vom Datensender gesetzt, um zu zeigen, dass die Endpunkte des Transportprotokolls ECN-fähig sind.
- Das Congestion-Experienced(CE)-Bit (Bit 7 im IPv4-TOS-Feld) wird vom Router gesetzt, um dem Endsystem Überlastung anzuzeigen. Treffen Pakete in einem Router mit einer vollen Queue ein, werden diese Pakete verworfen.

Die Markierungen im TCP-Header werden auf Sender- sowie auf Empfängerseite dazu genutzt, die TCP-Staukontrolle zu verbessern. Für weitere Details zum Zusammenspiel von ECN mit TCP siehe [5].

3 Endpoint Admission Control

Es ist allgemein erwiesen, dass strikte Zugangskontrolle einen wesentlichen Teil dazu beiträgt, dass in einem Netzwerk Garantien Ende-zu-Ende zur Verfügung gestellt werden können. Die traditionelle Zugangskontrolle, die in Abschnitt 2.1.1 beschrieben wurde, muss dazu Informationen über jeden Fluss besitzen. Die Entscheidungen werden von Routern innerhalb des Netzwerks getroffen.

In einer DiffServ-Architektur ist es leichter, die Verwaltung der Flüsse zu zentralisieren. In *Bandwidth Brokern* [1] wird z. B. die Zugangskontrollentscheidung an einem zentralen Ort für jede administrative Domäne gefällt. Der Einsatz von Bandwidth Brokern reduziert die Kosten erheblich, die für den Aufbau eines Flusses notwendig sind, da bei der Zugangskontrollprozedur nur ein Punkt in der Domäne angefragt werden muss. Diese Lösung skaliert aber nicht, da der Bandwidth Broker in Situationen mit hoher Flussaktivität mit Anfragen überhäuft wird.

Der Overhead, der durch die Verbindungsanfragen entsteht, kann zusätzlich reduziert werden, wenn die Entscheidungen nur an Endpunkten getroffen werden. Für diese sogenannte *Endpoint Admission Control* (EAC) wird keine explizite Unterstützung der Router benötigt. Diese verwerfen oder markieren kontrollierte Pakete, abgesehen von der Verwendung der Priorisierungsmechanismen von DiffServ, wie Best-Effort-Pakete. EAC ist ein Versuch, unter Benutzung der Best-Effort-Infrastruktur (mit ihren DiffServ-Erweiterungen) und durch Hinzufügen von Algorithmen an den Endpunkten[1] einen Echtzeitdienst zur Verfügung zu stellen.

Es sei erwähnt, dass im Gegensatz zu [21] der Begriff EAC nur im Zusammenhang mit einer Zugangskontrolle verwendet wird, die auf aktiven Messungen durch Endpunkte basiert.

3.1 Fundamentale Architektur

Die einzelnen Ansätze in der Literatur (siehe Abschnitt 3.5) unterscheiden sich erheblich im Detail, haben aber ähnliche Architekturen. In allen Ansätzen wird das Netzwerk zuerst geprüft, bevor die Daten eines Flusses gesendet werden können. Wenn sich nach dem Prüfen herausstellt, dass das Netzwerk zu stark

[1] Ein Endpunkt kann entweder ein Host oder ein Edge-Router sein.

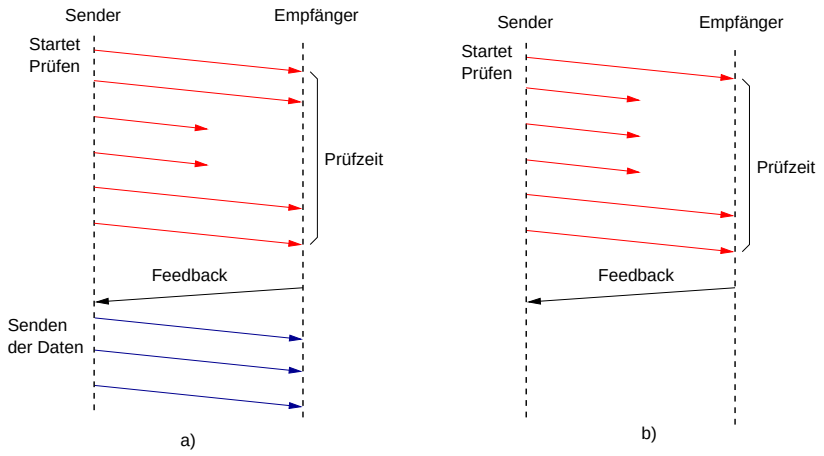


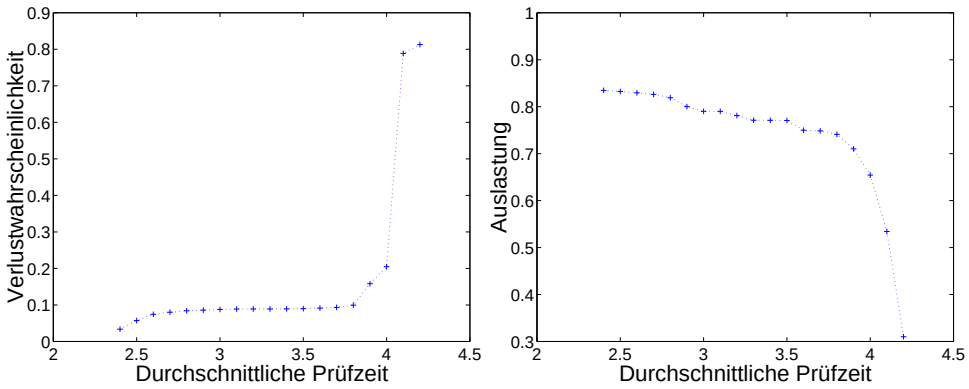
Abbildung 3.1: Zugangskontrollprozedur mit a) Akzeptanz und b) Zurückweisung

überlastet ist, dürfen die Daten nicht gesendet werden. Um das Mass der Überlastung in einer quantitativen Grösse darstellen zu können, werden während der Prüfphase die Anzahl verlorenen oder markierten Pakete gespeichert. Abhängig von diesem Wert wird dann die Zugangsentscheidung getroffen.

Datenpakete werden entweder mit gleicher (in-band) oder höherer Priorität (out-of-band) gesendet als Prüfpakete, aber immer mit höherer Priorität als Best-Effort-Verkehr. Zusammen mit der verwendeten Methode zur Überlastungsdetektion ergeben sich daraus vier grundlegende Entwürfe: in-band Dropping, out-of-band Dropping, in-band Marking und out-of-band Marking [8]. Beim in-band Dropping und out-of-band Dropping werden während des Prüfens nur die verloren gegangen Pakete gezählt, während beim in-band Marking und out-of-band Marking zusätzlich noch die markierten Pakete gezählt werden. Diese Markierung kann z. B. durch ein ECN-Bit realisiert werden und erfolgt dann, wenn ein Paket einen überlasteten Link passiert. Egal welche Technik eingesetzt wird, führt das Prüfen zu einer bedeutenden Aufbauverzögerung von einigen Sekunden. Nicht alle Echtzeitanwendungen werden dies einfach so tolerieren.

3.2 Die Zugangskontrollprozedur

Abbildung 3.1 illustriert die Phasen beim Aufbau eines kontrollierten Flusses mit Akzeptanz resp. Rückweisung. Zuerst initiiert der Sender die Prüfphase. Das Ziel dieser Prüfphase ist es, das Netzwerk so zu belasten, als würde bereits der Datenverkehr fliessen. Dies impliziert, dass die Rate der Prüfpakete gleich gross sein muss, wie die zusätzliche Last, die nach der Akzeptanz der Verbindung generiert wird. Betrachtet man Flüsse mit konstanter Rate, ist dies offensichtlich; Falls ein prüfender Fluss mit kleinerer Rate sendet, als der Datenfluss anfordert, kann dies zu einer Verschlechterung der Service-Qualität führen,



(a) Verlustwahrscheinlichkeit

(b) Auslastung

Abbildung 3.2: Illustration des Thrashing-Problems

wenn die Verbindung in die Datenphase wechselt. Pakete gleicher Länge sollten also mit der Datenrate gesendet werden, mit der man die darauf folgenden Daten verschicken möchte.

Die Dauer der Prüfphase kann der Sender selber wählen. Die Prüfpakete können je nach Ansatz verschiedene Informationen enthalten. So verwendet der Ansatz in Abschnitt 3.5.1 Angaben über die Prüfdauer, die Datenrate und einen Identifikator mit dem die Anforderungen unterschieden werden können. Das Ziel zählt die empfangenen Pakete, bis die Prüfdauer vorbei ist. Diese Zahl wird danach der Quelle kommuniziert. Basierend auf dieser Rückmeldung fällt die Quelle die Zugangskontrollentscheidung. Liegt der berechnete Anteil der verlorenen (oder markierten) Pakete unter einem Schwellenwert, wird der Fluss akzeptiert, und die Daten können gesendet werden.

Um diesen Mechanismus etwas genauer zu betrachten, verwenden wir ein einfaches Modell mit einem überlasteten Link mit Kapazität C. Jeder Fluss i sendet mit einer fixen Rate r_i. Die resultierende Verlustrate ist $\frac{\sum_i r_i - C}{\sum_i r_i}$. Damit ein Fluss gewährt wird, muss die Quelle für die Dauer der Prüfphase mit der Rate r_i Pakete senden. Das Ziel misst die resultierende Verlustrate und der Fluss wird akzeptiert, wenn sie unter einem Schwellenwert ϵ_i liegt. Sei z. B. die Kapazität des Links 10 Mbit/s und drei Flüsse mit den Raten 2, 3 und 8 Mbit/s aktiv. Die Verlustrate beträgt dann $\frac{3}{13}$. Für $\epsilon_1 = 0.1$ würde also der erste Fluss zurückgewiesen werden. Da alle beteiligten Flüsse mit konstanter Rate senden und kein zusätzlicher Verkehr hinzukommt, ist die Verlustrate in diesem Beispiel unabhängig von der Prüfdauer.

3.3 Probleme von EAC

Ist die Anzahl der prüfenden Flüsse hoch und liegt die Anzahl der akzeptierten Flüsse weit unter dem Wert, den der Link verarbeiten könnte, spricht man von Thrashing. Abbildung 3.2 illustriert diesen Zustand mit in-band Dropping. Die Flüsse kommen in einem Poisson-Prozess an, und es werden exponentielle Fluss-Lebenszeiten und exponentielle Prüfzeiten verwendet. Wenn man die Kurve in Abbildung 3.2(a) betrachtet, sieht man, dass die Verlustwahrscheinlichkeit mit der Länge der Prüfzeit zu Beginn nur sehr langsam zunimmt. Es ist aber eine scharfe Grenze zu erkennen, wo die Kurve steil ansteigt. Vor dieser Grenze ist die Verlustwahrscheinlichkeit recht gering. Dahinter strebt die Verlustwahrscheinlichkeit gegen 1. Das System kollabiert. Thrashing erhöht nicht nur die Verlustwahrscheinlichkeit, sondern verhindert auch, dass neu angekommene Flüsse akzeptiert werden. Dies obwohl die Auslastung fast null ist (siehe Abbildung 3.2(b)).

EAC-Algorithmen sollten so entworfen werden, dass sie Thrashing minimieren. Das in [8] vorgeschlagene Slow-Start-Verfahren wirkt dem Thrashing entgegen, indem die Prüfrate langsam erhöht wird, um Überlastung zu erkennen. Die Prüfrate wird dabei zu Beginn für eine bestimmte Zeit sehr klein gehalten. Ist die beobachtete Verlustrate kleiner als der Schwellenwert ε, wird die Prüfrate verdoppelt. Dieser Prozess wird solange wiederholt, bis die gewünschte Übertragungsrate erreicht ist. Der Sender erhält dadurch relativ früh ein Signal für Überlastung.

Ein weiteres Problem tritt auf, wenn das Prüfen in-band erfolgt und der Prüfverkehr Bandbreite braucht, die vorgängig kontrolliertem Verkehr zugeteilt wurde. Der Prüfverkehr stiehlt also Bandbreite, die ihm nicht zusteht. Dieses Problem ist daher unter dem Begriff *Bandwidth Stealing* bekannt. Messungen, die in Gegenwart von Bandwidth Stealing gemacht werden, widerspiegeln die Fähigkeit des Netzwerks, neue Flüsse unterstützen zu können, sehr schlecht. Die daraus folgenden ungenauen Messungen resultieren in übermässig viel gewährten Flüssen und in einer verminderten Dienstqualität für akzeptierte Flüsse.

3.4 Queueing und EAC

Damit eine Zugangskontrolle zwischen Endpunkten möglich ist, braucht es Unterstützung der dazwischenliegenden Netzwerkknoten. Router können diese Unterstützung durch die Verwendung einer der in Abschnitt 2.3 beschriebenen Queueing-Mechanismen sicherstellen. Nicht alle Mechanismen sind aber für EAC gleich gut geeignet. Priority Queueing ist hilfreich, um einem Fluss oder einer Dienstklasse absolute Priorität gegenüber dem anderen Verkehr zu geben. Wenn man ein DiffServ-Netzwerk mit integrierter EAC aufsetzt, könnte dieser Algorithmus sehr nützlich sein. Kontrollierter Verkehr könnte damit von den Routern priorisiert behandelt werden. Damit der vorhandene Best-Effort-Verkehr nicht vollständig verdrängt wird, ist es aber notwendig, dass

der kontrollierte Verkehr eine obere Grenze der verfügbaren Bandbreite erhält. Dies könnte in einer Priority Queue leicht mit einem Ratenbegrenzer (Rate Limiter) realisiert werden.

Scheduling-Mechanismen wie Fair Queueing, die eine Max-Min Fair Share Allokation durchführen, sollten für den EAC-kontrollierten Verkehr nicht verwendet werden. Da in diesen Mechanismen Flüsse voneinander isoliert werden, kann es nämlich zu Situationen kommen, wo kleinere Flüsse gewährt werden, obwohl deren Akzeptanz verhindert, dass der Service schon akzeptierten grösseren Flüssen zur Verfügung gestellt wird [8]. Zur Veranschaulichung dient wiederum das Modell mit dem überlasteten Link mit Kapazität C, diesmal existieren aber zwei Gruppen von Flüssen: die eine Gruppe sendet mit der Rate r_1, die andere mit der Rate r_2 (r_2 sei grösser als r_1). n_1 und n_2 sind die entsprechende Anzahl Flüsse, die im Moment aktiv (oder am Prüfen) sind. Wegen der Max-Min Fair Share Allokation wird im Modell ein Fluss in der ersten Gruppe so lange gewährt, wie $r_1(n_1 + n_2) < C$. Ein Fluss in der zweiten Gruppe wird so lange gewährt, wie $n_1 r_1 + n_2 r_2 < C$. Treffen kurze Zeit darauf ein paar Flüsse der ersten Gruppe ein, werden diese neu angekommenen Flüsse gewährt bis $r_1 \cdot (n_1 + n_2) = C$. Zu diesem Zeitpunkt geht die Verlustrate für die erste Gruppe gegen 0, während die der zweiten Gruppe $\frac{r_2 - r_1}{r_2}$ beträgt. Ist z. B. r_1 halb so gross wie r_2 nimmt die Verlustrate den unakzeptablen Wert 0.5 an. In diesen Situationen erhalten grosse Flüsse hohe Verluste, obwohl das Netzwerk in der Prüfphase nicht überlastet war.

RED ist sehr brauchbar für TCP, da es den Durchsatz in Stausituationen maximieren kann. Weiter kann RED die Bandbreite zwischen TCP-Datenströmen auf faire Weise aufteilen, da verlorene Pakete automatisch eine Reduktion der Rate eines Datenflusses hervorrufen. Datenströme, die nicht den Überlast-Verhinderungsmechanismen von TCP folgen, wie UDP-basierte Echtzeit- oder Multicast-Applikationen, können von RED nicht profitieren, da sie nicht auf Paketverlust reagieren. Werden jedoch EAC-Mechanismen verwendet, die auf Paketverlust basieren, präsentiert sich die Situation etwas anders. Die implizite Benachrichtigung durch Paketverlust wird von einem EAC-fähigen Endsystem erkannt und beeinflusst die Entscheidung, ob ein Fluss akzeptiert wird oder nicht. In Stausituationen werden prüfende Flüsse durch die Verwendung von RED früher zurückgewiesen. Dies erhöht die Dienstgüte der bereits akzeptierten Flüsse.

Auch ECN wurde primär für TCP entwickelt, kann aber auch in Verwendung mit UDP eingesetzt werden. Bei EAC-Mechanismen, die auf Paketmarkierungen basieren, ist ECN integraler Bestandteil und wird vorausgesetzt. Der in Abschnitt 3.5.3 genauer beschriebene Ansatz benutzt z. B. den *Virtual-Queue-Algorithmus*, um zu entscheiden, ob Pakete in einer Queue markiert werden. Die Markierung wird dann durch das Setzen des CE-Bits realisiert.

3.5 Spezifische Konzepte für EAC

Dieser Abschnitt liefert einen Überblick über existierende EAC-Konzepte, die nach der oben definierten Architektur aufgebaut sind. Die einzelnen Ansätze werden jeweils kurz vorgestellt und spezielle Themen herausgegriffen. Abschnitt 3.5.5 stellt speziellere Konzepte vor, die sich z. T. stark von der grundlegenden EAC-Architektur unterscheiden.

3.5.1 Admission Control Based on End-to-End Measurements

Elek, Karlsson et al. schlagen in [12] eine Zugangskontrolle vor, die auf Paketverlust basiert. Der beschriebene Service zur Lastkontrolle benötigt eine eigene Partition in der Link-Kapazität des Netzwerks. Die unbenutzte Kapazität kann vollumfänglich vom Best-Effort-Verkehr genutzt werden. Innerhalb der Service-Klasse wird zwischen zwei Prioritäten unterschieden. Das Prüfen erfolgt also out-of-band. Um den Weiterleitungsmechanismus zu erreichen, wird ein System mit zwei Priority Queues vorgeschlagen; Eine hochpriorisierte Queue für den akzeptierten Verkehr und eine für die Prüfpakete. Die Queue für die Prüfpakete kann nur wenige Pakete zwischenspeichern, um die Verzögerung minimal zu halten. Sie wird nur dann bedient, wenn die hochpriorisierte Queue leer ist.

Die Zugangskontrollprozedur erfolgt im Prinzip so wie in Abschnitt 3.2 beschrieben. Anstelle von Flüssen wird von sogenannten Sessions gesprochen, die bei der Zugangskontrollentscheidung akzeptiert oder zurückgewiesen werden. Eine Session beginnt mit der Prüfphase, in der der Sender das Netzwerk während einer selbst gewählten Zeit prüft. Jedes Prüfpaket enthält die Spezifikation der Prüfdauer, die Übertragungsrate und einen Identifikator für die aktuelle Session. Ist die Prüfdauer abgelaufen, wird ein Messprotokoll zum Sender geschickt, das die Anzahl erhaltener Prüfpakete beinhaltet. Dieses Messprotokoll wird mit einer hohen Priorität versehen, damit es mit einer kleinen Verlustwahrscheinlichkeit übertragen wird.

Aufgrund des Messprotokolls entscheidet der Sender über die Gewährung. Liegt die berechnete Verlustwahrscheinlichkeit unter einem Schwellenwert, kann die Quelle die Daten senden. Nach einer Zurückweisung erfolgt nicht sofort eine neue Anforderung, sondern es wird eine zufällig gewählte Zeit gewartet. Diese Zeit wird aus einer uniformen Verteilung berechnet und wird bei jeder Zurückweisung mit gleichem Empfänger verdoppelt.

Dieser Ansatz unterscheidet sich von den anderen Ansätzen in diesem Abschnitt hauptsächlich in der Verwendung eines fehlerkorrigierenden Codes für den kontrollierten Verkehr. Der Code (z. B. Forward-error Correction) wird benutzt, um Datenpaketverluste wiederherstellen zu können. Wird ein Fluss gewährt, sendet der Empfänger der Quelle periodisch eine Rückmeldung mit der aktuellen Verlustwahrscheinlichkeit, um den fehlerkorrigierenden Code zu verbessern. Dieser Mechanismus bringt die Anzahl nicht korrigierbarer Paketverluste auf einen tolerierbaren Wert.

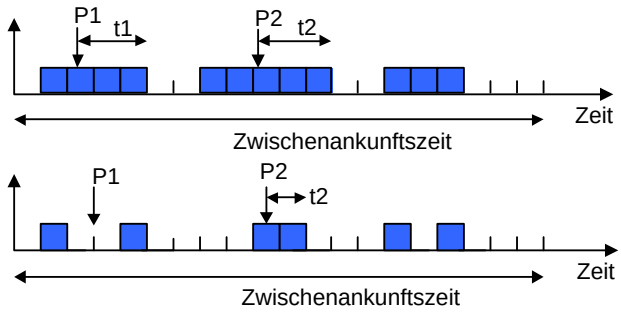


Abbildung 3.3: Graphische Veranschaulichung des Effekts eines kurzen PLT-Wertes

3.5.2 End-to-End Measurement-Based Admission Control

Bianchi, Capone et al. führen in [13] ein analytisches Modell ein, um die Leistungsfähigkeit von EAC-Mechanismen zu evaluieren. Gleichzeitig wird aber auch ein Verfahren vorgestellt, das auf Paketverlust basiert und Prüfpakete einer geringeren Prioritätsklasse als Datenpakete zuordnet. Die Zugangskontrollprozedur erfolgt wie in Abschnitt 3.2 beschrieben.

Speziell an diesem Ansatz ist die Verwendung eines Limits für die Lebenszeit eines Prüfpakets (Probing Packet Lifetime PLT), um die Überlastung von Prüfpaketen zu kontrollieren. Ein Prüfpaket wird verworfen, falls es nicht innerhalb einer bestimmten Zeit seit seiner Ankunft in der Queue weitergeleitet wird. Um zu verstehen, warum die PLT ein wichtiger Parameter ist, ist eine nähere Betrachtung der Abläufe auf Paketebene notwendig. Abbildung 3.3 stellt zwei Szenarien für die Belegung eines Links mit Datenpaketen graphisch dar. In diesem Beispiel wird als Zwischenankunftszeit die verstrichene Zeit zwischen zwei Paketen des selben Datenstroms bezeichnet. Es wird angenommen, dass eine Zwischenankunftszeit in 20 Schlitze unterteilt ist, was einer Link-Kapazität von 20 aktiven Verbindungen entspricht. Im ersten Szenario liegt die Linkbelegung bei 60% (12 aktive Verbindungen), im zweiten bei 30% (6 aktive Verbindungen). Wir betrachten nun zwei Prüfphasen, P1 und P2, welche zu verschiedenen Zeitpunkten gestartet werden. Ist die PLT grösser oder gleich der Zwischenankunftszeit, erhalten die Prüfpakete auf jeden Fall einen freien Schlitz für die Übertragung, da in beiden Szenarien der Link nicht überlastet ist. Ist die PLT also gross genug, ist es nicht möglich zu erkennen, wie viel Last dem Link zur Verfügung gestellt wird. Es kann nur erkannt werden, ob genug freie Kapazität verfügbar ist. Wird der Wert für die PLT viel kleiner als die Zwischenankunftszeit gewählt, kann die Lebenszeit der Prüfpakete ablaufen, bevor ein leerer Schlitz verfügbar wird. Wie man in der Abbildung sehen kann, ist der Zeitpunkt, an dem das Prüfpaket bedient wird, stark von der Last des Datenverkehrs abhängig. Die Wahrscheinlichkeit, dass ein Paket wegen dem Ablauf der PLT verloren geht, wächst somit mit der Last des Datenverkehrs. Die Konsequenz für die Zugangskontrolle ist nun der,

dass je kleiner der Wert für die PLT in jedem Netzwerknoten gehalten wird, desto früher kann Stau vom Messprozess an den Endpunkten erkannt werden. Zusammenfassend kann man sagen, dass die passende Einstellung der PLT-Werte eine gute Kontrolle über den Netzverkehr ermöglicht.

3.5.3 Distributed Admission Control

Kelly, Key et. al beschreiben in [17] ein sehr allgemeines Framework, in dem Zugangskontrollentscheidungen in Randpunkten oder Endsystemen gefällt werden. Im Internet, als Beispielanwendung dieses Frameworks, wird diese Information mit einem ECN-Bit (siehe Abschnitt 2.3.4.3) angezeigt. Zugangskontrollentscheidungen werden nach Ablauf der Prüfphase nun aufgrund der Anzahl markierter und verlorener Pakete gemacht. Wie genau die Entscheidung gefällt wird, ist nicht spezifiziert. Eine Möglichkeit wäre beispielsweise einen Fluss zu akzeptieren, wenn keine Sequenz von m Prüfpaketen markiert wurde.

Damit Pakete in den Routern markiert werden können, müssen sie natürlich entsprechende Funktionalität besitzen. Eine einfache Markierungsstrategie ist, ähnlich wie es RED (siehe Abschnitt 2.3.4.1) macht, alle Pakete zu markieren, wenn die Queue einen bestimmten Schwellenwert übersteigt. Diese Strategie basiert auf der Länge der Queue und hat den Nachteil, dass eine Reaktion auf Datenstau recht langsam erfolgt. Um dies zu umgehen, schlägt der Ansatz eine Markierungs-Strategie mit einer virtuellen Queue (Virtual Queue) vor. Diese Queue funktioniert genau gleich wie die reale Queue, hat aber nur einen Bruchteil der Service-Rate und der Pufferspeicher-Kapazität der realen Queue. Der Markierungsmechanismus läuft dann wie folgt ab: Wenn die virtuelle Queue ein Paket verliert, werden alle ausgehenden Pakete markiert bis sie wieder leer ist. Zu Beginn werden also die Pakete markiert, die verloren gegangen wären. Danach werden solange Pakete markiert, bis die Überlastungssituation vorüber ist. Folgender Pseudocode beschreibt den Algorithmus:

```
Für jede Ankunft eines Pakets
   Virtueller Byte-Zähler -= Bandbreite * (aktuelle Zeit -
   Zeit der letzten Paketankunft)
   IF Virtueller Byte-Zähler kleiner als 0
        Setze virtueller Byte-Zähler auf 0 zurück
        Setze Markierungs-Flag zurück
   IF Summe des virtuellen Byte-Zählers und
   Paketgrösse kleiner als virtueller Pufferspeicher
        Setze Markierungs-Flag
   ELSE
        Erhöhe virtuellen Byte-Zähler um Paketgrösse
   IF Markierungs-Flag gesetzt
        Setze ECN-Bit im IP-Header
```

Anwendungen auf der Empfängerseite, die das ECN-Bit in die Verarbeitung der Pakete mit einbeziehen, erhalten früh ein Signal für Stau im Netzwerk. Der Beginn einer Überlast kann zu einem gewissen Grad vorausgesehen werden.

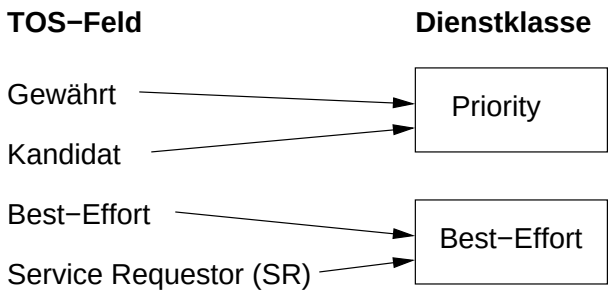


Abbildung 3.4: Abbildung von TOS-Markierungen auf QoS-Klassen

Dieser Anatz lässt sich mit dem in [19] beschriebenen nutzungsbezogenem Zahlungssystem kombinieren. Flüsse können dann so viel Verkehr senden, wie sie wollen. Dies ist möglich, da der Sender für jedes markierte Paket einen fixen Preis zahlen muss. Die Prüfphase könnte somit einem weiteren Zweck dienen. Nach der Prüfphase kann nämlich abgeschätzt werden, wie viel die Verbindung kosten wird. In dieser Situation ist die Zugangskontrolle ein Dienst, der durch Dritte (oder das Netzwerk selbst) angeboten wird, um für den Fluss einen garantierten Preis bereitzustellen.

3.5.4 DiffServ Enhanced Admission Control Scheme

Im Entwurf von Hill und Kung (siehe [20]) werden Eigenschaften aus EAC und der DiffServ-Architektur kombiniert, um individuellen Flüssen harte Garantien zusichern zu können. Es unterscheidet sich von den in den vorangegangenen Abschnitten beschriebenen Ansätzen darin, dass die Router die Grösse des gleichzeitigen Prüfverkehrs kontrollieren können. Damit wird das in Abschnitt 3.3 beschriebene Problem des Bandwidth Stealing gelöst.

3.5.4.1 Architektur

Der Ansatz hat zwei Komponenten, ein Reservierungsprotokoll und einen Zugangskontrollalgorithmus. Um den Nachrichten-Overhead zu minimieren, wird eine erweiterte DiffServ-Architektur vorgeschlagen, die es ermöglicht, den Status einer Entscheidung eines Flusses via einer Markierung im Paket zu kommunizieren. Die Markierung wird im TOS-Feld des Paket-Headers gespeichert und ermöglicht den Routern, zwischen gewährtem und nicht-gewährtem Verkehr zu unterscheiden.

Mögliche Markierungen, die ein Fluss erhalten kann, sind Best-Effort, *Service Requestor* (SR), Kanditat für Gewährung und gewährt. Gewährte Flüsse und Kandidaten erhalten hohe Dienstqualität, alle anderen Best-Effort-Service (siehe Abbildung 3.4). Pakete der ersten Klasse erhalten in der Queue der Router strikte Priorität gegenüber der zweiten Klasse. Flüsse, deren Pakete als gewährt markiert sind, wurden von allen Routern auf dem Pfad zwischen Sender

und Empfänger erfolgreich zugelassen, und es erfolgt eine implizite Reservierung. Die Reservierung ist implizit, da keine explizite Signalisierung notwendig ist, um die Bandbreite im Kern zu reservieren.

Da die Pakete von Kandidaten-Flüssen dieselbe Priorität erhalten wie die Pakete von gewährten Flüssen, erfolgt das Prüfen in-band. Um die Garantien, die gewährten Flüssen zugesichert wurden zu schützen, limitiert jeder Router die Menge an gleichzeitigem Kandidaten-Verkehr. Dazu verwalten die Router eine Liste, in der ein Identifikator für jeden als Kandidat ausgewählten Fluss steht. Mit diesem Mechanismus kann zwar sehr schön die Grösse des Prüfverkehrs kontrolliert werden, durch den zusätzlichen Verwaltungsaufwand entstehen aber auch Skalierungsprobleme. Diese halten sich jedoch im Rahmen, da die in den Routern zu speichernden Informationen nur von der Anzahl gleichzeitiger Kandidaten-Flüsse abhängig ist und nicht von der Anzahl gewährter Flüsse.

3.5.4.2 Zugangskontrollprozedur

Die Zugangskontrollprozedur läuft in drei Stufen ab. Eine Anforderung beginnt mit einem als Service Requestor markierten Fluss. Wenn alle Router entlang des Pfades verfügbare Ressourcen besitzen, wird der Fluss von den Routern als Kandidat für eine Gewährung ausgewählt. Während einer festgelegten Beobachtungsperiode überwacht nun der Egress-Router den Durchsatz der Kandidaten-Flüsse und sendet nach Ablauf der Beobachtungszeit den gemessenen Durchsatz dem sendenden Knoten. Ist die Menge der verfügbaren Bandbreite grösser oder gleich der Prüfrate, wird der Fluss akzeptiert.

3.5.5 Weitere Ansätze

Es gibt verschiedene speziellere Ansätze, die sich z. T. stark von den bis anhin vorgestellten Konzepten unterscheiden. Ein solcher Ansatz wird in [15, 16] beschrieben. Der Endpunkt ist hier kein Host, sondern ein Egress-Router. Während Hosts das Netzwerk prüfen müssen, um Überlastung zu bemerken, können Egress-Router Pfade passiv überwachen und dabei die vorhandene Last feststellen. Passives Prüfen kann eine genauere Schätzung über die vorhandene Last machen und hat zusätzlich den Vorteil, dass die Verzögerung, die durch die Prüfphase entsteht, wegfällt. Weder im Innern des Netzwerks noch an den Randknoten werden dabei die Zustände der Flüsse gespeichert. Die Zugangskontrolle und die Verwaltung der Ressourcen werden nur in Egress-Routern durchgeführt.

Ein Konzept, das ebenso auf ein aktives Prüfen des Netzwerks verzichtet, wird in [21] beschrieben. Es wird eine netzwerkbasierte Zugangskontrolle vorgeschlagen, die keine Wartezeit für das Set-Up eines Flusses aufweist und eine hohe Zugangswahrscheinlichkeit (und damit hohe Auslastung) hat. Dies wird erreicht, indem verfügbare Ressourcen mit sogenannten *Sink-Trees* offline strukturiert werden. Die Wurzel eines Sink-Trees ist der Egress-Router

und die Blätter sind die Ingress-Router. Im Baum sind Informationen über die Kapazität und die allozierte Bandbreite jedes Links auf dem Pfaden zu den Ingress-Routern gespeichert. Da jeder Egress-Router über einen eigenen Sink-Tree verfügt, hat jedes mögliche Paar von Quell- und Zielknoten einen eindeutigen Pfad im Sink-Tree. Wenn ein Fluss nun an einem Ingress-Router ankommt, kann mit Hilfe des passenden Sink-Trees die verfügbaren Ressourcen auf dem Pfad automatisch ermittelt werden. Es ist dann möglich, in jedem Knoten, in dem ein Fluss ankommt, eine Zugangsentscheidung zu fällen.

In [14] werden statt Paketverlust Verzögerungsschwankungen als Indikation für Überlastung benutzt. Im Entwurf erfolgt die Zugangskontrollprozedur wie üblich mit einer Prüfphase, in der verifiziert wird, ob genügend Ressourcen für eine neue Verbindung im Netzwerk vorhanden sind. Die Entscheidung, ob ein Aufruf akzeptiert oder zurückgewiesen wird, wird am Zielknoten auf Basis der gemessenen Verzögerungsschwankungen gefällt. Die Leistungsevaluation dieses Ansatzes zeigt, dass Dienstgüteanforderungen bezüglich der Verzögerung im Bereich von wenigen Millisekunden garantiert werden können. Weiter wird belegt, dass Verzögerungsschwankungen von Prüfpaketen ein effektiver Mechanismus ist, um die Überlastung im Netzwerk zu messen.

4 Der Netzwerksimulator ns-2

Dieses Kapitel gibt eine kurze Einführung in die Netzwerksimulations-Software, die für das Simulations-Framework verwendet wurde. Dies beinhaltet einen Überblick über die Fähigkeiten von ns-2 und eine Beschreibung von spezifischen Features, die für die Implementierung des Frameworks relevant waren. Letzteres enthält Aspekte wie Protokolle, Pakete und die Simulation von Endsystemen.

4.1 Wahl der Simulationssoftware

Die Anzahl der verfügbaren Netzwerksimulatoren ist heutzutage recht gross. Die wenigen guten Simulatoren, die auch in der Forschungsgemeinschaft einen guten Ruf geniessen, sind meist sehr teuer (z. B. Opnet [23]). Im Gegensatz dazu ist ns-2 frei verfügbar und ist seit einigen Jahren der De-facto-Standard für Netzwerksimulatoren für akademische Forschung. Die Software, die vom Information Sciences Institute (ISI) der University of Southern California (USC) als Teil des Virtual Internet Testbed (VINT) entwickelt wurde, wird seit 1995 ständig erweitert. Eine grosse Anzahl von Netzwerkkomponenten sind für ns-2 verfügbar. Es werden u.a die folgenden Technologien unterstützt:

- Punkt-zu-Punkt-Verbindungen
- Verschiedene Queueing-Mechanismen (Drop Tail, RED, CBQ, etc.)
- IP
- Multicast
- TCP, UDP und verschiedene experimentelle Transportprotokolle
- Applikationen und Verkehrsgeneratoren (Telnet, FTP, CBR, etc.)

4.2 Architektur von ns-2

Der Netzwerksimulator ns-2 ist ein ereignisbasierter Simulator für Computernetze und Netzwerkprotokolle. Er besteht aus den drei Hauptkomponenten Event-Scheduler, Netzwerkkomponenten-Bibliothek und Netzwerksetup-Bibliothek (siehe Abbildung 4.1). Der Event-Scheduler verwaltet Ereignisse

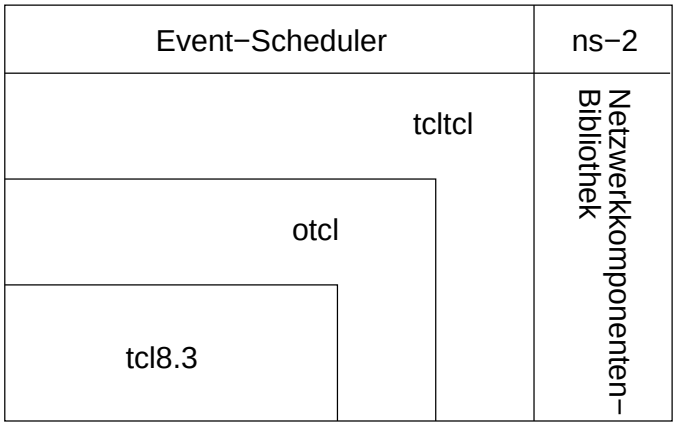


Abbildung 4.1: Hauptkomponten von ns-2

wie das Starten und Stoppen eines Verkehrsflusses oder das Speichern von Messwerten. Der Kern des Simulators ist die in C++ geschriebene Netzwerkkomponenten-Bibliothek. Sie enthält die Komponenten, die notwendig sind, um ein Netzwerk mit einem bestimmten Szenario simulieren zu können. Die Netzwerksetup-Bibliothek dient als Frontend und enthält einen Script-Interpreter, der es dem Benutzer ermöglicht, die erstellten Simulationen schnell anzupassen.

Neben der kompilierten C++-Klassenhierarchie besteht eine ähnliche Klassenhierarchie innerhalb des Interpreters. Die beiden Hierarchien haben eine enge Beziehung zueinander. Aus der Sicht des Benutzers besteht eine Eins-zu-eins-Relation zwischen einem Objekt in der kompilierten Hierarchie und einem in der interpretierten Hierarchie. Diese Dualität nutzt einerseits die Flexibilitätsvorteile einer Interpretersprache und andererseits die Effizienz von kompiliertem Code. Szenarien können mit der OTcl-Sprache, einer objekt-orientierten Version von Tcl, schnell erstellt und parametrisiert werden, wobei dort, wo Ausführungsgeschwindigkeit wichtig ist, C++ verwendet wird.

Topologien werden aus Knoten, Queues und Links zwischen den Knoten aufgebaut. Letztere simulieren physikalische Verbindungen und Netzwerk-Interfaces und definieren Netzwerkbedingungen wie Bandbreite, Verzögerung und Queue-Länge.

4.3 Pakete und Protokolle

Es ist wichtig, zwischen Paketen, die ns-2 simulieren muss und deren internen Respräsentation im Simulator zu unterscheiden. Der Unterschied kann leicht veranschaulicht werden, wenn man ein Paket auf der Netzwerkebene betrachtet. So enthält beispielsweise ein Paket für eine Audio-Übertragung in Wirklichkeit einen IP-Header gefolgt von UDP-und RTP-Header und Nutzlast. Diese vier Teile findet man auch in einem Paket im Simulator, es enthält

aber noch *alle* anderen Paket- oder Protokoll-Header, die ns-2 unterstützt, wie z. B. TCP, Ping, MAC usw. Einer davon ist der allgemeine Header (Common-Header), der wichtige Informationen wie die Paketgrösse (im Beispiel: Länge der IP-, UDP- und RTP-Header plus Länge der Nutzlast), Pakettyp und einen Zeitstempel enthält. Der Zeitstempel wird zur Messung der Verzögerung innerhalb von Routern verwendet.

Generell kann man sagen, dass die Paket-Header von ns-2 für ein spezifisches Protokoll nicht unbedingt mit den in den RFCs definierten Protokoll-Headern übereinstimmen. Beispielsweise enthält der IPv4-Header von ns-2 die Zieladresse, die Quelladresse und ein TTL-Feld. Das Übrige (wie z. B. TOS, Header-Prüfsumme, Optionen usw.) wurde weggelassen. Da das für EAC sehr wichtige TOS-Feld fehlt, mussten im Framework die Informationen über die Dienstklassen an anderer Stelle gespeichert werden. Das Framework nutzt dazu verschiedene Pakettypen, die in Abschnitt 5.3.1 genauer beschrieben werden.

4.4 Das Konzept der Agenten

Agenten sind in ns-2 Endsysteme, wo Netzwerkschicht-Pakete konstruiert und konsumiert werden. Sie dienen der Implementierung von Protokollen auf verschiedenen Schichten. In ns-2 sind zahlreiche Agenten für verschiedene Protokolle vorhanden. TCP steht z. B. mit mehreren Varianten wie Tahoe, Reno oder Vegas zur Verfügung. Der Datenfluss erfolgt in ns-2 immer von einem Agenten zu einer (unicast) oder mehreren (multicast) Senken. Der Verkehr wird durch einen Verkehrsgenerator oder durch eine simulierte Applikation erzeugt (siehe Abb. 4.2). Diese Objekte werden einem Agenten angehängt. Generierter Verkehr umfasst vier Typen:

- Exponentieller Verkehr mit On- und Off-Perioden.
- Pareto-verteilter Verkehr mit On- und Off-Perioden.
- Verkehr mit konstanter Bitrate (CBR).
- Traffic-Trace. Der Verkehr wird dabei aus einer Trace-Datei gelesen. Damit ist es beispielsweise möglich, den Verkehr einer realen Videoübertragung zu simulieren.

Als simulierte Applikationen stehen Telnet und FTP zur Verfügung.

Ist in einer Simulation der Verkehr einmal gestartet, fliesst er vom Knoten, an dem der Verkehr angehängt ist, zur korrespondierenden Senke. Dazwischen passieren Pakete unter Umständen Knoten, wo sie zwischengespeichert werden und nach einer vorgegeben Verzögerung weitergeleitet werden. Treffen die Pakete an der Senke ein, ist es abhängig vom verwendeten Protokoll, ob ein Verarbeitungsschritt erfolgt oder ob die Pakete einfach vernichtet werden.

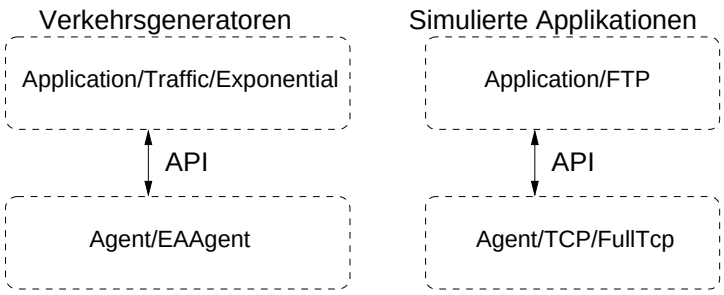


Abbildung 4.2: Beispiel für die Verknüpfung von Applikationen und Agenten

Letzteres ist z. B. bei UDP der Fall. Für das Simulations-Framework wurde ein neuer Agenttyp basierend auf dem UDP-Agenten entwickelt. Er wird in Abschnitt 5.3.1 beschrieben.

4.5 Queue-Management und Paket-Scheduling

Queues repräsentieren in ns-2 Einheiten, in denen Pakete gehalten oder verworfen werden. Das Paket-Scheduling bezieht sich auf den Entscheidungsprozess, der verwendet wird, um zu wählen, welche Pakete bedient oder verworfen werden sollen. Zur Zeit bietet ns-2 Unterstützung für die in Abschnitt 2.3 beschriebenen Queueing-Mechanismen Drop-Tail, RED, CBQ (einschließlich Priorität- und Round-Robin-Scheduler) und die Varianten des Fair Queueing einschliesslich Stochastic Fair Queueing (SFQ) und Deficit Round-Robin (DRR).

Eine wichtige Funktionalität einer Queue in ns-2 ist die Realisierung der Signalverzögerung, die in einem realen Netzwerk zwischen zwei Netzwerkknoten entsteht. Diese Verzögerung wird dadurch erreicht, dass eine Queue vom sendenden Nachbarn blockiert und reaktiviert wird. Paketverlust ist so implementiert, dass jede Queue ein "Verlustziel" (Drop Target) hat, also ein Objekt, das alle verworfenen Pakete empfängt. Dies kann nützlich sein, um z. B. Statistiken über verworfene Pakete zu führen.

Während der Implementierung des Frameworks wurden verschiedene Queues dem Simulator hinzugefügt. Sie implementieren die von ns-2 zur Verfügung gestellte Queue-Schnittstelle, welche die grundlegenden Funktionen einer Queue wie das Einfügen und Herausnehmen eines Pakets abstrahiert.

5 Implementierung des Frameworks in ns-2

Dieses Kapitel beschreibt das Design des Simulations-Frameworks für Endpoint Admission Control, dessen Einbettung in die ns-2-Architektur und die Implementierung in C++. Es werden einerseits die inneren Abläufe einer EAC-Simulation beschrieben und andererseits die Schnittstelle zum Benutzer dokumentiert. Da diese Schnittstelle recht komplex ist und Programmierkenntnisse voraussetzt, hat dieses Kapitel auch Tutorium-Charakter und enthält Code- sowie Anwendungsbeispiele.

Das Framework realisiert die EAC-Architektur, wie sie in Abschnitt 3.1 beschrieben wurde und stellt somit die grundlegende Funktionalität zur Verfügung, um mit den Ansätzen aus Kapitel 3 experimentieren zu können.

5.1 Grundlegendes Design

Das Framework wurde in die Version 2.1 Beta 8 von ns-2 integriert. Diese Version von ns-2 enthält über 30 verschiedene Pakete (Packages). Das UML-Paketdiagramm in Abbildung 5.1 stellt die wichtigsten Pakete dar. Das Paket *common* stellt Schnittstellen und Klassen zur Verfügung, die von allen anderen Paketen genutzt werden. Dazu gehören u.a. die Klassen *Node*, *Agent* und *Packet*, welche die Basisfunktionalität des Simulators enthalten.

Das Simulations-Framework wurde als Paket *eac* in ns-2 integriert und enthält zwei Teilpakete. Das *Probing-Modul* beinhaltet mehrere C++-Klassen und erweitert ns-2 um EAC-Funktionalität für UDP-basierten Verkehr. Es generiert den Prüfverkehr und trifft die Zugangskontrollentscheidung abhängig vom verwendeten Zugangskontrollalgorithmus. Zusätzlich stellt das Modul Schnittstellen zur Verfügung, um neue EAC-Konzepte schnell und einfach zu implementieren.

Das *Queue-Modul* übernimmt das Weiterleiten der Pakete in den Knoten innerhalb des Netzwerks und das Trennen von Best-Effort-Verkehr und kontrolliertem Verkehr. Es besteht hauptsächlich aus Queue-Implementierungen, die gewisse Pakete mit höherer Priorität weiterleiten oder Pakete markieren. Ein weiterer Bestandteil dieses Moduls ist ein Ratenbegrenzer, der das in Abschnitt 3.4 erwähnte Bandbreitenlimit für kontrollierten Verkehr realisiert.

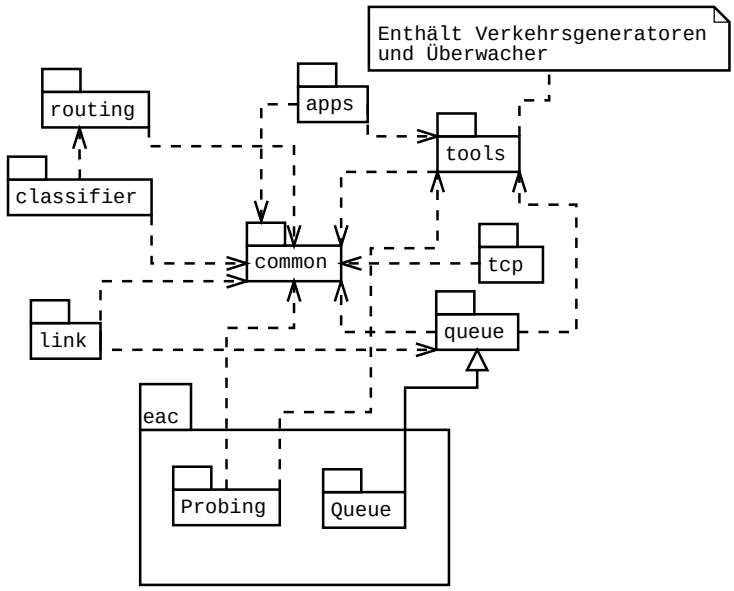


Abbildung 5.1: UML-Paketdiagramm mit den wichtigsten Paketen von ns-2 und dem *eac*-Paket

Als Frontend dient der OTcl-Interpreter von ns-2. Mit Hilfe von OTcl-Scripten kann der Benutzer seine Simulationen aufsetzen, konfigurieren und schnell abändern.

5.2 Modifikationen

5.2.1 Änderungen an existierenden Dateien

Das Framework konnte so implementiert werden, dass sehr wenig Änderungen in bereits existierendem Code gemacht werden mussten. Damit ist es möglich, das Framework auch in anderen Versionen als Version 2.1 Beta 8 benutzen zu können. Für gewisse Funktionalität war es aber unumgänglich, bestehende Klassen ein wenig anzupassen oder zu erweitern. Eine Liste der geänderten Dateien können in Anhang A.1 gefunden werden.

5.2.2 Neue Dateien

Neue Funktionalität wurde ns-2 konsequent mit C++-Klassen hinzugefügt. Jede Klasse befindet sich in einer separaten Header- und Quelltext-Datei im Unterverzeichnis */eac*. Zusätzlich wurden die Dateien gk.h und gk.cc aus der Version 2.1 Beta 9 ins Wurzel-Verzeichnis von ns-2 kopiert. Sie enthalten eine Implementierung einer virtuellen Queue nach Gibbens und Kelly [18]. Eine Liste mit den wichtigsten hinzugefügten Dateien findet sich in Anhang A.2.

5.3 Das Probing-Modul

Die Klassen und Funktionen in diesem Abschnitt können in ~ns/eac/Probing-Strategy, SlowStart, SimpleStrategy, EarlyReject, EAAgent, EASink {h, cc} gefunden werden.

5.3.1 Implementierung von EAC-fähigen Endsystemen

Um ein EAC-fähiges Endsystem simulieren zu können, wurden zwei neue Agenten hinzugefügt. Die Klasse *EAAgent* repräsentiert einen Endpunkt in einem Netzwerk, der als Sender agiert. Die Klasse *EASink* ist das Gegenstück auf der Empfängerseite. Ein EAAgent ist dafür verantwortlich, Prüf- oder Datenpakete zu erstellen und weiterzuleiten. Die Senke empfängt Prüf- und Datenpakete und speichert die Anzahl empfangener und verworfener (oder markierter) Prüfpakete. Die bestehende Klasse *Agent* wird für beide Klassen als Basisklasse verwendet.

Da EAC v.a. zur Übertragung von Echtzeitdaten wichtig ist und somit auf UDP basiert, erweitern beide Agenten die bestehende Implementierung eines UDP-Agenten. Wenn TCP-Simulationen gemacht werden wollen, müsste die EAC-Unterstützung in dem jeweiligen Agenten hinzugefügt werden.

5.3.1.1 Pakete senden und empfangen

Die Methode `sendmsg()` in der Klasse EAAgent erhält vom Simulator ein neues Paket, bei dem bereits ein paar Felder im Common-Header und im IP-Header ausgefüllt sind. Die wichtigsten Felder sind *ptype* (Pakettyp), *size* (Paketgrösse in Bytes), *src* (Quelladresse) und *dst* (Zieladresse). Sobald der EA-Agent Zugriff auf das Paket erhält, wird der Pakettyp mit einem der folgenden Werte überschrieben:

- **PT_PROBE** für Prüfpakete
- **PT_EADATA** für Datenpakete

Verlangt das verwendete Verfahren Unterstützung von ECN-Bits, wird dies zusätzlich durch das Setzen des ECT-Bits (siehe Abschnitt 2.3.4.3) im Header aktiviert. Danach wird das Paket zum nächsten Knoten weitergeleitet.

Abbildung 5.2 stellt den Beginn einer Prüfperiode graphisch dar. Mit `start()` initiiert der EAAgent die Prüfphase und ruft `probe()` der angehängten Prüfstrategie auf. Danach wird mit der Variable `expirationTime_` die Ablaufzeit des Prüfens festgelegt. Sind nach dieser Zeit keine Pakete beim Empfänger angekommen, wird das Prüfen abgebrochen und der Fluss zurückgewiesen. Damit wird verhindert, dass in einer Situation, in der keine Prüfpakete ankommen, das Netzwerk mit Prüfpaketen überhäuft wird. Um nach erfolgtem Prüfen eine Rückmeldung vom Empfänger erhalten zu können, muss ein

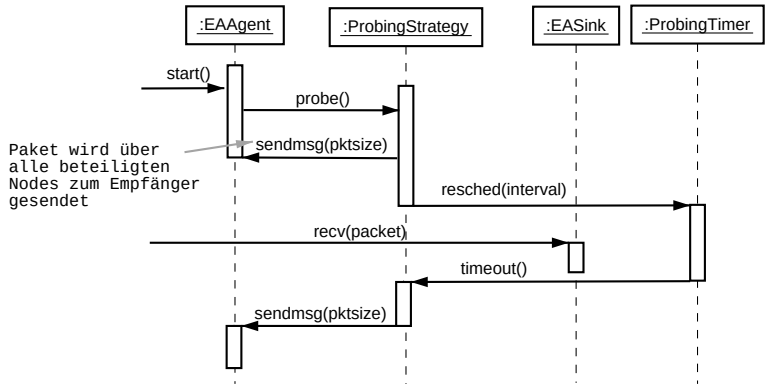


Abbildung 5.2: Sequenzdiagramm des Starts der Prüfphase

EAAgent auch Pakete empfangen können. Diese Funktionalität wird durch `recv()` zur Verfügung gestellt. Das eingehende Paket enthält im Reservierungs-Header ein Bit, das die Zugangskontrollentscheidung anzeigt. Ist dieser Wert 1, werden die Daten gesendet. Für den korrekten Ablauf dieser Prozedur ist es wichtig, dass das gesendete Signalisierungspaket am Ende einer Prüfphase nicht fallen gelassen wird. In einem realen System könnte dieses Problem z. B. mit einem Timer gelöst werden. Der Empfänger müsste dann nach Ablauf dieses Timers das Paket nochmals senden. Eine weitere Möglichkeit wäre, das Antwortpaket mehrmals mit hoher Priorität zu senden. Im Simulations-Framework wird dem Verlust des Signalisierungspakets entgegengewirkt, indem verhindert wird, dass es in einer Queue fallen gelassen wird.

Die wichtigste Methode der Klasse *EASink* ist die geerbte Methode `recv()`, die aufgerufen wird, wenn ein Paket beim Empfänger ankommt. Falls ein Prüfpaket empfangen wird, werden die Zähler für die empfangenen, gesendeten und verworfenen (oder markierten) Pakete aktualisiert. Mit der Variable `checkTime_` kann die Zeit festgelegt werden, bei der jeweils geprüft wird, ob der Fluss frühzeitig zurückgewiesen werden kann. Diese Funktionalität kann von Prüfstrategien verwendet werden, die in irgendeiner Form einen Early-Reject-Algorithmus implementieren (siehe dazu Abschnitt 6.2.2). Hat die Variable den Wert 0, wird erst nach Ablauf der Prüfzeit eine Entscheidung gefällt. Das Resultat der Entscheidung wird durch den Aufruf von `sendResponse()` dem Sender kommuniziert.

5.3.1.2 Konfiguration von EAAgent und EASink

Ein EAAgent-Objekt wird in OTcl über die Klasse *Agent/EndpointADC* instanziiert, ein EASink-Objekt über die Klasse *Agent/EASink*. Ein EAAgent-Objekt kann über folgende Variablen parametrisiert werden:

lifetime_ Dauer des Datenflusses.

packetSize_ Grösse der ausgehenden Pakete.

expirationTime_ Ablaufzeit der Prüfphase.

showDecision_ Boolean der anzeigt, ob die Zugangskontrollentscheidung jedes Flusses auf dem Bildschirm ausgegeben werden soll.

Zusätzlich unterstützen folgende Kommandos die Konfiguration auf der OTcl-Seite:

start Ruft die Methode `probe()` der angehängten Prüfstrategie auf und startet somit die Zugangskontrollprozedur.

stop Stoppt das weitere Generieren von Paketen durch die angehängte Verkehrsquelle.

attach-Strategy <strategy> Fügt eine Prüfstrategie hinzu.

add <ea_sink> Verbindet den Agenten mit der angegebenen Senke.

attach-monitor <admission_monitor> Fügt ein AdmissionMonitor-Objekt hinzu.

Die Klasse EASink verfügt über das Kommando `trace`, durch dessen Aufruf die Verzögerungen der eintreffenden Datenpakete aufgezeichnet werden. Beim Aufruf muss ein Parameter angegeben werden, der den Dateinamen der Trace-Datei enthält. Es sei erwähnt, dass diese Datei unter Umständen sehr gross werden kann.

5.3.1.3 Beispiel

Dieses Beispiel erstellt zwei EAC-fähige Endsysteme. Die Erstellung und Konfiguration der Objekte für die Prüfstrategie (`strategy`), der Netzwerkknoten (`node0, node1`) und des Verkehrs (`traffic`) wurden übersichtshalber weggelassen. Ein komplettes Beispiel-Script folgt später.

```
set ns [Simulator instance]
set eaAgent [new Agent/EndpointADC]
$ns attach-agent $node0 $eaAgent
$eaAgent set lifetime_ 300
$eaAgent set showDecision_ true
$eaAgent set packetSize [$traffic set packetSi-
ze_]
$eaAgent set expirationTime_ 10.0
$eaAgent attach-strategy $strategy
$traffic attach-agent $eaAgent
```

```
set eaSink [new Agent/EASink]
$eaSink trace delay.tr
$ns attach-agent $node1 $eaSink
$eaAgent add $eaSink
$ns at 1.0 "$eaAgent start"
```

5.3.2 Die Klasse ProbingStrategy

Um mit den in Kapitel 3 beschriebenen sowie mit neuen Ansätzen experimentieren zu können, ist es notwendig, die Algorithmen auswechselbar zu entwerfen. Diese Flexibilität wird durch die abstrakte Klasse *ProbingStrategy* erreicht. Sie benutzt das Strategy-Pattern, um Algorithmen zu kapseln und dient als gemeinsame Schnittstelle für Klassen, die einen Zugangskontrollalgorithmus implementieren.

Die Methode `probe()` sendet das erste Prüfpaket an den Empfänger und setzt die Zeit für das nächste fest. Die abstrakte Methode `timeout()` wird durch einen Timer immer dann aufgerufen, wenn die Zeit zwischen zwei zu sendenden Paketen abgelaufen ist. Sie wird von der jeweiligen Subklasse implementiert. Die Subklassen sind dafür verantwortlich, wie die Prüfphase im Detail erfolgt. Die Prüfphase dauert so lange, bis `stopProbing()` aufgerufen wird. Mit der ebenfalls öffentlichen Methode `decision()` wird einem Kunden mitgeteilt, ob mit der angegeben Anzahl verworfener und gesendeter Pakete eine negative oder positive Zugangskontrollentscheidung erfolgt. Das Messen von Paketverlust ist erst dann möglich, wenn die Prüfpakete auf der Senderseite nummeriert werden. Jedes ausgehende Paket erhält also eine Sequenznummer, womit der Empfänger fehlende Pakete erkennen kann, indem er nach Lücken in den beobachteten Sequenznummern Ausschau hält. Ist s_{max} die höchste und s_{min} die kleinste in einem Zeitintervall beobachtete Sequenznummer, und ist l die Anzahl der beobachteten Lücken in den Sequenznummern, so berechnet sich die Verlustrate r durch:

$$r = \frac{l}{s_{max} - s_{min}}.$$

Die Methode `decision()` macht nun nichts anderes, als die Verlustrate zu berechnen und diesen Wert mit dem Schwellenwert `epsilon_` zu vergleichen.

5.3.2.1 Konfiguration

Die C++-Klasse ProbingStrategy verfügt über zwei gebundene Instanzvariablen, die unter gleichem Namen via OTcl verändert werden können:

epsilon_ Dies ist der Schwellenwert für die Verlustrate. Flüsse werden zurückgewiesen, wenn dieser Schwellenwert erreicht ist.

probingTime_ Dauer der Prüfphase (in Sekunden).

Folgende Kommandos werden von der Klasse ProbingStrategy unterstützt und ergänzen die Konfiguration der Strategien auf der OTcl-Seite:

cngMarks Konfiguriert die Strategie so, dass Paketmarkierungen als Indikation für Überlast benutzt wird.

pktDrops Konfiguriert die Strategie mit Paketverlust als Indikation für Überlast.

5.3.3 ProbingStrategy Implementierungen

Im Moment sind drei C++-Klassen vorhanden, die von der Klasse ProbingStrategy erben:

- **SimpleStrategy** implementiert eine einfache Prüfstrategie, in der der Sender während der Prüfzeit mit einer fixen Rate Prüfpakete sendet. Am Ende der Prüfphase wird die Verlustrate berechnet und die Zugangskontrollentscheidung gefällt.
- **SlowStart** erhöht langsam die Übertragungsrate des Prüfverkehrs. Die Rate wird jeweils nach einem vorgegebenen Zeitintervall um einen Faktor erhöht. Ist nach einem Intervall die Verlustrate über dem Schwellenwert, wird der Fluss zurückgewiesen. Die Übertragungsrate für ein Intervall i wird durch

  $$r_i = \frac{r}{f^{(m-1)-i}}$$

  berechnet, wobei f der zu multiplizierende Faktor und m die Anzahl Perioden ist.
- **EarlyReject** prüft mit der Rate r für höchstens die Dauer der Prüfzeit. Ist nach einem vorgegebenen Zeitintervall die Verlustrate über dem Schwellenwert, wird der Fluss zurückgewiesen.

Diese Klassen können in einem OTcl-Script kreiert und konfiguriert werden. Die Namen der Klassen und Parameter sind im Folgenden beschrieben:

SimpleStrategy - Ein SimpleStrategy-Objekt wird in OTcl über die Klasse *PrStrategy/Simple* instanziiert. Sie enthält die gebundene Variable `rate_`, die die Übertragungsrate während der Prüfphase (in bit/s) angibt.

EarlyReject - Ein EarlyReject-Objekt wird in OTcl über die Klasse PrStrategy/EarlyReject instanziiert. Sie enthält die gebundene Variable `rate_`, die oben beschrieben wurde. Weiter kann mit dem Parameter `checkTime_`, das

Intervall bestimmt werden, nach dem jeweils überprüft wird, ob ein Fluss frühzeitig zurückgewiesen werden kann.

SlowStart - Ein SlowStart-Objekt wird in OTcl über die Klasse PrStrategy/SlowStart instanziiert. Die Instanzvariablen, die dieses Objekt parametrisieren sind:

rate_ Max. Übertragungsrate während der Prüfphase (in bit/s).

periodTime_ Dauer eines Intervalls mit gleicher Übertragungsrate (in Sekunden).

factor_ Faktor mit der die Übertragungsrate an den Intervallgrenzen multipliziert wird.

5.3.3.1 Beispiel

Dieses Beispiel-Script zeigt die Erzeugung und Konfiguration von zwei EAC-fähigen Endsystemen. Zuerst wird ein EAAgent an einen Knoten in der Topologie gebunden und mit Slow-Start konfiguriert. Danach wird er mit dem add-Kommando mit der Senke verbunden. Die Simulation endet nach fünf Sekunden und die Anzahl verworfener und gesendeter Pakete wird auf dem Bildschirm angezeigt.

```
# Simple EAAgent demo
set f [open out.tr w]
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]

$ns duplex-link $n0 $n1 1MB 2ms EADC

#create queue
set q [[$ns link $n0 $n1] queue]
set qmon [$ns monitor-ptype $n0 $n1 $f]
set l [$ns link $n0 $n1]
set queue_0 [new Queue/DropTail]
$queue_0 set limit_ 150
$queue_0 drop-target [$l set drophead_]
$q insertPrioQueue $queue_0
$q setBELimit 200

# Endpoint admission control agent
set eaAgent [new Agent/EndpointADC]
$ns attach-agent $n0 $eaAgent
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 125
$cbr set rate_ 500kb
$cbr attach-agent $eaAgent
```

```
# Configuration of SlowStart algorithm
set strategy [new PrStrategy/SlowStart]
$strategy set probingTime_ 5.0
$strategy set epsilon_ 0.05
$strategy set rate_ [$cbr set rate_]
$strategy pktDrops
$strategy set periodTime_ 1.0
$strategy set factor_ 2.0
$eaAgent attach-strategy $strategy
$eaAgent set lifetime_ 300
$eaAgent set expirationTime_ 10.0
$eaAgent set packetSize_ [$cbr set packetSize_]
$eaAgent set showDecision_ true

set sink [new Agent/EASink]
$ns attach-agent $n1 $sink
$eaAgent add $sink

$ns at 0.1 "$eaAgent start"

$ns at 5.1 "finish"
proc finish {} {
    global ns f qmon
    qmon printStats
    $ns flush-trace
    close $f
    exit 0
}
$ns run
```

5.3.4 Entwicklung neuer Prüfstrategien

Wenn das Framework um einen neuen EAC-Algorithmus erweitert werden soll, muss man eine neue Prüfstrategie in C++ definieren. Als Basisklasse dient die Klasse ProbingStrategy, die schon viel Funktionalität kapselt. Die abgeleitete Klasse muss die beiden rein-virtuellen Methoden `init()` und `timeout()` ersetzen. In der Methode `init()` werden alle Variablen initialisiert die zum Zeitpunkt der Objekterstellung noch nicht verfügbar sind. Dazu gehört u.a. die Variable `interval_`, die die Zeit zwischen dem ersten gesendeten Paket und dem darauffolgenden umfasst. Die Methode `timeout()` ist dafür verantwortlich, über seinen Agenten ein neues Paket zu senden und das Ereignis für das nächste zu sendende Paket festzulegen.

Um den Algorithmus via OTcl-Script konfigurieren zu können, müssen seine Paramater gebunden und gegebenenfalls Kommandos hinzugefügt werden. Das weitere Vorgehen ist nun sehr vom Algorithmus abhängig, den man implementieren will. Wenn Änderungen an den Endsystemen oder dem Queue-Modul notwendig sind, sollte unbedingt zuerst die Dokumentation zu diesen Einheiten studiert werden.

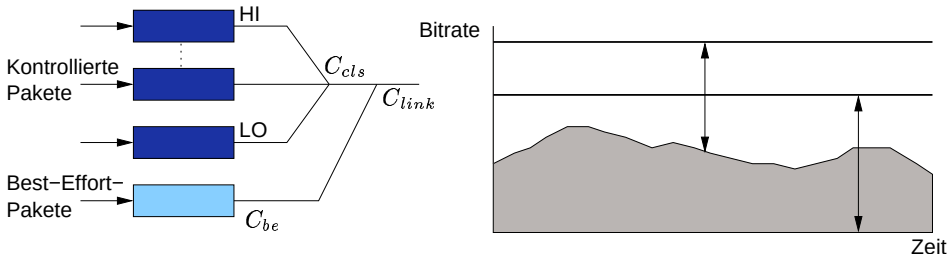


Abbildung 5.3: Paket-Scheduler in der EAQueue. Der kontrollierte Service erhält einen Bruchteil der Bandbreite. Die übrige Kapazität (weisse Fläche) kann vom Best-Effort-Service genutzt werden.

Zusammengefasst sind somit 4 Schritte notwendig, um dem Framework Unterstützung für einen neuen EAC-Algorithmus hinzuzufügen:

1. Eine neue Klasse schreiben mit ProbingStrategy als Basisklasse
2. Die Methoden `init()` und `timeout()` implementieren
3. Kommandos definieren und Variablen binden, damit die Klasse via OTcl konfiguriert werden kann
4. Gegebenenfalls Code in den Endsystemen und dem Queue-Modul hinzufügen

5.4 Das Queue-Modul

Die Klassen und Funktionen in diesem Abschnitt können in ~ns/eac/EAQueue, PriorityQueue, gk, GKPriority, RateLimiter, TSWRateLimiter {h, cc} gefunden werden.

5.4.1 Die Klasse EAQueue

Die Klasse EAQueue erbt von der abstrakten Klasse Queue und stellt zwei Queues zur Verfügung, um Best-Effort-Pakete und kontrollierte Pakete unabhängig voneinander zwischenzuspeichern. Die Queue für den Best-Effort-Verkehr wird durch die in ns-2 vorhandene Klasse PacketQueue realisiert, die nach dem FIFO-Prinzip arbeitet. Falls der Puffer voll ist, werden die zuletzt angekommenen Pakete verworfen (Drop Tail). Die Queue für den kontrollierten Verkehr ist eine Priority-Queue, welche weiter unten näher beschrieben wird.

Um die in Abschnitt 2.3.1 beschriebene Isolation zwischen TCP-Verkehr und kontrolliertem Verkehr zu realisieren, bedient ein Paket-Scheduler die Queue mit kontrolliertem Verkehr mit höherer Priorität als die Best-Effort-Queue.

Die Struktur des Schedulers ist in Abbildung 5.3 graphisch dargestellt. Der kontrollierte Verkehr kann wiederum in mehrere Prioritäts-Niveaus unterteilt werden. Dies ist z. B. dann notwendig, wenn zwischen Daten- und Prüfverkehr unterschieden werden soll. Die Bandbreite, den der kontrollierte Service erhalten soll, kann mit einem Ratenbegrenzer auf einen Bruchteil der Link-Kapazität begrenzt werden. Die übrige Kapazität (weisse Fläche in der Abbildung) kann vom Best-Effort-Service genutzt werden.

Die EAQueue wird von aussen durch die beiden Methoden `enque()` und `deque()` bedient. `enqueue()` speichert eingehende Pakete abhängig von ihrem Pakettyp in einer der beiden Queues. `deque()` realisiert den erwähnten Paket-Scheduler. Kontrollierte Pakete werden mit höherer Priorität aus der EAQueue herausgenommen, aber nur dann, wenn das Bandbreitenlimit noch nicht erreicht wurde.

5.4.1.1 EAQueue Konfiguration

In ns-2 werden Queue-Objekte bei der Erstellung eines Links erzeugt. Die Syntax eines Duplex-Links ist wie folgt:

```
set ns [new Simulator]
$ns duplex-link <node0> <node1> <bwdth> <dly> <queue_type>
```

Der Befehl kreiert einen Link von `<node0>` nach `<node1>` mit der angegebenen Bandbreite und Verzögerung. Der Link benutzt eine Queue vom Typ `<queue_type>`. Um eine EAQueue zu benutzen, muss dieser Parameter den Wert `EADC` erhalten. Folgende Kommandos werden unterstützt, um die EAQueue zu konfigurieren:

insertPrioQueue <queue> <ptype> Fügt ein Queue-Objekt für den kontrollierten Verkehr hinzu. Das Argument `<ptype>` ist optional und definiert die Verwaltung eines speziellen Pakettyps für die einzufügende Queue. Gültige Werte sind `EADATA` (für Datenpakete) und `PROBE` (für Prüfpakete). Wird kein Pakettyp angegeben, verwaltet die Queue Pakete jeden Typs.

setBELimit <limit> Setzt die Länge der Best-Effort Queue.

addQueueRate <max_rate> <rltype> Setzt das Limit für den kontrollierten Verkehr auf die angegebene Bandbreite. Der zweite Parameter ist optional und bezeichnet den Typ des Ratenbegrenzers. Der Standardwert und im Moment einziger gültiger Wert ist `TSW` (siehe auch Abschnitt 5.4.4).

Ein Beispiel wie EAQueues und ihre Kommandos benutzt werden, kann im folgenden Abschnitt gefunden werden.

5.4.2 Die Priority Queue

Da einige Ansätze aus Kapitel 3 Daten- und Prüfpakete in unterschiedliche Prioritätsklassen einstufen, wird für den kontrollierten Verkehr eine Queue benötigt, die wiederum zwischen mindestens zwei verschiedenen Prioritäten unterscheiden kann. Die Klasse PriorityQueue stellt diesen Priorisierungsmechanismus zur Verfügung. Sie ist eine reine C++-Klasse und ist nicht direkt über den OTcl-Interpreter zugänglich.

5.4.2.1 Queue hinzufügen

Eine Priority-Queue hat eine Liste mit Queue-Objekten, in denen eingehende Pakete gespeichert werden. Die Methode `insertQueue()` fügt ein Queue-Objekt ans Ende dieser Liste ein. Im Grunde genommen kann jede Klasse, die von der Klasse Queue erbt, der Liste hinzugefügt werden. In ns-2 sind mehrere solcher Klassen vorhanden. Sie implementieren Queueing-Mechanismen wie faires Queueing, RED oder Drop Tail. Die Konfiguration der einzufügenden Queue wie z. B. deren Länge muss ausserhalb der PriorityQueue erfolgen. Für eine vollständige Liste der vorhandenen Klassen und die dazugehörenden Konfigurationsparameter siehe [22].

5.4.2.2 Pakete einfügen und herausnehmen

Beim Aufruf der Methode `enque()` wird das eingehende Paket nach seinem Pakettyp überprüft. Falls eine spezielle Queue vorhanden ist, die den Pakettyp verwaltet, wird das Paket dieser Queue hinzugefügt. Die PriorityQueue ist somit nicht nur auf zwei Prioritätsklassen reduziert und ist mit weiteren Pakettypen erweiterbar. Wird `deque()` aufgerufen, wird aus der ersten Queue in der Liste, die nicht leer ist, ein Paket herausgenommen. Die zuerst eingefügte Queue erhält also die höchste, die zuletzt eingefügte die niedrigste Priorität.

5.4.2.3 Beispiel

Das folgende Beispiel zeigt die Benutzung der EAQueue und der Priority-Queue. Das Script erzeugt einen Link zwischen zwei Knoten mit einer EAQueue. In der zweiten Zeile wird das EAQueue-Objekt referenziert und mit einer Priority-Queue für Daten bzw. Prüfpakete konfiguriert. Das von der Klasse Queue zur Verfügung gestellte Kommando `drop-target` wird benötigt, um das Ziel der verworfenen Pakete zu setzen. Damit diese Pakete korrekt gezählt werden können, muss dieses Ziel mit demjenigen des Links übereinstimmen. In der letzten Zeile wird die Rate für den kontrollierten Verkehr auf 1 Mbit/s begrenzt.

```
$ns duplex-link $n0 $n1 10MB 20ms EADC
set q [[$ns link $n0 $n1] queue]
```

```
set l [$ns link $n0 $n1]
set queue_0 [new Queue/DropTail]
$queue_0 set limit_ 75
$queue_0 drop-target [$l set drophead_]
#queue for data traffic
$q insertPrioQueue $queue_0 EADATA
set queue_1 [new Queue/DropTail]
$queue_1 set limit_ 75
$queue_1 drop-target [$l set drophead_]
#queue for probing traffic
$q insertPrioQueue $queue_1 PROBE
$q setBELimit 200
$q addQueueRate 100000 TSW
```

5.4.3 Klassen mit einer virtuellen Queue

Für die in Abschnitt 3.5.3 beschriebene virtuelle Queue stellt ns-2 die Klasse GK (benannt nach den Autoren Gibbens und Kelly) zur Verfügung.

Konfigurationsparameter und Kommandos für GK-Objekte sind:

ecnlim_ Bruchteil der Pufferspeicher-Kapazität, die der virtuellen Queue im Verhältnis zur realen Queue zur Verfügung steht.

mean_pktsize_ Durchschnittliche Paketgrösse.

drop_front_ Boolean der angibt, ob Pakete am Anfang oder am Ende der Queue entfernt werden.

link <link> fügt der GK-Queue ein Link-Objekt hinzu.

Die Klasse GKPriority erweitert GK mit Funktionalität für das Virtual Dropping (siehe Abschnitt 6.4) und Unterstützung von mehreren Prioritätsstufen. Wie bei der PriorityQueue können Daten- und Prüfpakete separiert zwischengespeichert werden. Beim Virtual Dropping ist dies besonders wichtig, da bei voller virtueller Queue nur Prüfpakete und keine Datenpakete verworfen werden dürfen.

Die Konfiguration erfolgt analog zur GK-Queue. Mit dem Parameter `markMethod_` kann zusätzlich noch das Markieren von Paketen ein- bzw. ausgeschaltet werden. Hat die Variable den Wert `false`, ist die Queue für das Virtual Dropping konfiguriert.

Bemerkung: Der verwendete Algorithmus braucht Information über die Kapazität des ausgehenden Links, um die Länge der virtuellen Queue bei eintreffenden Paketen jeweils aktualisieren zu können. Es sollte daher immer nach der Instanziierung eines GK-Objekts das Kommando `link` aufgerufen werden.

5.4.3.1 Beispiel

Dieses Beispiel zeigt die Erstellung einer Queue vom Typ GKPriority. Die Queue wird so konfiguriert, dass bei Überlast Pakete markiert werden.

```
set l [$ns link $n0 $n1]
set queue_0 [new Queue/GK/Priority]
$queue_0 set ecnlim_ 0.9
$queue_0 set mean_pktsize_ 125
$queue_0 set qlimit0_ 75
$queue_0 set qlimit1_ 75
$queue_0 set markMethod_ true
$queue_0 link [$l set link_]
$queue_0 drop-target [$l set drophead_]
```

5.4.4 Hilfsklassen

5.4.4.1 Ratenbegrenzer

Um die Bandbreite des kontrollierten Verkehrs begrenzen zu können, stellt das Framework die Schnittstelle RateLimiter zur Verfügung. Die Methode `update()` aktualisiert den Status des entsprechenden RateLimiters und muss vom Kunden, typischerweise eine Queue-Klasse, bei jedem Entfernen eines Pakets aufgerufen werden. Die Implementierung dieser Methode ist vom entsprechenden Algorithmus zur Berechnung des Erreichens des Bandbreitenlimits abhängig. Im Moment ist die Klasse TSWRateLimiter vorhanden, die den in [24] beschriebenen Time-Sliding-Window-Algorithmus verwendet.

Die beiden erwähnten Klassen sind reine C++-Klassen und können nicht über den OTcl-Interpreter kreiert werden. Die Wahl eines konkreten RateLimiters erfolgt über das Kommando `addQueueRate` der EAQueue.

5.4.4.2 AdmissionMonitor

Die Klasse AdmissionMonitor führt eine Statistik über die Anzahl angenommener und zurückgewiesener Flüsse zwischen zwei Endsystemen. Mit dem Kommando `attach-monitor` des EAAgents werden alle getroffenen Entscheidungen gespeichert. Die Methoden `reject()` und `accept()` inkrementieren die korrespondierenden Zähler.

Ein AdmissionMonitor wird unter gleichem Namen auf der OTcl-Seite kreiert. Folgende Kommandos werden zusätzlich unterstützt:

clear setzt die Zähler wieder auf null zurück.

printStats gibt die aktuelle Statistik auf dem Bildschirm aus.

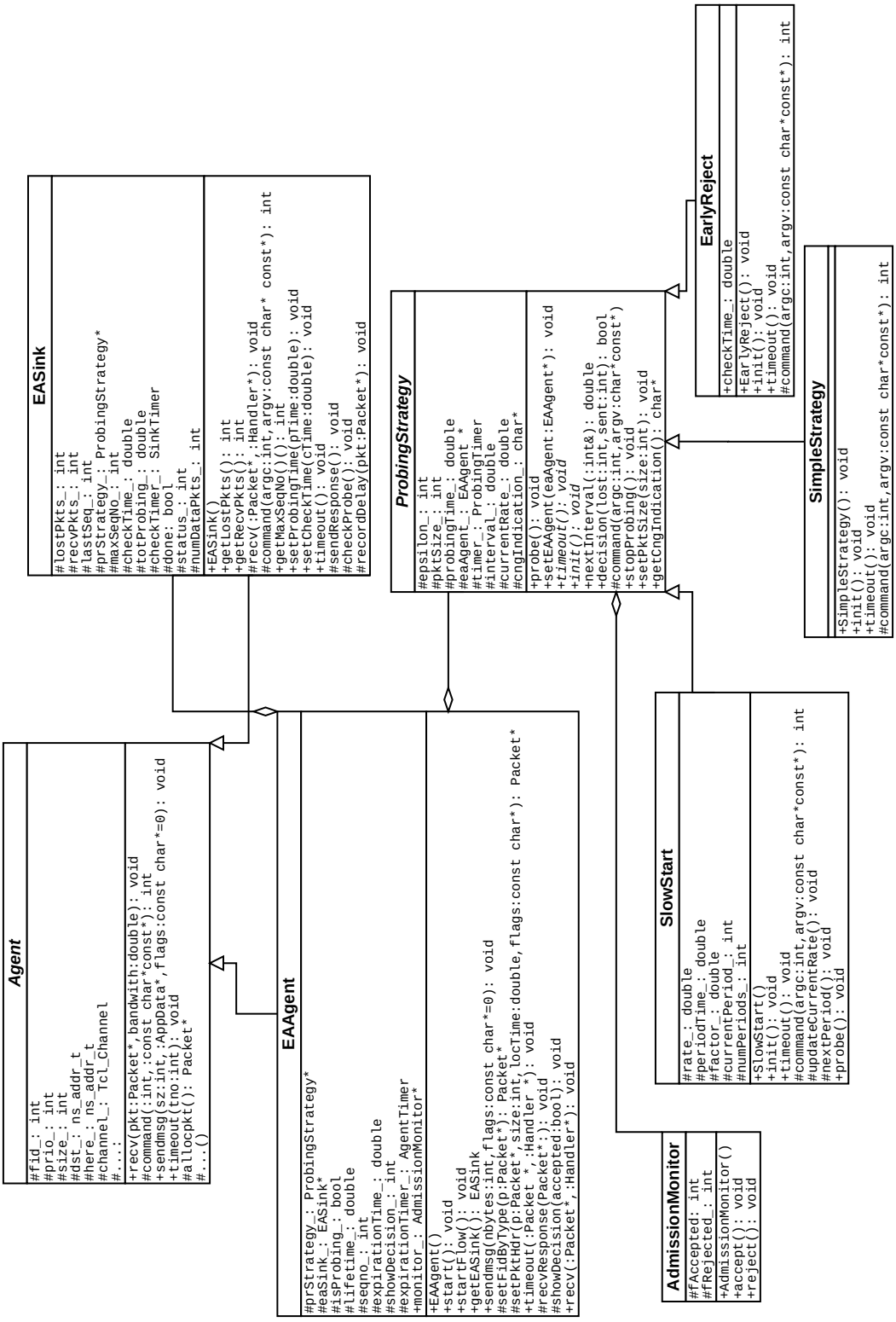


Abbildung 5.4: Klassendiagramm des Probing-Moduls

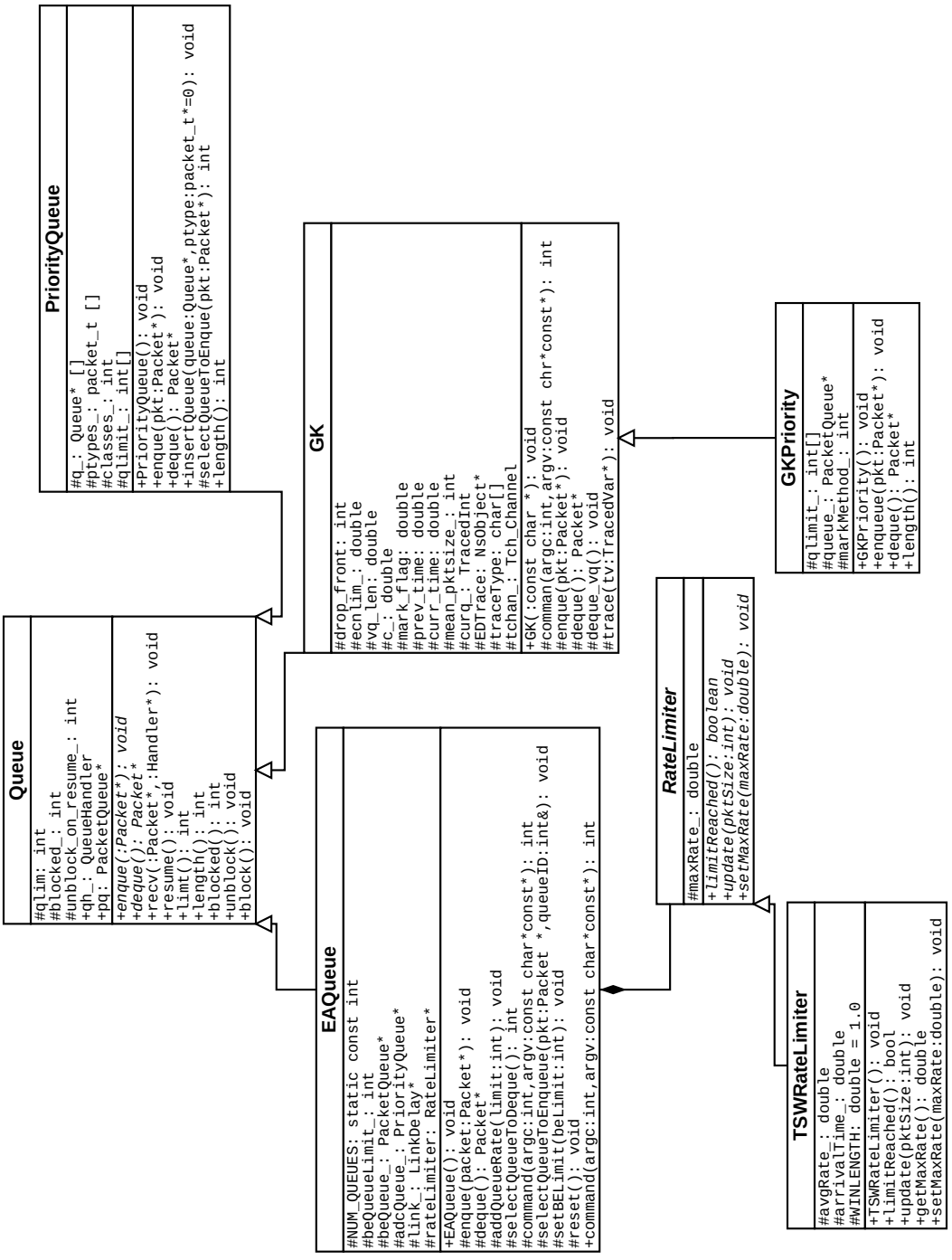


Abbildung 5.5: Klassendiagramm des Queue-Moduls

6 Experimentelle Resultate

Um das Verhalten des Simulations-Frameworks zu überprüfen, wurden Simulationen mit den aus der Literatur bekannten EAC-Algorithmen durchgeführt. Ausserdem wird ein bisher noch nicht untersuchter Ansatz evaluiert.

Die bei den Simulationen berücksichtigten Werte sind die Verlustrate, die Netzauslastung, die Paketverzögerung und die Zurückweisungswahrscheinlichkeit der Flüsse. Diese Werte dienen dazu, die Leistungsfähigkeit der getesteten Ansätze zu diskutieren.

Die Simulations-Szenarien zu Beginn des Kapitels wurden bewusst wie in [8] gewählt, um die Resultate miteinander vergleichen zu können.

6.1 Simulationsmethode

Alle Knoten verwenden die im letzten Kapitel beschriebene EAQueue mit einer Best-Effort- und einer Priority-Queue. Kontrollierter Verkehr hat strikte Priorität gegenüber Best-Effort-Verkehr. Die Auslastung in den jeweiligen Abbildungen ist definiert als der relative Anteil an übertragenen Daten in einem Zeitintervall zu maximal möglichen Übertragungen. Sie bezieht sich nur auf die Menge des Verkehrs, der EAC zur Verfügung steht, nicht auf die ganze Bandbreite eines Links. Prüfverkehr wurde nicht in die Auslastung mit einbezogen, da er keine signifikanten Übertragungen darstellt.

In allen Simulationen wird der kontrollierte Verkehr entsprechend einem Poisson-Prozess generiert. Anfragen für eine Gewährung erfolgen beim Sender mit durchschnittlicher Zwischenankunftszeit γ. Flüsse, die zurückgewiesen wurden, versuchen nicht, nochmals zu prüfen. Diese Vereinfachung ist zwar unrealistisch, man kann aber das erneute Versuchen als Teil des eintreffenden Poisson-Prozesses ansehen. Ist ein Fluss einmal akzeptiert, hat er, wie in [8], eine exponentielle Lebenszeit mit einem Durchschnitt von 300 Sekunden.

Wo nichts anderes erwähnt wird, bestehen die Verkehrsquellen aus Verkehr mit exponentiellen On- und Off-Zeiten von 500 ms, einer Burst-Rate von 256 kBit/s und einer Paketgrösse von 125 Bytes. Die Verkehrsquellen entsprechen einem Token-Bucket-Filter mit $B = 125$ und der durch die Burst-Rate gegebenen Rate r. Es wurde Verkehr mit variabler Bitrate (VBR) gewählt, da dies dem Verkehr von Echtzeitanwendungen am nächsten kommt. Die meisten Simulationen verwenden eine einfache Topologie mit einem einzigen überlasteten Link. Selbst wenn dies ein sehr vereinfachtes Szenario ist, ist es dennoch

Abbildung 6.1: Topologie für das Basisszenario

repräsentativ für ein komplexeres Netzwerk-Szenario, in dem auf dem Pfad zwischen Quelle und Ziel ein Link als Flaschenhals wirkt. In den Knoten haben die Best-Effort- und Priority-Queues eine maximale Queue-Länge von 150 Paketen. In Abschnitt 6.4.3 wird zusätzlich eine Topologie mit mehreren Links verwendet, um das Verhalten von EAC auch in einem grösseren Szenario beurteilen zu können.

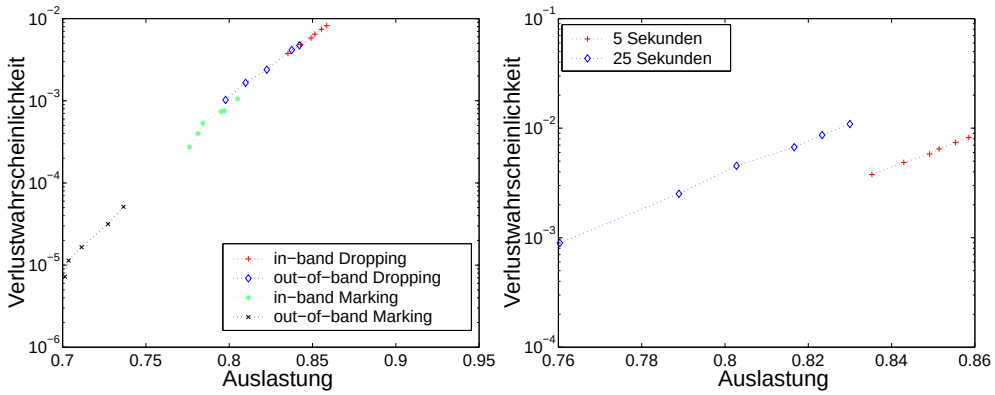
Alle Simulationen wurden 10 Mal mit unterschiedlichen Startwerten für die Zufallszahlen (Random-Seed) durchgeführt und gemittelt. Die einzelnen Simulationszeiten betrugen 1500 Sekunden, während die Daten für die ersten 300 Sekunden nicht berücksichtigt wurden. Da Netzwerksimulationen sehr rechenaufwändig sind und oft mehrere Stunden dauern, wurde als Hardware-Umgebung UBELIX [25], ein Linux-Cluster bestehend aus 32 Knoten mit je einem AMD Athlon 1400Mhz Prozessor und 1GB RAM gewählt.

6.2 Vergleich mit Resultaten aus der Literatur

Um die Resultate, die das Simulations-Framework liefert, überprüfen zu können, werden sie in diesem Abschnitt mit Resultaten aus der Literatur verglichen. Als Referenz dienen die Resultate aus [8]. Zusätzlich werden die Resultate mit dem Kriterium analysiert, ob erwartete bzw. erklärbare Messwerte erzeugt werden.

6.2.1 Basisszenario

Abbildung 6.1 veranschaulicht die Topologie für das Basisszenario. Der Link hat eine Bandbreite von 10 Mbit/s und ist mit Flüssen von A nach B belegt. Die Prüfzeit ist für alle Flüsse auf 5 Sekunden festgelegt, γ beträgt 3.5 Sekunden. In-band Marking und in-band Dropping wurde mit $\epsilon = 0, 0.01, 0.02, 0.03, 0.04$ und 0.05 getestet, out-of-band Marking und out-of-band Dropping mit $\epsilon = 0$, 0.05, 0.10, 0.15 und 0.20. Abbildung 6.2(a) zeigt die Verlustwahrscheinlichkeit gegenüber der Auslastung für die vier EAC-Algorithmen. Das Prüfen erfolgt durch Slow-Start mit einer Intervallzeit von einer Sekunde. Die Punkte in der Abbildung stellen für einen gegebenen Algorithmus die Verlustwahrscheinlichkeit und die Auslastung für verschiedene Werte von ϵ dar. Ein Aspekt, den man aus den Kurven herausgreifen kann, bezieht sich auf die Verlustwahrscheinlichkeit zu einem gegebenen Wert für die Auslastung (oder umgekehrt).



(a) Prüfdauer 5 Sekunden

(b) Prüfdauer 25 Sekunden (in-band Dropping)

Abbildung 6.2: Basisszenario

Algorithmen, die weniger Verluste zum gleichen Auslastungsniveau produzieren, sind als besser zu bewerten. Wie man sehen kann, erreicht out-of-band Marking eine Verlustrate im Bereich von 10^{-5}, während im Gegensatz dazu die minimale Verlustrate von in-band Dropping grösser also 10^{-3} ist. In-band Marking und out-of-band Dropping erreichen ein Verlustniveau dazwischen.

Um eine kleinere Verlustrate zu erhalten, könnte man die Prüfzeit erhöhen, was aber mit einer längeren Set-Up-Zeit verbunden ist. In Abbildung 6.2(b) wird dieses erweiterte Szenario mit in-band Dropping mit dem Basisszenario verglichen. Das Slow-Start-Prüfen dauert 25 Sekunden mit einer Intervallzeit von 5 Sekunden. In der Abbildung sieht man, dass die Verlustrate zwar gesunken ist, gleichzeitig aber die Auslastung wesentlich kleiner geworden ist. Dies liegt darin begründet, dass mehr Bandbreite von Prüfpaketen konsumiert wird.

Die Resultate für das Basisszenario und das erweiterte Basisszenario stimmen gut mit denen aus der Literatur überein. Die minimalen Verlustraten und die Auslastung haben im Vergleich zur Referenz sehr ähnliche Werte. Abweichungen kommen dadurch zu Stande, dass nicht genau das gleiche Szenario nachgebildet werden konnte, da z. T. Angaben fehlten.

6.2.2 Hohe Last

Das Basisszenario wird nun so erweitert, dass weit mehr Verkehr um einen Zugang konkurriert. Die Zwischenankunftszeit γ wird deshalb verkleinert und beträgt nun eine Sekunde. In diesem Szenario ist die Last sehr hoch und beträgt ungefähr 400% der Link-Kapazität. Es kann zu Thrashing kommen, wobei wenig Flüsse akzeptiert werden (siehe Abschnitt 3.3). Das Verhalten unter

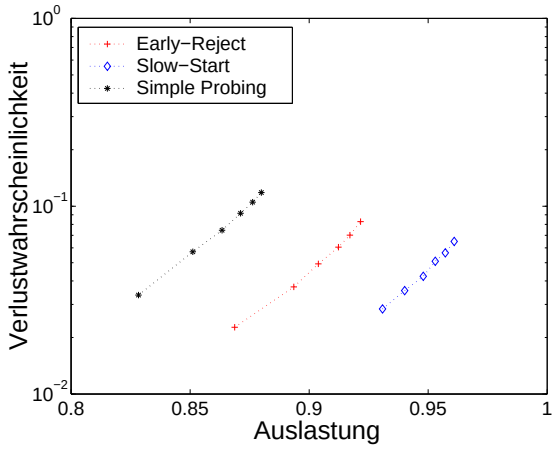


Abbildung 6.3: In-band Dropping mit hoher Last ($\gamma = 1.0s$)

hoher Last wird exemplarisch anhand in-band Dropping mit den drei Prüfvarianten *Slow-Start, Simple Probing* und *Early-Reject* gezeigt. Slow-Start wurde bereits in Abschnitt 3.3 beschrieben. Simple Probing ist ein trivialer Algorithmus, der für die Dauer der Prüfphase mit konstanter Rate sendet und frühestens nach Ablauf der Prüfzeit eine Entscheidung fällt. Early-Reject versucht zu erkennen, ob ein Fluss frühzeitig zurückgewiesen werden kann. Der Algorithmus prüft mit Rate r für höchstens die Dauer der Prüfzeit. Übersteigt aber in irgendeinem (vorgegeben) Intervall der Prozentsatz der verlorenen (oder markierten) den Schwellenwert ϵ, wird der Fluss zurückgewiesen und das Prüfen abgebrochen. In der Simulation beträgt diese Intervallzeit eine Sekunde. Early-Reject ist eine Version des Prüfens zwischen Simple Probing und Slow-Start: Es prüft mit konstanter Prüfrate (wie Simple Probing) aber nicht unbedingt bis ans Ende der Prüfzeit (wie Slow-Start).

In Abbildung 6.3 sieht man, dass die drei Prüfvarianten ähnliche Verlustraten aufweisen. Die grösste Abweichung ist bei Simple Probing und Slow-Start für $\epsilon = 0.05$ zu erkennen und liegt bei 0.053. In der Auslastung sind die Unterschiede hingegen markant. Die geringe Auslastung von Simple Probing kommt dadurch zu Stande, dass infolge Thrashing wenig Flüsse akzeptiert werden. Bei Slow-Start ist die Auslastung mit über 92% deutlich grösser. Slow-Start schafft es, Thrashing zu minimieren, indem es eintreffendem Prüfverkehr nicht erlaubt, Zugänge zu verhindern [8]. Wie erwartet, liegt die Kurve von Early-Reject genau dazwischen.

6.3 Verzögerungsmessungen

In der Forschung wurden bisher vor allem Paketverlust und die Auslastung zur Leistungsanalyse von EAC-Konzepten hinzugezogen. Echtzeitanwendungen wie IP-Telefonie haben aber auch strenge Anforderungen bezüglich der

Paketverzögerung (weniger als 150 ms Mund-zu-Ohr-Verzögerung für gute Sprachqualität), die während der ganzen Verbindungszeit erfüllt werden müssen. Die Analyse der Verzögerungskomponenten auf dem Pfad zwischen Quelle und Ziel zeigt, dass 100-150 ms für Kompression, Paketisierung, Jitter-Kompensation, Signallaufzeit, etc verbraucht wird. Da bleiben nur noch wenige Millisekunden für die Queue-Verzögerung innerhalb der zahlreichen Router auf dem Pfad [14]. Es stellt sich die Frage, ob die verschiedenen Ansätze diese Anforderung erfüllen können. Weiter ist von Interesse, wie gut die Verzögerung von den Endpunkten kontrolliert werden kann.

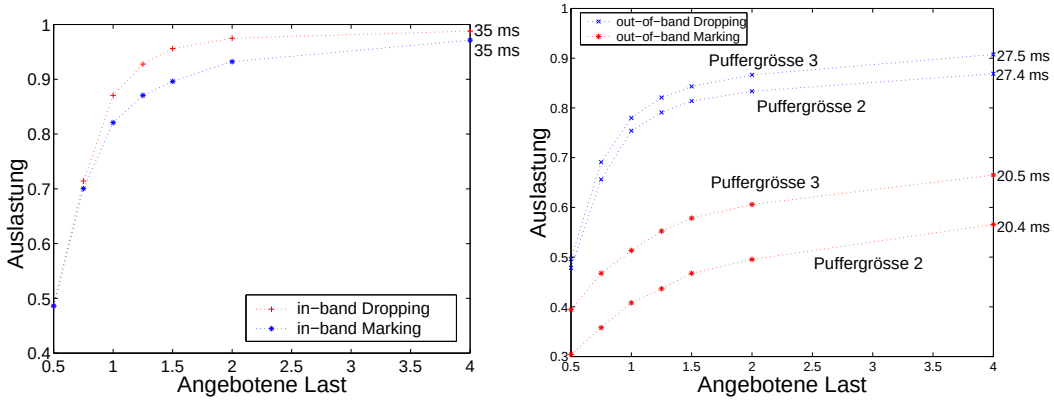
Um die Leistungsfähigkeit der EAC-Verfahren bezüglich der eben erwähnten Aspekte evaluieren zu können, wurden Simulationen mit Verzögerungsmessungen durchgeführt. Im Simulations-Framework kann die Unterstützung für Messungen dieser Art in den Senken aktiviert werden (siehe Abschnitt 5.3.1). Zur Simulation diente das Szenario aus Abschnitt 6.2.1. Es wurde aber so angepasst, dass es in etwa mit einem realen IP-Telefonie-Szenario übereinstimmt. Die durchschnittliche Lebenszeit eines Flusses beträgt nur noch 3 Minuten, was der Dauer eines durchschnittlichen Telefongesprächs entspricht. Die maximale Set-Up-Zeit wurde auf eine Sekunde reduziert. Wiederum wird der angebotene Verkehr gemäss eines Poisson-Prozesses generiert. Die angebotene Last ρ berechnet sich dann durch

$$\rho = \frac{\lambda}{\mu} \cdot \frac{r_{avg}}{C},$$

dabei ist λ (Flüsse / s) die Erzeugungsrate der Flüsse, $1/\mu$ (s) die durchschnittliche Lebensdauer eines Flusses, r_{avg} die durchschnittliche Übertragungsrate und C die Kapazität des Links. Der Schwellenwert ϵ beträgt in allen Simulationen 0, um einen möglichst kleinen Paketverlust zu erreichen. Wie in Abschnitt 3.5.1 angedeutet, kann die Verzögerung in EAC-Verfahren, die out-of-band prüfen, durch eine kleine Queue für den Prüfverkehr minimiert werden. In den Simulationen beträgt die Länge dieser Queues zwei bzw. drei Pakete.

6.3.1 In-band-Prüfen

Abbildung 6.4 zeigt die akzeptierte Last gegenüber der angebotenen Last für einen Link mit 10 Mbit/s und 20 ms Signalverzögerung. Die 99. Percentile der Ende-zu-Ende-Verzögerung, die in der Simulation bei 4-facher Last gemessen wurden, sind ebenfalls angegeben. In Abbildung 6.4(a) ist ersichtlich, dass dort, wo das Prüfen in-band erfolgt, die Auslastung sehr hoch ist. Bei diesen Ansätzen werden viele Flüsse akzeptiert, und es wird weitgehend über 80% der Link-Kapazität ausgenutzt. Die Verzögerung ist aber deutlich höher als bei Ansätzen, die out-of-band prüfen. Dies liegt daran, dass die Verzögerungskomponente nur wenig in die Messungen der Prüfphase einfliesst. Die Zugangskontrolle berücksichtigt nur die Anzahl markierter oder verworfener Pakete in ihren Entscheidungen. Im Grunde genommen kann man sagen, dass in Situationen, wo Pakete in einer Queue verworfen werden, sich auch eintreffende Pakete verzögern. Die Verzögerung innerhalb einer Queue kann aber



(a) In-band-Prüfen

(b) Out-of-band-Prüfen

Abbildung 6.4: Auslastung vs. angebotene Last. Die 99. Percentile der Ende-zu-Ende-Verzögerung sind bei 4-facher Last angegeben.

schon lange bevor die ersten Pakete fallen gelassen oder markiert werden unakzeptable Ausmasse annehmen. Somit können hohe Verzögerungen schlechter durch Paketverlust reflektiert werden, je grösser die Queues auf dem Pfad zwischen Sender und Empfänger sind.

6.3.2 Out-of-band-Prüfen

Wird out-of-band geprüft, kann die Verzögerung durch die Verwendung von kleinen Pufferspeichern in den Knoten begrenzt werden. In Abbildung 6.4(b) sieht man, dass bei out-of-band Dropping die Verzögerung mit 27.4 ms (27.5 ms bei Puffergrösse 3) recht klein ist. Zieht man noch die Signalverzögerung von 20 ms ab, bleiben nur noch ein paar Millisekunden für die Queue-Verzögerung übrig. Gleichzeitig erreicht die akzeptierte Last ansprechende Werte, bei Überlastung des Links liegt sie über 0.75.

Ganz anders präsentiert sich die Situation bei out-of-band Marking. Abbildung 6.4(b) zeigt, dass die Puffergrösse in der Queue ein kritischer Parameter ist. Bei einer Puffergrösse von zwei Paketen nimmt die Auslastung unakzeptable Werte an. Die übriggebliebene Bandbreite geht zwar nicht verloren, da sie vom Best-Effort-Service verwendet werden kann, dennoch wird sehr wenig kontrollierter Verkehr akzeptiert. Eine mögliche Erklärung könnte in der Verwendung der virtuellen Queue liegen. Pakete werden bekanntlich markiert, sobald Pakete in der virtuellen Queue verloren gehen. Da diese nochmals kleiner ist als die reale Queue, geschieht dies unter Umständen sehr bald. Der auf 0 festgesetzte Schwellenwert ϵ lässt indes keine markierten Pakete zu und weist deshalb sehr viele Flüsse fälschlicherweise zurück. Eine Erhöhung des Pufferspeichers um eins resultiert bereits in einer deutlich besseren Leistungsfä-

higkeit. Die Verzögerungszeiten bei 4-facher Last bleiben dabei in etwa gleich klein.

Die Relevanz der Puffergrösse bringt einen grossen Nachteil mit sich. Sie kann von den Endpunkten nicht eingestellt werden, sondern muss in den Kern-Routern gesetzt werden. Es ist somit schwierig mit den hier vorgestellten EAC-Verfahren die resultierende Paketverzögerung zu regulieren.

6.4 Virtual Dropping

Wie in Abschnitt 6.2 gesehen, unterscheiden sich die Resultate der beiden Algorithmen erheblich, die Prüfpakete mit einer separaten Priorität versehen. Der wirkliche Unterschied zwischen out-of-band Marking und out-of-band Dropping ist aber nicht die Verwendung von Markierungen, sondern die Verwendung einer virtuellen Queue wie sie in Abschnitt 3.5.3 beschrieben wurde. Man könnte nun die Idee der virtuellen Queue verwenden, um zu entscheiden, wann etwas mit Prüfpaketen gemacht werden soll. Anstatt sie zu markieren, könnten sie lediglich verworfen werden. Dieser *Virtual Dropping* Ansatz würde kein ECN-Bit mehr brauchen, aber immer noch früh ein Signal für Überlastung liefern. Dies ist aber nur mit out-of-band Marking möglich, da die Prüfpakete in einer eigenen Queue gespeichert werden, welche es dem Router ermöglicht, Prüfpakete und nicht Datenpakete zu verwerfen. Im Zusammenhang mit in-band Marking ist Virtual Dropping unbrauchbar, da als unerwünschter Nebeneffekt zusätzlich noch Datenpakete verloren gehen würden.

Die Idee des Virtual Dropping entstammt aus [8], wo auch am Rande erwähnt wird, dass dieser Ansatz gleich gute Resultate liefern müsste wie out-of-band Marking. Es wurde bis jetzt noch nirgends bewiesen, dass diese Behauptung stimmt. Um diese Lücke zu schliessen, wurde die Unterstützung für Virtual Dropping dem Simulations-Framework hinzugefügt und die Simulationensergebnisse mit denen von out-of-band Marking verglichen.

6.4.1 Basisszenario

Abbildung 6.5 zeigt die Resultate von Virtual Dropping gegenüber out-of-band Marking unter Verwendung des in Abschnitt 6.2.1 beschriebenen Szenarios. Es lässt sich auf Anhieb erkennen, dass beide Verfahren in Bezug auf die Verlustwahrscheinlichkeit und die Auslastung sehr ähnliche Resultate liefern. Beide erreichen für die verschiedenen Werte von ϵ eine Verlustrate im Bereich von 10^{-5} und eine Auslastung zwischen 70% und 74%. Out-of-band Marking hat zwar die etwas kleinere Verlustwahrscheinlichkeit, dafür kann Virtual Dropping eine höhere Auslastung vorweisen. Die Unterschiede sind aber minim und täuschen in der Abbildung ein bisschen, da der Graph in y-Richtung gestreckt wurde.

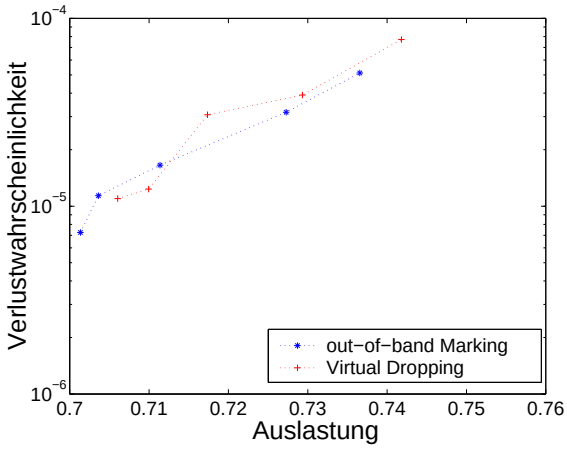


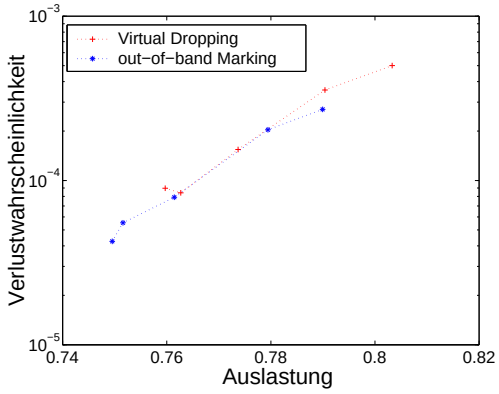
Abbildung 6.5: Basisszenario mit Virtual Dropping

6.4.2 Erweiterte Verkehrsquellen

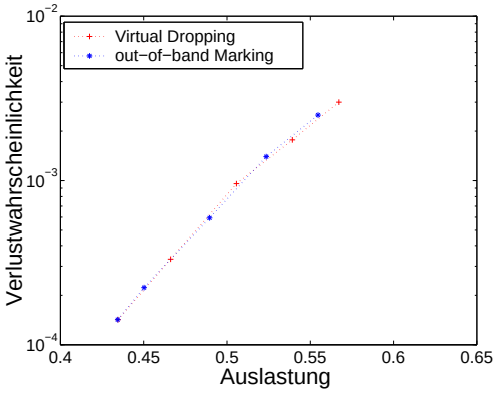
Im Folgenden wird die Leistungsfähigkeit von Virtual Dropping unter hoher Last und mit weiteren Verkehrsquellen unterschiedlicher Charakteristik evaluiert. Dies beinhaltet Pareto-verteilte Quellen, Quellen mit hoher Burst-Rate und eine Spur von realem Verkehr. Letzteres umfasst den generierten Verkehr einer Videoanwendung, die einen MPEG-codierten Videofilm ins Netz sendet. Die Verkehrsinformationen werden dem Simulator mittels einer Spur-Datei (Trace-File) zur Verfügung gestellt, in welcher Einträge mit zwei Feldern gespeichert sind. Das erste Feld enthält die Zeit bis das nächste Paket generiert wird, das zweite die Länge des nächsten Pakets.

In Abbildung 6.6 sind die Resultate von vier Szenarien wiederum mit der Verlustwahrscheinlichkeit und der Auslastung dargestellt. In Abbildung 6.6(a) wurde das Szenario aus Abschnitt 6.2.2 verwendet, bei dem die Flüsse mit geringen Abständen ankommen. In Abbildung 6.6(b) und 6.6(c) besteht der Verkehr aus On/Off-Quellen mit exponentieller Verteilung bzw. Pareto-Verteilung. In Abbildung 6.6(d) ist die Quelle eine Spur vom Film Star Wars [9], welcher eine Paketgrösse von 200 Bytes verwendet. Die Spur wurde so umgeformt, dass sie einem Token Bucket mit $r = 800$ kBit/s und $B = 200$ kb entspricht [8].

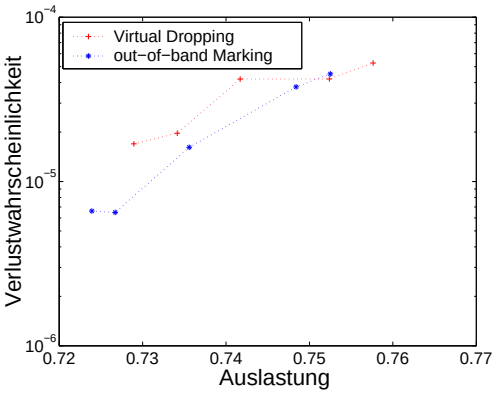
Es wird nicht jeder Graph von Abbildung 6.6 detailliert diskutiert, dafür wird eine kurze Zusammenfassung der Resultate geliefert. In Abbildung 6.6(b) kann man sehr schön sehen, wie Virtual Dropping und out-of-band Marking fast genau die gleichen Resultate liefern. Bei hoher Last ist der Unterschied etwas grösser. Der Markierungsmechanismus kann hier die Verlustrate sehr tief halten. Dies kann dadurch erklärt werden, dass das Signal für Überlastung die Quelle unter Umständen schneller erreicht als bei Virtual Dropping. Zur Veranschaulichung nehmen wir an, dass zu einer Zeit t die virtuelle Queue im ersten Knoten voll wird und kurz darauf eine grosse Anzahl von Prüfpaketen ankommt. Die Queue muss wiederum kurze Zeit später für die Dauer der



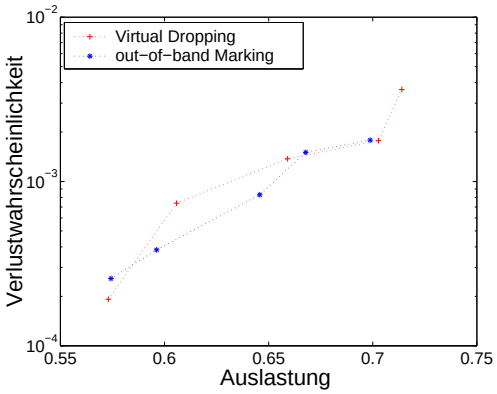
(a) Hohe Last



(b) Exponentieller Verkehr mit einer Burst-Rate von 1024 kBit/s. On-Zeit 125 ms, Off-Zeit 875 ms.



(c) Pareto On/Off. Burst-Rate 256 kBit/s, On-Zeit 500 ms, Off-Zeit 500 ms.



(d) Star Wars Trace

Abbildung 6.6: Experimente mit unterschiedlichen Verkehrscharakteristiken

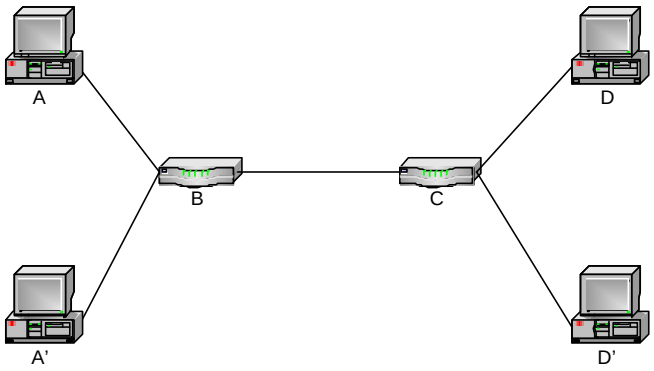


Abbildung 6.7: Topologie für das Multi-Link-Szenario

Überlastung jedes eintreffende Prüfpaket verwerfen. Betrachten wir nun einen prüfenden Fluss, der bis zum Zeitpunkt t alle Pakete übertragen konnte, danach aber bis zum Ablauf der Prüfzeit keine mehr. Im Falle von out-of-band-Marking ist dies nicht so schlimm. Der Empfänger bekommt nach Eintritt der Überlastung noch mehrere markierte Pakete und kann dem Sender eine korrekte Rückmeldung geben. Im Falle von Virtual Dropping weiss der Empfänger nichts von der aufgetretenen Überlastung, da der berechnete Paketverlust null zu sein scheint. Dieser Wert ist falsch, da der Empfänger verlorene Pakete nur durch Lücken in den Sequenznummern der Pakete erkennen kann und eine solche Lücke im schlechtesten Fall nie entsteht. Dieses Problem könnte dadurch gelöst werden, dass der Sender dem Empfänger die Gesamtzahl der zu sendenden Pakete mitteilt. Am Ende der Prüfphase könnte der Empfänger in jedem Fall die Anzahl der verlorenen Pakete korrekt bestimmen. Dieses Verfahren ändert jedoch nichts an der Tatsache, dass die Information, dass eine Überlastung im Netz aufgetreten ist, unter Umständen später eintrifft als bei der Verwendung von Paketmarkierungen.

6.4.3 Multi-Link-Szenario

Bis jetzt beruhten alle Simulationen auf einer Topologie mit nur einem Link. Was passiert bei mehreren Links? Um diese Frage zu beantworten wird die Topologie in Abbildung 6.7 verwendet. Flüsse existieren im Multi-Link-Szenario von den Endpunkten A bzw. A' nach D bzw. D'. Die Wahl des jeweiligen Zielknotens wird dabei von der Simulation zufällig bestimmt. Es wurde ein homogenes Szenario simuliert, in dem alle Links eine Bandbreite von 10 Mbit/s und eine Verzögerung von 20 ms haben. Der Schwellenwert ϵ ist auf 0.05 fixiert.

Tabelle 6.1 zeigt die Verlustwahrscheinlichkeit über dem überlasteten Link von Knoten B nach D und die durchschnittliche Zurückweisungswahrscheinlichkeit der vier möglichen Gruppen von Flüssen. Wie man sehen kann ist die Anzahl der zurückgewiesenen Flüsse in allen Verfahren über 50%. Dies liegt daran, dass der Link zwischen B und D als Flaschenhals wirkt und dadurch

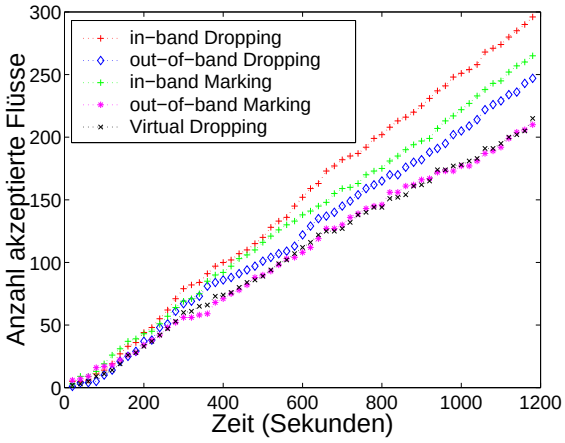


Abbildung 6.8: Anzahl akzeptierter Flüsse im Verlauf der Simulation

viele Pakete verworfen oder markiert werden. Die Diskussion in Abschnitt 6.2.1 zeigte, dass in-band Dropping eine hohe Auslastung aufweist. Der Grund dafür liefert die gemessene Zurückweisungswahrscheinlichkeit in Tabelle 6.1. Sie ist im Vergleich zu den anderen Prüfverfahren recht tief. Da mehr Flüsse akzeptiert werden, ist aber auch die Verlustrate entsprechend hoch. Betrachtet man die Werte von out-of-band Marking und Virtual Dropping, kann man wiederum sehen, wie ähnlich sich diese beiden Verfahren sind. Die Rate der blockierten Flüsse ist praktisch identisch.

Auch in Abbildung 6.8, wo die Anzahl akzeptierter Flüsse im Intervall von 20 Sekunden aufgezeichnet sind, zeigt sich das gleiche Bild. Die Kurven von out-of-band Marking und Virtual Dropping sind annähernd deckungsgleich, die anderen heben sich hingegen recht deutlich voneinander ab.

Algorithmus	Zurückweisungs-wahrscheinlichkeit	Zurückweisungs-wahrscheinlichkeit
In-band Dropping	0.54754	0.03100
Out-of-band Dropping	0.60280	0.00609
In-Band Marking	0.59869	0.00511
Out-of-band Marking	0.65222	0.000041
Virtual Dropping	0.65334	0.000079

Tabelle 6.1: Zurückweisungs- und Verlustwahrscheinlichkeiten im Multi-Link-Szenario

6.5 Zusammenfassung der Simulationsergebnisse

Die zu Beginn dieses Kapitels präsentierten Simulationsergebnisse haben gezeigt, dass das Simulations-Framework korrekte Resultate liefert. Die Messungen weichen von denen in der Literatur nur wenig ab.

Die Simulationen mit einem IP-Telefonie-Szenario haben ergeben, dass EAC-Verfahren, bei denen nicht zwischen Prüf- und Datenverkehr unterschieden wird, keine Garantien bezüglich der Paketverzögerung liefern können. Die Indikation des Paketverlusts an den Endpunkten reflektiert die vorhandene Paketverzögerung im Netzwerk nur schlecht. Verfahren, die Prüf- und Datenverkehr separieren, können hingegen eine Verzögerung von wenigen Millisekunden garantieren. Die Puffergrösse für den Prüfverkehr in den Routern darf dazu nur wenige Pakete gross sein.

Desweitern wurde mit Virtual Dropping ein Ansatz evaluiert, dem eine virtuelle Queue zu Grunde liegt, der aber keine ECN-Unterstützung an den Endpunkten braucht. Dieser Ansatz liefert in den meisten Fällen die gleichen Ergebnisse wie out-of-band Marking. Bei hoher Last ist die Verlustrate etwas höher, da die Information, dass eine Überlastung im Netz aufgetreten ist, unter Umständen später eintrifft als bei der Verwendung von ECN.

7 Zusammenfassung und Ausblick

Verschiedene Arbeiten haben Endpoint Admission Control als mögliche Alternative zu IntServ vorgeschlagen, um einen Soft-Echtzeitdienst unterstützen zu können. Statt komplexe und nicht-skalierbare Signalisierungsprotokolle zu verwenden, kombiniert EAC durchdachte, Host-basierte Algorithmen mit einer traditionellen Best-Effort-Infrastruktur, die nur zwei zusätzliche Prioritätsstufen verlangt.

Die Diskussion der Architektur von EAC hat zwei grundlegende Entwürfe nahegelegt: Sollen zur Indikation von Überlastung Pakete verworfen oder markiert werden, und soll in-band oder out-of-band geprüft werden. Eine Kombination der beiden Varianten ist das sogenannte Virtual-Dropping-Verfahren. Dieses Verfahren benutzt die Idee der virtuellen Queue, um Überlastung zu erkennen. Anstatt Pakete zu markieren, werden sie aber einfach verworfen.

Unabhängig vom grundlegenden Entwurf können zum Prüfen des Netzwerks unterschiedliche Strategien angewandt werden. Eine effiziente Strategie ist das Slow-Start-Verfahren, bei welchem die Übertragungsrate des Prüfverkehrs vom Sender langsam erhöht wird. Durch die Verwendung des Slow-Start-Verfahrens wird eine bevorstehende Überlast im Netzwerk frühzeitig erkannt und das Problem des Thrashings gelöst.

Das während der Arbeit entstandene Simulations-Framework stellt die grundlegende EAC-Architektur zur Verfügung und implementiert die gängigsten Prüfstrategien. Das Framework sollte die zukünftige Forschung im Bereich EAC erleichtern, da es so ausgelegt ist, dass neue Verfahren einfach evaluiert werden können. Abhängig von der Komplexität der zu testenden Architektur sind zusätzliche Implementierungsschritte notwendig.

Um das Framework überprüfen zu können, wurden Simulationen mit den ersten drei Ansätzen aus Abschnitt 3.5 durchgeführt und die Resultate mit denen aus der Literatur verglichen. Es konnte gezeigt werden, dass das Framework korrekte Resultate liefert. Weiter wurde der Virtual-Dropping-Ansatz implementiert und Messungen durchgeführt. Die Simulationsergebnisse zeigen, dass Virtual Dropping die gleiche Leistung erreichen kann wie das entsprechende Markierungsverfahren. Die Analyse der Paketverzögerung hat ergeben, dass es mit den EAC-Verfahren, die auf Paketverlust basieren, schwierig ist, die resultierende Verzögerung zu kontrollieren. Mit dem in Abschnitt 3.5.5 erwähnten Ansatz, der auf Verzögerungsschwankungen basiert, könnte die Verzögerung besser kontrolliert werden. Es wäre durchaus interessant, diesen Ansatz mit den traditionellen EAC-Konzepten zu vergleichen. Es könn-

te sich desweitern lohnen, ein Verfahren zu untersuchen, das sowohl Verzögerungsschwankungen als auch Paketverlust zur Indikation von Überlastung verwendet.

Ein Thema, das sicherlich noch untersucht werden muss, ist das Zusammenspiel von EAC und Multicast. Es gibt im Moment noch keinen Ansatz, der skalierbare Multicast-Sitzungen unterstützt. Verschiedene Ideen sind vorhanden, konkrete Ergebnisse fehlen aber. Das Simulations-Framework kann zur Analyse aller erwähnten Themen verwendet werden. Um eine Multicast-fähige Zugangskontrolle unterstützen zu können, wären wohl einige Erweiterungen notwendig. Mit weniger Aufwand verbunden wäre hingegen die Implementierung einer Architektur, die auf Verzögerungsschwankungen basiert.

A Anhang

A.1 Liste der Änderungen im ns-2-Quellcode

Die folgende Liste enthält die Dateien, die während der Entwicklung in der ns-2.1b8-Distribution geändert werden mussten. Die zweite Spalte gibt jeweils an, welche Änderungen vorgenommen wurden.

Datei/en	**Änderungen**
trafgen.h/app.h	Methoden `start()` und `stop()` öffentlich gemacht.
packet.h	Pakettypen `PT_PROBE` und `PT_EADATA` hinzugefügt. Neues Feld im Common-Header hinzugefügt.
queue.h	Methode `length()` virtuell gemacht, damit abgeleitete Klassen sie überschreiben können.
ns-lib.tcl	Die Methode `simplex-link` erweitert, sodass EADC-Links kreiert werden können. Neue Zeile für den Aufruf des link-Kommandos einer GK-Queue hinzugefügt. Methode `monitor-queue-ptype` zur Erstellung eines PTQueueMonitor-Objekts hinzugefügt.
ns-default.tcl	Standardwerte für die Parameter der neu hinzugefügten Klassen definiert.
ns-link.tcl	Methode `init-monitor-ptype` hinzugefügt. Diese Methode initialisiert einen PTQueueMonitor.

A.2 Liste der hinzugefügten Dateien

Die folgende Liste enthält die wichtigsten Dateien des Frameworks. Sie befinden sich im Unterverzeichnis */eac.*

Datei	Beschreibung
ProbingStrategy.h/.cc	Abstrakte Klasse für EAC-Algorithmen.
SimpleStrategy.h/.cc	Einfacher Algorithmus mit fixer Prüfrate.
SlowStart.h/.cc	Implementierung des Slow-Start-Algorithmus.
EarlyReject.h/.cc	Implementierung des Early-Reject-Algorithmus.
AdmissionMonitor.h/.cc	Hilfsklasse, die als Zähler für die angenommenen und zurückgewiesenen Flüsse dient.
EAAgent.h/.cc	Repräsentiert einen Endpunkt im Netzwerk, der als Quelle agiert.
EASink.h/.cc	Repräsentiert einen Endpunkt im Netzwerk, der als Empfänger agiert.
EAQueue.h/.cc	Queue für einen EAC-fähigen Router.
PriorityQueue.h/.cc	Queue mit mehreren Prioritätsstufen.
RateLimiter.h/.cc	Schnittstelle für einen Ratenbegrenzer.
TSWRateLimiter.h/.cc	Ratenbegrenzer, der mit dem Time-Sliding-Window-Algorithmus funktioniert.
GKPriority.h/.cc	Erweiterte GK-Queue mit Unterstützung von mehreren Prioritätsstufen.
PTQueueMonitor./.cc	Klasse zur Überwachung einer Queue. Speichert Informationen pro Pakettyp.

A.3 Installationsanleitung für das Simulations-Framework

Das Simulations-Framework wurde für ns-2.1b8 geschrieben, es sollte aber auch mit anderen ns-Versionen genutzt werden können. Es ist als Paket mit Quellcode erhältlich und muss zusammen mit dem installierten ns-Code kompiliert werden. Es wird vorausgesetzt, dass ns-2 bereits korrekt installiert und lauffähig ist.

Folgende Schritte sind zur Installation des Frameworks notwendig:

1. Laden Sie die Datei *eac_patch.tar.gz* herunter.
2. Wechseln Sie ins Wurzelverzeichnis der ns-Distribution (typischerweise ns-2.1b8) und öffnen Sie das Archiv wie folgt:
   `tar xzvf <dir>/eac_patch.tar.gz`
3. Führen Sie das `install`-Script im *eac*-Unterverzeichnis aus. Mit diesem Script werden die in Anhang A.1 aufgelisteten Dateien gepacht.
4. Fügen Sie die folgenden Zeilen in den Abschnitt *OBJ_CC* des Makefiles im ns-Wurzelverzeichnis hinzu:

```
eac/EAAgent.o eac/EASink.o eac/PTQueueMonitor.o \
```

```
eac/ProbingStrategy.o eac/EAQueue.o \
eac/PriorityQueue.o eac/TSWRateLimiter.o gk.o \
eac/GKPriority.o eac/TBPolicer.o eac/AdmissionMonitor.o \
eac/SimpleStrategy.o eac/SlowStart.o eac/EarlyReject.o \
```

5. Führe Sie `make clean` gefolgt von `make` aus.

6. Lassen Sie die Tests für das Framework laufen. Die fünf Tests sollten fehlerfrei beendet werden.
   ```
   cd tcl/test
   ./test-all-endpoint-adc quiet
   ```

A.4 Default-Werte in ns-default.tcl

```
Agent/EndpointADC set packetSize_ 1000
Agent/EndpointADC set lifetime_ 300
Agent/EndpointADC set expirationTime_ 10.0
Agent/EndpointADC set showDecision_ false
PrStrategy set epsilon_ 0.0
PrStrategy set probingTime_ 5.0
PrStrategy set debug_ false
PrStrategy/Simple set rate_ 128k
PrStrategy/EarlyReject set checkTime_ 1.0
PrStrategy/EarlyReject set rate_ 128k
PrStrategy/SlowStart set factor_ 2.0
PrStrategy/SlowStart set periodTime_ 1.0
PrStrategy/SlowStart set rate_ 128k
AdmissionMonitor set fRejected_ 0
AdmissionMonitor set fAccepted_ 0
Queue/GK set ecnlim_ 0.95
Queue/GK set mean_pktsize_ 1000
Queue/GK set curq_ 0
Queue/GK set drop_front_ 0
Queue/GK/Priority set qlimit1_ 75
Queue/GK/Priority set qlimit0_ 75
Queue/GK/Priority set markMethod_ true
Queue/EndpointADC set beQueueLimit_ 200
```

Glossar

AF (Assured Forwarding): Eine spezifische PHB-Gruppe, die IP-Pakete in vier Klassen unterteilt.

ATM (Asynchronous Transfer Mode): Eine von der ITU für Fernnetze mit hohen Bitraten entwickelte Norm, die auf dem Prinzip des asynchronen Zeitmultiplexverfahrens basiert.

AQM (Active Queue Management): Zusammenfassung von Mechanismen, die die Queueing-Verzögerung innerhalb von Routern kontrollieren. AQM-Mechanismen zeigen Endsystemen Netzüberlastung an, indem sie Pakete verwerfen oder markieren.

BA (Behaviour Aggregate): Ein Behaviour Aggregate bezeichnet eine Menge von Paketen mit demselben DS-Codepunkt, die in eine gewisse Richtung fliessen.

Bandwidth Broker: Eine Komponente, die Ressourcen in einem DiffServ-Netz verwaltet.

Best-Effort-Service: Das Standard-Verhalten eines IP-Netzwerks ohne QoS. Datenpakete werden so schnell wie möglich und mit geringstem Aufwand zugestellt.

CBQ (Class Based Queueing): Paket-Scheduling-Mechanismus, der die Verteilung der Übertragungsraten zwischen unterschiedlichen Verkehrsklassen festlegt.

CBR (Constant Bit Rate): Eine Datenübertragung mit konstanter Bit-Rate.

Classifier: Eine Einheit, die Pakete aufgrund ihres Headers und vordefinierter Regeln auswählt.

DiffServ (Differentiated Services): Ein von der IETF entwickeltes Framework, um skalierbare und flexible Service-Differenzierung zur Verfügung zu stellen.

DS-Codepunkt: Ein spezifischer Wert des DSCP-Teils im DS-Feld.

DS-Domäne: Zusammenhängendes Netz von Knoten mit einer gemeinsamen DiffServ-Konfiguration.

DS-Feld: Das TOS-Feld in IPv4-Headern oder das Traffic-Class-Feld in IPv6-Headern.

Dropper: Einheit, die Pakete aufgrund definierter Regeln verwirft.

EAC (Endpoint Admission Control): Zugangskontrolle, die auf aktiven Messungen durch Endpunkte basiert.

ECN (Explicit Congestion Notification): Ein Verfahren, bei dem ein Router durch eine einfache Markierung eines Bits im IP-Header eine drohende Überlastung mitteilen kann.

EF (Expedited Forwarding): Ein PHB, der dazu verwendet wird, um eine virtuelle Standleitung zu kreieren.

Egress-Knoten: Ein Grenzknoten, der den Datenverkehr aus der DS-Domäne zur benachbarten DS-Domäne bzw. zur nicht-DS-Domäne behandelt.

FEC (Forward Error Correction): Eine in der Datenübertragung angewandte Technik zur Erkennung und Korrektur von Übertragungsfehlern.

Flowspec: Flussspezifkation. Enthält TSpec und RSpec.

IETF (Internet Engineering Task Force): Eine zur Internet Society gehörende technische Arbeitsgruppe, die sich mit der Standardisierung von Entwicklungsstrukturen im Zusammenhang mit dem Internet befasst.

Ingress-Knoten: Ein Grenzknoten, der den von aussen kommenden Verkehr beim Eintritt in die DS-Domäne behandelt.

IntServ (Integrated Services): Ein von der IETF entwickeltes Framework, um individuelle Dienstgüte für individuelle Datenströme zur Verfügung zu stellen.

Mikrofluss: Der Datenfluss zwischen zwei Anwendungen.

Marker: Eine Einheit, die Pakete mit einem bestimmten Codepunkt versieht.

MPEG (Motion Picture Expert Group): Normungsgruppe mit dem Ziel, die Standardisierung von Regeln zur Kompression von Laufbildern zu erreichen.

ns-2 Eine Netzwerksimulationssoftware, die vom Information Sciences Institute (ISI) der University of Southern California (USC) entwickelt wurde.

RED (Random Early Detection): Algorithmus zur Verbesserung der TCP-Staukontrolle, der innerhalb von Routern durchgeführt wird.

PHB (Per-Hop-Behavior): Das Verhalten eines DS-Knotens gegenüber einem Behaviour Aggregate.

RIO (Random Early Detection In and Out) Erweiterter RED-Mechanismus, der zwei RED-Algorithmen zwei Prioritäten zuweist.

Rspec (Request Specification): Anforderungsspezifikation, die die vom Netzwerk verlangte Dienstgüte beschreibt.

RSVP (Resource Reservation Setup Protocol): Signalisierungsprotokoll, das im Zusammenhang mit IntServ entwickelt wurde.

SLA (Service Level Agreement): Vertrag, der den Service definiert, den ein Kunde erwarten kann.

Shaping: Das Begrenzen eines Datenstroms auf eine gewisse Paket- oder Bitrate durch verzögertes Senden der Pakete.

TCP (Transmission Control Protocol): Zuverlässiges verbindungsorientiertes Protokoll der Transportschicht.

TSpec (Traffic Specification): Verkehrsspezifikation, die die Variation der angeforderten Bandbreite über der Zeit beschreibt.

Token-Bucket-Filter: Ein Mechanismus, der gebraucht werden kann, um Verkehr zu charakterisieren.

TOS (Type of Service): Feld im IPv4-Header zur Priorisierung von Paketen nach unterschiedlichen Kriterien.

Traffic Conditioning: Erzwingen gewisser Regeln bezüglich eines Datenstroms durch Metering, Marking, Shaping und/oder Policing.

Policing: Das Verwerfen bestimmter Pakete mit Hilfe eines Droppers.

QoS (Quality of Service): Zusammenfassung verschiedener Kriterien, die die Güte eines Dienstes im Netz hinsichtlich Geschwindigkeit und Zuverlässigkeit beschreiben.

UDP (User Datagram Protocol): Unzuverlässiges verbindungsloses Protokoll der Transportschicht.

VBR (Variable Bit Rate): Eine Datenübertragung mit variabler Bit-Rate.

WFQ (Weighted Fair Queueing): Mechanismus für das Paket-Scheduling innerhalb von Routern.

WRED (Weighted RED): WRED kombiniert die Möglichkeiten des RED-Algorithmus mit den IP-Prioritäten, um eine bevorzugte Behandlung von Paketen mit höherer Priorität zu erreichen

Zugangskontrolle (Admission Control): Eine Routine zur Überprüfung der Gewährbarkeit einer QoS-Anforderung.

Literaturverzeichnis

[1] An Architecture for Differentiated Services,
Blake, Black, Carlson, Davis, Wang, Weiss
Internet RFC 2475, IETF 1998

[2] Resource Reservation Protocol,
Braden, Zhang, Berson, Herzog, Jamin,
Internet RFC 2205, IETF 1997

[3] Integrated Services in the Internet Architecture: an Overview,
R. Braden, D. Clark, S. Shenker,
Internet RFC 1633, IETF 1994

[4] A Framework for Integrated Services Operation over DiffServ Networks,
Y. Bernet u. a.,
Internet RFC 2998, IETF 2000

[5] A Proposal to add Explicit Congestion Notification (ECN) to IP,
K. Ramakrishnan, S. Floyd
Internet RFC 2481, IETF 1999

[6] A Report on Some Recent Developments in TCP Congestion Control
S. Floyd,
IEEE Communications Magazine, April 2001

[7] Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service,
S. Jamin, S. J. Shenker, P. B. Danzig,
IEEE Infocom 1997

[8] Endpoint Admission Control: Architectural Issues and Performance,
L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, H. Zhang,
ACM Sigcomm 2000

[9] Analysis, modeling and generation of self-similar VBR video traffic,
M. Garret, W. Willing,
Proc. ACM Sigcomm, September 1994

[10] A proposal to add explicit congestion notification (ECN) to IP,
K. Ramakrishnan, S. Floyd,
Internet RFC 2481, 1999

[11] Measurement-Based Admission Control for Bufferless Multiplexers,
M. Reisslein,
International Journal of Communication Systems, Vol. 14, Oktober 2001

[12] Admission Control on End-to-End Measurements,
V. Elek, G. Karlsson, R. Rönngren,
IEEE Infocom 2000

[13] Throughput Analysis of End-to-End Measurement-Based Admission Control in IP,
G. Bianchi, A. Capone, C. Petrioli,
IEEE Infocom 2000

[14] Endpoint Admission Control with Delay Variation Measurements for QoS in IP Networks,
G. Bianchi, F. Borgonovo, A. Capone,
ACM Sigcomm Computer Communications Review, Vol. 32, April 2002

[15] Egress Admission Control,
C. Cetinkaya, V. Kanodia, E.W.Knightly,
IEEE Infocom 2000

[16] Scalable Services via Egress Admission Control,
C. Cetinkaya, V. Kanodia, E.W.Knightly,
IEEE Transactions on Multimedia: Special Isuue on Multimedia over IP, Vol. 3, No. 1, März 2001

[17] Distributed Admission Control
F. P. Kelly, P. B. Key, S. Zachary,
IEEE Journal on Selected Areas in Communications, Vol. 18, 2000

[18] Distributed connection acceptance control for a connectionless network,
R. J. Gibbens, F. P. Kelly
Teletraffic Congress Edinburgh, 1999

[19] Resource pricing and the evolution of congestion control
R. J. Gibbens, F. P. Kelly
Automatica, Vol. 35, 1999

[20] A Diff-Serv Enhanced Admission Control Scheme,
R. Hill, HT Kung,
IEEE Globecom 2001

[21] Endpoint Admission Control: Network Based Approach,
B. K. Choi, R. Bettati,
Proceedings of the International Conference on Distributed Computing Systems, April 2001

[22] The ns Manual (formerly ns Notes and Documentation),
K. Fall, K. Varadhan
http://www.isi.edu/nsnam/ns/ns-documentation.html

[23] Opnet Website, http://www.opnet.com

[24] Explicit Allocation of Best effort Delivery Service,
D. Clark and W. Fang,
IEEE/ACM Transactions on Networking, Vol. 6, 1998

[25] University of Berne Linux Cluster
http://www.clc.unibe.ch/