Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Masterarbeit

Design, Implementierung und Analyse eines flexiblen Dienstgütemodells für zeitsensitive Kommunikationsdienste

Robin Laidig

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer/in:	Dr. rer. nat. Frank Dürr
Beginn am:	03. Mai 2021
Beendet am:	03. November 2021

Kurzfassung

Viele vernetzte Anwendungen, wie beispielsweise vernetzte Regelungssysteme, sind latenzsensitive Systeme, die beschränkte maximale Verzögerungen im Netzwerk benötigen. Typische Quality-of-Service-Modelle (QoS-Modelle) unterteilen darum den Netzwerkverkehr in die beiden Klassen „Echtzeit“ und „Best-Effort“. Solange sich ein Strom an die vereinbarte Verkehrsspezifikation hält, erhält sein gesamter Netzwerkverkehr die Garantien des Echtzeitverkehrs. Diese Garantien sind aus Sicht des Netzwerks teuer, da sie reservierte Ressourcen benötigen. Es muss aber nicht für jede Netzwerkübertragung einer Anwendung eine deterministische Garantie für die Verzögerung vorhanden sein. Beispielweise genügt es für die Stabilität eines Networked Control System (NCS), wenn nur eine wohldefinierte Teilmenge an Sensorwerten mit deterministischen Echtzeitgarantien transportiert wird. Alle darüber hinaus erhaltenen Werte sind nicht mehr kritisch für die Stabilität, können aber die Performance (Regelgüte) des Systems verbessern. Für diese Werte genügt es, wenn sie mit schwächeren Garantien oder gar mit Best-Effort-Garantien übermittelt werden. An diesem Beispiel wird deutlich, dass ein Spektrum an Qualität existiert, das von garantierten kleinen Latenzen bis zu keinen Latenzgarantien reicht. Das typische zwei-Klassen-Modell bietet jedoch nicht die nötige Flexibilität, um dieses Spektrum abzudecken. Hierfür werden flexiblere QoS-Modelle benötigt, die eine feingranulare Abstufung zwischen der Echtzeit- und Best-Effort-Klasse unterstützen. Ein möglicher Ansatz ist die Erweiterung des bekannten IntServ-Modells, welches im ersten Teil dieser Arbeit untersucht wird. Um die Skalierungsprobleme von IntServ zu überwinden, wird außerdem ein neu konzipiertes Modell auf Basis einer Multi-Level-Feedback-Queue (MLFQ) vorgestellt. Dieses Modell benötigt keine separaten Warteschlangen pro Strom bei jedem Router und ermöglicht es, Ströme anhand ihres Sendeverhaltens dynamisch in verschiedene Prioritäten einzuordnen. Dabei ist die Einordnung für die Anwendung berechenbar, sodass ihr ein Anreiz zur Selbstregulierung gegeben wird. Anwendungen, die sich aus Sicht des Netzwerks kooperativ verhalten und wenige, kleine Bursts senden, sollen durch eine höhere Priorität mit einem besseren Quality-of-Service belohnt werden. Je höher die Priorität eines Stroms bei den einzelnen Routern ist, desto bessere Garantien und niedrigere Verzögerungen erhält er. Gleichzeitig profitiert das Netzwerk von den kooperativen Strömen, da sie einen geringeren Ressourcenverbrauch und eine bessere Ressourcennutzung ermöglichen. Für dieses Modell werden sowohl die konzeptionellen Eigenschaften und Berechnungen der Garantien gezeigt, als auch praktische Implementierungsdetails. Die Funktionsweise und Parametrisierung dieses neuen Modells werden zum Abschluss mit einem Netzwerksimulator evaluiert. Dabei konnte gezeigt werden, dass die berechneten Garantien zuverlässig eingehalten werden und die Verzögerungen häufig sogar über 50% geringer sind. Außerdem wurde gezeigt, dass das Verfahren die Ausgangsbandbreite eines Routers zu 100% nutzen kann und keine Bandbreite durch die zugesicherten Garantien verloren geht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

MLFQ Multi Level Feedback Queue

NCS Networked Control System

QoS Quality of Service

WFQ Weighted Fair Queueing

1 Einleitung

Die Vision des Internets der Dinge (englisch: Internet of Things, IoT), bei welchem nahezu alle Geräte über das Internet miteinander vernetzt sein werden, wird zunehmend Realität. Sowohl in „intelligenten“ Alltagsgegenständen wie der Kaffeemaschine, die automatisch Kaffee nachbestellt, oder den Lampen, die über eine „Smart Home“ App ferngesteuert werden, als auch in der Industrie zeigt sich dieser Trend. In der Industrie ist dabei die Rede von einer intelligenten Fabrik, welche auch als „Industrie 4.0“ oder im Englischen als „Industrial Internet of Things (IIoT)“ bezeichnet wird. Dort werden die Maschinen, Roboter und Werkzeuge vernetzt, um die Abläufe zu automatisieren und die menschlichen Arbeiter*innen optimal zu unterstützen.
Aus technischer Sicht handelt es sich bei diesen „intelligenten“ Geräten um eingebettete vernetzte Systeme, die mit Mikrocontrollern oder Kleinstrechnern ausgestattet sind. Sie verfügen über Kommunikationstechnologien wie Bluetooth, WLAN oder Ethernet und häufig zusätzlich über Sensoren und Aktoren, um mit der Umgebung zu interagieren. In diesem Kontext wird daher auch häufig der Begriff „Cyber-Physical System“ verwendet, welcher aussagt, dass physische Prozesse von Computersystemen gesteuert und geregelt werden. Solche Systeme werden daher oft auch als vernetzte Regelungssysteme (engl. Networked Control Systems) bezeichnet, um die Steuerungs- und Regelungsfunktionalität zu betonen.
Die wesentlichen Komponenten eines vernetzten Regelungssystems sind eine Menge von vernetzten Sensoren, die den aktuellen Zustand der Anlage erfassen und übermitteln. Auf die Sensoren reagieren die Regler und senden Steuersignale an die verschiedenen Aktoren, die sich in den Maschinen und Robotern befinden [DN16]. All diese Signale und Sensorwerte werden über ein Kommunikationsnetzwerk übertragen, welches alle Komponenten verbindet. Für diese Netzwerke, sowohl im Heimgebrauch als auch in der Industrie, haben sich die IEEE Standards 802 durchgesetzt. Der Standard 802.11 [IEE18b] spezifiziert beispielsweise das bekannte Funknetzwerk WLAN und der Standard 802.3 [IEE18a] das Kabelnetzwerk Ethernet.

Cyber-Physical Systems bzw. vernetzte Regelungssysteme sind generell latenzsensitive Systeme, die zur Steuerung und Regelung von sicherheitskritischen Anwendungen eingesetzt werden. Eine Verletzung der Zeitanforderungen bei der Kommunikation von Sensorwerten und Steuersignalen kann gravierende oder gar katastrophale Folgen haben. Ein Beispiel hierfür ist das Betätigen eines Notaus-Schalters innerhalb einer Fabrik. Es gibt Normen, die festlegen, wie lange es höchstens dauern darf, bis eine Maschine nach Betätigen dieses Schalters zum Stillstand kommt [DIN19]. Würde das Netzwerk versagen und die Informationen zu spät eintreffen, könnten schlimmstenfalls Menschen zu Schaden kommen.
Entsprechend hohe Anforderungen werden deshalb an die Kommunikationsmechanismen und Netzwerktechnologien gestellt: Sie müssen die Fähigkeit zur Echtzeitkommunikation besitzen. Das bedeutet, dass es eine definierte maximale Verzögerung gibt, nach welcher jedes Netzwerkpakete garantiert beim Ziel angekommen ist. Es muss also eine garantierte obere Schranke für die Verzögerung existieren.

Die Kommunikationstechnik unterstützt häufig allerdings nur ein Best-Effort Servicemodell, bei welchem es zu Verzögerungen und Verlusten kommen kann. Besonders häufig tritt dies auf, wenn das Netzwerk überlastet ist. Die Router, welche für das Weiterleiten von Netzwerkpaketen zuständig sind, haben nur eine beschränkte Bandbreite und Pufferplatz zur Verfügung. Falls der eingehende Netzwerkverkehr größer ist, als ihre Ausgangsbandbreite, füllt sich ihr Puffer. Noch bevor ihr Puffer voll wird, fangen Router typischerweise an Pakete zu verwerfen und machen dadurch die Sender auf die Überlastsituation aufmerksam. Bei einer anhaltenden Überlastung des Routers laufen die Puffer schließlich voll und der gesamte ankommende Netzwerkverkehr wird verworfen [FJ93]. Dadurch ist es möglich, dass manche Pakete erst nach vielen Versuchen oder schlimmstenfalls nie bei ihrem Ziel ankommen. Es gibt bei Best-Effort-Verkehr also im allgemeinen keine Garantie, wann ein Paket spätestens am Ziel ankommt und es kann auch nicht garantiert werden, dass es überhaupt ankommt. Dieses Verhalten ist nicht akzeptabel für Cyber-Physical Systems.

Bisherige Lösungsansätze garantieren für die Verzögerung eine obere Schranke, indem Ressourcen reserviert werden. Ein Ansatz besteht darin, pro Netzwerkstrom Bandbreite auf jedem Router zu reservieren, sodass schließlich eine garantierte minimale Bandbreite für den gesamten Strom zur Verfügung steht. Über diese minimale Bandbreite kann dann eine garantierte maximale Verzögerung berechnet werden. Diesen Ansatz verwendet z.B. IntServ mit dem „Resource Reservation Protocol (RSVP)“ [BCS94; ZDE+93]. Eine andere Möglichkeit besteht darin, die Belegung des Netzwerks in feste Zyklen mit mehreren Zeitschlitzen einzuteilen. Ein Zeitschlitz kann dann exklusiv einer Anwendung bzw. der Echtzeitkommunikation zugewiesen werden, gemäß dem „Time Division Multiple Access (TDMA)“-Prinzip. Dadurch kann anhand der Länge des Zeitschlitzes und der Frequenz des Zyklus eine garantierte Bandbreite und damit maximale Verzögerung berechnet werden. Dieser Ansatz wird beispielsweise von „Time-Sensitive Networking (TSN) [IEE16a]“ verfolgt.

Damit bleiben allerdings noch offene Probleme und Raum für Verbesserungen. Die bisherigen Lösungsansätze unterscheiden für Netzwerkverkehr meist zwei „Quality-of-Service (QoS)“ Modelle: Echtzeitkommunikation mit garantierten oberen Schranken für die Verzögerung und „Best-Effort-Service“ ohne jegliche Garantien. Best-Effort-Verkehr kann mit einfachen Algorithmen und geringem Ressourcenverbrauch implementiert werden. Um dagegen die deterministischen oder probabilistischen Schranken für Echtzeitkommunikation garantieren zu können, ist die Zusicherung von Ressourcen durch Ressourcenreservierung notwendig.
Es müssen möglicherweise aber nicht für den gesamten Netzwerkverkehr einer Anwendung teure obere Schranken garantiert werden. So wurde zum Beispiel gezeigt, dass es für die Stabilität eines „Networked Control System (NCS)“ ausreichend ist, wenn nur eine wohldefinierte Teilmenge an Sensorwerten mit deterministischen Garantieren transportiert wird [LCD+19]. Alle darüber hinaus erhaltenen Sensorwerte sind nicht mehr kritisch für die Stabilität und könnten daher auch über ressourcensparenden Best-Effort-Verkehr übertragen werden. Eine solche Abstufung wird jedoch von klassischen QoS Modellen nicht unterstützt. Sie liefern die selben Garantien für den kompletten Verkehr einer Anwendung, solange diese sich an die vereinbarte Spezifikation hält.
Selbst wenn nun die zusätzlichen Sensorwerte der Anwendung nur noch über Best-Effort-Verkehr übertragen würden, sind diese für die Anwendung durchaus interessant. Denn sie können dazu beitragen die Performanz des Systems zu erhöhen, wenn sie mit geringer Latenz ankommen. Mittels Best-Effort-Verkehr werden jedoch auch Informationen übertragen, die keinen Vorteil durch Zeitgarantien erhalten. Hierfür können große Dateiübertragungen oder Backups als Beispiele genannt

werden. Es existieren also durchaus weitere Abstufungen der Priorität innerhalb des Best Effort Verkehrs. Die klassischen QoS Modelle unterscheiden jedoch den Best Effort Verkehr nicht weiter, sondern behandeln jeden enthaltenen Strom gleich. Deshalb würden die zusätzlichen Sensorwerte keine höhere Priorität erhalten als Ströme, die nicht von einer pünktlichen Auslieferung profitieren.

Das Ziel dieser Arbeit besteht darin, flexiblere QoS Modelle zu untersuchen, welche mehr als nur die beiden Klassen „Echtzeitverkehr“ und „Best Effort Verkehr“ unterscheiden. Dafür bedarf es einer formalen Definition des gesuchten flexiblen QoS Modells.
Ein möglicher Kandidat eines solchen Modells könnte die Kombination von IntServ [BCS94] mit „Weighted Fair Queuing [DKS89]“ und dem Token Bucket Modell [SW97] sein. Es soll daher zunächst untersucht werden, inwieweit es dem gesuchten Modell entspricht.
Allerdings ist bekannt, dass IntServ aufgrund des Bedarfs von separaten Warteschlangen pro Strom eine kaum realisierbare Lösung ist [HK00; WM03]. Deshalb soll ferner untersucht und verglichen werden, wie komplex die gefundenen Modelle hinsichtlich ihrer Implementierung sind. Weiter sollen State-of-the-Art QoS Modelle hinsichtlich ihrer Verteilung der Verzögerungen analysiert werden und mit den gefundenen Modellen verglichen werden.
Abschließend sollen die ausgewählten Mechanismen implementiert und ihr Verhalten evaluiert werden. Dafür darf beispielsweise der Netzwerksimulator OMNet++ [VH08] und die Erweiterung NeSTiNg [FHC+19] verwendet werden.

Die Beiträge dieser Arbeit sind eine detaillierte Analyse der Verzögerungen von IntServ sowie die Konzipierung eines neuen, flexiblen und skalierbaren QoS-Modells auf Basis einer MLFQ. Es werden konkrete Formeln vorgestellt, mit welchen die garantierte maximale Ende-zu-Ende-Verzögerung von einzelnen Strömen im IntServ-Modell berechnet werden kann. Dazu wird eine mögliche Anpassung vorgestellt, um mit IntServ verschiedene Performance-Klassen mit garantierten maximalen Verzögerungen außerhalb des Stabilitätsbereichs zu realisieren. Auch für das neue QoS-Modell werden die Formeln vorgestellt, um die garantierte maximale Ende-zu-Ende-Verzögerung zu berechnen, sodass ein direkter Vergleich möglich ist. Des Weiteren werden Implementierungsdetails des neuen Modells erläutert und dessen Funktionsweise und Parametrisierung mit einem Netzwerksimulator evaluiert.

Diese Arbeit ist folgendermaßen strukturiert: In Kapitel 2 wird Hintergrundwissen zur behandelten Thematik zusammengefasst und anschließend einen Überblick über aktuelle Arbeiten in diesem Forschungsbereich gezeigt. Darauf folgt das zugrundeliegende System-Modell dieser Arbeit in Kapitel 3 sowie eine detaillierte Problemstellung. In Kapitel 4 wird zunächst das bekannte IntServ Modell mit Weighted Fair Queuing untersucht. Dabei werden die maximalen Verzögerungen von Netzwerkpaketen im Detail analysiert und berechnet. Im zweiten Teil dieses Kapitels wird der Entwurf eines neuen QoS-Modells auf Basis des Multi-Level-Feedback-Queue-Verfahrens erläutert. Zu diesem neuen Modell werden Implementierungsdetails in Kapitel 5 gezeigt und in Kapitel 6 folgt die Evaluation des Modells mithilfe eines Netzwerksimulators. Den Abschluss dieser Arbeitet bildet eine Zusammenfassung und ein Ausblick in Kapitel 7.

2 Hintergrundwissen und verwandte Arbeiten

2.1 Scheduling

Der Anglizismus „Scheduling“ bedeutet auf Deutsch etwa Ablaufplanung oder Maschinenbelegungsplanung [Heu13; Sch13]. Anfang des 19. Jahrhunderts gewann das Thema durch die Massenproduktion von Gütern erstmals für die Industrie an Bedeutung. Für die Informatik spielt das Scheduling seit den ersten Computern eine große Rolle, da damit die Zuteilung der Computerressourcen wie CPU, Arbeitsspeicher oder Netzwerknutzung an die einzelnen Anwendungen geregelt werden. Deshalb befassen sich seit Mitte des 19. Jahrhunderts viele Arbeiten mit den theoretischen Konzepten und praktischen Anwendungen von Scheduling Algorithmen [Pin12]. Da im Fokus dieser Arbeit Netzwerkanwendungen stehen, werden im Folgenden einige Scheduling Algorithmen für diesen Bereich vorgestellt.

Netzwerkstrom / Paketstrom Um die folgenden Algorithmen zu beschreiben, wird der Begriff „Netzwerkstrom“ (vom englischen „flow“) verwendet. Die Bezeichnung „Paketstrom“ ist ein Synonym.

Definition 2.1.1
Ein Netzwerkstrom ist eine Menge von Netzwerkpaketen, welche den selben Ursprung und das selbe Ziel haben und die gleiche Route verwenden. Sie benötigen die selbe Servicequalität bei jedem Router / Gateway entlang ihrer Route. Jedes Netzwerkpaket kann mithilfe des Paket Headers eindeutig einem Netzwerkstrom zugeordnet werden. (Nach Shreedhar und Varghese [SV96])

Round Robin (RR) Der Begriff „Round Robin“ stammt aus der Zeit der Französischen Revolution und bezeichnete das kreisförmige Unterzeichnen einer Petition, damit die Initiatoren nicht erkenntlich waren [Bre98; Hen98]. Beim Round Robin Scheduling Algorithmus für Netzwerkströme werden die Pakete der einzelnen Ströme immer im Kreis bearbeitet bzw. aus der Warteschlange entfernt [Nag87]. Abbildung 2.1 illustriert den Algorithmus nochmals.

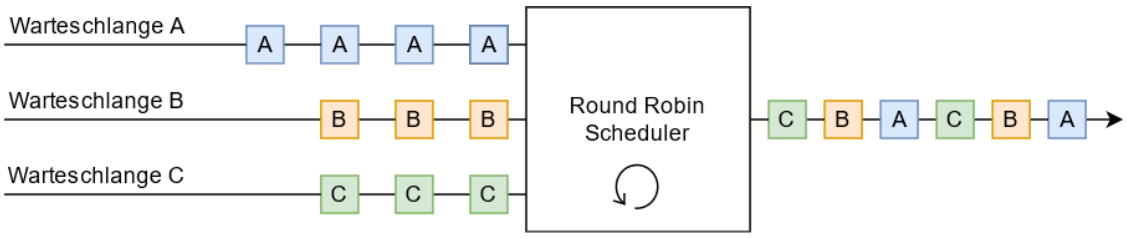

Abbildung 2.1: Illustration des Round Robin Scheduling Algorithmus.

So kann sichergestellt werden, dass jeder Netzwerkstrom gleich häufig zum Zug kommt, falls für alle

gleichzeitig Pakete auf die Bearbeitung warten. Allerdings garantiert dies noch keine faire Aufteilung der Bandbreite, denn die Paketgrößen können variieren. Somit würde ein Netzwerkstrom mit großen Paketen mehr Bandbreite bekommen, als ein Netzwerkstrom mit kleinen Paketen. Demers et al. haben deshalb den verbesserten bit-by-bit Round Robin Algorithmus vorgestellt [DKS89] (auch „Fair Queuing“ genannt). Hier bekommt jeder Netzwerkstrom pro Durchlauf ein Bit zugeteilt, diese werden aufsummiert, bis die wartende Paketgröße erreicht ist. Dann wird das Paket in einem Stück versandt und der Zähler zurück auf Null gesetzt.

Weighted Fair Queuing (WFQ) In ihrer abschließenden Diskussion von „Fair Queuing“ haben Demers et al. [DKS89] eine mögliche Erweiterung des Algorithmus vorgestellt. Das sogenannte „Weighted Fair Queuing“ arbeitet wie bit-by-bit Round Robin, allerdings können den einzelnen Netzwerkströmen Gewichte w_i zugeordnet werden. In jeder Runde erhält der Strom dann nicht nur ein Bit sondern w_i Bits, wodurch er entsprechend mehr oder weniger Bandbreite bekommt. Der Algorithmus wird in Abbildung 2.2 verdeutlicht.

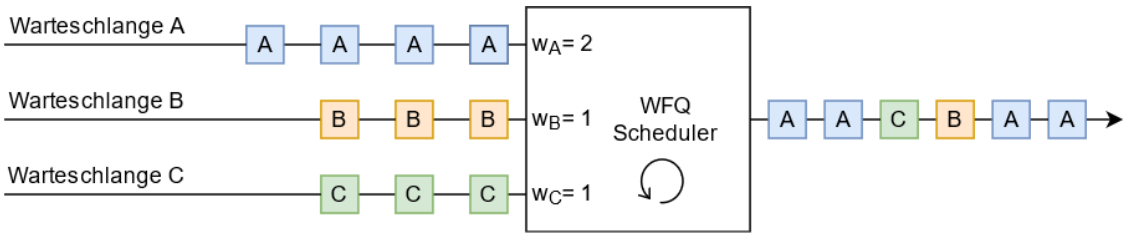

Abbildung 2.2: Illustration des WFQ Scheduling Algorithmus.

Bennet und Zhang [BZ96] haben gezeigt, dass für jeden Netzwerkstrom eine garantierte minimale Bandbreite gr_i berechnet werden kann:

$$gr_i = \frac{w_i}{\sum_{j=1}^{N} w_j} \cdot r \tag{2.1}$$

Dabei bezeichnet N die Anzahl der Netzwerkströme, welche um die Gesamtbandbreite r konkurrieren.

Priority Queuing (PQ) Beim Priority Queuing werden alle eingehenden Pakete klassifiziert und je nach Priorität in verschiedene Warteschlangen eingeordnet. Die Warteschlange mit der höchsten Priorität wird bevorzugt und alle Pakete darin werden weitergeleitet, bis sie leer ist. Erst dann kommt die nächst niedrigere Warteschlange zum Zug. Sobald ein Paket in einer höheren Warteschlange ankommt, wird die Übertragung unterbrochen und die höhere Priorität abgearbeitet [ATMT04; HT11; IIHA12]. Abbildung 2.3 illustriert den Algorithmus mit zwei Prioritäten.
Im Gegensatz zu Round Robin und Weighted Fair Queuing ist Priority Queuing kein fairer Algorithmus [RAL04]. Deshalb kann es passieren, dass Netzwerkströme mit geringer Priorität „verhungern“. Das bedeutet, dass ihre Pakete nie weitergeleitet werden und tritt auf, wenn kontinuierlich Pakete mit höherer Priorität vorhanden sind [ML89].

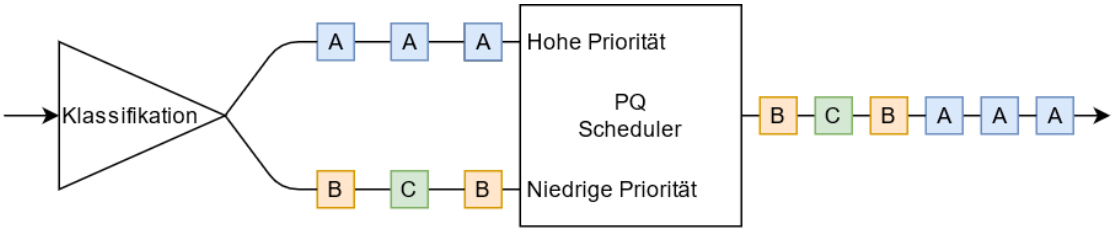

Abbildung 2.3: Illustration des PQ Scheduling Algorithmus mit zwei Prioritäten.

Multi Level Feedback Queue (MLFQ) Dieser Scheduling Algorithmus ist bekannt durch seine Anwendung beim Prozess-Scheduling, für Netzwerkanwendungen wurde er bisher selten verwendet. Der Algorithmus wird an dieser Stelle erläutert, da er in einem späteren Ansatz dieser Arbeit verwendet wird.
Die Multi Level Feedback Queue wurde erstmals von Corbato et al. [CMD62] vorgestellt. Der ursprüngliche Kontext ist ein Computersystem, welches von mehreren Benutzern geteilt wird, allerdings nicht über ausreichenden Arbeitsspeicher für alle Nutzerprogramme verfügt. Deshalb müssen die einzelnen Programme unterbrochen und ausgelagert werden, bis sie wieder an der Reihe sind. Mit Hilfe des Algorithmus werden die einzelnen Programme zur Laufzeit in verschiedene FIFO-Warteschlangen mit abgestuften Prioritäten eingeordnet. Ein neues Programm startet in der höchsten Prioritätsstufe. Wie beim Priority Queuing bereits erläutert, wird die Warteschlange höchster Priorität exklusiv bearbeitet, bis sie leer ist. Allerdings gibt es pro Warteschlange eine maximale Ausführungszeit, das sogenannte Zeitquantum. Ist das Programm nach Ablauf dieser Zeit noch nicht fertig, so wird es unterbrochen und in die nächst niedrigere Warteschlange eingeordnet. Hier ist das Zeitquantum höher, jedoch kommt die Anwendung nun nur noch zum Zug, wenn alle höheren Warteschlangen leer sind. Ein Prozess hat keine Möglichkeit wieder eine höhere Priorität zu erhalten.
Auf das Netzwerk übertragen, könnte statt der Ausführungszeit die Burst Größe verwendet werden. Sendet ein Netzwerkstrom große Bursts so verringert sich nach und nach seine Priorität. Ströme, welche nur kurze Bursts senden, erhalten dagegen die höchste Priorität. Zur Veranschaulichung zeigt Abbildung 2.4 ein mögliches Szenario eines MLFQ-Schedulers für Netzwerkströme.

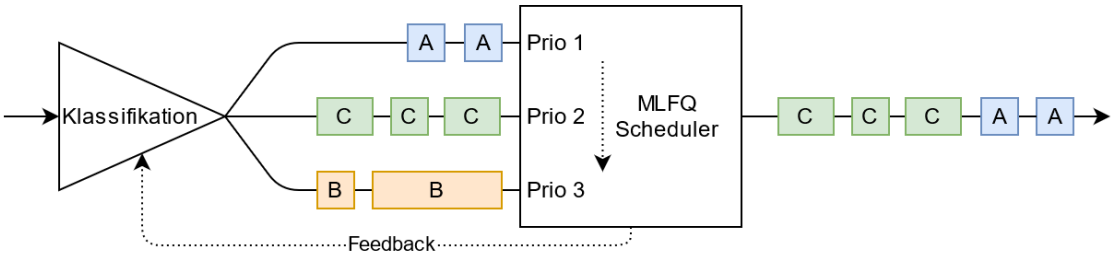

Abbildung 2.4: Eine mögliche MLFQ für Netzwerkströme. Die Ströme werden anhand ihrer Burst-Größen in der Vergangenheit in die Warteschlangen entsprechender Priorität eingeordnet.

2.2 Traffic Shaping

Traffic Shaping (übersetzt Verkehrsformung) gibt dem Netzwerkverkehr eine bestimmte Form, welche abhängig von der vereinbarten Spezifikation und des Verfahrens ist. Im Folgenden werden zwei bekannte Verfahren des Traffic Shaping erläutert: Das Leaky-Bucket- und das Token-Bucket-Modell.

Leaky-Bucket Die Metapher des Leaky-Buckets (auf Deutsch „löchriger Eimer“) wurde erstmals von Okun [Oku15] erwähnt. Allerdings stand sie im Kontext eines Gedankenexperiments bei der Vermögensverteilung zwischen den Armen und Reichen. Turner [Tur86] hat das Leaky-Bucket Modell für die Formung von Netzwerkverkehr vorgestellt. Dabei wird die Analogie eines Eimers (=Warteschlange) verwendet, welcher mit Wasser (=Pakete) gefüllt wird. Durch ein Loch, im Boden des Eimers, fließt das Wasser mit konstanter Rate ab. Kurzzeitig kann auch mehr nachgefüllt werden, als abließen kann, je nach Kapazität des Eimers. Das Modell wird in Abbildung 2.5 nochmals veranschaulicht.

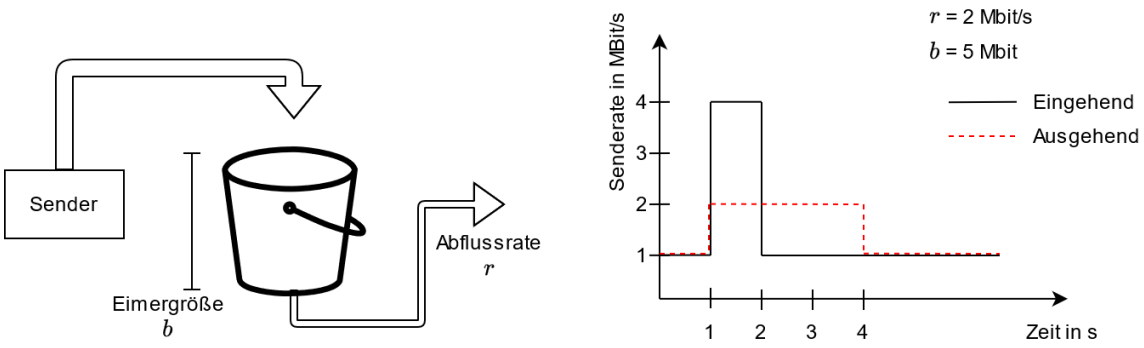

Abbildung 2.5: Illustration des Leaky-Bucket Modells (links) und dessen glättende Auswirkung auf die Senderate (rechts).

Das Leaky-Bucket-Modell wird spezifiziert durch die Eimergröße b und die Abflussrate r. Der Eimer puffert eingehende Bursts, welche mit der spezifizierten Abflussrate weitergeleitet werden. Dadurch ist die Senderate nach dem Leaky-Bucket auf maximal r beschränkt, eingehende Bursts werden entsprechend geglättet.

Token-Bucket Der Token-Bucket (auf Deutsch „Wertmarken-Eimer“) ist ein weiteres Traffic Shaping Verfahren, welches die Metapher eines Eimers verwendet [SW97]. Allerdings hat dieser Eimer keine Löcher, sondern wird mit Tokens (Wertmarken) gefüllt. Das Token-Bucket-Modell wird ebenfalls spezifiziert durch die Eimergröße b und die Token-Nachfüllrate r. Der Netzwerkverkehr benötigt Tokens um passieren zu dürfen. Anders als beim Leaky-Bucket gibt es hier keine Beschränkung der Bandbreite. Der Netzwerkverkehr wird mit der maximalen Bandbreite ins Netzwerk geschickt, solange Tokens vorhanden sind. Die Token-Nachfüllrate gibt also die mittlere Senderate vor, kurzzeitig kann jedoch auch mehr gesendet werden, falls zuvor Tokens angespart wurden. Dies wird als „Burst“ bezeichnet, was auf Deutsch soviel wie Sprengung bzw. platzen bedeutet. In Abbildung 2.6 ist eine Illustration des Modells sowie ein Diagramm des Verhaltens zu sehen.

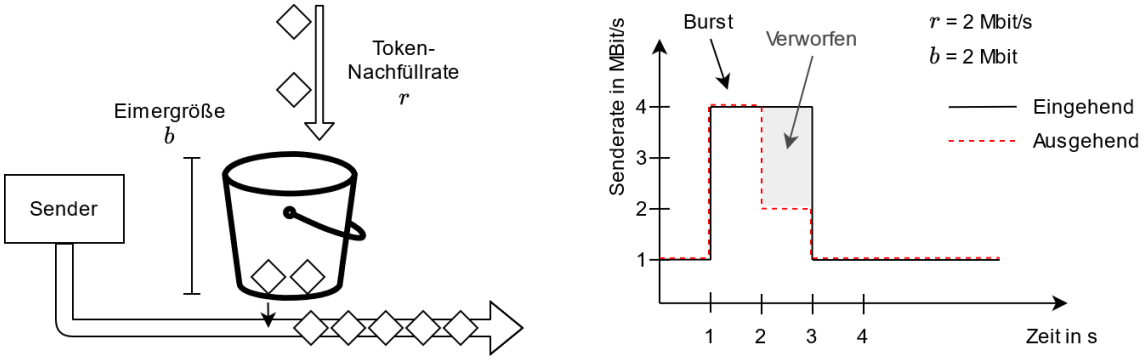

Abbildung 2.6: Illustration des Token-Bucket Modells (links) und dessen Auswirkung auf die Senderate (rechts).

Da kein Puffer für überschüssigen Netzwerkverkehr vorhanden ist, werden Pakete verworfen oder als ungültig markiert, wenn keine Tokens zur Verfügung stehen [TT99]. Dies ist im rechten Teil der Abbildung ebenfalls dargestellt.

2.3 Quality of Service

Ein einfaches und häufig implementiertes Service Modell für Netzwerkverkehr ist „Best-Effort“. Auf Deutsch übersetzt bedeutet es so viel wie „bestmögliches Bemühen“. Die Algorithmen *bemühen* sich um bestmögliche Leistung, es werden allerdings keinerlei Garantien gemacht [BS98]. Dies ist unzureichend für Anwendungen, die besondere Anforderungen an das Netzwerk haben [MS99]. Ein Beispiel hierfür ist VOIP Telefonie, bei welcher eine zu lange Verzögerung als störend wahrgenommen wird. Deshalb wurden verschiedene Modelle entwickelt, mit welchen bestimmte Service-Qualitäten (Englisch: Quality of Service, kurz QoS) garantiert werden können. Diese Modelle garantieren beispielsweise eine minimale Bandbreite oder maximale Verzögerung für den Netzwerkverkehr, indem Ressourcen reserviert werden. Dabei werden zwei Herangehensweisen unterschieden: IntServ und DiffServ. Beide werden im Folgenden erläutert.

IntServ Das Integrated Services Modell (kurz: IntServ) wurde im Rahmen eines RFCs von Braden et al. vorgestellt [BCS94]. Die Kernidee ist die Reservierung von Bandbreite *pro Netzwerkstrom* bei jedem genutzten Router. Als erstes müssen dazu die beteiligten Netzwerkströme spezifiziert werden. Hierfür wird das Leaky- oder Token-Bucket Modell verwendet. Anschließend wird mithilfe des Ressource-Reservation-Protocol (RSVP) [Wro+97; ZDE+93] Bandbreite bei jedem Router entlang des Pfads reserviert. Die Router verwenden Weighted Fair Queuing mit einer separaten Warteschlange pro Netzwerkstrom. Sie teilen mithilfe der vorgestellten Formel (2.1) die jeweilige garantierte Bandbreite über die Gewichte zu. Dadurch kann eine maximale Verzögerung für jeden Strom berechnet werden, wie in Kapitel 4.1 ausführlich gezeigt wird. Somit können für jeden Netzwerkstrom individuelle Garantien hinsichtlich der Bandbreite und Verzögerung vergeben werden.
Dieser Ansatz hat allerdings einen Nachteil: Jeder Router benötigt Zustandsinformationen bzw. eine separate Warteschlange für jeden Netzwerkstrom, der ihn passiert. In der Praxis laufen teilweise Millionen einzelner Ströme durch die großen Knotenpunkte des Internets. Dies würde bedeuten,

dass diese Router entsprechend viele Warteschlangen benötigen. Warteschlangen in Routern sind allerdings begrenzt und teuer, weshalb der IntServ Ansatz nicht gut skaliert werden kann. Deshalb hat sich der Ansatz in der Praxis nicht für das Internet durchgesetzt [HK00; WM03].

DiffServ Blake et al. haben das Differential Services Modell (kurz: DiffServ) vorgestellt [BBC+98]. Im Gegensatz zum IntServ Modell werden hier nicht einzelne Ströme sondern *Anwendungsklassen* betrachtet. Netzwerkströme werden klassifiziert und einer Anwendungsklasse zugeordnet. Die Router behandeln alle Pakete in der selben Anwendungsklasse gleich. Beispielsweise wird aller VOIP Netzwerkverkehr einer Klasse zugeordnet und von den Routern bevorzugt behandelt. Nun gibt es keine Garantien mehr für jede einzelne VOIP Verbindung, wie es bei IntServ der Fall wäre, sondern nur noch für den VOIP Verkehr im Allgemeinen.
Das Verhalten der Router (Englisch: Per-Hop Behavior), also der Umgang mit Paketen einer bestimmten Klasse, wurde unabhängig von der Architektur definiert. Lediglich für das Standardverhalten wurde die Best-Effort-Weiterleitung festgelegt [NBBB98]. Weitere mögliche Verhaltensweisen sind das Expedited Forwarding [DCB+02] und das Assured Forwarding [HBWW99].

- Expedited Forwarding bietet typischerweise die bestmögliche Qualität und geringsten Verzögerungen. Dafür sollten Pakete dieser Klasse möglichst auf leere Queues treffen und sofort weitergeleitet werden. Eine mögliche Implementierung ist die Verwendung einer Priority Queue.
- Assured Forwarding garantiert die Zustellung der Pakete, solange der Netzwerkverkehr die Spezifikation nicht überschreitet. Dafür wird für jede Anwendungsklasse eine Warteschlange verwendet und Bandbreite reserviert. Mithilfe eines Token-Buckets wird der Netzwerkverkehr pro Anwendungsklasse spezifiziert. Falls die Spezifikation überschritten wird, läuft der Token-Bucket leer und die Router beginnen, Pakete zu verwerfen. Die Reservierung kann wie für IntServ mittels WFQ implementiert werden. Im Gegensatz zu IntServ wird hierfür jedoch nur eine Queue pro Anwendungsklasse mit Assured Forwarding benötigt, unabhängig von der Anzahl der Netzwerkströme.

Der DiffServ Ansatz ist skalierbar, da es typischerweise nur eine geringe Menge an Klassen gibt. Die Router müssen keine Zustandsinformationen für die einzelnen Netzwerkströme halten, sondern nur für die konstante Anzahl der Klassen. Aus diesem Grund hat sich dieser Ansatz gegenüber dem IntServ Modell durchgesetzt. Ein Nachteil von DiffServ sind die schwächeren Garantien, verglichen zu IntServ. Es können keine einzelnen Garantien für einen Netzwerkstrom mehr vergeben werden, sondern nur noch für eine Anwendungsklasse. Außerdem wird ein globales Management benötigt, um sicherzustellen, dass der Netzwerkverkehr die reservierten Ressourcen nicht überlastet. Denn der Netzwerkverkehr einer Anwendungsklasse (z.B. VOIP-Verkehr) kann von beliebig vielen Sendern gleichzeitig generiert werden.

Three Color Marker Eine Möglichkeit, Ströme anhand ihres Sendeverhaltens zu klassifizieren, wurde für DiffServ mit dem „Two Rate Three Color Marker“ [HG99] vorgestellt. Dieser Klassifizierer ordnet die Ströme anhand ihres Sendeverhaltens in drei Klassen ein: Rot, Gelb und Grün. Dafür werden zwei Token-Buckets verwendet, deren Token-Rate als „Peak Information Rate (PIR)“ und „Committed Information Rate“ bezeichnet wird. Es gilt dabei PIR $\geq$ CIR. Der Three Color Marker klassifiziert die Pakete, indem er für jedes eingehende Paket die entsprechende Anzahl Tokens aus jeweils beiden Token-Buckets entfernt. Befinden sich genügend Tokens in beiden Buckets, es

wurde also weder die PIR noch die CIR überschritten, so wird das Paket grün markiert. Falls nur im Token-Bucket der PIR genügend Tokens vorhanden sind, so wird das Paket gelb markiert. Der Strom hat damit die Committed Information Rate überschritten, befindet sich jedoch noch innerhalb der Peak Information Rate. Sollten beide Token-Buckets nicht über ausreichend Tokens verfügen, wird das Paket rot markiert, denn der Strom hat beide Raten überschritten.
Mit diesem Ansatz können nur drei Klassen unterschieden werden. Außerdem werden die Pakete lediglich mit der jeweiligen Farbe markiert, es wird nicht definiert, wie der QoS für die verschiedenen Klassen umgesetzt werden soll. Es sind somit keine Garantien für die maximalen Verzögerungen in den einzelnen Farbklassen vorhanden.

2.4 Echtzeitkommunikation

Für viele Anwendungen ist es nicht ausreichend, wenn die Netzwerkkommunikation nur einem Best-Effort-Modell enstpricht. Besonders wenn Prozesse in der realen Welt gesteuert werden sollen, ist es häufig nötig, dass Netzwerkverkehr eine beschränkte Kommunikationsverzögerung (engl. Network Delay) und eine beschränkte Varianz der Verzögerung (engl. Jitter) hat. Ein Beispiel hierfür sind zeitlich genau abgestimmte Roboterbewegungen einer Produktionslinie. Würden dort Steuerungsinformationen verspätet oder überhaupt nicht ankommen, bedeutet dies unpräzise oder gar unkontrollierbare Bewegungen. Die Standard-Ethernet-Technologie unterstützt nur Best-Effort-Netzwerkeverkehr, weshalb für zeitkritische Anwendungen in der Vergangenheit separate Bus-Systeme verwendet wurden [DN16]. Da die Ethernet-Technologie heute weit verbreitet ist, wurden mehrere Technologien vorgestellt, um Ethernet mit der Fähigkeit für Echtzeitkommunikation zu erweitern. Beispiele hierfür sind SERCOS III [Sch04] oder PROFINET [TV99]. Ein Problem dieser Erweiterung ist jedoch, dass sie untereinander nicht kompatibel sind.

IEEE Time-sensitive Networking (TSN) Aufgrund dieser Inkompatibilität hat das Institute of Electrical and Electronics Engineers (IEEE) einen Standard für deterministische Echtzeitkommunikation über Ethernet entwickelt [DN16]. Dieser Standard, genannt IEEE 802.1Qbv [IEE16a] oder Time-sensitive Networking (TSN), verwendet einen TDMA-Ansatz, bei welchem die Belegung des Netzwerks in feste Zeitschlitze eingeteilt wird. Dieser Ansatz wird auch Time-Aware-Shaper genannt. Damit die Zeitschlitze bei allen Routern synchronisiert sind, wird eine präzise Synchronisation der Uhren benötigt. Dafür kann beispielsweise das Precision Time Protocol (PTP) [IEE20a] verwendet werden.
Sowohl der Echtzeitverkehr, als auch der Best-Effort-Verkehr, erhalten einen eigenen Zeitschlitz und eine eigene Warteschlange bei jedem Switch. Während seines Zeitschlitzes erhält der Echtzeitverkehr exklusiven Zugriff auf das Netzwerk, wodurch er vom Best-Effort-Verkehr entkoppelt ist. Dafür sind auf jedem Switch sogenannte „Gatter“ (auf Englisch: Gates) installiert, welche je nach Zeitschlitz den Zugriff einer Warteschlange auf das Netzwerk freigeben oder sperren. Eine Illustration der verschiedenen Warteschlangen und Gatter eines TSN Switches befindet sich in Abbildung 2.7.
Die Öffnungszeiten der Gatter können mithilfe eines „Programmable Gate Driver“ programmiert werden, um die Zuteilung der Zeitschlitze zu den Warteschlangen individuell vorzunehmen. Wie genau die Zeitschlitze eingeteilt werden müssen, um eine garantierte maximale Verzögerung zu erreichen, ist jedoch nicht mehr Teil des Standards [DN16].

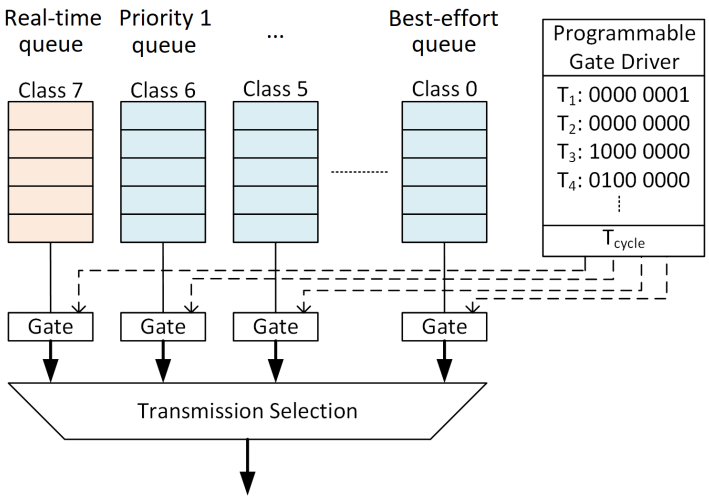

Abbildung 2.7: Die Architektur eines Ports von einem TSN Switch. Der Echtzeitverkehr (Scheduled Traffic) und Best-Effort Verkehr erhalten getrennte Warteschlangen und der Zugriff auf das Netzwerk wird über programmierbare Gatter (Gates) geregelt. [DN16]

Für die Einteilung der Zeitschlitze werden Ablaufpläne (Englisch: Schedules) erstellt. Die dafür vorgestellten TSN-Planungsverfahren können grundsätzlich in zwei Klassen eingeordnet werden:

1. **Reine Ablaufplanung mit gegebenen Routen:** Ein Vertreter dieser Klasse ist das „No-wait Packet Scheduling“ welches von Dürr und Nayak [DN16] vorgestellt wurde. Sie haben das bekannte „No-wait Job-Shop Scheduling Problem“ [MP02] auf Netzwerkpakete übertragen und damit das Problem formalisiert, einen Ablaufplan für die TSN-Zeitschlitze zu erstellen. Dieses Problem kann als Integer Linear Program (ILP) [Wol20] formuliert werden und mit einem ILP-Solver gelöst werden. Allerdings ist das Problem NP-schwer, weshalb dieser exakte Lösungsansatz nicht skalierbar ist. Darum haben Dürr und Nayak einen heuristischen Optimierungsalgorithmus für das Problem vorgestellt, mit welchem nahezu optimale Ablaufpläne effizient berechnet werden können.

2. **Gemeinsame Planung von Ablaufplänen und Routen:** Das Problem des Routings und das Erstellen eines Ablaufplans wird zusammen gelöst. Der Vorteil von Algorithmen dieser Klasse liegt in der höheren Erfolgsrate, eine Lösung zu finden. Es können sowohl die Lösungen gefunden werden, die mit vorgegebenen Routen von den Algorithmen der vorherigen Klasse gefunden werden können und zusätzlich weitere Lösungen auf alternativen Routen. Der Ansatz von Schweissguth et al. [SDT+17] ist ein Vertreter dieser Klasse. Sie formalisieren und lösen das Routing-Problem und das Erstellen eines Ablaufplans gemeinsam in Form eines ILP. Da im weiteren Verlauf dieser Arbeit jedoch angenommen wird, dass die Routen bereits existieren, sind Ansätze dieser Klasse hier weniger relevant.

Vernetzte Regelungssysteme Die TSN-Planungsverfahren erstellen einen Ablaufplan, welcher deterministische Garantien für die Netzwerkverzögerung liefert. Diese Garantien gelten auf Netzwerkebene für die einzelnen Pakete, der Inhalt der Pakete spielt dabei keine Rolle. Die Anwendungen befinden sich auf einer höheren Abstraktionsebene und nutzen das Netzwerk zum übermitteln verschiedener Daten. Für sie ist der Inhalt, also die übermittelten Daten von Bedeutung. Auf dieser

Anwendungsebene und auf der Netzwerkebene setzt die Arbeit von Linsenmayer et al. [LCD+19] an. Sie abstrahieren das Netzwerk mithilfe eines Regelungskreises und haben gezeigt, dass zwei Arten von Daten unterschieden werden können: Stabilitäts- und Performancedaten. Stabilitätsdaten sind die minimal notwendigen Daten, um die Stabilität des Regelungssystems sicherzustellen. Alle weiteren Daten werden als Performancedaten bezeichnet und können die Performance des Systems erhöhen, werden jedoch nicht zwingend benötigt. Linsenmayer et al. schlagen daher vor, nur die Stabilitätsdaten mit deterministischen Garantien zu übermitteln und für die Performancedaten ein einfaches Best-Effort Modell zu verwenden. Hierfür unterteilen sie die Netzwerknutzung in periodisch wiederkehrende Zeit-Slots, wobei bestimmte Slots für die Stabilitätsdaten reserviert sind. Die restlichen Slots stehen für den Best-Effort Verkehr mit den Performancedaten zur Verfügung und werden opportunistisch genutzt. Diese opportunistischen Sendevorgänge werden mithilfe eines Token-Buckets reglementiert.
Die vernetzten Regelungssysteme sind ein Anwendungsfall, der auch in dieser Arbeit unterstützt werden soll. Das Token-Bucket-Modell wird dabei für die Spezifikation des Netzwerkverkehrs verwendet, welcher von der Anwendung erzeugt wird. Auf Grundlage des Token-Buckets kann die Anwendung selbst über ihre Sendevorgänge bestimmen. Das Ziel dieser Arbeit ist es zum einen, der Regelungsanwendung wohldefinierte QoS-Zusagen in Abhängigkeit ihres Sendeverhaltens zu geben. Damit kann die Anwendung selbst eine Analyse der Regelgüte (engl. Quality of Control) auf Grundlage der erwarteten Nachrichtenverzögerung durchführen. Zum anderen wird versucht der Anwendung definierte Anreize zur Selbstregulierung des Netzwerkverkehrs zu geben. Dadurch entstehen aus Sicht des Netzwerks günstige Verkehrsmuster wie beispielsweise möglichst wenige/kleine Bursts.

Asynchronous Shaper Alle bisher erläuterten TSN-Planungsverfahren verwenden die genannten Zeitschlitze, die durch präzise synchronisierte Uhren ermöglicht werden. Einen grundlegend verschiedenen Ansatz, der keine zeitliche Synchronisation benötigt, haben Specht und Samii mit dem Asynchronous Shaper (ursprünlich Urgency-based Shaper) [SS16] vorgestellt. Anstatt Pakete synchron in festen Zyklen zu bearbeiten, wie es beim Time-Aware-Shaper der Fall ist, erfolgt die Bearbeitung asynchron bei jedem Router. Die Idee hinter diesem Ansatz ist die Verwendung von Traffic Shapern (Leaky bzw. Token-Buckets) bei jedem Router. Die genaue Funktionsweise wurde unter dem IEEE Standard 802.1Qcr [IEE20b] standardisiert.
Wie beim Time-Aware-Shaper ist das Ziel des Asynchronous Shaper deterministische Garantien für die Kommunikationsverzögerung zu geben. Dabei wird nur das Worst-Case-Verhalten der Anwendungen betrachtet. Im Fokus dieser Arbeit stehen dagegen ein Spektrum von abgestuften QoS-Garantien in Abhängigkeit vom Sendeverhalten der Anwendung.

OMNeT++ Simulator und NeSTiNg-Erweiterung Neben dem erwähnten Time-Aware-Shaper, der vom TSN Standard spezifiziert wurde, können zusätzlich auch andere Scheduling Mechanismen wie Priority Queuing verwendet werden. Außerdem spezifiziert der TSN Standard weitere komplexe Mechanismen wie beispielsweise Frame Preemption [IEE16b], die noch dazu verwendet werden können. Dadurch entsteht im allgemeinen ein schwer analysierbares Netzwerk, in welchem sowohl Netzwerkverkehr mit Echtzeitgarantien als auch nicht-Echtzeitverkehr transportiert wird [FHC+19]. Um solche komplexen TSN-Netzwerke analysieren zu können, haben Falk et al. die NeSTiNg Erweiterungen [FHC+19] für den OMNeT++ Netzwerksimulator [VH08] vorgestellt. Mit dem ereignisbasierten OMNeT++ Simulator können entworfene Netzwerkalgorithmen implementiert

und evaluiert werden, ohne dafür tatsächliche Hardware-Switches zu benötigen. Die NeSTiNg Erweiterung fügt zusätzlich die Funktionalität von TSN-Switches zum Simulator hinzu. Sie beinhaltet unter anderem den zuvor gezeigten TSN-Scheduling Algorithmus mit programmierbaren Gattern. Im späteren Verlauf dieser Arbeit wird der OMNeT++ Simulator verwendet, um das neu entworfene Scheduling-Verfahren zu implementieren und zu evaluieren.

3 System-Modell und Problemstellung

In diesem Kapitel werden die getroffenen Annahmen über das System-Modell erläutert, welches dieser Arbeit zugrunde liegt. Anschließend wird die Problemstellung der Arbeit nochmals im Detail ausgeführt.

3.1 System-Modell

In diesem Abschnitt wird beschrieben, welche Eigenschaften und Annahmen über das zugrundeliegenden System für diese Arbeit gelten. Das System besteht aus den folgenden Komponenten: Ein Netzwerk mit Routern, Sendern und Empfängern (Endsysteme) sowie der Anwendung (eine Abstraktionsebene höher).

Das Netzwerk besteht aus einer Menge von Routern, welche miteinander verbunden sind und Netzwerkpakete weiterleiten. Die Kapazität entlang der Routen (Link-Kapazität) ist beschränkt, weshalb die Bandbreite mittels Scheduling aufgeteilt werden muss und Pakete bei den Routern gepuffert werden müssen. Die Router verwenden zum Puffern Paketwarteschlangen. Sie verfügen nur über einen begrenzten Pufferspeicher, weshalb die Anzahl und Länge der Paketwarteschlangen begrenzt ist. Des Weiteren wird von gegebenen, eindeutigen Routen ausgegangen, welche sich während eines Sendevorgangs nicht verändert. Daher werden die Einflüsse von dynamischem Routing nicht betrachtet. Der Fokus liegt stattdessen auf dem Scheduling von Paketen entlang der gegebenen Routen.

Die Endsysteme befinden sich am Rand des Netzwerks. Die Empfänger sind jeweils direkt mit einem Router verbunden, und die Sender werden über einen Traffic-Shaper (Token-Bucket) an das Netzwerk angebunden. Die Menge der Sender und Empfänger ist dynamisch, wodurch das Scheduling dynamisch angepasst werden muss. Aufgrund der beschränkten Link- und Pufferkapazität müssen Mechanismen zur Zugangsbeschränkung (Admission-Control) verwendet werden, um eine gewisse Qualität sicherzustellen.

Die Anwendung bzw. der betrachtete Anwendungsfall sind vernetzte Regelungssysteme. Für sie sind zwei Aspekte wesentlich: Stabilität und Regelgüte (Performance). Der Netzwerkverkehr vom Sender zum Empfänger kann daher in zwei Teile unterteilt werden. Einen deterministischen Anteil, der für die Stabilität benötigt wird, und einen Performance-Anteil, der für die Sicherheit des Systems (Stabilität) nicht kritisch ist, aber die Regelgüte beeinflusst. Es werden nur

Unicast-Kommunikationsbeziehungen angenommen. Eine Multicast-Kommunikation zwischen einem Sender und mehreren Empfängern kann die vorgeschlagenen Konzepte grundsätzlich nutzen, wird aber hier nicht weiter betrachtet.

3.2 Problemstellung

Quality-of-Service-Modelle unterscheiden typischerweise zwei Klassen: Echtzeitkommunikation und Best-Effort-Service. Für die Echtzeitkommunikation wird eine garantierte Zeitschranke für die Verzögerung eingehalten, während für Best-Effort-Verkehr keinerlei Garantien gemacht werden. Dieses Modell ist ausreichend, um die Stabilität der Anwendung zu garantierten: Solange eine definierte Menge an Sensorwerten innerhalb der Zeitschranke ankommt, ist das System stabil. Zusätzliche Sensorwerte, die rechtzeitig ankommen, können die Performance der Anwendung verbessern. Da sie jedoch nicht kritisch sind, würden sie in diesem Modell nur mit Best-Effort-Service übermittelt werden und keine garantierte Zeitschranken einhalten.

Wie bereits erwähnt, wird für den kritischen Verkehr *eine* Zeitschranke definiert. Es wäre aber auch ein QoS-Modell mit feingranularer Abstufung innerhalb einer Anwendung denkbar, welches *mehrere* Zeitschranken und Klassen definiert. In Abbildung 3.1 wird der folgende Ansatz illustriert. Die einzelnen Komponenten werden anschließend genauer erläutert und definiert.

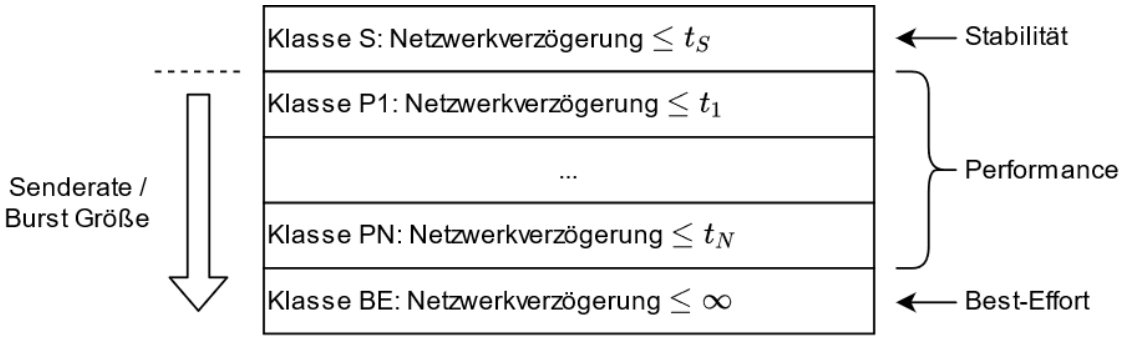

Abbildung 3.1: Illustration einer möglichen feingranularen Abstufung innerhalb eines QoS-Modells.

Definition 3.2.1 (Klasse S: Stabilitäts-Klasse)
Diese Klasse garantiert die Zeitschranke t_S, welche für die Stabilität der Anwendung benötigt wird.

Definition 3.2.2 (Klasse P1-PN: Performance-Klassen)
Die Performance-Klassen garantieren die Zeitschranken t_1 bis t_N, wobei gilt, dass $t_1 < t_2 < \ldots < t_N < \infty$. Die Service-Qualität nimmt somit ab (die garantierte maximale Verzögerung nimmt zu), je höher die Nummer der Klasse ist. Sie sind für zusätzliche, nicht stabilitätskritische Sensorwerte gedacht. Die Einstufung in die Performance-Klassen ist abhängig vom Sendeverhalten der Anwendung.

In Abbildung 3.1 ist die Stabilitätsklasse und die Performance-Klassen mit ihren Zeitschranken eingezeichnet. Nach der letzten Performance-Klasse folgt die bekannte Best-Effort Klasse, hier benannt als `Klasse BE`. In dieser Klasse wird nur noch Best-Effort-Service ohne eine garantierte Zeitschranke angeboten.
Anmerkung: Obwohl Klasse S in der Abbildung an höchster Stelle liegt, ist keine Beziehung zwischen t_S und t_1 vorgegeben. Somit ist es theoretisch möglich, dass manche der oberen Performance-Klassen sogar geringere Verzögerungen als Klasse S garantieren.

In diesem feingranularen Modell bekommt nun jede Anwendung den Service von Klasse S für die benötigte Anzahl an Sensorwerten zugesichert, sodass die Stabilität garantiert ist. Jeder darüber hinausgehender Netzwerkverkehr fällt in die übrigen Klassen. Hier sollen nun die Anwendungen anhand ihres Verhaltens in der Vergangenheit, also die Senderate und Burst-Größe, dynamisch in die jeweiligen Klassen eingeteilt werden. Eine aus Netzsicht „gutmütige“ Anwendung, die mit niedriger Rate sendet bzw. kleine Bursts erzeugt, soll mit besserem QoS belohnt werden. Dagegen soll eine Anwendung, die viel sendet oder große Bursts produziert, einen schlechteren QoS und schließlich nur noch Best-Effort-Service erhalten. Diese Zuteilung in die QoS-Klassen soll vorhersagbar sein, sodass die Anwendung damit planen kann und ihr Verhalten entsprechend anpassen kann. Außerdem soll zwischen den Anwendungen eine Fairness gelten: Keine Anwendung kann sich unrechtmäßig einen besseren QoS verschaffen, als der, welcher ihr aufgrund ihres Verhaltens zusteht. In der folgenden Abbildung 3.2 werden das typische zwei-Klassen-QoS und der neuen Ansatz gegenübergestellt.

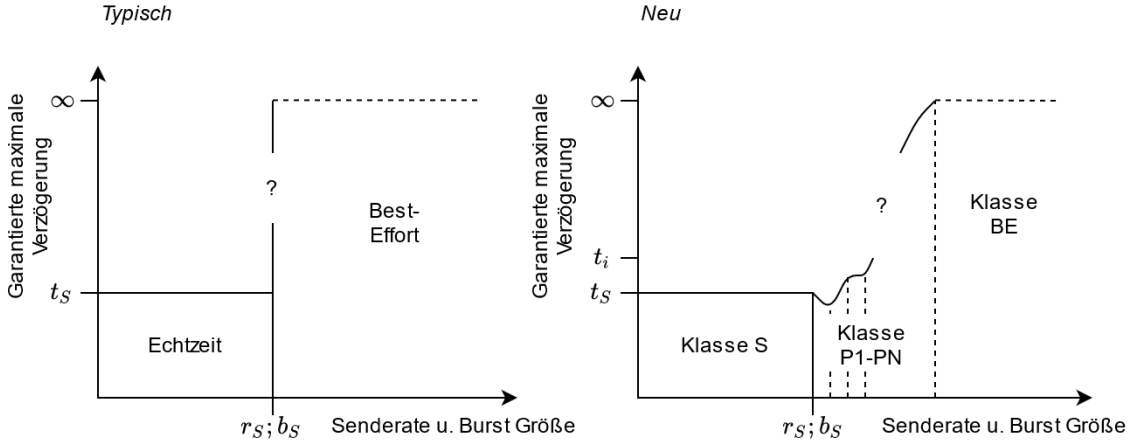

Abbildung 3.2: Gegenüberstellung des typischen QoS Modells (links) und des neuen Ansatzes (rechts). Verglichen wird die garantierte maximale Verzögerung bei steigender Senderate bzw. Burst Größe.

Auf der linken Seite befindet sich ein klassisches QoS Modell mit Echtzeit- und Best-Effort-Klasse. Solange die Senderate und Burst Größe innerhalb der Spezifikation liegen ($\leq r_S$ und $\leq b_S$), wird eine Verzögerung von maximal t_S garantiert. Unklar und im ersten Schritt zu analysieren, ist das Verhalten bekannter QoS Modelle nach Überschreiten der Spezifikation.
Rechts wird ein mögliches Diagramm des neuen Ansatzes gezeigt. Wie beim klassischen Ansatz wird für den Echtzeitverkehr (Klasse S) die Schranke t_S garantiert. Anschließend folgen die Klassen

P1 - PN, welche ihre eigenen Schranken für die maximale Verzögerung garantieren (der Übersicht halber nur als t_i eingezeichnet). Solch ein feingranulares Modell gilt es zu entwerfen, implementieren und dessen Verhalten zu evaluieren.

4 Analyse und Entwurf alternativer QoS Modelle

4.1 Analyse von IntServ mit WFQ

Im Folgenden werden die maximalen Verzögerungen eines Netzwerks mit IntServ und WFQ analytisch berechnet. Dieser Abschnitt ist in drei Schritte unterteilt: Zuerst wird die Verzögerung vom Sender über das Token-Bucket-Modell zum ersten Router genauer betrachtet. Anschließend werden die Verzögerungen von Weighted Fair Queuing zwischen den Routern untersucht. Schließlich wird mithilfe des Fluid-Modells die Gesamtverzögerung von Sender zu Empfänger berechnet.

Verzögerungen und Berechnungen mit dem Token-Bucket Der betrachtete Netzwerkstrom F_1 für die Berechnungen sei mithilfe eines Token-Buckets (r_1, b_1) spezifiziert. Als Erstes wird nun die Verzögerung vom Sender, über den Token-Bucket, zum ersten Router R_1 untersucht. In Abbildung 4.1 ist eine Übersicht der Komponenten zu finden.

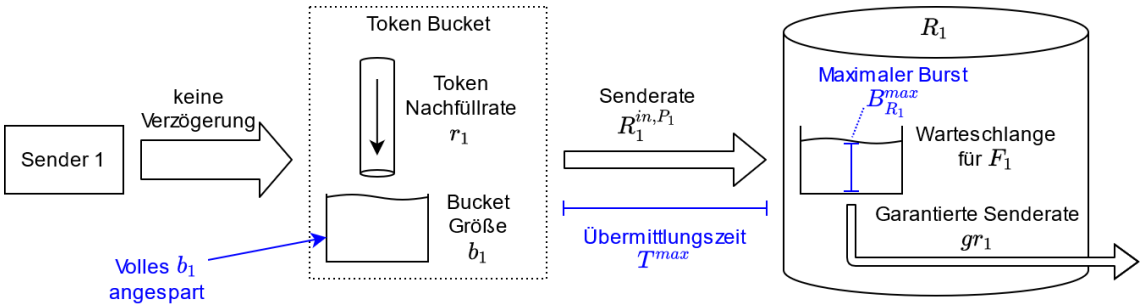

Abbildung 4.1: Die Komponenten vom Sender, über den Tocken-Bucket, zum ersten Router R_1.

Es wird angenommen, dass vom Sender zum Token-Bucket ohne Verzögerung Daten übertragen werden können. Diese Annahme kann getroffen werden, da der Token-Bucket z.B. vom Sender direkt implementiert werden kann. Die zu sendenden Daten liegen dann im Hauptspeicher des Sender-Hosts und können direkt an das Betriebssystem übergeben werden. Die maximale Verzögerung entsteht, wenn der Token-Bucket komplett mit angesparten Tokens gefüllt ist und ein Burst von maximaler Größe gesendet wird. Dieser Burst hat die Größe b_1 und wird mit der kompletten verfügbaren Senderate $R_{R_1}^{in,P_1}$ an den ersten Router R_1 übermittelt. Dadurch lässt sich die Übermittlungszeit T^{max} des Bursts zu R_1 berechnen:

$$T^{max} = \frac{b_1}{R_1^{in,P_1}} \tag{4.1}$$

Nach dieser Zeit wurde der komplette Burst an R_1 übertragen. Bereits während der Übermittlung wird ein Teil des Bursts über die garantierte Senderate gr_1 von R_1 an den nächsten Router weitergeschickt. In der dedizierten Warteschlange für F_1 befindet sich nach T^{max} also nicht mehr der komplette Burst

b_1. Für die Berechnung der maximalen Verzögerung muss lediglich der verbleibende Burst $B_{R_1}^{max}$ berücksichtigt werden. Allerdings kann R_1 nicht während der kompletten Übertragungszeit T^{max} bereits mit dem Weiterleiten beginnen, sondern erst nachdem das erste Paket vollständig an ihn übertragen wurde. Die Zeit, während welcher bereits Pakete weitergeleitet werden, entspricht also $T^{max} - \frac{S^{ep}}{R_1^{in,P_1}}$. S^{ep} bezeichnet die Größe des ersten Pakets des Bursts. Die Größe des verbleibenden Bursts $B_{R_1}^{max}$ kann dadurch berechnet werden mit:

$$\begin{aligned} B_{R_1}^{max} &= b_1 - (T^{max} - \frac{S^{ep}}{R_1^{in,P_1}}) \cdot gr_1 \\ &= b_1 - \frac{b_1 - S^{ep}}{R_1^{in,P_1}} \cdot gr_1 \end{aligned} \tag{4.2}$$

Verzögerungen von Weighted Fair Queuing Zunächst wird von einer simplen Netzwerktopologie mit zwei Strömen ausgegangen, welche beide über einen Flaschenhals-Link laufen. Die Topologie wird in Abbildung 4.2 veranschaulicht.

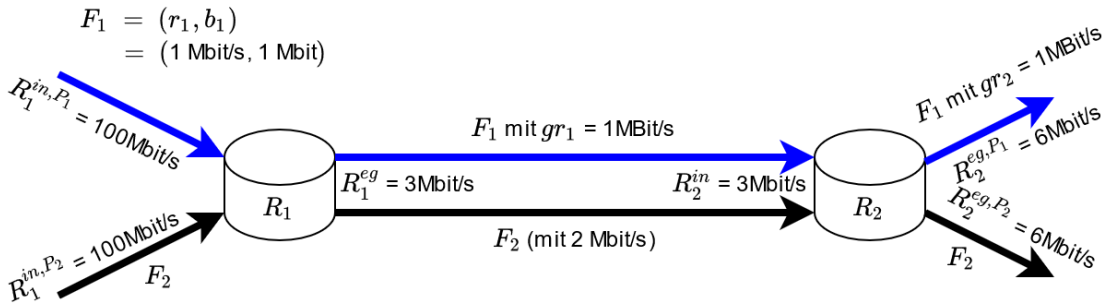

Abbildung 4.2: Topologie mit Flaschenhals-Link für nachfolgende Berechnungen

Für die Berechnungen wird jeweils Strom F_1 betrachtet. Wie zuvor erläutert, wird F_1 anhand des Token-Buckets (r_1, b_1) spezifiziert. Beide Router R_1 und R_2 implementieren IntServ mit WFQ. Für die Verbindung zwischen den beiden Routern, im Folgenden auch als Flaschenhals bezeichnet, wird für F_1 Bandbreite reserviert. Auch für die nächste Verbindung ab R_2 wird Bandbreite reserviert. Diese reservierte, garantierte Bandbreite gr_1 und gr_2 muss mindestens so groß sein, wie die spezifizierte Token-Rate r_1. Der Übersichtlichkeit halber wird im Folgenden angenommen, dass $gr_1 = gr_2 = r_1$. Theoretisch könnte R_2 nach dem Flaschenhals auch mehr Bandbreite reservieren ($gr_2 > r_1$), dieser Fall wird zu einem späteren Zeitpunkt genauer untersucht. Router 1 hat zwei Eingänge (Ports) mit jeweils einer maximalen Eingangsrate von $R_1^{in,P_1} = R_1^{in,P_2}$ = 100 Mbit/s. Er hat nur einen Ausgang, dessen maximale Bandbreite mit R_1^{eq} bezeichnet wird.

Da WFQ „work-conserving“ arbeitet, kann ein Netzwerkstrom die Bandbreite des jeweilig anderen Stroms mitverwenden, falls einer der Ströme weniger als seine garantierte Bandbreite sendet. Im Folgenden soll jedoch die garantierte maximale Verzögerung untersucht werden, weshalb vom schlechtesten Fall ausgegangen werden muss: Alle anderen Ströme (hier F_2) nutzen ihre Bandbreite vollständig. Somit kann F_1 jeweils nur mit gr_1 und gr_2 senden. Die maximale Verzögerung entsteht, wenn F_1 einen komplett vollen Token Bucket b_1 angespart hat und nun einen Burst von maximaler Größe sendet. Mit den bereits vorgestellten Formeln (4.1,4.2), kann die Dauer und Größe des

verbleibenden Bursts in R_1 berechnet werden. Es wird dabei angenommen, dass die Größe des ersten Pakets S^{ep} = 1 kbit ist:

$$\begin{aligned} T^{max} &= \frac{b_1}{R_1^{in,P_1}} \\ &= \frac{1 \cdot 10^6 bit}{100 \cdot 10^6 bit/s} \\ &= \frac{1}{100} s = 10ms \end{aligned}$$

$$\begin{aligned} T^{max} - \frac{S^{ep}}{R_1^{in,P_1}} &= 10 \cdot 10^{-3} s - \frac{1 \cdot 10^3 bit}{100 \cdot 10^6 bit/s} \\ &= 10ms - 0,01ms \\ &= 9,99ms \end{aligned} \tag{4.3}$$

$$\begin{aligned} B_{R_1}^{max} &= b_1 - (T^{max} - \frac{S^{ep}}{R_1^{in,P_1}}) \cdot gr_1 \\ &= 1 \cdot 10^6 bit - (9,99 \cdot 10^{-3} s \cdot 1 \cdot 10^6 bit/s) \\ &= 1000 \cdot 10^3 bit - 9,99 \cdot 10^3 bit \\ &= 990,01 kbit \end{aligned}$$

Dieser Burst muss nun mit der garantierten Rate gr_1 weitergeleitet werden, wodurch sich die Verzögerung $d_{R_1}^{max}$ berechnen lässt:

$$\begin{aligned} d_{R_1}^{max} &= \frac{B_{R_1}^{max}}{gr_1} \\ &= \frac{990,01 \cdot 10^3 bit}{1 \cdot 10^6 Bit/s} \\ &= 990,01 \cdot 10^{-3} s = 990,01ms \end{aligned} \tag{4.4}$$

Auswirkungen der Token-Bucket-Parameter Um die Auswirkungen von r_1 und b_1 auf die maximale Verzögerung zu veranschaulichen, werden die in Abbildung 4.2 angegebenen Werten verwendet und zuerst b_1 und anschließend r_1 variiert. Mit einer Veränderung von r_1 wird analog gr_1 angepasst, sodass $r_1 = gr_1$ weiterhin gilt. Zur Variation von r_1 und b_1 wurde die Formel für $d_{R_1}^{max}$ umgeformt:

$$\begin{aligned} d_{R_1}^{max} &= \frac{B_{R_1}^{max}}{gr_1} \\ &= \frac{b_1 - (T^{max} - \frac{S^{ep}}{R_1^{in,P_1}}) \cdot gr_1}{gr_1} \\ &= \frac{b_1 - \frac{b_1 - S^{ep}}{R_1^{in,P_1}} \cdot gr_1}{gr_1} \\ &= \frac{b_1}{gr_1} - \frac{b_1 - S^{ep}}{R_1^{in,P_1}} \\ &= \frac{b_1}{gr_1} - \frac{b_1}{R_1^{in,P_1}} + \frac{S^{ep}}{R_1^{in,P_1}} \\ &= \frac{b_1}{r_1} - \frac{b_1}{100 \cdot 10^6 bit/s} + 0,01ms \end{aligned} \tag{4.5}$$

Damit wurde ein Diagramm für verschiedene b_1 und r_1-Werte geplottet. Abbildung 4.3 zeigt die maximale Verzögerung von Router R_1 ($d_{R_1}^{max}$) in Abhängigkeit von b_1 bzw. r_1.
An Abbildung 4.3 wir deutlich, dass die maximale Verzögerung in linearer Abhängigkeit zur Bucket Größe b_1 steht. Die Token Rate r_1 (bzw. die garantierte Rate gr_1) ist umgekehrt proportional zur maximalen Verzögerung, da ihr Kehrwert in die Berechnung einfließt.
Die Anwendung kann beeinflussen, wie viele Daten sie in das Netzwerk schickt, dies entspricht der Burstgröße. Sie hat also direkte Kontrolle über b_1. Eine Änderung der Token Rate oder garantierten Rate ist für sie allerdings kaum praktikabel. Um die Senderate kümmert sich der Traffic-Shaper, welcher den Token Bucket gemäß Spezifikation implementiert. Selbst wenn die Anwendung nur „langsam“ Daten senden würde (also kleine Bursts mit Pausen dazwischen), so würde der Traffic Shaper diese Bursts mit maximaler Senderate übermitteln. Um die garantierte Rate zu ändern, müsste die Anwendung erneut eine Verbindung aufbauen und Bandbreite reservieren. Dies ist nicht praktikabel, da es je nach Implementierung zwei wesentliche Nachteile hätte:

- Die alte Reservierung könnte zunächst freigegeben werden und anschließend eine neue erstellt werden. Dies würde jedoch eine Verletzung des QoS während der Transition bedeuten. Außerdem könnten die neue Reservierung fehlschlagen, falls nicht mehr genügend Ressourcen verfügbar sind. Dies könnte passieren, wenn die neue Reservierung mehr Ressourcen benötigt als bisher oder zwischenzeitlich von einem anderen Strom eine Reservierung durchgeführt wurde.
- Alternativ könnte deshalb zunächst die neue Reservierung angefordert werden und die alte erst nach Erfolg freigegeben werden. Dadurch entstünde jedoch ein erhöhter Bandbreitenbedarf während des Übergangs. Womöglich ist für diese kurzzeitige, doppelte Reservierung nicht genügend Bandbreite vorhanden und sie schlägt fehl, obwohl für jede einzelne Reservierung ausreichend Ressourcen vorhanden wären.

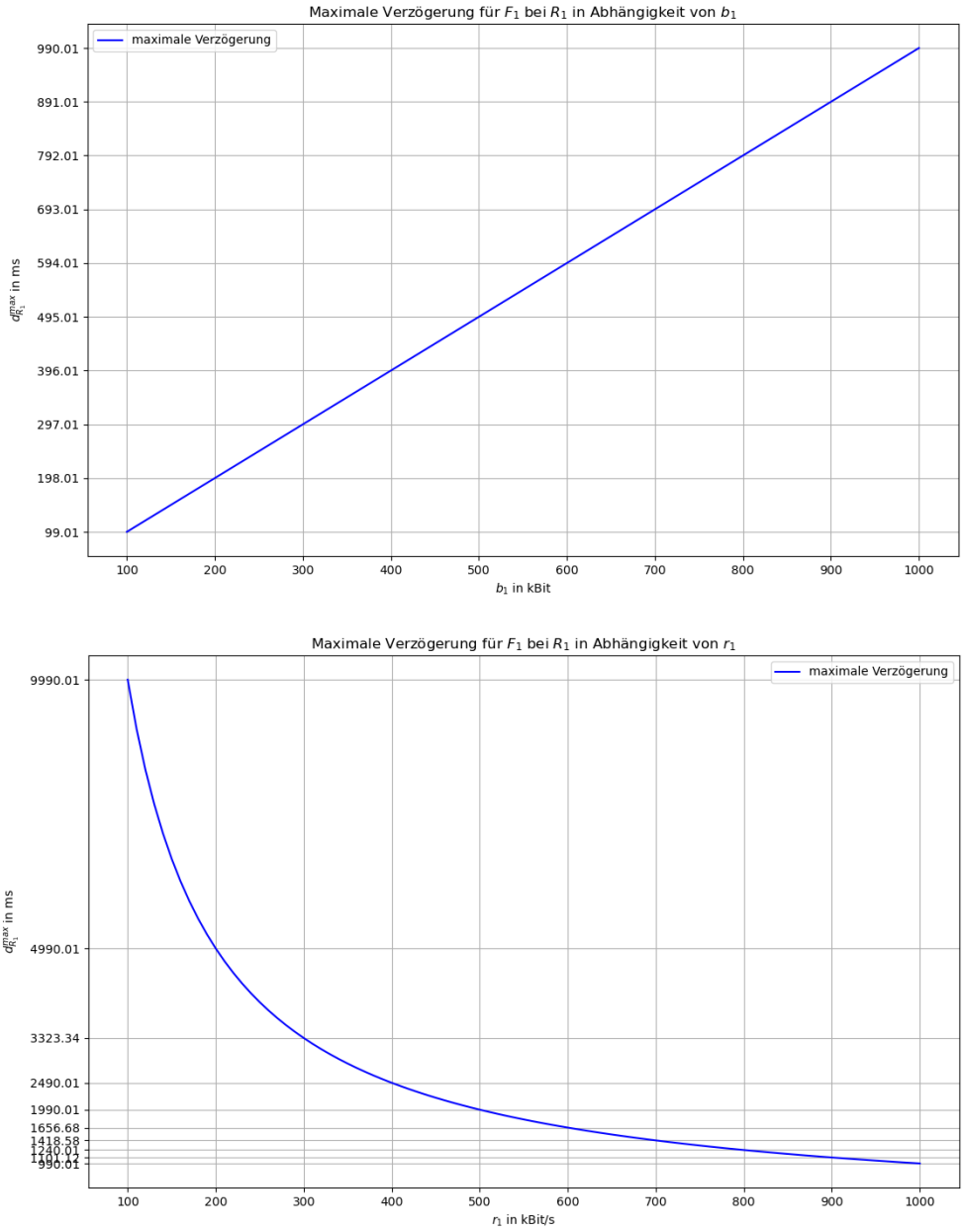

Abbildung 4.3: Maximale Verzögerung für F_1 bei R_1 in Abhängigkeit von b_1 (oben) und r_1 (unten). Für die Berechnung von b_1 wurde r_1 = 1 Mbit/s angenommen. Umgekehrt wurde für die Berechnung von r_1 angenommen, dass b_1 = 1 Mbit.

Berechnung der Gesamtverzögerung Bisher wurde nur die Verzögerung bei R_1 betrachtet. Um die Gesamtverzögerung für F_1 zu berechnen, wird im Folgenden für die Argumentation ein Fluid Modell [Jac88] verwendet. Dabei wird angenommen, dass sich ein Netzwerkstrom ähnlich wie eine Flüssigkeit verhält. Falls zu wenig Bandbreite zur Verfügung steht (= eine zu „dünne“ Leitung), so staut es sich an dieser Stelle. Das bedeutet, dass der Füllstand einer Warteschlange steigt, welche sich in einem Router vor einem Flaschenhals befindet. Interessant ist besonders, dass es zu keinem

Stau kommt, falls eine dünne Leitung in einer dickeren Leitung mündet. Übertragen bedeutet dies, dass die Warteschlange eines Routers leer bleibt, falls seine Eingangsbandbreite kleiner oder gleich seiner Ausgangsbandbreite ist.
Doch wie lässt sich nun für ein Fluid-Modell der Worst-Case ermitteln, für welchen die maximale Verzögerung entsteht? In der Analogie zu einer Flüssigkeit entstünde die maximale Verzögerung für den letzten Tropfen, der in das Netzwerk geschüttet wird. Unter der Annahme, dass er immer an letzter Stelle bleiben wird, stößt er bei jeder Engstelle auf den längstmöglichen Stau. Da im tatsächlichen Netzwerk Pakete unterwegs sind, wird dieser „letzte Tropfen“ als das letzte Paket übersetzt, welches im Zuge des Bursts verschickt wird. Für die Berechnungen wird angenommen, dass dieses Paket eine Größe von S^{lp} = 1 kBit hat. Damit kann modelliert werden, auf welche Wartezeiten das letzte Paket stößt, wie in Abbildung 4.4 zu sehen ist.

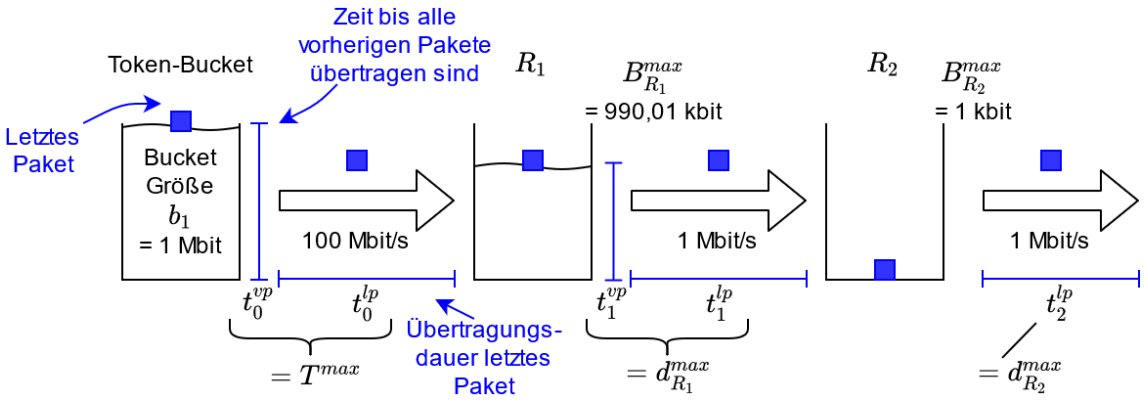

Abbildung 4.4: Modellierung der Wartezeiten des letzten Paketes, welches als Teil des Bursts übermittelt wird.

An dieser Modellierung werden mehrere interessante Eigenschaften deutlich. Für jeden Übertragungsschritt muss das letzte Paket jeweils warten, bis alle vorherigen Pakete in der Warteschlange übertragen wurden (t^{vp}). Dies entspricht im ersten Schritt $\frac{b_1 - 1\text{ kbit}}{100\text{ Mbit/s}}$. Anschließend benötigt das letzte Paket die Zeit t^{lp} um selbst übertragen zu werden. Für den ersten Übertragungsschritt also:

$$t_0^{vp} + t_0^{lp} = \frac{b_1 - 1\text{ kbit}}{100\text{ Mbit/s}} + \frac{1\text{ kbit}}{100\text{ Mbit/s}} = \frac{b_1}{100\text{ Mbit/s}} = T^{max} \tag{4.6}$$

Es ist somit nicht nötig, das letzte Paket gesondert zu betrachten.
Betrachtet man den nächsten Übertragungsschritt von R_1 zu R_2, so zeigt sich, dass die Warte- und Übertragungszeit des letzten Pakets $\frac{B^{max}_{R_1}}{1\text{ Mbit/s}} = d^{max}_{R_1}$ entspricht. Für die Gesamtverzögerung muss also der Wert $d^{max}_{R_1}$ später in die Berechnungen einfließen. Für eine weitere Engstelle bei R_2 muss erneut die verbleibende Burstgröße berechnet werden, wie in der folgenden Formel (4.7) gezeigt.
In der Abbildung ist jedoch keine erneute Engstelle bei R_2, da die reservierte Bandbreite des vorherigen Routers (R_1) kleiner oder gleich der reservierten Bandbreite von R_2 ist. Somit kommt es zu keinem Stau. Das letzte Paket landet in einer leeren Warteschlange und kann sofort weitergeleitet werden. Die maximale Burstgröße an dieser Stelle entspricht also der Größe des letzten Pakets. Damit kann folgende allgemeine Aussage über die maximale Burstgröße bei Router i gemacht werden:

$$B^{max}_{R_i} = \begin{cases} S^{lp}, & \text{wenn } \exists j < i : gr_j \leq gr_i \\ b_1 - (T^{max}_{R_{i-1}} - \frac{S^{ep}}{gr_{i-1}}) \cdot gr_i, & \text{ansonsten} \end{cases} \tag{4.7}$$

Dabei ist $T^{max}_{R_{i-1}}$ definiert als:

$$T^{max}_{R_{i-1}} = \frac{b_1}{gr_{i-1}} \tag{4.8}$$

Eine wichtige Überlegung an dieser Stelle ist die Frage, ob bei zwei aufeinanderfolgenden Engstellen die Burstgröße als Differenz des ursprünglichen Bursts b_1 oder als Differenz des vorherigen Bursts $B^{max}_{R_{i-1}}$ berechnet wird (der untere Fall von Formel (4.7)). Tatsächlich muss der ganze Burst b_1 betrachtet werden, wie in der Formel vermerkt. Denn obwohl sich bei Router R_{i-1} nur ein maximaler Burst $B^{max}_{R_{i-1}}$ in der Warteschlange befindet, so leitet er trotzdem den kompletten Burst b_1 an R_i weiter. Die Differenz zum Burst b_1 ist lediglich bereits „abgeflossen“, weshalb er nicht mehr komplett in der Warteschlange von R_{i-1} liegt. Aus Sicht von R_i kommt aber ohne Unterbrechung der komplette Burst b_1 an. Somit dauert auch die Übertragung des Bursts nicht nur $d^{max}_{R_{i-1}}$ sondern die komplette Übertragungszeit $T^{max}_{R_{i-1}}$.
Falls sich der Flaschenhals bereits bei einem vorherigen Router befand (der obere Fall von Formel (4.7)), so kommt es zu keinem Stau. Dies gilt auch, wenn es eine weitere Engstelle gibt, welche jedoch über eine größere oder gleiche Bandbreite verfügt, als der Flaschenhals. Ein Beispiel wäre eine Topologie mit Flaschenhals von 1 Mbit/s, gefolgt von einer Rate von 3 Mbit/s und schließlich erneuter Engstelle von 2 Mbit/s. Ab dem Flaschenhals beträgt die tatsächliche Datenrate nur noch 1 Mbit/s, die Pakete werden lediglich mit den schnelleren Raten weitergeleitet, gefolgt von Pausen. Somit staut es sich bei der erneuten Engstelle nicht, da die eingehenden Pakete noch immer schneller weitergeleitet werden können, als sie durch den Flaschenhals kommen.
Die letzten noch fehlenden Werte für Berechnung der Gesamtverzögerung, sind die Verarbeitungszeit d^V (Processing-Delay) und die Ausbreitungsverzögerung d^A (Propagation-Delay). Die Verarbeitungszeit ist eine konstante Zeit, welche von den Routern benötigt wird, um Pakete zu verarbeiten. Der Wert dieser Konstanten wird als $d^V = 1$ ms festgelegt. Die Ausbreitungsverzögerung ist abhängig von verschiedenen physikalischen Eigenschaften einer Verbindung zwischen zwei Routern. Beispiele dafür sind die Länge der Verbindung und das verwendete Medium (Kabel, Funk). Um die Topologien besser vergleichbar zu machen, wird im Folgenden angenommen, dass jede Verbindung die gleiche Ausbreitungsverzögerung $d^A = 1$ ms hat.
Alle Werte für die Berechnung wurden zur Übersicht nochmals in Abbildung 4.5 eingetragen.

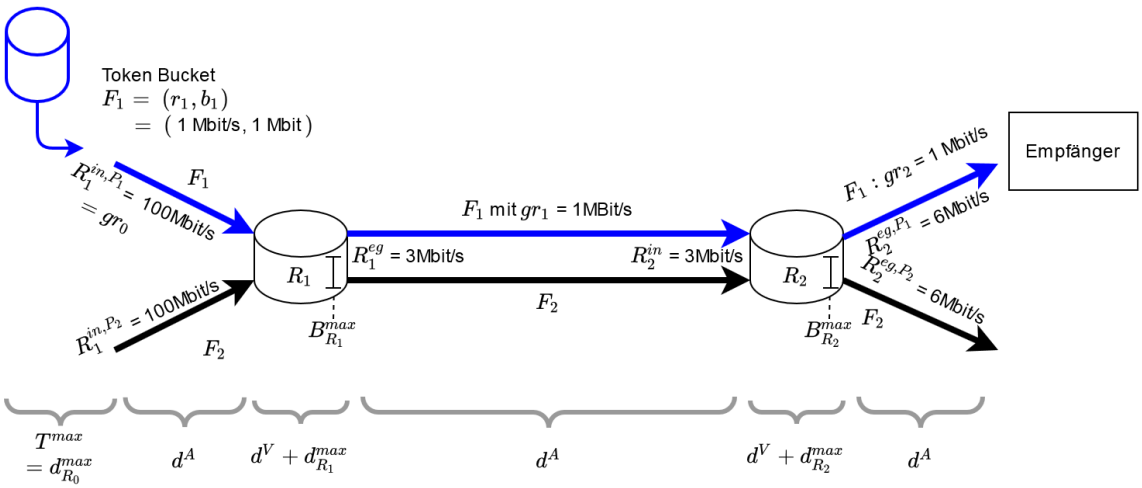

Abbildung 4.5: Übersicht der Topologie mit den verschiedenen Werten, welche für die Gesamtverzögerung berücksichtigt werden.

Die maximale Gesamtverzögerung kann somit berechnet werden mit:

$$\begin{aligned} d_{gesamt}^{max} &= d_{R_0}^{max} + d^A + d^V + d_{R_1}^{max} + d^A + d^V + d_{R_2}^{max} + d^A \\ &= \frac{b_1}{R_1^{in,P_1}} + 1ms + 1ms + \frac{B_{R_1}^{max}}{gr_1} + 1ms + 1ms + \frac{B_{R_2}^{max}}{gr_2} + 1ms \\ &= \frac{1 \cdot 10^6\ bit}{100 \cdot 10^6\ bit/s} + \frac{990,01 \cdot 10^3\ bit}{1 \cdot 10^6\ bit/s} + \frac{S^{lp}}{gr_2} + 5ms \\ &= 10ms + 990,01ms + \frac{1 \cdot 10^3\ bit}{1 \cdot 10^6\ bit/s} + 5ms \\ &= 10ms + 990,01ms + 1ms + 5ms \\ &= 1006,01ms \end{aligned} \tag{4.9}$$

Da die Eingangsbandbreite gr_1 von R_2 gleich wie die reservierte Ausgangsbandbreite gr_2 ist, wird die Warteschlange für F_1 immer leer bleiben. Deshalb gilt $B_{R_2}^{max} = S^{lp} = 1$ kbit.
Mit dieser Formel kann nun auch die Gesamtverzögerung in Abhängigkeit von b_1, gr_1 und gr_2 angegeben werden, indem die Summanden entsprechend eingesetzt werden. Der Übersichtlichkeit halber wird R_1^{in,P_1} im Folgenden gr_0 bezeichnet:

$$\begin{aligned} d_{gesamt}^{max} &= d_{R_0}^{max} + d^A + d^V + d_{R_1}^{max} + d^A + d^V + d_{R_2}^{max} + d^A \\ &= \frac{b_1}{gr_0} + 1ms + 1ms + \frac{B_{R_1}^{max}}{gr_1} + 1ms + 1ms + \frac{B_{R_2}^{max}}{gr_2} + 1ms \\ &= \frac{b_1}{gr_0} + \frac{b_1 - (d_{R_0}^{max} - \frac{S^{ep}}{gr_0}) \cdot gr_1}{gr_1} + \frac{S^{lp}}{gr_2} + 5ms \\ &= \frac{b_1}{gr_0} + \frac{b_1 - (\frac{b_1}{gr_0} - \frac{S^{ep}}{gr_0}) \cdot gr_1}{gr_1} + \frac{S^{lp}}{gr_2} + 5ms \\ &= \frac{b_1}{gr_0} + \frac{b_1}{gr_1} - \frac{b_1}{gr_0} + \frac{S^{ep}}{gr_0} + \frac{S^{lp}}{gr_2} + 5ms \\ &= \frac{b_1}{gr_1} + \frac{1\text{ kbit}}{gr_0} + \frac{1\text{ kbit}}{gr_2} + 5ms \\ &\leq \frac{b_1}{r_1} + \frac{2\text{ kbit}}{r_1} + 5ms \end{aligned} \tag{4.10}$$

Diese Formel gilt für alle Kombinationen von gr_1 und gr_2, bei denen $gr_1 \leq gr_2$ erfüllt ist, also der Flaschenhals bei gr_1 liegt. Man kann direkt erkennen, dass die Verzögerung wieder linear von b_1 abhängig ist und umgekehrt proportional zu gr_1 und gr_2. Da gr_1, gr_2 und R_1^{in,P_1} mindestens so groß wie r_1 sein müssen, ist die Abschätzung mit r_1 möglich.

Gesamtverzögerung für allgemeine Topologien mit n Routern Aus dem vorherigen Abschnitt ist bekannt, wie die Verzögerung berechnet werden kann, wenn die Topologie aus zwei Routern und einem Flaschenhals direkt beim ersten Router besteht. Mit dem Fluid-Modell kann damit direkt eine Formel für beliebige Topologien abgeleitet werden, für welche der Flaschenhals ebenfalls beim ersten Router liegt. Gegeben sei die Topologie in Abbildung 4.6. Darin liegt der Flaschenhals bei R_1, gefolgt von beliebig vielen schnelleren Verbindungen.

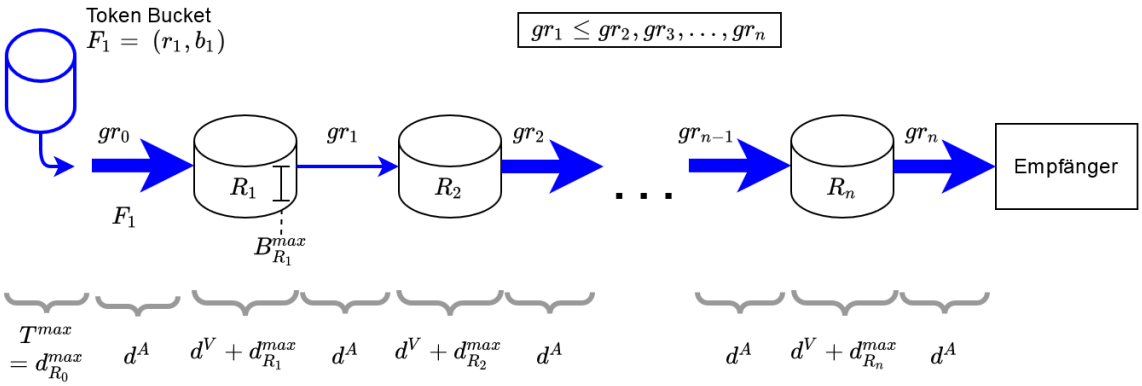

Abbildung 4.6: Eine Topologie beliebiger Länge mit Flaschenhals zwischen R_1 und R_2.

Die reservierten Bandbreiten aller Router nach R_1 sind größer oder gleich der garantieren Rate gr_1, der Flaschenhals liegt also zwischen R_1 und R_2. Dadurch staut sich der Netzwerkverkehr bei R_1 und der maximale Burst $B_{R_1}^{max}$ kann wie zuvor berechnet werden. Anschließend staut es sich jedoch bei keinem der Router mehr, weshalb nur noch die Größe des letzten Pakets S^{lp} für die Berechnung der Verzögerung benötigt wird. Unter der Annahme, dass die Verarbeitungszeit d^Q für jeden Router gleich ist, kann damit folgende Formel für die Gesamtverzögerung d_{gesamt}^{max} verwendet werden:

$$
\begin{aligned}
d_{gesamt}^{max} &= d_{R_0}^{max} + d_{R_1}^{max} + \sum_{i=2}^{n} d_{R_j}^{max} + nd^V + (n+1)d^A \\
&= \frac{b_1}{gr_0} + \frac{B_{R_1}^{max}}{gr_1} + \sum_{i=2}^{n} \frac{B_{R_i}^{max}}{gr_i} + nd^V + (n+1)d^A \\
&= \frac{b_1}{gr_0} + \frac{b_1 - (\frac{b_1}{gr_0} - \frac{S^{ep}}{gr_0}) \cdot gr_1}{gr_1} + \sum_{i=2}^{n} \frac{S^{lp}}{gr_i} + nd^V + (n+1)d^A \\
&= \frac{b_1}{gr_0} + \frac{b_1}{gr_1} - \frac{b_1}{gr_0} + \frac{S^{ep}}{gr_0} + \sum_{i=2}^{n} \frac{S^{lp}}{gr_i} + nd^V + (n+1)d^A \\
&= \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_0} + \sum_{i=2}^{n} \frac{S^{lp}}{gr_i} + nd^V + (n+1)d^A \\
&\leq \frac{b_1}{r_1} + \frac{S^{ep}}{r_1} + \frac{(n-1) \cdot S^{lp}}{r_1} + nd^V + (n+1)d^A
\end{aligned}
\tag{4.11}
$$

Für jeden Router R_i mit gr_i nach dem Flaschenhals erhöht sich also die Verzögerung um $\frac{S^{lp}}{gr_i} + d^V + d^A$.

Gesamtverzögerung für monoton fallende Engstellen Nun ist bekannt, wie die Verzögerung bei beliebig vielen Routern nach dem Flaschenhals berechnet werden kann. Es bleibt die Frage, welchen Einfluss die Übertragungsschritte vor dem Flaschenhals haben. Dazu wird als nächstes der Fall betrachtet, dass mehrere Engstellen aufeinander folgen, deren Bandbreiten kontinuierlich abnehmen (monoton fallen). Dies wird beispielhaft an einer Topologie durchgerechnet und schließlich

der allgemeine Fall betrachtet. Es sei eine Topologie gegeben, in welcher zwei Engstellen direkt aufeinander folgen. Die garantierte Rate der ersten Engstelle betrage 2 Mbit/s, die der zweiten 1 Mbit/s. Die zweite Engstelle ist damit der Flaschenhals. In Abbildung 4.7 wird die Topologie nochmals dargestellt.

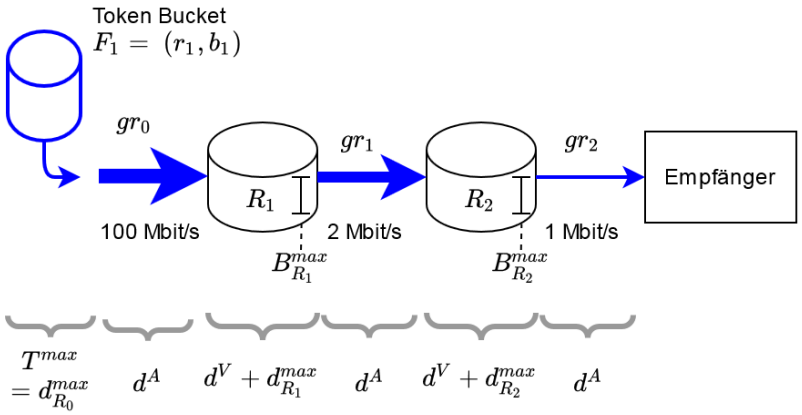

Abbildung 4.7: Eine Topologie mit kontinuierlicher Abnahme der Bandbreite. Der Übersichtlichkeit halber wurde statt R_1^{in,P_1} die Bezeichnung gr_0 verwendet.

Intuitiv handelt es sich bei diesem Fall lediglich um eine Verkettung des Falls mit einer Engstelle. Im Beispiel gibt es zwei Engstellen, somit muss die Berechnung der ersten Engstelle (100 Mbit/s -> 2 Mbit/s) wie bisher durchgeführt werden. Anschließend kommt die Verzögerung der zweiten Engstelle (2 Mbit/s -> 1 Mbit/s) hinzu. Zur bisherigen Summe kommen also lediglich die Summanden für den Router R_2 hinzu. Die Formel zur Berechnung der maximalen Burstgröße wurde bereits für den allgemeinen Fall bei Router R_i definiert (4.7). Deshalb kann die maximale Burstgröße wie bisher für R_1 und auch für R_2 berechnet werden. Da zur Berechnung der Verzögerung $d^{max}_{R_2}$ nur die maximale Burstgröße $B^{max}_{R_2}$ und die garantierte Rate gr_2 benötigt wird, sind damit alle Werte bekannt. Somit lässt sich die Gesamtverzögerung für diese Topologie berechnen mit:

$$
\begin{aligned}
d^{max}_{gesamt} &= d^{max}_{R_0} + d^A + d^V + d^{max}_{R_1} + d^A + d^V + d^{max}_{R_2} + d^A \\
&= T^{max}_{R_0} + 1ms + 1ms + \frac{B^{max}_{R_1}}{gr_1} + 1ms + 1ms + \frac{B^{max}_{R_2}}{gr_2} + 1ms \\
&= \frac{b_1}{gr_0} + \frac{b_1 - (T^{max}_{R_0} - \frac{S^{ep}}{gr_0}) \cdot gr_1}{gr_1} + \frac{b_1 - (T^{max}_{R_1} - \frac{S^{ep}}{gr_1}) \cdot gr_2}{gr_2} + 5ms \\
&= \frac{b_1}{gr_0} + (\frac{b_1}{gr_1} - T^{max}_{R_0} + \frac{S^{ep}}{gr_0}) + (\frac{b_1}{gr_2} - T^{max}_{R_1} + \frac{S^{ep}}{gr_1}) + 5ms \\
&= \frac{b_1}{gr_0} + (\frac{b_1}{gr_1} - \frac{b_1}{gr_0} + \frac{S^{ep}}{gr_0}) + (\frac{b_1}{gr_2} - \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_1}) + 5ms \\
&= \frac{b_1}{gr_2} + \frac{S^{ep}}{gr_0} + \frac{S^{ep}}{gr_1} + 5ms \\
&= \frac{1 \cdot 10^6\ bit}{1 \cdot 10^6\ bit/s} + \frac{1 \cdot 10^3\ bit}{100 \cdot 10^6\ bit/s} + \frac{1 \cdot 10^3\ bit}{2 \cdot 10^6\ bit/s} + 5ms \\
&= 1000ms + 0,01ms + 0,5ms + 5ms = 1005,51ms
\end{aligned}
\tag{4.12}
$$

Im Allgemeinen gilt für eine Topologie, bei welcher kontinuierlich die reservierte Bandbreite abnimmt, die folgende Formel. Es gilt $gr_0 > gr_1 > ... > gr_n$ wobei der Flaschenhals bei R_n liegt:

$$\begin{aligned} d^{max}_{gesamt} &= d^{max}_{R_0} + d^A + d^V + d^{max}_{R_1} + d^A + d^V + d^{max}_{R_2} + ... + d^A + d^V + d^{max}_{R_n} + d^A \\ &= T^{max}_{R_0} + \sum_{i=1}^{n} d^{max}_{R_i} + nd^V + (n+1)d^A \\ &= \frac{b_1}{gr_0} + \sum_{i=1}^{n} (\frac{b_1}{gr_i} - \frac{b_1}{gr_{i-1}} + \frac{S^{ep}}{gr_{i-1}}) + nd^V + (n+1)d^A \\ &= \frac{b_1}{gr_n} + \sum_{i=0}^{n-1} \frac{S^{ep}}{gr_i} + nd^V + (n+1)d^A \end{aligned} \tag{4.13}$$

Nimmt die reservierte Bandbreite kontinuierlich ab, so erhöht sich die Gesamtverzögerung für jeden Router R_i mit gr_i vor dem Flaschenhals um $\frac{S^{ep}}{gr_i} + d^V + d^A$.

Gesamtverzögerung für nicht-monoton fallende Engstellen Es ist nun bekannt, welche Verzögerung durch monoton fallende Engstellen entsteht und welchen Einfluss schnellere Verbindungen nach dem Flaschenhals haben. Damit verbleibt nur noch der Fall, dass mehrere Engstellen und schnellere Verbindungen gemischt vor dem Flaschenhals auftreten. Die Bandbreite vor dem Flaschenhals fällt also nicht-monoton. Da die Berechnungen für Verbindungen nach dem Flaschenhals bereits bekannt sind, müssen nur die Verbindungen bis zum Flaschenhals betrachtet werden. Es wird deshalb angenommen, dass sich der Empfänger direkt nach dem Flaschenhals befindet, um die Berechnungen zu verkürzen. Eine mögliche Topologie für diesen Fall ist in Abbildung 4.8 dargestellt. Dort befindet sich zu Beginn eine Engstelle mit 2 Mbit/s, gefolgt von einer schnelleren Verbindung mit 3 Mbit/s und anschließendem Flaschenhals mit 1 Mbit/s.

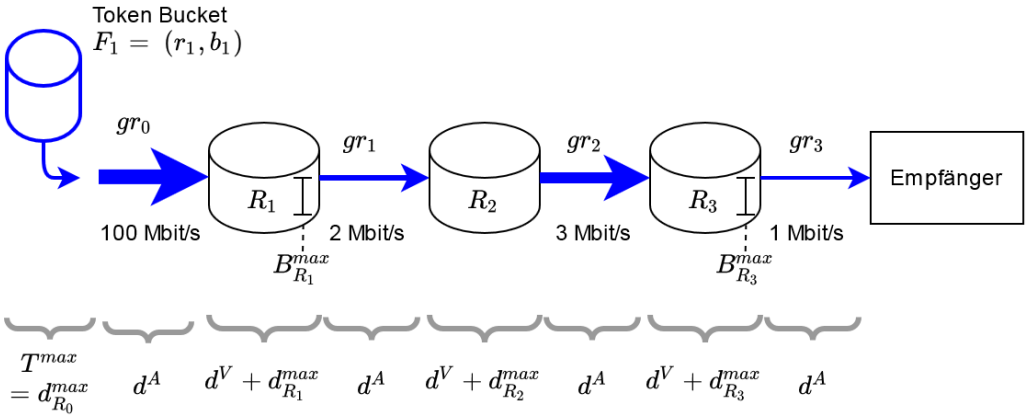

Abbildung 4.8: Eine Topologie mit mehreren Engstellen und einer schnelleren Verbindung dazwischen. Der Übersichtlichkeit halber wurde statt R_1^{in,P_1} die Bezeichnung gr_0 verwendet.

Die Verzögerungen im Beispiel für $d^{max}_{R_0}, d^{max}_{R_1}$ und $d^{max}_{R_2}$ sind bereits mit den bekannten Formeln

berechenbar. Ein Problem wird jedoch sichtbar bei der Berechnung von $d_{R_3}^{max}$. Mit den bisherigen Formeln würde gelten:

$$\begin{aligned} d_{R_3}^{max} &= \frac{B_{R_3}^{max}}{gr_3} \\ &= \frac{b_1 - (T_{R_2}^{max} - \frac{S^{ep}}{gr_2}) \cdot gr_3}{gr_3} \\ &= \frac{b_1 - (\frac{b_1}{gr_2} - \frac{S^{ep}}{gr_2}) \cdot gr_3}{gr_3} \end{aligned} \tag{4.14}$$

Das Problem liegt bei der Verwendung von $T_{R_2}^{max}$ an dieser Stelle. Dies würde bedeuten, dass R_3 den *kompletten* Burst b_1 mit der Geschwindigkeit von 3 Mbit/s empfängt. Tatsächlich kann R_2 die *einzelnen Pakete* mit 3 Mbit/s weiterleiten, allerdings entstehen dazwischen Pausen, da die Engstelle bei R_1 nur 2 Mbit/s liefert. Somit beträgt die Dauer, bis der komplette Burst bei R_3 angekommen ist $\frac{b_1}{gr_1} = T_{R_1}^{max}$. Entsprechend muss auch $\frac{S^{ep}}{gr_1}$ statt $\frac{S^{ep}}{gr_2}$ verwendet werden. Die korrekte Formel für $d_{R_3}^{max}$ des Beispiels ist also:

$$d_{R_3}^{max} = \frac{b_1 - (\frac{b_1}{gr_1} - \frac{S^{ep}}{gr_1}) \cdot gr_3}{gr_3} \tag{4.15}$$

Damit kann die Gesamtverzögerung des Beispiels berechnet werden mit:

$$\begin{aligned} d_{gesamt}^{max} &= d_{R_0}^{max} + d^A + d^V + d_{R_1}^{max} + d^A + d^V + d_{R_2}^{max} + d^A + d^V + d_{R_3}^{max} + d^A \\ &= \frac{b_1}{gr_0} + \frac{b_1 - (\frac{b_1}{gr_0} - \frac{S^{ep}}{gr_0}) \cdot gr_1}{gr_1} + \frac{S^{lp}}{gr_2} + \frac{b_1 - (\frac{b_1}{gr_1} - \frac{S^{ep}}{gr_1}) \cdot gr_3}{gr_3} + 3d^V + 4d^A \\ &= \frac{b_1}{gr_3} + \frac{S^{ep}}{gr_0} + \frac{S^{ep}}{gr_1} + \frac{S^{lp}}{gr_2} + 3d^V + 4d^A \end{aligned} \tag{4.16}$$

Im Allgemeinen muss also die Übermittlungszeit der *zuletzt passierten Engstelle* verwendet werden, um die Größe des verbleibenden Bursts zu berechnen. Dies zu formalisieren wäre kompliziert, es ist jedoch nicht nötig, um die finale Formel für beliebige Topologien aufzustellen. Man kann folgende Beobachtung an der Berechnung des Beispiels machen: Die Verzögerung ist identisch zur Topologie, in welcher R_2 und R_3 getauscht sind. Dadurch entsteht eine Topologie mit monoton fallenden Engstellen vor dem Flaschenhals, welche mit den bisherigen Formeln berechnet werden kann.

Diese Eigenschaft gilt auch für allgemeine Topologien, was im Folgenden gezeigt wird. Es sei eine Topologie gegeben mit einer ersten Engstelle und Bandbreite gr_1, gefolgt von beliebig vielen schnelleren Verbindungen $(gr_2, ..., gr_{n-1})$ und schließlich dem Flaschenhals gr_n. Es wird nun gezeigt, dass die Verzögerung dieser Topologie identisch ist zur Topologie, in welcher direkt auf gr_1 der Flaschenhals mit gr_n folgt und die schnelleren Verbindungen danach liegen. In Abbildung 4.9 sind die beiden Topologien nochmals illustriert.

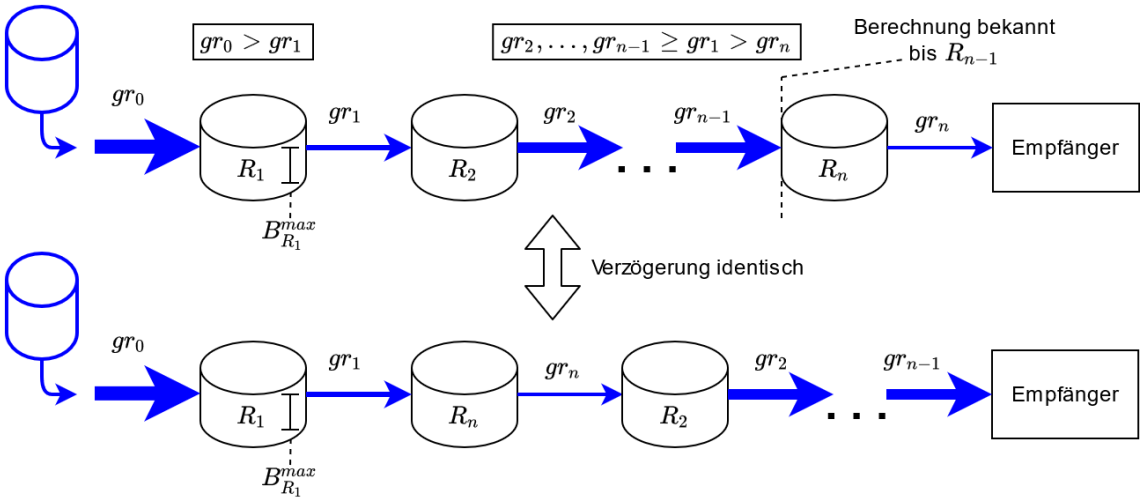

Abbildung 4.9: Zwei allgemeine Topologien, welche umgeordnet werden können und die gleiche Verzögerung haben.

Betrachtet man für die obere Topologie nur den Teil bis zu Router R_n, also ohne den tatsächlichen Flaschenhals R_n, so läge der Flaschenhals bei R_2 und es handelt sich um eine monoton fallende Engstelle (da $gr_0 > gr_1$). Dieser Fall ist in der Abbildung mit der gestrichelten Linie eingezeichnet. Die Formeln dafür sind bekannt und es lässt sich die Verzögerung ohne R_n berechnen:

$$d^{max}_{gesamt(ohne\ R_n)} = \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_0} + \sum_{i=2}^{n-1} \frac{S^{lp}}{gr_i} + (n-1)d^V + (n)d^A \tag{4.17}$$

Für die Gesamtverzögerung der oberen Topologie fehlt an dieser Stelle nur noch die Verzögerung von R_n zum Empfänger. Wie im Beispiel zuvor muss für die Berechnung von $d^{max}_{R_n}$ die Übermittlungszeit der *zuletzt passierten Engstelle* verwendet werden. Diese Engstelle liegt bei R_1, somit kann $d^{max}_{R_n}$ wie folgt berechnet werden:

$$d^{max}_{R_n} = \frac{b_1 - (\frac{b_1}{gr_1} - \frac{S^{ep}}{gr_1}) \cdot gr_n}{gr_n} = \frac{b_1}{gr_n} - \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_1} \tag{4.18}$$

Außerdem kommt für den Router R_n wieder die Verarbeitungszeit d^V hinzu, sowie die Ausbreitungsverzögerung d^A der Verbindung von R_n zum Empfänger. Addiert man die Verzögerung vom Sender bis R_n und die Verzögerung von R_n zum Empfänger, so erhält man die Gesamtverzögerung der oberen Topologie:

$$\begin{aligned} d^{max}_{gesamt,oben} &= d^{max}_{gesamt(ohne\ R_n)} + d^{max}_{R_n} + d^V + d^A \\ &= \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_0} + \sum_{i=2}^{n-1} \frac{S^{lp}}{gr_i} + (n-1)d^V + (n)d^A + \frac{b_1}{gr_n} - \frac{b_1}{gr_1} + \frac{S^{ep}}{gr_1} + d^V + d^A \\ &= \frac{b_1}{gr_n} + \frac{S^{ep}}{gr_0} + \frac{S^{ep}}{gr_1} + \sum_{i=2}^{n-1} \frac{S^{lp}}{gr_i} + nd^V + (n+1)d^A \end{aligned} \tag{4.19}$$

Nun wird die Verzögerung für die umgeformte (untere) Topologie berechnet. Hierfür kann die Formel für monoton fallende Engstellen und für beliebige Verbindungen nach dem Flaschenhals verwendet werden. Die Gesamtverzögerung beträgt somit:

$$d^{max}_{gesamt,unten} = \frac{b_1}{gr_n} + \frac{S^{ep}}{gr_0} + \frac{S^{ep}}{gr_1} + \sum_{i=2}^{n-1} \frac{S^{lp}}{gr_i} + nd^V + (n+1)d^A \tag{4.20}$$

Beide Formeln sind identisch, woraus folgt, dass die umgeordnete Topologie die identische maximale Gesamtverzögerung hat. Streng genommen wurde damit nur gezeigt, dass eine Topologie, mit zwei Engstellen (R_1 und R_n) und schnelleren Verbindungen dazwischen, umgeordnet werden kann. Für eine Topologie mit noch mehr Engstellen kann die Argumentation aber wiederholt angewandt werden.
Eine Topologie, bestehend aus mehreren Engstellen und schnelleren Verbindungen dazwischen, kann also für die Berechnung so umgeordnet werden, dass alle schnelleren Verbindungen nach dem Flaschenhals liegen. Dadurch verbleibt eine Topologie mit monoton fallenden Engstellen bis zum Flaschenhals und anschließend beliebig viele schnellere Verbindungen als der Flaschenhals.

Formel für die Gesamtverzögerung einer beliebigen Topologie An den zuvor betrachteten Fällen wurden folgende Eigenschaften deutlich:

- Zur Berechnung kann die Topologie so umgeordnet werden, dass vor dem Flaschenhals nur monoton fallende Engstellen liegen.
- Befinden sich vor dem Flaschenhals nur monoton fallende Engstellen, so erhöht sich die Gesamtverzögerung für jeden Router R_i mit gr_i vor dem Flaschenhals um $\frac{S^{ep}}{gr_i} + d^V + d^A$.
- Für einen beliebigen Router R_j mit gr_j nach dem Flaschenhals, erhöht sich die Verzögerung jeweils um $\frac{S^{lp}}{gr_j} + d^V + d^A$.
- Für den Flaschenhals R_f mit gr_f erhöht sich die Gesamtverzögerung um $\frac{b_1}{gr_f} + d^V + d^A$

Die Berechnung der Verzögerung von Verbindungen vor und nach dem Flaschenhals unterscheidet sich lediglich an der Größe des ersten bzw. letzten Pakets des Bursts b_1. Unter der Annahme, dass das erste und letzte Paket die gleiche Größe haben, also $S = S^{ep} = S^{lp}$, kann eine einfache Formel für die Gesamtverzögerung aufgestellt werden:

$$d^{max}_{gesamt} = \frac{b_1}{gr_f} + \sum_{i=0, i\neq f}^{n} \frac{S}{gr_i} + nd^V + (n+1)d^A \tag{4.21}$$

Es bezeichnet dabei gr_f die Bandbreite des Flaschenhalses und n die Anzahl der Router.

Theorem IntServ1: Kleine Bursts haben niedrigere garantierte Verzögerungen als große Bursts

Dieses Theorem wird mit der zuvor aufgestellten Formel (4.21) gezeigt. Es sei ein Token-Bucket (b,r) gegeben sowie eine Topologie mit reservierter Bandbreite gr_i für jeden Router R_i. Es wird angenommen, dass jeder Router genau die Token-Rate r reserviert, es gilt also $\forall i : gr_i = r$. Die Anwendung sendet nun periodisch Bursts der Größe $b(m) = \frac{b}{m}$. Dabei hängt die Periode p von der Anzahl der Bursts (m) pro Zeiteinheit ab, sodass in jedem Fall die durchschnittliche Senderate der Token-Rate r entspricht. Die Anwendung kann also pro Zeiteinheit m Bursts der Größe $\frac{b}{m}$ senden. Die Periode $p(m)$ kann berechnet werden mit:

$$p(m) = \frac{b(m)}{r} = \frac{b}{m \cdot r} \tag{4.22}$$

Eine Illustration von verschiedenen Perioden und Burst Größen wird in Abbildung 4.10 gezeigt.

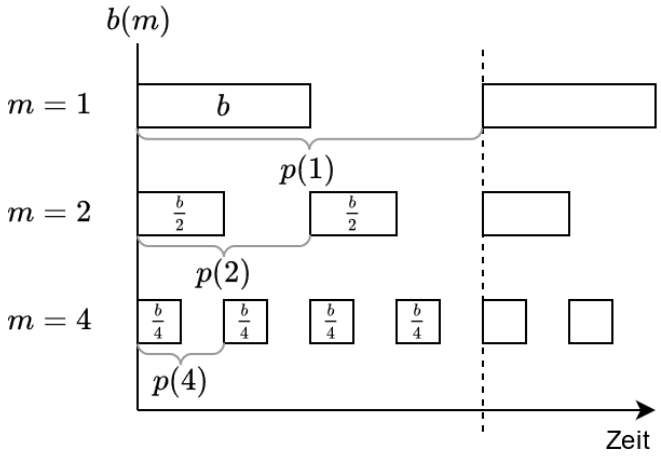

Abbildung 4.10: Beispiele verschiedener Burst-Größen und entsprechenden Perioden, sodass die durchschnittliche Senderate r gleich bleibt.

Mit der zuvor aufgestellten Formel (4.21) kann die Gesamtverzögerung in Abhängigkeit von m berechnet werden:

$$\begin{aligned} d^{max}_{gesamt}(m) &= \frac{b(m)}{gr_f} + \sum_{i=0, i\neq f}^{n} \frac{S}{gr_i} + nd^V + (n+1)d^A \\ &= \frac{b(m)}{r} + \frac{n \cdot S}{r} + nd^V + (n+1)d^A \\ &= \frac{b}{m \cdot r} + \frac{n \cdot S}{r} + nd^V + (n+1)d^A \end{aligned} \tag{4.23}$$

Mithilfe der Ableitung kann das Minimum berechnet werden:

$$\begin{aligned} \frac{d}{dm}(d^{max}_{gesamt}(m)) &= -\frac{b}{m^2 \cdot r} \\ 0 \overset{!}{=} -\frac{b}{m^2 \cdot r} &\xrightarrow{m\to\infty} 0 \end{aligned} \tag{4.24}$$

Die Ableitung geht für $m \to \infty$ gegen 0. Um eine minimale Verzögerung zu erhalten, müsste der Burst also unendlich oft geteilt werden, was einer Burst-Größe von 0 entspräche. Es stellt sich die Frage, welche Verzögerung für $m \to \infty$ verbleiben würde. Geht man von einem Fluid Modell

ohne Pakete aus, so könnte m beliebig groß gewählt werden. Für $m \to \infty$ würde der erste Term der Formel $(\frac{b}{m \cdot r})$ gegen 0 gehen. Da keine Pakete betrachtet werden, entspräche die Paketgröße $S = 0$ wodurch der Term $\frac{n \cdot S}{r}$ ebenfalls 0 wird. Es würden somit nur die konstanten Werte der Verarbeitungszeit d^V und Ausbreitungsverzögerung d^A übrig bleiben. Tatsächlich kann m allerdings nicht beliebig vergrößert werden, da irgendwann die minimale Paketgröße erreicht ist. Die Bursts bestünden dann lediglich aus einem Paket der minimalen Paketgröße, wodurch das „erste“ und „letzte“ Paket ein und dasselbe sind. Es würde als gelten, dass $b = S^{min} = S^{ep} = S^{lp} = S$. Die minimale Verzögerung unter Berücksichtigung von Paketen konvergiert für $m \to \infty$ somit gegen:

$$\begin{aligned} \lim_{m \to \infty} d^{max}_{gesamt}(m) &= \frac{S^{min}}{r} + \frac{n \cdot S}{r} + nd^V + (n+1)d^A \\ &= \frac{(n+1) \cdot S^{min}}{r} + nd^V + (n+1)d^A \end{aligned} \tag{4.25}$$

In Abbildung 4.11 wurde die maximale Verzögerung in Abhängigkeit von $\frac{b}{m}$ nochmals geplottet. Dabei beträgt b = 50 kbit, r = 100 kbit/s, $S = S^{min}$ = 1 kbit, $d^V = 1ms$ und $d^A = 1ms$. Die Anzahl der Router n wurde ebenfalls variiert und im selben Plot eingezeichnet.

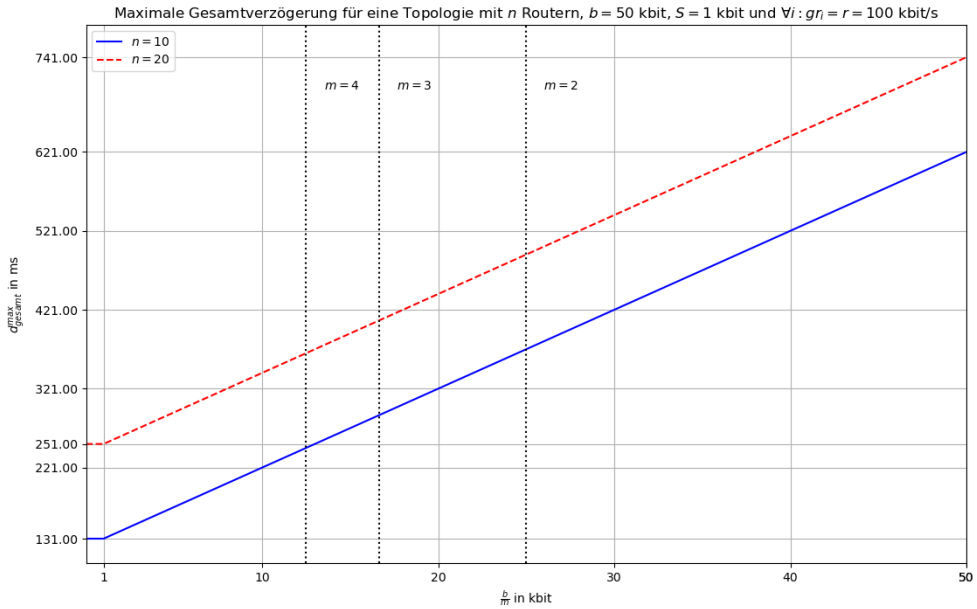

Abbildung 4.11: Ein Plot der maximalen Verzögerung in Abhängigkeit von $\frac{b}{m}$. Für $\frac{b}{m}$ = 1 kbit hat der Burst die minimale Paketgröße erreicht, dadurch sinkt die minimale Verzögerung nicht weiter.

Am Plot kann man den Punkt erkennen, an welchem die Burstgröße soweit verkleinert wurde, dass sie nur noch einem Paket entspricht (bei $\frac{b}{m}$ = 1 kbit). Außerdem wird deutlich, dass die Verzögerung bis zu diesem Punkt linear abhängig von der Burstgröße ist. Es kann also festgehalten werden, dass die garantierte Verzögerung geringer ist, je kleiner der Burst ist. □

Vergleich von IntServ mit dem neuen QoS Klassen-Modell der Problemstellung Nachdem die Verzögerungen von IntServ nun im Detail untersucht wurden, kann das Modell mit dem neuen Ansatz aus der Problemstellung verglichen werden. Das gezeigte Theorem besagt, dass die garantierte Verzögerung von IntServ für kleine Bursts geringer ist, als für große Bursts. Es gibt somit eine Abstufung innerhalb der garantierten Verzögerung des Modells, abhängig von der tatsächlich gesandten Burstgröße. Diese Abstufung steht in linearer Abhängigkeit zur Burstgröße, wie am Plot des Theorems (Abb. 4.11) deutlich wurde.
Allerdings verhält sich IntServ dennoch nur wie ein typisches 2-Klassen QoS Modell, wenn die Anwendung Bandbreite für die Stabilität reserviert. Angenommen, die Anwendung benötigt für die Stabilität einen Token-Bucket mit (r_S, b_S). Wenn sie mit dieser Spezifikation eine Verbindung über IntServ reserviert, so erhält sie die benötigten Garantien für die Stabilität. Sobald die Anwendung jedoch größere Bursts sendet, was in den Performance-Bereich des neuen Ansatzes fallen würde, so überschreitet sie die Reservierung und erhält keine Garantien mehr. Die Abstufungen von IntServ gelten nur innerhalb der Reservierung und würden hier keine Vorteile liefern. Dieses Verhalten wird in Abbildung 4.12a nochmals illustriert.
IntServ kann leicht in das neue Modell mit mehreren Klassen umgebaut werden. Anstatt eine Reservierung nur für die Stabilität durchzuführen, könnte eine etwas „größere“ Reservierung angefordert werden, sodass noch Garantien bis zur Klasse PN gelten (Tatsächlich muss nicht mehr Bandbreite reserviert werden, wie später erläutert wird). Dazu kann die lineare Abhängigkeit der Verzögerung zur Burstgröße genutzt werden. Anstatt nur eine Reservierung für (r_S, b_S) durchzuführen, wird für einen größeren Burst b_n reserviert. Dadurch bleiben die Garantien für die Stabilität erhalten und der anschließende Performance-Bereich bietet weiterhin eine garantierte maximale Verzögerung. Dieser Ansatz wird in Abbildung 4.12b verdeutlicht.

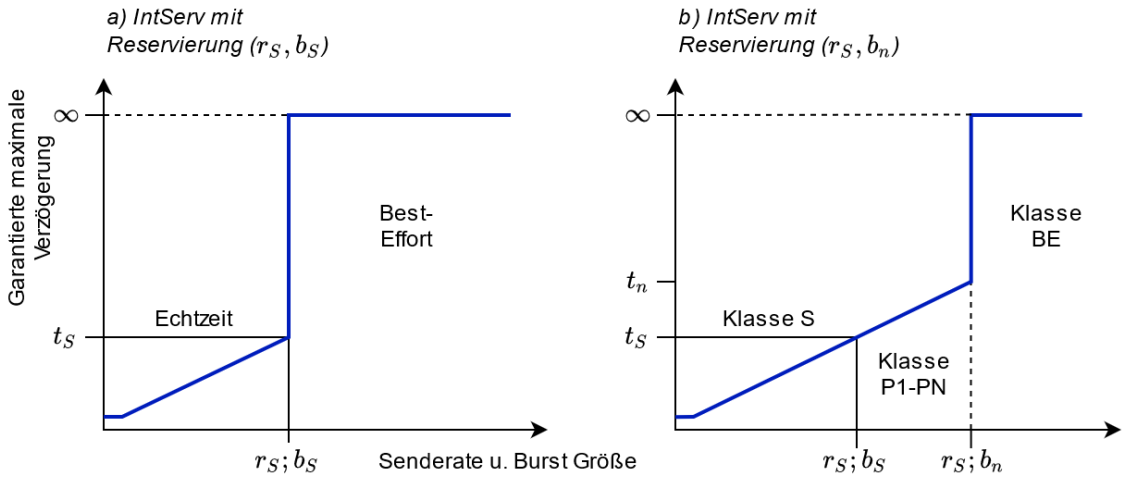

Abbildung 4.12: Die garantierten Verzögerungen von IntServ wurden in das 2-Klassen QoS-Modell eingetragen (a). Außerdem wird ein möglicher Ansatz zu Umsetzung der Performance Klassen mit IntServ veranschaulicht (b).

Mit diesem Ansatz müsste keine zusätzliche Bandbreite gegenüber der reinen „Stabilitätsreservierung“ (r_S, b_S) reserviert werden. Die Reservierung wird weiterhin so durchgeführt, dass die maximale Verzögerung für die Stabilität garantiert wird. Lediglich die Größe des Token-Buckets wird nachträglich auf b_n vergrößert, damit die größeren Bursts nicht in die Klasse BE fallen. Diese

Änderung muss nur zwischen der Anwendung und dem Token-Bucket durchgeführt werden, die bestehende Reservierung bei den Routern benötigt keine Veränderung. Damit muss die Anwendung jedoch zusätzlich ihr Sendeverhalten kontrollieren, damit die garantierte Verzögerung erhalten bleibt. Dies wird mit dem folgenden Theorem gezeigt:

Theorem IntServ2: Bei einem vergrößerten Token-Bucket b_n muss die Anwendung ihr Sendeverhalten kontrollieren, um die garantierte Verzögerung für die Stabilität zu erhalten.
Es sei eine Reservierung mit einem Token-Bucket (r_S, b_S) für die Stabilität gegeben. Die garantierte maximale Verzögerung d^{max}_{gesamt,b_S} (der Einfachheit halber im Folgenden abgekürzt als d_{b_S}) ist somit kleiner oder gleich der maximalen Verzögerung d_S, welche für die Stabilität benötigt wird. Nun wird der Token-Bucket auf $b_n > b_S$ vergrößert, ohne Änderungen an den Reservierungen der Router durchzuführen. Es wird angenommen, dass die zugewiesenen Warteschlangen für diesen Netzwerkstrom bei allen Routern mindestens so groß wie b_n sind.
Solange die Anwendung nicht mehr als b_S Tokens anspart, verhält sich der angepasste Token-Bucket identisch wie der nicht vergrößerte Token-Bucket. Spart sie mehr als b_S Tokens an, so kann sie auch einen größeren Burst senden. Der Extremfall ist ein komplett voller Bucket, wodurch sie einen Burst der Größe b_n senden kann. Die maximale Verzögerung dieses Bursts (bezeichnet mit d_{b_n}) ist größer als d_{b_S} und kann mithilfe der zuvor gezeigten Formel berechnet werden:

$$\begin{aligned} d_{b_n} &= \frac{b_n}{gr_f} + \sum_{i=0, i\neq f}^{n} \frac{S}{gr_i} + nd^V + (n+1)d^A \\ &> \frac{b_s}{gr_f} + \sum_{i=0, i\neq f}^{n} \frac{S}{gr_i} + nd^V + (n+1)d^A = d_{b_s} \end{aligned} \tag{4.26}$$

Bei allen bisherigen Berechnungen konnte immer implizit angenommen werden, dass ein Burst von maximaler Größe überall auf eine leere Warteschlange trifft. Dies gilt, denn bisher war die Zeit zwischen zwei Bursts maximaler Größe begrenzt durch $\frac{b_S}{r_S}$, also die Dauer bis ein leerer Token Bucket komplett gefüllt wurde. Da am Flaschenhals eine Bandbreite $gr_f \geq r_S$ reserviert ist, benötigt der Flaschenhals höchstens genauso lange um den vorherigen Burst weiterzuleiten. Alle anderen Router haben die gleiche oder höhere reservierte Bandbreite und benötigen somit ebenfalls nicht länger. Jeder Router benötigt somit höchstens $\frac{b_S}{gr_f} \leq \frac{b_S}{r_S}$ um den vorherigen Burst weiterzuleiten. Dadurch trifft jeder Burst bei einer Token-Bucket Größe von b_S auf eine leere Warteschlange.
Durch den größeren Token-Bucket ist diese Annahme nicht mehr korrekt. Dies kann an einem einfachen Beispiel verdeutlicht werden: Die Anwendung hat einen vollen Token-Bucket b_n angespart und sendet zuerst einen Burst $b_n - b_S$ und direkt im Anschluss einen Burst b_S. Der Burst b_S trifft nun auf eine bereits gefüllte Warteschlange durch den vorherigen Burst. Aus Sicht des Netzwerks handelt es sich um *einen* Bursts der Größe b_n, für welchen soeben gezeigt wurde, dass die Verzögerung größer als d_{b_S} ist. Die Anwendung muss nun also zusätzlich berücksichtigen, wie ihr vorheriges Sendeverhalten war, wenn sie stabilitätskritische Bursts sendet. Sie muss solange warten bis sie sich sicher sein kann, dass die Warteschlange am Flaschenhals leer genug ist, um die Stabilitätsdaten in der benötigten Zeitschranke weiterzuleiten. Diese Wartezeit t_w ist abhängig von der Größe des

vorherigen Bursts (bezeichnet als b_{vor}) und des gewünschten neuen Bursts (bezeichnet mit b_{neu}). Dabei gilt:

$$\begin{aligned} b_{vor} &\leq b_n - b_S \\ b_{neu} &\leq b_S \\ t_w &= \frac{b_{vor} - (b_S - b_{neu})}{r_S} \end{aligned} \tag{4.27}$$

Falls der neue Burst kleiner als b_S ist, muss die Anwendung nicht solange warten, bis ihr Burst sicher auf eine *leere* Warteschlange trifft. Es genügt, wenn der neue Burst auf einen Rest trifft, solange dadurch höchstens ein Burst der Größe b_S entsteht. Denn damit beträgt die Verzögerung höchstens d_{b_S}, was für die Stabilität benötigt wird.
Die Anwendung muss also bei einem nachträglich vergrößerten Token-Bucket b_n ihr Sendeverhalten kontrollieren und gegebenenfalls zwischen zwei Burst warten, damit die benötigte maximale Verzögerung nicht überschritten wird. □

Bisher noch nicht beleuchtet wurde die Frage, ob die Anwendung überhaupt genügend Tokens zur Verfügung hat, um größere Bursts zu senden. Falls die Anwendung für die Stabilität eine mittlere Datenrate von r_S benötigt, verbraucht sie alle Tokens und spart langfristig keine Tokens an. Dieser Ansatz funktioniert somit nur, wenn die Reservierung von Anfang an etwas größer ist, als die benötigte Bandbreite für die Stabilität.
Wie bereits erwähnt, hat IntServ den Nachteil, dass es schlecht skaliert. Für jeden Netzwerkstrom müssen die beteiligten Router Zustandsinformationen speichern und es wird eine separate Warteschlange pro Strom benötigt. Auch wenn die Eigenschaften und mögliche Anpassungen des Modells vielversprechend aussehen, hat IntServ durch diesen Nachteil kaum praktische Relevanz. Deshalb wird im Folgenden der Entwurf eines neuen Modells vorgestellt.

4.2 Entwurf eines neuen Modells nach dem Prinzip einer MLFQ

Anforderungen an ein neues Modell Bevor genauer auf das entwickelte Modell eingegangen wird, folgt zunächst eine Übersicht über die Eigenschaften, die ein neues Modell haben sollte. Diese Eigenschaften sind:

- Die Verkehrsspezifikation der Netzwerkströme soll mittels Token-Buckets vorgenommen werden. Der Token-Bucket hat für sehr große Bursts bereits eine gewisse Glättungseigenschaft, indem er Pakete markiert, welche die Spezifikation überschreiten. Dies passiert beispielsweise, wenn die Anwendung Bursts sendet, die größer als ein vollständig gefüllter Token-Bucket sind. Diese markierten Pakete sollen direkt in den Best-Effort-Verkehr fallen und werden vom vorgestellten Modell nicht weiter behandelt. Der Fokus liegt nur auf Bursts, die innerhalb der vereinbarten Token-Bucket-Parameter liegen. Wenn im Folgenden der Begriff „Burst“ verwendet wird, sind damit immer Bursts innerhalb der Token-Bucket-Spezifikation gemeint.
- Es sollen mehrere Performance-Klassen mit unterschiedlichen garantierten Zeitschranken existieren. Insbesondere soll es kein Aushungern der niedrigen Klassen geben.
- Die einzelnen Ströme sollen dynamisch den Klassen zugewiesen werden, abhängig vom Sendeverhalten der Anwendung. Dabei sollen die Ströme bei einer Verhaltensänderung auch wieder in bessere Klassen aufsteigen können. Eine Anwendung, welche wenig und mit möglichst konstanter Bitrate sendet, soll einen besseren Service erhalten, als eine Anwendung die viel sendet und eine hohe Varianz bei der Bitrate hat. Anwendungen mit möglichst kleinen und regelmäßigen Bursts erhalten also einen guten Service, während große, unregelmäßige Bursts zu einem schlechten Service führen.
  Dies soll als Anreiz für die Anwendungen dienen, ihren Netzwerkverkehr möglichst gering und ohne große Bursts zu gestalten. Dadurch wird die hohe Netzwerkbelastung durch Bursts eingedämmt. Außerdem werden so Hochlastsituationen vermieden, was sich positiv auf die Latenz aller Anwendungen auswirkt.
- Admission Control: Bei drohender Überlast sollen die Garantien von bestehenden Strömen erhalten bleiben und neue Ströme abgelehnt werden.
- Es sollen keine dedizierten Warteschlangen pro Strom bei den Routern verwendet werden. Stattdessen soll die Anzahl der Warteschlangen konstant sein, unabhängig von der Anzahl der Netzwerkströme.
  Wie bereits im Abschnitt zu IntServ erwähnt, ist die Verwaltung einer eigenen Warteschlange pro Strom aufwändig und nicht gut skalierbar. Einfache Netzwerkelemente wie TSN-Switches haben nur eine fixe und geringe Anzahl von Warteschlangen, was die Anzahl der Ströme stark limitieren würde. Deshalb soll der neue Ansatz keine dedizierte Warteschlange je Strom benötigen.

Das neue Modell basiert auf der Funktionsweise einer MLFQ. Im Folgenden wird als Erstes erläutert, wie das ursprünglich für Prozess-Scheduling entwickelte MLFQ-Verfahren für das Scheduling von Netzwerkströmen umgesetzt werden kann. Anschließend wird es schrittweise erweitert, bis daraus das entworfene Modell entsteht. Dabei wird das Verfahren zunächst um einen Mechanismus erweitert, der das Aufsteigen in höhere Prioritäten ermöglicht. Anschließend wird eine weitere Modifikation erläutert, die das Aushungern der niedrigeren Prioritäten verhindert und auch für sie eine maximale Verzögerung garantiert. Daraufhin wird ein Admission-Control-Mechanismus hinzugefügt, der

sich darum kümmert, dass Verbindungen bei drohender Überlast abgelehnt werden. Schließlich wird gezeigt, wie für das neue Modell die garantierte maximale Verzögerung berechnet werden kann.

Wiederholung des MLFQ-Verfahrens Die Multi Level Feedback Queue stammt aus dem Kontext des Prozess-Scheduling und trennt dort die interaktiven Prozesse, welche die CPU nur kurzzeitig benötigen, von den langlaufenden Batch-Prozessen. Das Ziel ist eine möglichst kurze Antwortzeit für die interaktiven Prozesse zu gewährleisten. Es handelt sich um ein präemptives Scheduling-Verfahren. Das bedeutet, dass ein aktiver Prozess unterbrochen wird, wenn er die CPU für eine maximale Zeit verwendet hat und seine Berechnungen noch nicht abgeschlossen sind. Die Dauer der CPU-Verwendung des Prozesses wird auch als CPU-Burst bezeichnet. Die maximale Zeit, für welche er die CPU verwenden darf, wird Zeitquantum genannt. Die MLFQ besteht aus mehreren FIFO-Warteschlangen mit absteigenden Prioritäten. Diese Warteschlange werden nach dem Priority-Queuing Prinzip bearbeitet. Es wird immer die Warteschlange mit der höchsten Priorität abgearbeitet, nur wenn sie leer ist, wird die Warteschlange mit der nächst niedrigeren Priorität bedient. Nur wenn die höchsten beiden Warteschlangen leer sind, wird die dritte Priorität bearbeitet usw.
Für jede dieser Warteschlangen ist ein Zeitquantum definiert, wobei die höchste Priorität das kürzeste Zeitquantum verwendet. Mit jeder darunterliegenden Priorität nimmt das Zeitquantum zu, Prozesse in den niedrigen Prioritäten können somit längere CPU-Bursts ohne Unterbrechung durchführen. Allerdings haben sie eine niedrigere Priorität und sind dadurch seltener an der Reihe. Es kann auch passieren, dass Prozesse mit niedriger Priorität verhungern, falls immer Prozesse mit höherer Priorität ausführungsbereit sind.
Ein neuer Prozess startet in der höchsten Priorität. Falls sein CPU-Burst vor Ablauf des Zeitquantums abgearbeitet ist, gibt er die CPU freiwillig ab, da er z.B. auf Benutzereingaben oder I/O warten muss. In diesem Fall verbleibt er in der aktuellen Priorität und wird wieder am Ende der entsprechenden Warteschlange eingereiht. Falls er unterbrochen werden musste, wird seine Priorität verringert und er wird in die nächst niedrigere Warteschlange eingeordnet. Das heißt, interaktive Prozesse mit kurzen CPU-Bursts behalten eine hohe Priorität. Batch-Prozesse mit langen CPU-Bursts hingegen sinken immer weiter in der Priorität ab, je länger ihre CPU-Bursts sind und je öfter sie daher unterbrochen werden. Eine Möglichkeit wieder in höhere Prioritäten aufzusteigen, gibt es bei diesem Verfahren nicht.

Umsetzung des MLFQ-Verfahrens für Netzwerkströme und Netzwerkbursts Nun wird erläutert, wie das MLFQ-Verfahren für das Scheduling von Netzwerkströmen umgesetzt werden kann. Dazu ist zunächst eine Übersicht des MLFQ-Verfahrens mit Netzwerkkomponenten in Abbildung 4.13 illustriert. Im Folgenden werden die benötigten Anpassungen erläutert.
Übertragen auf das Netzwerk entsprechen die CPU-Bursts den Netzwerkbursts, die durch den Token-Bucket gesendet werden können. Durch die Paketierung können die Netzwerkbursts nicht an beliebiger Stelle unterbrochen werden, sondern nur paketweise. Anhand der Größe der Netzwerkbursts können die Ströme klassifiziert werden und ihre Pakete in die entsprechende Warteschlange eingeordnet werden. Die erste Herausforderung stellt die Messung des Netzwerkbursts dar. Beim CPU-Scheduling wird der erste Prozess aus der höchsten Warteschlange ausgewählt und ausgeführt, bis sein CPU-Burst beendet ist, oder das Zeitquantum abgelaufen ist. Bei Netzwerkbursts könnte nun ebenfalls das erste Paket aus der höchsten Warteschlange genommen werden. Anschließend

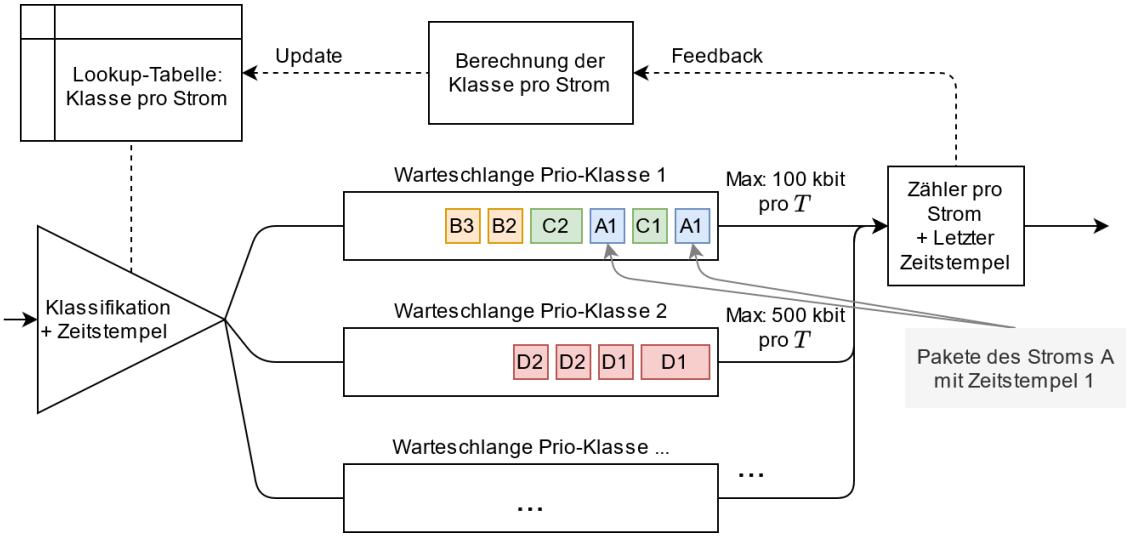

Abbildung 4.13: Eine mögliche Umsetzung des MLFQ-Verfahrens für das Scheduling von Netzwerkströmen.

werden solange weitere Pakete weitergeleitet, bis ein Paket eines anderen Stroms an der ersten Position steht oder eine maximale Anzahl von Bits weitergeleitet wurde. Dieser Ansatz funktioniert jedoch nicht zuverlässig um die tatsächliche Burstgröße zu erkennen, denn dabei wird angenommen, dass ein Burst an einem Stück in der Warteschlange liegt. Beim MLFQ-Verfahren verwenden mehrere Netzwerkströme dieselbe Warteschlange, wodurch sich die Pakete der Ströme vermischen können. Daher wird ein anderer Mechanismus benötigt, um die Burstgrößen zu messen und damit die Netzwerkströme den entsprechenden Klassen und Warteschlangen zuzuordnen. Im Folgenden werden zwei Möglichkeiten vorgestellt, um die Burstgrößen zu messen und die Ströme zu klassifizieren.

- **Klassifizierung per-Hop:** Die erste Möglichkeit besteht darin, die übertragenen Bits pro Strom innerhalb eines festgelegten Messintervalls T (in Sekunden) zu messen. Dieser Ansatz wurde in der Übersichtsgrafik (Abbildung 4.13) verwendet. Diese Messung findet bei jedem Router, also „per-Hop“ statt. Zunächst wird jedes eingehende Paket mit einem Zeitstempel versehen, der anzeigt in welchem Messintervall T das Paket angekommen ist. Weiter wird pro Strom i ein Zähler C_i (in Bits) benötigt, der mit jedem übermittelten Paket des aktuellen Messintervalls um die Paketgröße erhöht wird. Um festzustellen, ob ein Paket bereits zum nächsten Messintervall gehört, wird zusätzlich der zuletzt gesehene Zeitstempel als Zustand gehalten. Sobald ein Paket mit größerem Zeitstempel aus der Warteschlange genommen wird, werden alle Zähler ausgewertet und zurück auf Null gesetzt.
  Für die Auswertung ist für jede Priorität P ein maximaler Zählerwert C_P^{max} definiert. Wenn $C_i > C_{P_i}^{max}$ (P_i bezeichnet die aktuelle Priorität von Strom i) wurde der Maximalwert überschritten und ein Prioritätsabstieg findet statt. Denn wenn ein Strom viele Bits innerhalb der Zeitspanne gesendet hat (besonders wenn $C_i > C_{P_i}^{max}$), kann daraus geschlossen werden, dass er einen großen Burst oder mehrere kleine Burst in kurzem zeitlichem Abstand gesendet hat. Der Strom i wird in die nächste niedrigere Warteschlange der Priorität $P_i + 1$ abgestuft. Diese Abstufung wird ab diesem Zeitpunkt für neu eintreffende Pakete des Stroms durchgeführt

und sie werden in die niedrigere Warteschlange eingereiht. Die Pakete, welche sich noch in der Warteschlange der höheren Priorität befinden, werden beim Prioritätsabstieg jedoch nicht verschoben. Denn eine Verschiebung bereits eingereihter Pakete ist nicht effizient umsetzbar. Dazu müsste die gesamte Warteschlange durchlaufen werden und möglicherweise Pakete aus der Mitte entfernt werden, was für FIFO-Warteschlangen nicht effizient möglich ist. Alle Pakete, die vor dem Prioritätsabstieg beim Router angekommen sind, werden also noch mit der ursprünglichen Priorität bearbeitet.
Um die ankommenden Pakete in die korrekte Warteschlange einzuordnen, muss der Router die aktuelle Prioritätsklasse jedes Stroms speichern. Dieser Ansatz benötigt somit einen Speicherplatz für den zuletzt gesehenen Zeitstempel, einen Zähler pro Strom und einen Eintrag in einer Lookup-Tabelle für die Klasse pro Strom. Diese Werte müssen als Zustand beim Router verwaltet werden. Je nach Präzision des Zählers, kann dieser Zustand mit wenigen Bytes pro Strom gespeichert werden.

- **Klassifizierung on-the-edge:** Eine zweite Möglichkeit kommt komplett ohne Zustandsinformationen beim Router aus. Anstatt den Burst beim Router zu messen, übernimmt eine Instanz beim Zugangspunkt zum Netzwerk („on-the-edge“) diese Messung. Ein Kandidat für die Messung wäre z.B. ein Classifier direkt nach dem Token-Bucket. Dieser Wert der Burstgröße wird anschließend an jedes Paket angefügt, bevor es ins Netzwerk geschickt wird. Außerdem können zusätzliche Informationen über die Sende-Vergangenheit hinzugefügt werden, beispielsweise die Größe des größten Bursts innerhalb eines Zeitfensters oder ein gleitender Mittelwert. Mit diesen Werten können die Router den Netzwerkstrom der korrekten Klasse zuordnen, ohne dafür Zustand speichern zu müssen. Der Zustand befindet sich stattdessen in den Paketen, was eine Erhöhung des Netzwerkverkehrs bedeutet.
  Dieser Ansatz hat starke Ähnlichkeit zum DiffServ-Modell, bei welchem die Ströme zu Beginn klassifiziert werden. Die angehangene Zustandsinformation entspricht dieser Klassifizierung, wenn jeder Router die selben Schwellwerte und Algorithmen verwendet, um die Priorität des Stroms daraus zu bestimmen. In diesem Fall kann der Klassifizierer nach dem Token-Bucket auch direkt die Berechnung übernehmen und die Prioritätsklasse bestimmen. Die Router können jedoch auch individuelle Schwellwerte festlegen, anhand ihrer verfügbaren Bandbreite. Ein weiterer Nachteil dieses Ansatzes ist die schwierige Überlastvermeidung, was an späterer Stelle erläutert wird.

Aufgrund der Nachteile der Klassifizierung on-the-edge wird im Folgenden die Klassifizierung per-hop verwendet.

Erweiterung für Prioritätsaufstieg Beim ursprünglichen MLFQ-Verfahren für Prozess-Scheduling gibt es keine Möglichkeit, aus einer niedrigen Priorität wieder in eine höhere aufzusteigen. Bei Prozessen ist dies meist nicht nötig, da diese ihr Verhalten typischerweise nicht ändern. Anwendungen, welche sowohl interaktive als auch rechenintensive Aufgaben implementieren, teilen diese Aufgaben typischerweise in mehrere Threads (Subprozesse) auf. Eine gängige Aufteilung ist beispielsweise ein User-Interface-Thread für Interaktionen mit dem Benutzer und separate Worker-Threads für rechenintensive Aufgaben. User-Interface-Threads warten die meiste Zeit auf Benutzereingaben (I/O von Eingabegeräten) und benötigen nur kurze CPU-Bursts um auf die Eingaben zu reagieren. Worker-Threads hingegen benötigen lange CPU-Bursts um z.B. aufwändige Berechnungen durchzuführen. Sowohl User-Interface-Threads als auch Worker-Threads ändern

ihr CPU-Burst-Verhalten typischerweise nicht.
In modernen Betriebssystemen erfolgt das Scheduling auf Thread-Basis (zumindest für sogenannte Kernel-Level-Threads), weshalb der Begriff Prozess und Thread hier synonym verwendet werden kann. Es genügt für das Prozess-Scheduling deshalb, die Prozesse bzw. Threads einmalig zu klassifizieren und der entsprechenden Warteschlange zuzuordnen.
Für das Scheduling von Netzwerkströmen gilt diese Eigenschaft im allgemeinen nicht. Ein Netzwerkstrom, der eine Zeitlang große Bursts gesendet hat, kann durchaus sein Verhalten ändern und anschließend wieder kleine Bursts senden. Ein Sensor kann beispielsweise einmal täglich eine Zusammenfassung mit Wartungsinformationen senden, welche in Form eines großen Bursts gesendet wird. Für den Rest des Tages sendet er dagegen nur kleine Bursts mit aktuellen Sensorwerten. Den Sensor aufgrund des seltenen großen Bursts dauerhaft in eine niedrige Prioritätsklasse einzuordnen, wäre nicht optimal. Besser ist es, ihn nach einer gewissen Zeit ohne große Bursts wieder in eine der höheren Klassen hochzustufen.
Mit der gezeigten Klassifizierung per-Hop kann die Funktionalität des Prioritätsaufstiegs leicht implementiert werden. Es muss dazu nur die Berechnung der Prioritätsklasse (P_i) pro Strom i angepasst werden. Anstatt wie bisher nur den Zählerwert C_i des Stroms mit dem Maximalwert der aktuellen Klasse $C_{P_i}^{max}$ zu vergleichen, muss lediglich ein Vergleich mit dem Maximalwert der nächst höheren Priorität ($C_{P_i-1}^{max}$) hinzugefügt werden. Es gibt also zwei Vergleiche, einen für den Prioritätsabstieg und einen für den Prioritätsaufstieg:

$$\begin{aligned} \text{Prioritätsabstieg: } & C_i > C_{P_i}^{max} \\ \text{Prioritätsaufstieg: } & C_i < C_{P_i-1}^{max} \end{aligned} \tag{4.28}$$

Um ein häufiges Alternieren zwischen zwei Prioritäten zu vermeiden, kann zusätzlich eine Hysterese verwendet werden. Dazu werden kleinere Werte für den Aufstieg verwendet, als für den Abstieg. Dies wird hier jedoch nicht benötigt, da ein häufiger Wechsel bereits durch eine Aufstiegsverzögerung verhindert wird, wie im Folgenden erläutert wird.
Der Prioritätsabstieg soll nach jedem Messintervall überprüft und gegebenenfalls durchgeführt werden, damit Ströme mit großen Bursts schnellstmöglich abgestuft werden. Der Prioritätsaufstieg soll dagegen nicht nach jedem Messintervall möglich sein, damit abgestufte Ströme nicht sofort wieder aufsteigen können. Typischerweise folgt nach großen Bursts eines Stroms eine Sendepause, da der Token Bucket erst wieder aufgefüllt werden muss. Wäre der Aufstieg nach jedem Messintervall möglich, würden Ströme, die ausschließlich große Bursts senden, während ihrer Sendepausen direkt wieder aufsteigen. Das gewollte Verhalten ist aber, Ströme mit ausschließlich großen Bursts in den niedrigen Prioritäten zu halten.
Darum wird ein Mechanismus benötigt, der den Aufstieg erst nach einer gewissen Zeit T^{auf} ermöglicht. Diese Aufstiegsverzögerung ist für jeden Strom gleich lang und wird in Abhängigkeit des Messintervalls T und einer Konstanten k definiert:

$$T^{auf} = k \cdot T \tag{4.29}$$

Ein Strom kann also frühestens nach k Messintervallen wieder aufsteigen. Sobald ein Strom abgestuft wurde, soll die Aufstiegsverzögerung für ihn beginnen. Die verbleibende Wartezeit t_i^{auf} bis zum nächst möglichen Aufstieg ist deshalb individuell für jeden Strom i und abhängig vom Zeitpunkt seiner letzten Abstufung. Falls der Strom zwischenzeitlich jedoch erneut in die gleiche Priorität eingeordnet wird, so wird die Aufstiegsverzögerung zurückgesetzt. Dadurch wird sichergestellt, dass ein Strom erst nach k Messintervallen ohne Überschreitung des Schwellwerts $C_{P_i-1}^{max}$ wieder aufsteigen kann. Algorithmus 4.1 enthält den Prioritätsauf- und abstieg mit individueller Aufstiegsverzögerung

Algorithmus 4.1 Prioritätsabstieg und Prioritätsaufstieg - individueller Beginn von t_i^{auf}

```
procedure BerechneKlasse({C^max}, {C}, {P}, {t^auf}, k)
    for all i do
        if C_i > C^max_{P_i} then    // Prioritätsabstieg
            P_i ← P_i + 1
            t_i^auf ← k
        else if t_i^auf = 0 ∧ P_i > 0 ∧ C_i < C^max_{P_i−1} then    // Prioritätsaufstieg
            P_i ← P_i − 1
        else if C_i < C^max_{P_i−1} then    // Aufstiegsverzögerung
            t_i^auf ← t_i^auf − 1
        else
            t_i^auf ← k    // Zurücksetzen der Aufstiegsverzögerung
        end if
    end for
end procedure
```

t_i^{auf}.
Ein Nachteil dieses Algorithmus ist der zusätzliche Zustand, welcher für den individuellen Zähler t_i^{auf} der Aufstiegsverzögerung benötigt wird. Je nach Größe von k beträgt dieser Zustand allerdings nur 1–2 Bytes pro Strom, was technisch problemlos möglich ist.
Die Funktionalität des Algorithmus kann auch mit einer Variante ohne zusätzlichen Zustand pro Strom angenähert werden. Dabei wird ein globaler Zähler der Größe $2k+1$ verwendet, welcher mit jedem Messzyklus verringert wird. Der Aufstieg ist für alle Ströme nur im Zyklus möglich, in welchem der Zähler den Wert Null erreicht. Nach diesem Aufstiegszyklus wird der Zähler wieder auf $2k+1$ gesetzt. Dadurch ist ein Aufstieg nur alle $2k+1$ Messzyklen möglich. Die Aufstiegsverzögerung eines Stroms kann dadurch zwischen $2k+1$ (falls er im Aufstiegszyklus abgestuft wurde) und 1 liegen. Im Mittel beträgt sie somit $\frac{2k+1-1}{2} = k$. Diese Variante nähert also das Verhalten des Algorithmus 4.1 an, bei welchem die Aufstiegsverzögerung genau k beträgt, solange kein Zurücksetzen ausgelöst wird. Diese Variante hat den Vorteil, dass sie keinen zusätzlichen Zustand pro Strom benötigt. Ihre Effektivität und mögliche Nachteile werden im späteren Verlauf der Arbeit evaluiert.

Verhindern von Aushungern Wie bereits erwähnt, können beim MLFQ-Verfahren die niedrigen Prioritäten aushungern, wenn ständig Prozesse bzw. Pakete in der höchsten Priorität vorhanden sind. Um dies für das entworfene MLFQ-Verfahren für Netzwerkströme zu verhindern, werden im Folgenden sogenannte Platzhalter-Pakete (im Englischen: Dummy-Pakete) verwendet. Diese Platzhalter-Pakete sind ein logisches Konzept, um die Funktionsweise zu verdeutlichen. Die tatsächliche Implementierung weicht davon ab, liefert jedoch die gleiche Funktionalität. Genaueres kann im Kapitel zur Implementierung (5) gefunden werden.
Ein Platzhalter-Paket ist ein besonderes Paket, das vom Router selbst in die Warteschlangen gelegt werden kann. Solch ein Platzhalter-Paket wird zu Beginn in alle Warteschlangen gelegt, außer in die Warteschlange niedrigster Priorität (W_n). Wenn dieses Platzhalter-Paket in der höchsten Warteschlange (W_0) an der Reihe ist und weitergeleitet werden soll entfernt der Router es aus der Warteschlange und leitet stattdessen Pakete aus der nächst niedrigeren Warteschlange (W_1)

weiter. Anschließend fügt er wieder ein Platzhalter-Paket ans Ende der Warteschlange W_0 hinzu. Das Platzhalter-Paket hat die Größe S_{D_0} in Bits, diese Größe kann entsprechend der gewünschten Garantien angepasst werden. Die nächst niedrigere Warteschlange kann dadurch S_{D_0} Bits weiterleiten, was auch mehreren Paketen entsprechen kann.
Dieses Verfahren setzt sich fort bis zur vorletzten Warteschlange: In W_1 befindet sich ein Platzhalter-Paket der Größe S_{D_1}, bei dessen Abarbeitung die Warteschlange W_2 Pakete weiterleiten darf usw. und schließlich ein Platzhalter-Paket in W_{n-1} zur Weiterleitung von Paketen in W_n. In der niedrigsten Warteschlange wird kein Platzhalter-Paket mehr benötigt, da es keine darunterliegende Warteschlange gibt. Abbildung 4.14 illustriert das Verfahren für drei Warteschlangen.

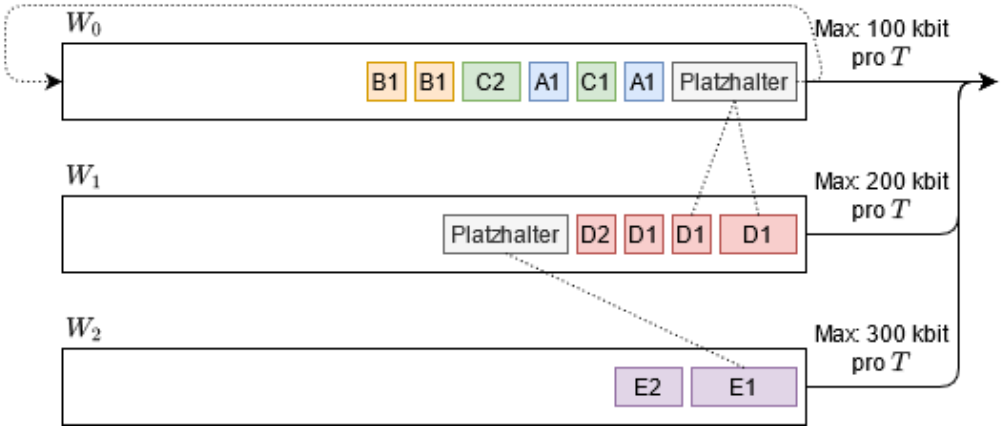

Abbildung 4.14: Eine MLFQ für Netzwerkscheduling mit drei Performance-Klassen. In W_0 und W_1 werden Platzhalter-Pakete eingereiht, um das Aushungern der jeweiligen niedrigeren Priorität zu verhindern.

Für die Größe des Platzhalter-Pakets (S_D) muss in jeder Warteschlange gelten, dass sie mindestens so groß gewählt wird wie die maximale Paketgröße der eingehenden Pakete. Ansonsten könnte es passieren, dass ein größeres Paket als S_D in die darunterliegende Warteschlange eingereiht wird und nie weitergeleitet werden kann. Denn überschüssige Bits der Platzhalter-Pakete können nicht angespart werden. Es dürfen pro Platzhalter-Paket nur maximal S_D Bits aus der darunterliegenden Warteschlange weitergeleitet werden, auch wenn beim vorherigen Platzhalter-Paket nicht die volle Kapazität verwendet wurde. Diese Design-Entscheidung wurde getroffen, um die Berechnung der Garantien zu vereinfachen.
Es können auch mehrere Platzhalter-Pakete pro Warteschlange eingereiht werden oder ein größerer Wert für S_D gewählt werden, um häufiger Pakete der niedrigeren Prioritäten weiterzuleiten. Damit wird jedoch mehr und mehr die Priorisierung ausgehebelt. Vollständig aufgehoben ist die Priorisierung bei zwei Warteschlangen W_0 und W_1, wenn die Hälfte von W_0 mit Platzhalter-Paketen gefüllt ist. Das System verhält sich dann im Mittel wie ein Round-Robin-Scheduler, da die Hälfte der Zeit Pakete aus W_1 durch die Platzhalter-Pakete weitergeleitet werden. Im Folgenden wird nur von einem Platzhalter-Paket pro Warteschlange ausgegangen, die Berechnung werden trotzdem möglichst allgemein gehalten. N_{D_0} bezeichnet die Anzahl der Platzhalter-Pakete in Warteschlange W_0.
Das Verhalten der MLFQ ändert sich durch die Platzhalter-Pakete nicht, wenn die Warteschlangen der hohen Prioritäten leer sind. Falls keine Netzwerkströme in der höchsten Priorität Pakete senden, so

befindet sich nur das Platzhalter-Paket in W_0. Dieses wird nun ständig entfernt, entsprechend Pakete der darunter liegenden Warteschlange W_1 weitergeleitet und anschließend wieder ein Platzhalter-Paket in W_0 gelegt. Falls auch W_1 nur das Platzhalter-Paket enthält wird entsprechend die nächste Warteschlange W_2 bedient, bis eine nichtleere Warteschlange gefunden wurde oder die niedrigste Priorität erreicht ist. Dies entspricht dem Verhalten einer nicht-modifizierten MLFQ.
Falls die Warteschlange höchster Priorität ständig gefüllt ist, ändert sich das Verhalten im Vergleich zum ursprünglichen MLFQ-Verfahren. Auch bei einer komplett gefüllten Warteschlange soll mindestens einmal pro Messintervall T das Platzhalter-Paket aus W_0 entfernt und Pakete von W_1 weitergeleitet werden. Ohne diese Erweiterung hatte W_0 die komplette Ausgangsbandbreite R^{eg} zur Verfügung, ihre minimale Bandbreite mr_0 entsprach somit R^{eg}. Nun verringert sie sich entsprechend um den Anteil der Platzhalter-Pakete:

$$mr_0 = R^{eg} \cdot \frac{|W_0| - (N_{D_0} \cdot S_{D_0})}{|W_0|} = R^{eg} \cdot h_0 \tag{4.30}$$

Der Bruch kann auch als Konstante h_0 mit $0 \leq h_0 \leq 1$ interpretiert werden, welche den prozentualen Anteil der Gesamtbandbreite angibt, die für W_0 zur Verfügung steht. W_1 erhält den abgezogenen Anteil als minimale Bandbreite mr_1. Dieser prozentuale Anteil kann ebenfalls mithilfe einer Konstanten angeben werden, sie wird als $l_0 = 1 - h_0$ bezeichnet. Falls noch eine darunterliegende Warteschlange W_2 existiert, muss für die Platzhalter-Pakete entsprechend wieder Bandbreite abgegeben werden:

$$mr_1 = R^{eg} \cdot \frac{(N_{D_0} \cdot S_{D_0})}{|W_0|} \cdot \frac{|W_1| - (N_{D_1} \cdot S_{D_1})}{|W_1|} = R^{eg} \cdot l_0 \cdot h_1 \tag{4.31}$$

Da es sich bei den Platzhalter-Paketen nur um ein logisches Konzept handelt, werden im Folgenden die übersichtlicheren Konstanten verwendet. Allgemein gilt damit für die minimale Bandbreite mr_i einer Warteschlange W_i:

$$mr_i = \begin{cases} R^{eg} \cdot h_0, & \text{wenn } i = 0 \\ R^{eg} \cdot \prod_{j=0}^{n-1} l_j, & \text{wenn } i = n \\ R^{eg} \cdot \prod_{j=0}^{i-1} (l_j) \cdot h_i, & \text{sonst} \end{cases} \tag{4.32}$$

Erweiterung um einen Admission-Control-Mechanismus Die Zugriffskontrolle (auf Englisch: Admission-Control) kümmert sich darum, dass neue Netzwerkströme zugelassen oder abgelehnt werden. Dabei wird geprüft, ob die verfügbaren Netzwerkressourcen ausreichend sind, um den neuen Strom aufzunehmen, ohne die Zusicherungen an bereits zugelassene Ströme zu verletzen. Nach dem Zulassen müssen die Ressourcen für den neuen Strom dauerhaft zu Verfügung stehen bzw. bis der Strom von Seiten der Anwendung beendet wurde. Ein nachträgliches Ablehnen eines Stroms ist nicht vorgesehen. Die Admission-Control darf also nicht mehr Ströme zulassen, als die verfügbaren Ressourcen leisten können.
Für das neu entworfene Modell kann die Zugriffskontrolle wie folgt durchgeführt werden:

1. Die Anwendung, welche einen neuen Netzwerkstrom beginnen möchte, sendet ein besonderes Admission-Control-Paket. Dieses Paket beinhaltet eine eindeutige Identifikation des Netzwerkstroms, beispielsweise die IP Adresse + Stromnummer, falls die Anwendung bereits andere Netzwerkströme verwendet. Dieses Paket wird nun an das gewünschte Ziel adressiert und verschickt.

2. Das Paket wird zum gewünschten Ziel geroutet. Jeder Router entlang des Pfads entscheidet, ob er genügend Ressourcen zur Verfügung hat, um einen neuen Strom aufzunehmen. Der Algorithmus für diese Entscheidung wird im Anschluss erläutert. Hat der Router genügend Ressourcen, notiert er sich den möglichen neuen Strom auf einer „Warteliste“ und sendet das Admission-Control-Paket weiter. Hat der Router nicht genügend Ressourcen, so stoppt er das Paket und schickt ein „Ablehnungspaket“ zurück an die Anwendung. In dieses Paket kopiert er die eindeutige Identifikation des Netzwerkstroms.

3. Empfängt ein Router ein Ablehnungspaket, so identifiziert und entfernt er den betroffenen Strom von seiner Warteliste und leitet das Ablehnungspaket weiter. Schließlich erreicht es die Anwendung und sie erhält die Information, dass nicht genügend Ressourcen zur Verfügung stehen.

4. Falls kein Router den neuen Strom abgelehnt hat, erreicht das Admission-Control-Paket schließlich das gewünschte Ziel. Dieses generiert ein „Zustimmungspaket“, in welches wieder die Identifikation des Stroms kopiert wird, und schickt es zurück an die Anwendung.

5. Jeder Router entlang des Pfads reagiert auf das Zustimmungspaket, indem er den neuen Strom von seiner Warteliste in die Lookup-Tabelle überträgt. Der neue Strom erhält die höchste Prioritätsklasse. Jeder Router fügt außerdem eine Übersicht über seine Prioritäts-Maximalwerte C_P^{max} und die jeweilige garantierte maximale Verzögerung pro Priorität dem Zustimmungspaket hinzu. Genaueres zur Berechnung der Verzögerung wird im späteren Abschnitt „Berechnung der maximalen Ende-zu-Ende Verzögerung“ erläutert. Anschließend wird das Paket weitergeleitet. Schließlich erreicht das Zustimmungspaket die Anwendung und sie erhält die Information, dass ihr Strom zugelassen wurde. Außerdem erhält sie eine Übersicht über die Prioritäts-Maximalwerte C_P^{max} aller Router entlang des Pfads.

Offen ist noch der Algorithmus für die Entscheidung eines Routers, ob genügend Ressourcen für einen neuen Strom verfügbar sind. Dazu muss zunächst festgehalten werden, um welche Ressourcen es sich handelt und wann diese vollständig ausgelastet sind.
Die Admission-Control bei einem nicht-Echtzeit Betriebssystem überprüft beim Zulassen neuer Prozesse, ob diese noch effizient ausgeführt werden können. Die limitierende Ressource ist dabei der Hauptspeicher, da man „Seitenflattern“ (auf Englisch: Thrashing) vermeiden will. Beim Thrashing werden exzessiv Speicherseiten durch die virtuelle Speicherverwaltung gewechselt, da nicht genügend Hauptspeicher vorhanden ist und zu viele Seiten auf die Festplatte ausgelagert wurden. Genaueres dazu kann z.B. in der Arbeit von Denning [Den68] gefunden werden. Solange es nicht zu Thrashing kommt, können neue Prozesse zugelassen werden.
Der Hauptspeicher als limitierende Ressource kann nicht direkt auf das Netzwerk übertragen werden, da das Threshing-Problem dort nicht gegeben ist. Aus Sicht jedes einzelnen Routers ist die zu vermeidende Situation eine komplett gefüllte Warteschlange, denn dann können ankommende Pakete nicht mehr gepuffert werden und müssen verworfen werden. Im Folgenden wird die Warteschlange höchster Priorität des Routers betrachtet. Um ein Überlaufen dieser Warteschlange zu vermeiden genügt es nicht, alleine den aktuellen Füllstand der Warteschlange zu betrachten. Denn darin sammeln sich nur Pakete der Ströme, welche momentan senden. Netzwerkströme, die gerade eine Sendepause machen, haben folglich keine Pakete in der Warteschlange. Würden nun neue Ströme zugelassen werden, solange die Warteschlange nicht vollständig gefüllt ist, riskiert man eine Überlastung. Denn sobald pausierende Ströme wieder zu senden beginnen, wäre nicht mehr genügend Platz in der Warteschlange. Jeder Router muss bei seiner Entscheidung deshalb alle Ströme berücksichtigen, welche theoretisch senden könnten.

Im „schlechtesten“ Fall (aus Sicht des Warteschlangenfüllstands) befinden sich alle Ströme in der höchsten Priorität und senden gleichzeitig einen Burst der maximal zulässigen Größe (C_0^{max}) pro Messintervall. Daraus lässt sich die maximale Anzahl der Bits berechnen, die pro Messintervall in der höchsten Priorität anfallen, indem die Anzahl der Ströme (N) mit C_0^{max} multipliziert wird. Im theoretischen schlechtesten Fall, würde diese Menge an Bits an einem Stück beim Router ankommen und muss komplett in der Warteschlange Platz finden. Dies ist in der Praxis nicht unbedingt realistisch, da der Router nur eine begrenzte Eingangsbandbreite besitzt, es kann damit aber eine sichere obere Schranke angegeben werden.
Für die Größe der Warteschlange höchster Priorität ($|W_0|$) muss daher gelten:

$$|W_0| \geq (N+1) \cdot C_0^{max} \tag{4.33}$$

Falls die Platzhalter-Pakete tatsächlich als Pakete implementiert werden, muss der Platzbedarf dieser Pakete in der Warteschlange zusätzlich berücksichtigt werden. Auf die gezeigte Formel (4.33) muss dann der Term $N_{D_0} \cdot S_{D_0}$ dazu addiert werden.
Ein weiterer limitierender Faktor ist die Ausgangsbandbreite des Routers. Falls er pro Messintervall mehr Pakete bzw. Bits empfängt, als er weiterleiten kann, so füllt sich die Warteschlange und wird irgendwann voll sein. Es muss daher für einen zusätzlichen Strom gewährleistet sein, dass die minimale Bandbreite mr_0 groß genug ist, um alle theoretisch anfallenden Bursts abarbeiten zu können:

$$mr_0 \cdot T \geq (N+1) \cdot C_0^{max} \tag{4.34}$$

Da alle benötigten Werte bereits zur Zeit des Entwurfs bekannt sind, kann Rechenaufwand eingespart werden und im Voraus die maximal unterstützte Anzahl der Ströme berechnet werden. Sie wird als M bezeichnet und kann wie folgt berechnet werden:

$$M = \lfloor \min(\frac{|W_0|}{C_0^{max}}, \frac{mr_0 \cdot T}{C_0^{max}}) \rfloor \tag{4.35}$$

Jeder Router kann diesen Wert für M einmalig berechnen und braucht beim Zulassen eines neuen Stroms lediglich zu prüfen, ob $N+1 \leq M$. Dadurch wird ein Überlaufen der Warteschlange höchster Priorität verhindert.
Für die niedrigeren Prioritäten kann es jedoch noch immer zum Überlaufen der Warteschlangen und damit zu Paketverlust kommen. Denn je niedriger die Priorität der Warteschlange ist, desto geringer wird die minimale Bandbreite und gleichzeitig steigt die maximale Burstgröße C^{max}. Es müssen somit weitere Bedingungen für M aufgestellt werden, die zusätzlich erfüllt sein müssen, um auch das Überlaufen der niedrigeren Prioritäten zu verhindern. Diese Bedingungen schränken allerdings die Anzahl der Ströme weiter ein, die zugelassen werden können. Der Paketverlust von Strömen mit niedriger Priorität kann durch eine konservative Planung und Reservierung von Bandbreite verhindert werden, allerdings auf Kosten der Anzahl der zugelassenen Ströme. An dieser Stelle muss abgewogen werden, ob ein Überlaufen der Warteschlangen niedriger Priorität und der damit verbundene Paketverlust in Kauf genommen werden soll, um mehr Ströme aufnehmen zu können. Dadurch würden die niedrigeren Prioritäten keine feste Garantie mehr bekommen, dass ihre Pakete zugestellt werden. Sie würden nur noch eine Zusicherung bekommen, dass ein Paket entweder nach Ablauf einer Zeit t garantiert am Ziel angekommen ist, oder aufgrund von Überlast verworfen wurde. Zu verspäteten Paketen würde es nicht kommen. Dies könnte als weiterer Anreiz gesehen werden, sich möglichst gut aus Sicht des Netzwerks zu verhalten, um in den hohen Prioritäten zu bleiben. Im Folgenden werden die Bedingungen vorgestellt, mit welchen das Überlaufen der Warteschlange W_1 verhindert werden kann. Es wird anschließend anhand eines Beispiels gezeigt, wie sich die

Anzahl der maximal möglichen Ströme M dadurch reduzieren würde.
Um die Bedingungen zur Verhinderung des Überlaufens von W_1 aufzustellen, muss zunächst der Worst-Case ermittelt werden. Dieser ist nicht direkt offensichtlich. Die minimale Bandbreite mr_1 erhält W_1, wenn die Warteschlange W_0 komplett gefüllt ist. Das würde jedoch bedeuten, dass sich M Ströme in W_0 befinden, wodurch kein Strom mehr in W_1 sein kann, da pro Router nicht mehr als M Ströme zugelassen werden können. Es können sich somit zwischen 0 und $M-1$ Ströme in W_0 befinden und die verbleibenden Ströme in den niedrigeren Warteschlangen. Für den Worst-Case aus Sicht von W_1 befinden sich nur Ströme in W_0 und W_1, denn ein Strom in W_1 konkurriert ansonsten nur mit $M-n$ Strömen in W_0 und W_1, falls sich n Ströme in den Warteschlangen der Priorität > 1 befinden.
W_1 erhält abhängig von der Aufteilung der Ströme mehr Bandbreite als mr_1. Würden sich alle Ströme in W_1 befinden, so würde W_1 die komplette Ausgangsbandbreite R^{eg} (abzüglich der Bandbreite für niedrigere Prioritäten) erhalten. Denn dann würde in W_0 nur das Platzhalter-Paket liegen und ständig an der Reihe sein. Mit folgender Formel kann die tatsächliche Bandbreite tr_1 berechnet werden, die W_1 abhängig von der Anzahl der Ströme M_0 in W_0 erhält:

$$tr_1 = R^{eg} \cdot \frac{l_0}{\frac{M_0}{M} \cdot h_0 + l_0} \cdot h_1 \tag{4.36}$$

Falls noch eine darunterliegende Warteschlange W_2 existiert, muss dafür Bandbreite abgegeben werden. Dies wird durch die Multiplikation mit h_1 berücksichtigt.
Mit der tatsächlichen Bandbreite tr_1 können nun die Bedingungen aufgestellt werden, die das Überlaufen von W_1 verhindern. Wie zuvor für die höchste Priorität, muss wieder die Warteschlange groß genug sein, um alle theoretisch anfallenden Bursts puffern zu können. Dabei befinden sich im Worst-Case wie bisher alle Ströme in W_1:

$$|W_1| \geq M \cdot C_1^{max} \tag{4.37}$$

Weiter muss die tatsächliche Bandbreite groß genug sein, um M_1 abarbeiten zu können. Mit M_1 wird die Anzahl Ströme in W_1 bezeichnet:

$$\forall M_1 \in \{0, ..., M\} : tr_1 \cdot T \geq M_1 \cdot C_1^{max} \tag{4.38}$$

Um die Bedingung ohne Quantor angeben zu können, kann auch das Minimum verwendet werden:

$$\min_{M_1}\left(\frac{tr_1 \cdot T}{M_1 \cdot C_1^{max}}\right) \geq 1 \tag{4.39}$$

Dabei kann folgende Umformung vorgenommen werden:

$$\begin{aligned}
\frac{tr_1 \cdot T}{M_1 \cdot C_1^{max}} &= R^{eg} \cdot \frac{l_0}{\frac{M-M_1}{M} \cdot h_0 + l_0} \cdot h_1 \cdot T \cdot \frac{1}{M_1 \cdot C_1^{max}} \\
&= \frac{R^{eg} \cdot l_0 \cdot h_1 \cdot T}{C_1^{max}} \cdot \frac{1}{\frac{M-M_1}{M} \cdot h_0 \cdot M_1 + l_0 \cdot M_1} \\
&= \frac{mr_1 \cdot T}{C_1^{max}} \cdot \frac{1}{M_1 \cdot h_0 - \frac{M_1^2}{M} \cdot h_0 + l_0 \cdot M_1} \\
&= \frac{mr_1 \cdot T}{C_1^{max}} \cdot \frac{1}{M_1 - M_1^2 \cdot \frac{h_0}{M}}
\end{aligned} \tag{4.40}$$

Für die quadratische Gleichung unter dem Bruchstrich können die beiden Nullstellen bestimmt werden:

$$\begin{aligned} 0 &= M_1 - M_1^2 \cdot \frac{h_0}{M} = M_1 \cdot (1 - M_1 \cdot \frac{h_0}{M}) \\ &\Rightarrow \text{Nullstelle 1: } 0 \\ &\Rightarrow \text{Nullstelle 2: } \frac{M}{h_0} \end{aligned} \tag{4.41}$$

Da sich die Extremstelle einer Parabel genau in der Mitte zwischen den beiden Nullstellen befindet, kann sie durch den Mittelwert der beiden Nullstellen bestimmt werden:

$$M_1^{ex} = \frac{M}{2 \cdot h_0} \tag{4.42}$$

Anstelle des Minimums in der Bedingung (4.39) kann somit direkt die Extremstelle geprüft werden:

$$\begin{aligned} \min_{M_1}(\frac{tr_1 \cdot T}{M_1 \cdot C_1^{max}}) &= \frac{mr_1 \cdot T}{C_1^{max}} \cdot \frac{1}{\frac{M}{2 \cdot h_0} - \frac{M^2}{4 \cdot h_0^2} \cdot \frac{h_0}{M}} \\ &= \frac{mr_1 \cdot T}{C_1^{max}} \cdot \frac{1}{\frac{M}{2 \cdot h_0} - \frac{M}{4 \cdot h_0}} \\ &= \frac{mr_1 \cdot T}{C_1^{max}} \cdot \frac{1}{\frac{M}{4 \cdot h_0}} \\ &= \frac{4 \cdot h_0 \cdot mr_1 \cdot T}{C_1^{max} \cdot M} \overset{(4.39)}{\geq} 1 \\ &\Rightarrow \frac{4 \cdot h_0 \cdot mr_1 \cdot T}{C_1^{max}} \geq M \end{aligned} \tag{4.43}$$

Damit gibt es nun vier Bedingungen, die für M erfüllt sein müssen, um ein Überlaufen der höchsten Priorität und der zweithöchsten Priorität zu verhindern. Dies sind die bereits bekannten beiden Bedingungen der höchsten Priorität welche sicherstellen, dass W_0 groß genug ist, um alle Bursts aufzunehmen (4.33) und dass die Ausgangsrate ausreichend ist, um alle Bursts abzuarbeiten (4.34). Für die zweithöchste Priorität muss ebenfalls die Warteschlange W_1 genügend Platz für alle theoretisch anfallenden Bursts haben (4.37). Und schließlich muss die Bandbreite in dieser Priorität groß genug sein, um alle Bursts abarbeiten zu können (4.43). Die Anzahl der Ströme M, die unter diesen Bedingungen zugelassen werden kann, lässt sich somit wie folgt bestimmen:

$$M = \lfloor \min(\frac{|W_0|}{C_0^{max}}, \frac{mr_0 \cdot T}{C_0^{max}}, \frac{|W_1|}{C_1^{max}}, \frac{4 \cdot h_0 \cdot mr_1 \cdot T}{C_1^{max}}) \rfloor \tag{4.44}$$

Um zu verdeutlichen, welchen Einfluss die Bedingungen auf M haben, werden in der folgenden Tabelle 4.1 verschiedene Beispiele mit konkreten Werten gezeigt. Dabei wird angenommen, dass die Warteschlangengröße ($|W_0|$ und $|W_1|$) jeweils 1 Gbit = 125 MB beträgt. In der Praxis ist die maximale Warteschlangengröße vom verfügbaren Pufferspeicher des verwendeten Geräts abhängig. Dabei gibt es Unterschiede zwischen Data Center-Switches und Routern. Die Pufferspeicher von Data-Center-Switches sind typischerweise relativ klein und betragen je nach Modell 10 – 100 MB (80 – 800 Mbit) [BDHL16; Edg20]. Im Vergleich dazu ist der Pufferspeicher von Routern deutlich

größer. Dieser kann beispielsweise 3 – 16 GB (24 – 128 Gbit) betragen [Jun20]. Die Annahme, dass die Warteschlangengröße 1 Gbit entspricht ist somit ein Mittelwert, der von Routern ohne Probleme erfüllt werden kann und nahe am Möglichen von Data Center-Switches liegt.

Tabelle 4.1: Die Berechnung der maximalen Anzahl der Ströme M, verdeutlicht an konkreten Beispielen. Es sei $|W_0| = |W_1| = 1$ Gbit, $C_0^{max} = 100$ kbit, $T = 1$s und es gebe nur die beiden Prioritäten 0 und 1 (also $h_1 = 1$).

R^{eg}	h_0	C_1^{max}	$\frac{\|W_0\|}{C_0^{max}}$	$\frac{mr_0 \cdot T}{C_0^{max}}$	$\frac{\|W_1\|}{C_1^{max}}$	$\frac{4 \cdot h_0 \cdot mr_1 \cdot T}{C_1^{max}}$	Endgültiges M
10 Mbit/s	0,75	1000 kbit	10000	75	1000	7,5	7
1 Gbit/s	0,75	1000 kbit	10000	7500	1000	750	750
1 Gbit/s	0,75	500 kbit	10000	7500	2000	1500	1500
1 Gbit/s	0,75	200 kbit	10000	7500	5000	3750	3750
1 Gbit/s	0,9	1000 kbit	10000	9000	1000	360	360
1 Gbit/s	0,55	1000 kbit	10000	5500	1000	990	990

Beim Erhalten eines Admission-Control-Pakets überprüft jeder Router also implizit die vorgestellten Bedingungen 4.44, indem er vergleicht, ob $N + 1 \leq M$. Wird M nicht überschritten sendet er das Admission-Control-Paket weiter, da er genügend Ressourcen für den neuen Strom besitzt. Falls eine der Bedingungen nicht erfüllt ist, verfügt der Router nicht über genügend Ressourcen um einen neuen Strom aufzunehmen und sendet stattdessen ein Ablehnungspaket zurück. Diese Bedingungen verhindern das Überlaufen der Warteschlange höchster und zweithöchster Priorität.

Berechnung der maximalen Ende-zu-Ende Verzögerung Abhängig von den Prioritäten, in welchen ein Strom bei den Routern entlang seines Pfads eingeordnet ist, kann die maximale Ende-zu-Ende Verzögerung vom Sender zum Empfänger berechnet werden. Diese setzt sich, wie bereits bei der Berechnung der Verzögerung von IntServ, aus drei Komponenten zusammen: Dem Queuing-Delay pro Router $d_{R_i}^{max}$, der Ausbreitungsverzögerung d^A und der Verarbeitungszeit d^V. Für d^A und d^V können wie zuvor konstante Werte angenommen werden, es gelte wieder d^A = 1ms und d^V = 1ms.
Das Queuing-Delay unterscheidet sich in der Berechnung im Vergleich zu IntServ. Bei IntServ wird pro Strom eine eigene Warteschlange verwendet, weshalb die Pakete häufig auf leere Warteschlangen treffen, siehe hierzu die Analyse in Kapitel 4.1. Dabei genügt es, bei allen Routern, mit Ausnahme des Flaschenhalses, nur die Verzögerung des ersten bzw. letzten Pakets eines Bursts zu betrachten (Gleichung 4.21). Da sich beim neu entworfenen Modell mehrere Ströme eine Warteschlange teilen, kann nicht mehr von leeren Warteschlangen ausgegangen werden. Stattdessen muss im schlechtesten Fall bei jedem Router von einer vollen Warteschlange ausgegangen werden, wie im Folgenden erläutert wird.
Es wird zunächst nur der Fall betrachtet, dass der Strom F_1 sich bei jedem Router in der höchsten Priorität befindet. Für einen Burst b_1 des Stroms F_1 gilt somit, dass $b_1 \leq C_0^{max}$. Der Strom wird daher überall in die Warteschlange höchster Priorität (W_0) eingeordnet. Durch die Admission Control

ist sichergestellt, dass diese Warteschlangen nicht überlaufen, also genügend Platz für einen Burst der erlaubten Größe C_0^{max} vorhanden ist. Sie können aber vollständig ausgelastet sein. Dadurch trifft der Burst b_1 im schlechtesten Fall bei jedem Router auf eine fast volle Warteschlange, die nur noch Platz für C_0^{max} hat. Es müssen somit zuerst alle Pakete in der Warteschlange vor dem Burst abgearbeitet werden und anschließend der Burst selbst. Dies entspricht dem Abarbeiten von $|W_0| - C_0^{max} + b_1$. In der Warteschlange befinden sich auch die Platzhalter-Pakete, wodurch die abgegebene Bandbreite für niedrigere Prioritäten automatisch berücksichtigt ist. Deshalb gilt:

$$d_R^{max} = \frac{|W_0| - C_0^{max} + b_1}{R^{eg}}, \text{ wenn } F_1 \text{ bei } R \text{ in } W_0 \tag{4.45}$$

Befindet sich der Strom bei einem Router R beispielsweise nur in der dritthöchsten Priorität, so erhält er durch die Platzhalter-Pakete dort nur einen Anteil $R^{eg} \cdot l_0 \cdot l_1$ der Bandbreite. Die zweithöchste Priorität erhält $R^{eg} \cdot l_0$ der Bandbreite und gibt durch die Platzhalter-Pakete davon den Anteil l_1 an die dritthöchste Priorität ab. Auch hier ist die abgegebene Bandbreite an eine eventuell darunterliegende vierte Prioritäten durch die Platzhalter-Pakete in der Warteschlange W_2 bereits berücksichtigt. Die Verzögerung beträgt somit:

$$d_R^{max} = \frac{|W_2| - C_2^{max} + b_1}{R^{eg} \cdot l_0 \cdot l_1}, \text{ wenn } F_1 \text{ bei } R \text{ in } W_2 \tag{4.46}$$

Im Allgemeinen kann das Queueing-Delay eines Routers R_i, bei welchem der Strom die Priorität p erhält, berechnet werden mit:

$$d_{R_i}^{max} = \frac{|W_p| - C_p^{max} + b_1}{R_i^{eg} \cdot \prod_{j=0}^{p-1} l_j} \tag{4.47}$$

Zur Verdeutlichung der Formel illustriert Abbildung 4.15 den Warteschlangenfüllstand im schlechtesten Fall eines Routers mit drei Prioritäten.

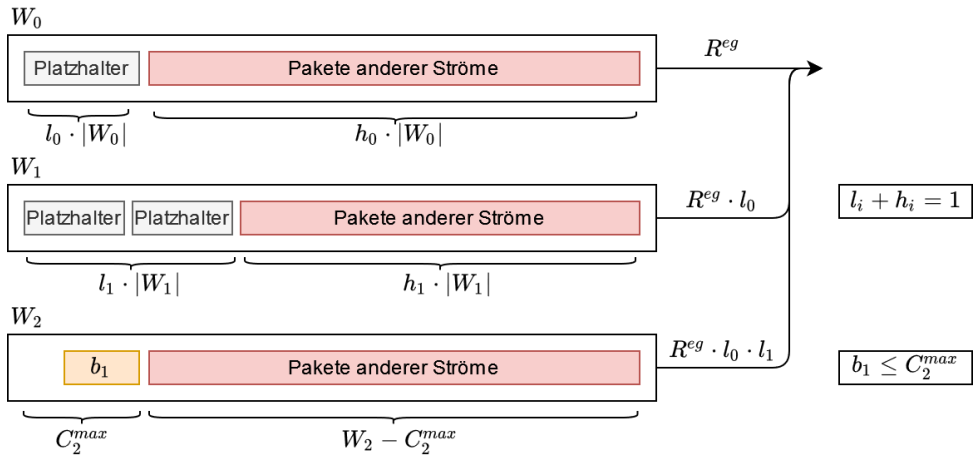

Abbildung 4.15: Illustration des Warteschlangenfüllstands im schlechtesten Fall eines Routers mit drei Prioritäten. Der Strom F_1 befindet sich in der dritten und niedrigsten Priorität.

Zur Berechnung der Gesamtverzögerung wird davon ausgegangen, dass der Strom F_1 mit einem Token-Bucket (r_1, b_1) spezifiziert wird. Er durchläuft anschließend mehrere Router $R_1, R_2, \ldots$, welche

eine Ausgangsbandbreite R^{eg} und Eingangsbandbreite R^{in} besitzen und das neu entworfene Modell implementieren. Zunächst wird eine Beispieltopologie mit zwei Routern untersucht, bevor eine allgemeine Formel aufgestellt wird. Diese Beispieltopologie ist in der folgenden Abbildung 4.16 illustriert.

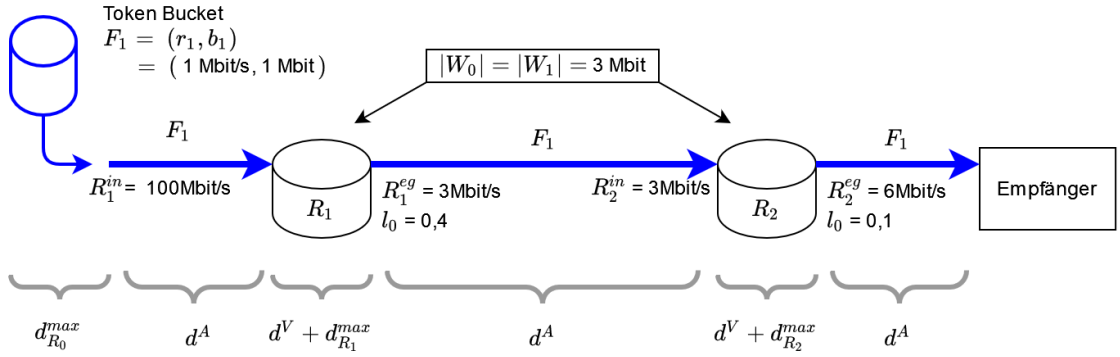

Abbildung 4.16: Beispieltopologie mit zwei Routern, identisch mit der Beispieltopologie für IntServ (Abbildung 4.5). Bei IntServ konnten theoretisch 3 x 1 Mbit Ströme über den Flaschenhals laufen. Um möglichst vergleichbar zu sein, wurde die Warteschlangengröße auf 3 Mbit begrenzt, sodass im neuen Modell auch 3 x 1 Mbit Ströme aufgenommen werden können.

Zunächst wird angenommen, dass F_1 sich bei beiden Routern in der höchsten Priorität befindet und beide Router die selben Maximalwerte C_0^{max} verwenden. Weiter wird davon ausgegangen, dass F_1 einen Burst maximal zulässiger Größe sendet, also $b_1 = C_0^{max}$. Die Verzögerung $d^{max}_{R_0}$ des Token-Buckets kann wie zuvor bei IntServ mit $\frac{b_1}{R_1^{in}}$ berechnet werden. Damit ergibt sich folgende maximale Verzögerung für die Beispieltopologie:

$$\begin{aligned} d^{max}_{gesamt} &= d^{max}_{R_0} + d^A + d^V + d^{max}_{R_1} + d^A + d^V + d^{max}_{R_2} + d^A \\ &= \frac{b_1}{R_1^{in}} + \frac{|W_0|}{R_1^{eg}} + \frac{|W_0|}{R_2^{eg}} + 3 \cdot d^A + 2 \cdot d^V \\ &= \frac{1 \cdot 10^6 Bit}{100 \cdot 10^6 Bit/s} + \frac{3 \cdot 10^6 Bit}{3 \cdot 10^6 Bit/s} + \frac{3 \cdot 10^6\ Bit}{6 \cdot 10^6 Bit/s} + 5ms \\ &= 10ms + 1000ms + 500ms + 5ms \\ &= 1515ms \end{aligned} \tag{4.48}$$

Falls F_1 bei beiden Routern in der zweithöchsten Priorität eingeordnet wurde und ein Burst maximaler Größe ($b_1 = C_1^{max}$) gesendet hat, kann die maximale Verzögerung folgendermaßen berechnet werden:

$$\begin{aligned} d_{gesamt}^{max} &= d_{R_0}^{max} + d^A + d^V + d_{R_1}^{max} + d^A + d^V + d_{R_2}^{max} + d^A \\ &= \frac{b_1}{R_1^{in}} + \frac{|W_1|}{R_1^{eg} \cdot l_0} + \frac{|W_1|}{R_2^{eg} \cdot l_0} + 3 \cdot d^A + 2 \cdot d^V \\ &= \frac{1 \cdot 10^6 Bit}{100 \cdot 10^6 Bit/s} + \frac{3 \cdot 10^6 Bit}{3 \cdot 10^6 Bit/s \cdot 0,4} + \frac{3 \cdot 10^6\ Bit}{6 \cdot 10^6 Bit/s \cdot 0,1} + 5ms \\ &= 10ms + 2500ms + 5000ms + 5ms \\ &= 7515ms \end{aligned} \tag{4.49}$$

Im Allgemeinen kann die maximale Verzögerung für $n \geq 1$ Router mit folgender Formel berechnet werden:

$$d_{gesamt}^{max} = \frac{b_1}{R_1^{in}} + \sum_{i=1}^{n} d_{R_i}^{max} + nd^V + (n+1)d^A \tag{4.50}$$

Berechnung der maximalen Ende-zu-Ende Verzögerung unter Berücksichtigung der Admission-Control Die zuvor berechnete maximale Verzögerung stellt eine obere Schranke dar, die unabhängig vom konkreten Admission-Control-Verfahren gilt. Bei der Verwendung des zuvor vorgestellten Admission-Control-Algorithmus können noch präzisere Aussagen über die maximale Verzögerung gemacht werden. In der allgemeinen Berechnung wird angenommen, dass im schlechtesten Fall alle Warteschlangen fast vollständig gefüllt sind. Beim gezeigten Admission-Control-Verfahren kann der maximale Füllstand der höchsten und zweithöchsten Priorität durch M genauer angegeben werden. Dadurch muss die abgegebene Bandbreite an die nächst niedrigere Priorität berücksichtigt werden (mr_0 statt R^{eg}), da in M keine Platzhalter-Pakete enthalten sind. Es gilt für den schlechtesten Fall für einen Strom F_1 in der höchsten Priorität:

$$d_R^{max} = \frac{M \cdot C_0^{max}}{mr_0}, \text{ wenn } F_1 \text{ bei } R \text{ in } W_0 \tag{4.51}$$

Und falls F_1 sich in der zweithöchsten Priorität befindet gilt:

$$\begin{aligned}
d_R^{max} &= \frac{M_1^{ex} \cdot C_1^{max}}{tr_1}, \text{ wenn } F_1 \text{ bei } R \text{ in } W_1 \\
&= \frac{\frac{M}{2 \cdot h_0} \cdot C_1^{max}}{R^{eg} \cdot \frac{l_0}{\frac{M_0}{M} \cdot h_0 + l_0} \cdot h_1} \\
&= \frac{M \cdot C_1^{max} \cdot (\frac{M_0}{M} \cdot h_0 + l_0)}{2 \cdot h_0 \cdot R^{eg} \cdot l_0 \cdot h_1} \\
&= \frac{M \cdot C_1^{max} \cdot (\frac{M - \frac{M}{2 \cdot h_0}}{M} \cdot h_0 + l_0)}{2 \cdot h_0 \cdot mr_1} \\
&= \frac{M \cdot C_1^{max} \cdot ((1 - \frac{1}{2 \cdot h_0}) \cdot h_0 + l_0)}{2 \cdot h_0 \cdot mr_1} \\
&= \frac{M \cdot C_1^{max} \cdot ((h_0 - \frac{1}{2}) + l_0)}{2 \cdot h_0 \cdot mr_1} \\
&= \frac{M \cdot C_1^{max} \cdot (1 - \frac{1}{2})}{2 \cdot h_0 \cdot mr_1} \\
&= \frac{M \cdot C_1^{max} \cdot \frac{1}{2}}{2 \cdot h_0 \cdot mr_1} \\
&= \frac{M \cdot C_1^{max}}{4 \cdot h_0 \cdot mr_1} \overset{(4.43)}{\leq} T
\end{aligned} \tag{4.52}$$

An dieser Stelle kann beobachtet werden, dass die maximale Verzögerung durch die Länge des Messintervalls T beschränkt ist. Dies folgt aus dem Umstellen der Formel 4.43. Intuitiv kann diese Eigenschaft dadurch begründet werden, dass M so gewählt wird, dass alle Pakete in der Warteschlange W_1 einmal pro Messintervall T abgearbeitet werden können. Dadurch wird sichergestellt, dass sich keine Pakete in der Warteschlange anstauen können, da es sonst zu einem Überlaufen führen würde. Darum verweilt ein Burst höchstens die Zeit T in einem Router. Somit kann T auch dazu verwendet werden, eine bestimmte maximale Verzögerung für einen Router vorzugeben.
Unter der Annahme, dass F_1 jeweils in der zweithöchsten Priorität eingeordnet ist und $M = 3$ bei beiden Routern der Beispieltopologie (Abb. 4.16) gilt, kann folgende maximale Verzögerung garantiert werden:

$$\begin{aligned}
d_{gesamt}^{max} &= d_{R_0}^{max} + d^A + d^V + d_{R_1}^{max} + d^A + d^V + d_{R_2}^{max} + d^A \\
&= 10ms + \frac{M \cdot C_1^{max}}{4 \cdot h_0 \cdot mr_1} + \frac{M \cdot C_1^{max}}{4 \cdot h_0 \cdot mr_1} + 3 \cdot d^A + 2 \cdot d^V \\
&= \frac{3 \cdot 1 \cdot 10^6 \; Bit}{4 \cdot 0,6 \cdot 3 \cdot 10^6 \cdot 0,4 \; Bit/s} + \frac{3 \cdot 1 \cdot 10^6 \; Bit}{4 \cdot 0,9 \cdot 6 \cdot 10^6 \cdot 0,1 \; Bit/s} + 15ms \\
&= 1041,7ms + 1388,9ms + 15ms \\
&= 2445,6ms
\end{aligned} \tag{4.53}$$

Berechnung der individuellen Verzögerung beim Sender Es ist aus den vorherigen Abschnitten bekannt, wie die Ende-zu-Ende Verzögerung im Allgemeinen und unter Berücksichtigung des vorgestellten Admission-Control-Mechanismus berechnet werden kann. Diese Berechnung

benötigt allerdings viele Router-spezifische Parameter ($M, h_0, |W_0|, ...$), die einem Sender nicht bekannt sind. Abhängig vom eigenen Sendeverhalten soll aber jeder Sender die Möglichkeit haben, die eigenen Prioritäten bei den Routern und die damit verbundene maximale Verzögerung vorhersagen zu können. Um diese Anforderung der Problemstellung zu erfüllen, können die Router ihre eigenen maximalen Verzögerungen (d_R^{max}) jeder Priorität vorausberechnen. Für die Berechnungen sind nur die festgelegten Router-Parameter notwendig, wodurch jeder Router die Berechnungen nur einmalig durchführen muss. Sobald ein neuer Strom über die Admission-Control zugelassen wurde, können dann mit dem Zustimmungspaket folgende Werte pro Router R_i mitgeschickt werden:

- Die maximale Verzögerung des Routers $d_{R_i}^{max}$ jeder Priorität
- Die maximale Burst-Größe C^{max} pro Messintervall und Priorität
- Die Länge des Messintervalls T
- Die Verarbeitungszeit d^V
- Die Ausbreitungsverzögerung d^A

Die Ausbreitungsverzögerung d^A kann ebenfalls vom Router für alle direkten Nachbarn gemessen und gespeichert werden. Anhand des nächsten Hops des Zustimmungspakets kann jeder Router den zugehörigen Wert d^A auswählen.
Mit dem Eintreffen des Zustimmungspakets beim Sender hat dieser damit alle benötigten Werte, um die Priorität und Verzögerung seines Stroms zu berechnen. Mithilfe von C^{max} und T kann der Sender bestimmen, welche maximalen Burst-Größen und mit welcher durchschnittlichen Rate er senden darf, um in einer bestimmten Priorität zu bleiben. Durch $d_{R_i}^{max}, d^V$ und d^A kann er die maximale Ende-zu-Ende Verzögerung schließlich bestimmen.

Zusammenfassung des entworfenen Modells Das neu entworfene Modell verwendet pro Router eine feste Anzahl von Warteschlangen, unabhängig von der Anzahl der Netzwerkströme. Dadurch ist der Verwaltungsaufwand pro Router gering, insbesondere geringer als bei IntServ und daher skalierbar mit der Anzahl an Strömen. Die Warteschlangen repräsentieren verschiedene Prioritäten in welche die Netzwerkströme dynamisch anhand ihres Sendeverhaltens eingeordnet werden. Das Verfahren basiert auf dem Prinzip einer MLFQ und wurde um einen Aufstiegsmechanismus und einem Mechanismus zum Verhindern des Aushungerns niedriger Prioritäten erweitert. Es ist für die Sender der Ströme berechenbar, welche Prioritäten sie bei den einzelnen Routern erhalten. Mit dem vorgestellten Admission-Control-Verfahren kann sichergestellt werden, dass die beiden höchsten Prioritäten vor Überlast und Paketverlust durch überlaufende Warteschlangen geschützt sind. Damit ist es möglich, für Ströme in der höchsten und zweithöchsten Priorität eine maximale Verzögerung zu garantieren. Diese Verzögerung kann berechnet werden mit:

$$d_{gesamt}^{max} = \frac{b_1}{R_1^{in}} + \sum_{i=1}^{n} d_{R_i}^{max} + nd^V + (n+1)d^A \tag{4.54}$$

Dabei ist die Verzögerung pro Router $d_{R_i}^{max}$ abhängig von der Priorität des Stroms und des verwendeten Admission-Control-Verfahrens (Siehe hierzu die Gleichungen (4.47)(4.51) (4.52)).
Auch für Ströme in den niedrigeren Prioritäten kann mit derselben Formel eine maximale Verzögerung berechnet werden. Jedoch gibt es keine harten Garantien mehr, denn Pakete können verloren gehen. Bei Überlast gilt nur noch, dass ein Paket entweder nach der berechneten

maximalen Verzögerung beim Empfänger ankommt, oder aufgrund einer überfüllten Warteschlange verworfen wurde. Zu verspäteten Paketen kommt es dagegen nicht.
Das Messintervall T, welches ein Design-Parameter eines Routers darstellt, kann zur Beschränkung der maximalen Verzögerung in den höchsten beiden Prioritäten eines Routers verwendet werden. Damit ist es möglich, einzelne Router oder komplette Routen mit einer gewünschten maximalen Verzögerung zu entwerfen. Davon abhängig sind die Bandbreite pro Strom und die Anzahl der möglichen Ströme, welche sich bei niedrigeren Verzögerungen entsprechend verringern.

5 Implementierung

In diesem Kapitel werden die verwendete Simulationsumgebung und Implementierungsdetails des neu entworfenen Scheduling-Verfahrens erläutert. Es wird zunächst die Simulationsumgebung und dessen generelle Funktionsweise erklärt. Im Anschluss folgen Details zu den implementierten Komponenten und deren Schnittstellen.

5.1 Einbindung in die Simulationsumgebung Omnet++

Das entworfene Scheduling-Verfahren nach dem Prinzip einer MLFQ wurde zur Evaluierung mit dem Simulator Omnet++ [VH08] (Version 5.5.1 unter Ubuntu 20.04) implementiert. Für den Omnet++ Simulator gibt es zusätzlich das INET-Framework, welches fertige Implementierungen der meisten Internet-Protokolle enthält. Darin enthalten sind z.B. die IPv4- und IPv6-Protokolle, TCP und UDP sowie verschiedene Link-Layer-Protokolle wie Ethernet und IEEE 802.11 (WLAN). Für diese Arbeit werden jedoch keine spezifischen Eigenschaften von Layer 1- und 2-Technologien angenommen, wie beispielsweise die Echtzeiterweiterungen für TSN. Auch das IP-Protokoll wird nicht benötigt, da von bereits vorgegebenen Routern ausgegangen wird und somit keine Routing-Funktionalität notwendig ist. Und auch die IP-Adressierung wird nicht benötigt, denn um die maximale Ende-zu-Ende-Verzögerung zu evaluieren, genügt es eine Linien-Topologie mit Cross-Traffic zu implementieren. Dafür ist keine explizite Adressierung der Endsysteme nötig. Es wurde daher auf das INET Framework verzichtet, da die enthaltenen Protokolle nicht benötigt wurden.

Generelle Funktionsweise von Omnet++ Der Simulator Omnet++ basiert auf der Programmiersprache C++ und ermöglicht es, Netzwerkkomponenten (bezeichnet als Module) in zwei Schritten zu implementieren. Als erstes wird eine sogenannte Network-Description (NED) Datei angelegt, welche die Ein- und Ausgänge sowie weitere Konfigurationsinformationen über das neue Modul enthält. Anschließend wird das Verhalten in einer separaten Datei in C++ implementiert.
Da der Simulator ereignisbasiert arbeitet, muss eine wesentliche Funktion namens „handleMessage(Message)“ implementiert werden. Diese Funktion wird von der Simulation ausgeführt, sobald eine Nachricht bzw. ein Paket beim Modul ankommt. Da es keine „wait“-Operationen gibt, müssen Verzögerungen mithilfe von „self-messages“ modelliert werden. Dabei handelt es sich um Nachrichten, die vom Modul an sich selbst adressiert werden und nach einer vorgegebenen Zeit ankommen. Mithilfe einer weiteren NED-Datei kann schließlich ein Netzwerk definiert werden, das aus verschiedenen Modulen besteht und die einzelnen Verbindungen festlegt. Dieses Netzwerk kann mit dem integrierten Simulator grafisch dargestellt und in verschiedenen Geschwindigkeitsmodi simuliert werden. Im Anschluss an die Simulation können gesammelte statistische Daten ausgewertet und visualisiert werden. Dazu werden sogenannte „Signale“ verwendet, die vom Modul implementiert werden und beliebige Informationen emittieren können.

Übersicht über die entwickelten Komponenten (Module) Es wurden drei Module implementiert, um das Verfahren zu evaluieren: Das Modul Token-Bucket-Sender, das MLFQ-Router-Modul und das Empfänger-Modul. Alle Module können für die Simulationen beliebig oft instanziiert werden und mit verschiedenen Parametern konfiguriert werden. Die Verbindungen (Channels) zwischen den Modulen können mit einer beliebigen Bandbreite und Ausbreitungsverzögerung konfiguriert werden. Sie geben gleichzeitig das Routing vor, weshalb sie während der Simulation nicht geändert werden können. Genaueres dazu wird im Abschnitt des MLFQ-Routers erläutert. Die Funktionsweise der einzelnen Module wird in den folgenden Abschnitten erklärt. Außerdem wurde eine einfache Paket-Klasse entwickelt, auf die zum Abschluss kurz eingegangen wird.

5.2 Token-Bucket-Sender

Dieses Modul dient dazu, Netzwerkverkehr zu generieren. Es kann sowohl zur Evaluation von Ende-zu-Ende Messungen verwendet werden, als auch für Cross-Traffic zur Lastsimulation. Die Schnittstellen und Konfigurationsparameter des Token-Bucket-Senders sind in Abbildung 5.1 illustriert.

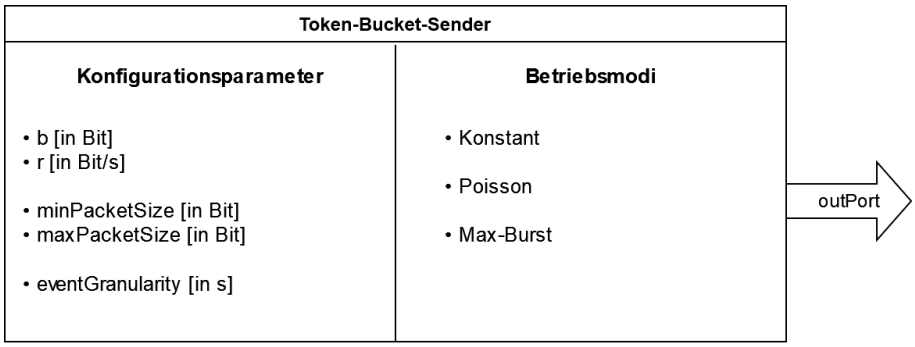

Abbildung 5.1: Übersicht der Schnittstellen und Konfigurationsparameter des Token-Bucket-Sender-Moduls.

Zusätzlich zu der bekannten Strom-Spezifikation mit Token-Rate r und Eimergröße b kann das Modul in verschiedenen Modi betrieben werden:

- Konstant: Bei diesem Modus werden mit konstanter Rate Pakete der minimalen Paketgröße gesendet. Die Anzahl der Pakete ist abhängig von der Token-Rate r. Um zu verhindern, dass die Sendeereignisse bei allen Sendern synchron mit dem Token-Nachfüllereignis stattfinden, wurden die Sendezeitpunkte randomisiert. Die Zeit zwischen den Sendeereignissen wird gleich verteilt zufällig aus dem Intervall $[0, 2 \cdot \texttt{eventGranularity}]$ gewählt, sodass sie im Mittel `eventGranularity` beträgt.
- Poisson: Das Sendeverhalten wird mit einer Poisson-Verteilung modelliert. Die Paketgröße ist dabei gleichverteilt über das Intervall `[minPaketSize, maxPacketSize]`. Der Erwartungswert λ für die Anzahl der Sendeereignisse pro Sekunde wird daher festgelegt auf:

$$\lambda \cdot \frac{\texttt{minPacketSize} + \texttt{maxPacketSize}}{2} = r \Rightarrow \lambda = \frac{2 \cdot r}{\texttt{minPacketSize} + \texttt{maxPacketSize}} \quad (5.1)$$

Dadurch werden pro Sekunde λ Pakete verschickt, die insgesamt $r \cdot 1\ s$ Bits umfassen. Somit entspricht die mittlere Senderate der Token-Rate r. Die Verteilung der Sendeereignisse erfolgt zufällig gemäß der Poisson-Verteilung. Der Zeitpunkt des nächsten Sendeereignisses t_i wird dazu berechnet mit:

$$t_i = t_{i-1} - \frac{\log(1.0 - \text{uniform}(0,1))}{\lambda} \tag{5.2}$$

Mit $uniform(0,1)$ wird eine gleich verteilte Zufallszahl aus dem Intervall $[0,1[$ generiert.

- Max-Burst: Dieser Modus simuliert einen Sender, welcher nur Bursts maximaler Größe sendet. Der Sender wartet dabei, bis der Eimer komplett gefüllt ist und sendet dann Pakete zufälliger Größe, bis zum ersten Mal nicht mehr genügend Tokens zur Verfügung stehen. Anschließend wartet er erneut, bis der Eimer komplett gefüllt ist. Um zu verhindern, dass der Beginn eines Bursts synchron mit dem Token-Nachfüllereignis stattfindet, wurde auch hier der Sendezeitpunkt randomisiert. Wie zuvor beim Konstant-Modus wurde eine Zufallsverzögerung gleich verteilt aus dem Intervall $[0, 2 \cdot \texttt{eventGranularity}]$ gewählt, sodass sie im Mittel `eventGranularity` beträgt. Diese Zufallsverzögerung hat nur Einfluss auf den Beginn des Bursts, die einzelnen Pakete innerhalb des Bursts werden ohne Verzögerung geschickt, sobald das Medium frei ist.

5.3 MLFQ-Router

Das Herzstück der Implementierung ist das MLFQ-Router-Modul. Abbildung 5.2 zeigt eine Übersicht über die Konfigurationsparameter und Schnittstellen der Moduls.

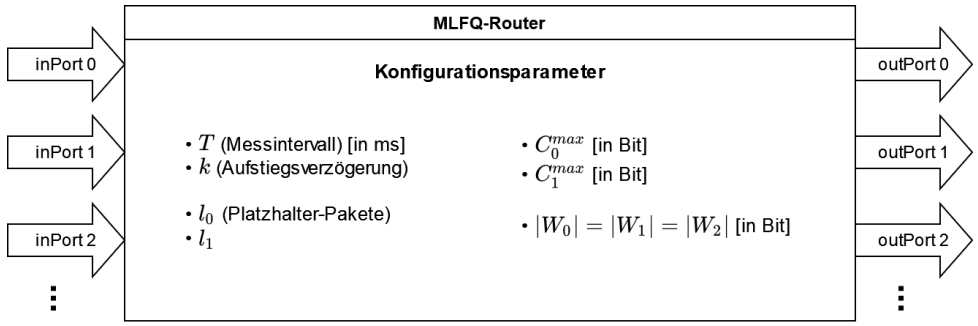

Abbildung 5.2: Übersicht der Schnittstellen und Konfigurationsparameter des MLFQ-Router-Moduls.

Es wurde zur Demonstration ein Router mit drei Prioritäten implementiert. Der Router speichert pro Strom die aktuelle Priorität sowie die Anzahl der Bits, die im aktuellen Messintervall gesendet wurden. Im Router befinden sich drei Warteschlangen für die Prioritäten 0 – 2, deren Länge der Einfachheit halber identisch ist. Außerdem können die Schwellwerte C_0^{max} und C_1^{max} sowie der prozentuale Anteil der Platzhalter-Pakete l_0 bzw. l_1 beliebig konfiguriert werden. Auch das Messintervall T und die Länge der Aufstiegsverzögerung k kann variiert werden. Für die Aufstiegsverzögerung wurden beide vorgestellten Varianten (allgemeiner Zähler ohne Zustand pro Strom und individueller Zähler pro Strom) implementiert.

Das Modul besitzt eine variable Anzahl an Ein- und Ausgängen (`inPort` bzw. `outPort`). Jedes Paket wird über den Ausgang mit der gleichen Nummer weitergeschickt, wie die Nummer des Eingangs, über die es empfangen wurde. Damit können vor Simulationsbeginn die Routen der einzelnen Ströme festgelegt werden. Intern werden alle Pakete in die 3 Warteschlangen nach dem MLFQ-Prinzip eingeordnet, es handelt sich bei den Ein- und Ausgängen also nicht um separate Warteschlangen. Um beispielsweise einen Flaschenhals mit Cross-Traffic zu simulieren, kann der gemessene Strom mit Token-Bucket an `inPort 0` und der Empfänger an `outPort 0` angeschlossen werden. Der Cross-Traffic kann an die anderen Ports angeschlossen werden, z.B. an `inPort 1` und `outPort 1`.
Die Platzhalter-Pakete wurden nicht als tatsächliche Pakete implementiert, sondern lediglich als Zähler. Mit jedem weitergeleiteten Paket wird der Zähler um die entsprechende prozentuale Anzahl an Bits erhöht. Falls ein Paket in den darunterliegenden Warteschlangen bereit steht und der Zählerwert genügend Bits umfasst, wird die nächst niedrigere Priorität bedient und der Zähler zurückgesetzt. Sollten alle darunterliegenden Warteschlangen leer sein, so wird der Zähler ebenfalls zurückgesetzt, da keine Platzhalter-Pakete angespart werden können.
Die Verarbeitungszeit d^V (Processing Delay) soll pro Router einen konstanten Wert haben. Im ereignisbasierten Simulator existiert jedoch keine Verarbeitungszeit, alle Berechnungen werden ohne Zeitverlust durchgeführt. Daher müsste jedes eingehende Paket für exakt d^V beim Router festgehalten bzw. pausiert werden, bevor es verarbeitet wird, um die Verarbeitungszeit zu simulieren. Dies ist im ereignisbasierten Simulator aufwändig zu implementieren, da hierfür pro Paket eine zusätzliche Selbst-Nachricht benötigt wird, um die Dauer des Festhaltens zu simulieren. Stattdessen wurde die Verarbeitungszeit mit der Ausbreitungsverzögerung verrechnet, was leicht implementiert werden kann. Endet eine Verbindung bei einem Router, so beträgt ihre implementierte Ausbreitungsverzögerung $d^A + d^V$ anstatt d^A. Beträgt beispielsweise $d^A = 0{,}05$ ms und $d^V = 0{,}05$ ms, so wird für Verbindung, die am Router endet eine Ausbreitungsverzögerung von 0,1 ms implementiert. Der Router bearbeitet die Pakete dafür sofort und fügt keine Verarbeitungszeit hinzu.

5.4 Empfänger und Paket-Klasse MLFQ-Paket

Das Empfänger-Modul nimmt Pakete entgegen und entfernt sie aus dem Netzwerk. Es ist außerdem dafür zuständig, die Ende-zu-Ende-Verzögerung der ankommenden Pakete zu messen und für die Statistik aufzuzeichnen. Es besitzt nur einen Eingang (`inPort`) und keine weiteren Konfigurationsparameter. Daher wurde auf eine Übersichtsgrafik verzichtet.

Paket-Klasse MLFQ-Paket Standardmäßig besitzen Pakete in Omnet++ ein Attribut, das den Erstellungszeitpunkt enthält. Mithilfe dieses Attributs kann der Empfänger die Ende-zu-Ende-Verzögerung berechnen. Um die Pakete der verschiedenen Sender den richtigen Prioritäten zuordnen zu können, wurden die Standard-Pakete um ein zusätzliches Attribut `senderID` erweitert. Dabei handelt es sich um eine eindeutige Nummer des Senders.

6 Evaluierung

In diesem Kapitel wird die korrekte Funktionsweise des neu konzipierten Modells evaluiert. Dazu wird als erstes untersucht, ob die dynamische Einordnung in die jeweiligen Prioritäten funktioniert. Anschließend werden die beiden vorgestellten Algorithmen der Aufstiegsverzögerung hinsichtlich ihrer korrekten Funktionsweise evaluiert. Dabei wird auch der Einfluss des Messintervalls T auf die korrekte Funktionsweise der Algorithmen untersucht. Im darauffolgenden Abschnitt wird die garantierte Ende-zu-Ende-Verzögerung in verschiedenen Szenarien untersucht. Dabei wird evaluiert, ob die berechneten maximalen Verzögerungen eingehalten werden. Abschließend wird die effiziente Nutzung der Ausgangsbandbreite eines Routers evaluiert.

6.1 Aufbau der Evaluierung

Zur Evaluierung wird der Netzwerksimulator Omnet++ und die dafür vorgestellte Implementierung des neu konzipierten Modells verwendet. Die Implementierung wurde bereits im vorherigen Kapitel 5 genauer erläutert. Für die Sender wurden verschiedene Sende-Modi implementiert, um unterschiedliche Lastmodelle zu simulieren (vgl. Kapitel 5):

- Konstant-Modus: Dieser Modus stellt den bestmöglichen Fall aus Sicht des Netzwerks dar, bei welchem kontinuierlich Pakete mit konstanter Größe geschickt werden. Es handelt sich somit um einen Strom mit konstanter Bitrate ohne Paketbursts. Dieser Modus ist besonders geeignet, um Cross-Traffic zu simulieren, welcher die komplette Bandbreite ausnutzt.
- Poisson-Modus: Mit diesem zufälligen Modus kann der durchschnittliche Fall simuliert werden. Durch die Poisson-Verteilung der Sendeereignisse wird eine Anwendung mit zufälligen Bursts und einer gegebenen durchschnittlichen Bitrate modelliert.
- Max-Burst-Modus: Durch diesen Modus wird der schlechteste Fall aus Sicht des Netzwerks simuliert. Es werden nur Bursts maximaler Größe geschickt, gefolgt von Pausen, in welchen erneut Tokens angespart werden.

Als „Setups“ werden die Topologien mit den verwendeten Konfigurationsparametern bezeichnet, welche für die verschiedenen Messungen verwendet wurden. Für alle Setups wird eine Ausbreitungsverzögerung $d^A = 0{,}05$ ms pro Verbindung angenommen. Dies entspricht einer Distanz von 10 km bei $\frac{2}{3}$ Lichtgeschwindigkeit (typische Signalausbreitungsgeschwindigkeit in einem Kupfer- oder Glasfasermedium). Außerdem wird eine Verarbeitungszeit $d^V = 0{,}05$ ms pro Router angenommen, was in etwa dem gemessenen Mittel zwischen Hardware- und Software-Switches in der Arbeit von Dürr und Kohler entspricht [DK+14]. Im Folgenden werden die einzelnen Setups vorgestellt. Zur einfacheren Referenzierung wurden sie jeweils mit einem Namen versehen:

Single-Router-Setup Dieses Setup besteht nur aus drei Komponenten: Einem Token-Bucket-Sender, einem MLFQ-Router und einem Empfänger. Die Topologie wird in Abbildung 6.1 illustriert.

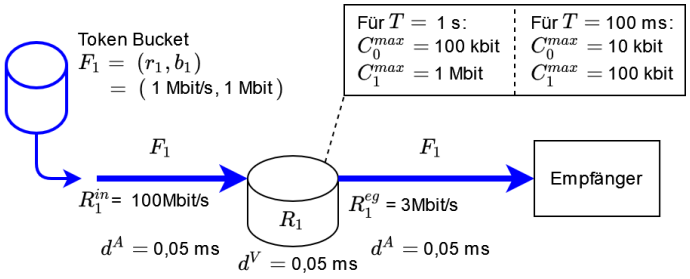

Abbildung 6.1: Die Topologie des Single-Router-Setups. Sie besteht aus einem Sender, der über einen Router mit dem Empfänger verbunden ist.

Durch die höhere Eingangsbandbreite (100 Mbit/s) des Routers im Vergleich zu dessen Ausgangsbandbreite (3 Mbit/s) handelt es sich um einen Flaschenhals, bei welchem sich die Pakete stauen. Die Parameter des Token-Buckets und die Schwellwerte des Routers sind so gewählt, dass der Strom des Senders in Priorität 2 fällt. Da es in diesem Setup keine weiteren Ströme gibt, wurden die Platzhalter-Pakete deaktiviert, um die Übersichtlichkeit zu erhöhen.

Cross-Traffic-Setup Bei diesem Setup gibt es zusätzlichen Cross-Traffic, um ein Netzwerk unter Last zu simulieren. Abbildung 6.2 zeigt die Topologie des Setups.

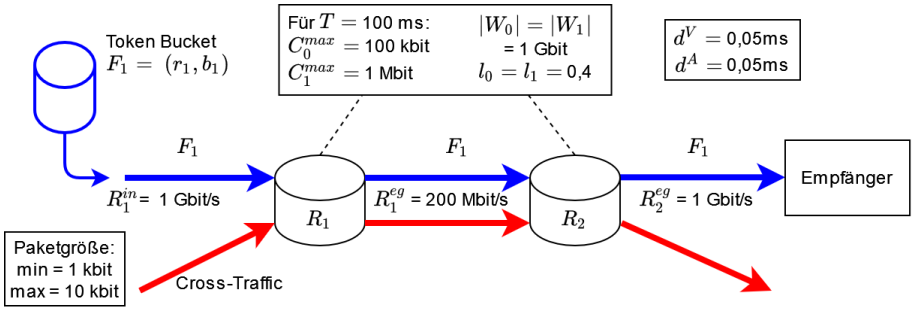

Abbildung 6.2: Die Topologie des Cross-Traffic-Setups. Die beiden Router werden mit zusätzlichem Cross-Traffic belastet.

Für dieses Setup wurde sowohl der Token-Bucket-Sender als auch der Cross-Traffic variabel gehalten, sodass verschiedene Kombinationen evaluiert werden können. Der Cross-Traffic kann sowohl als einzelner Strom implementiert werden, oder als mehrere kleine Ströme. Dadurch können verschiedene Netzwerkbelastungen simuliert werden, die sich unterschiedlich auf den zu evaluierenden Strom F_1 auswirken. Beide Router verwenden das Messintervall T = 100 ms sowie die Schwellwerte C_0^{max} = 100 kbit und C_1^{max} = 1 Mbit. Der Flaschenhals befindet sich bei R_1, da dieser nur über eine Ausgangsbandbreite von 200 Mbit/s verfügt.

6.2 Dynamische Einordnung der Priorität

Bei dieser Messung besteht das Ziel darin, die korrekte Auf- und Abstufung eines Stroms zu untersuchen. Das Verfahren soll Ströme dynamisch in die verschiedenen Prioritäten einordnen, wenn sich deren Sendeverhalten verändert.

Es wird für diese Messung die Topologie des Single-Router-Setups verwendet. Um einen dynamischen Sender zu simulieren, arbeitet der Token-Bucket-Sender im Konstant-Modus und verändert seine Token-Rate. Zu Beginn verwendet er eine Token-Rate von 50 kbit/s, diese wird nach 5 s auf 500 kbit/s erhöht. Zum Zeitpunkt 10 s wird sie erneut erhöht auf 5 Mbit/s und zum Zeitpunkt 15 s wieder auf die ursprünglichen 50 kbit/s zurückgesetzt. Für die Parametrisierung des Routers wird T = 100 ms und k = 11 mit der individuellen Aufstiegsverzögerung verwendet.
Durch die Schwellwerte C_0^{max} = 10 kbit und C_1^{max} = 100 kbit (bei T = 100 ms) fällt der Strom mit jeder Erhöhung in die nächst niedrigere Priorität. Dadurch wird eine Abstufung zum Zeitpunkt 5 s und 10 s erwartet. Mit dem Zurücksetzen der Token-Rate auf 50 kbit/s befindet sich der Strom wieder unter dem Schwellwert der höchsten Priorität und sollte daher in Priorität 0 aufsteigen.
Um dieses Verhalten zu evaluieren, wird die Priorität des Stroms im Verlauf der Zeit gemessen. Die Ergebnisse werden in Abbildung 6.3 illustriert.

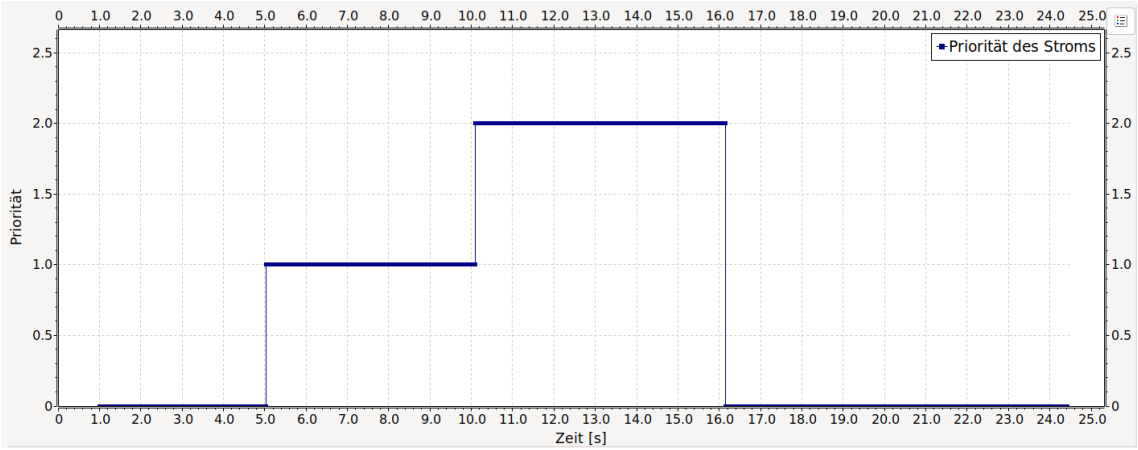

Abbildung 6.3: Ergebnisse der gemessenen Priorität des Stroms im Verlauf der Zeit.

Am zeitlichen Verlauf der Priorität des Stroms wird deutlich, dass die Abstufung innerhalb eines Messintervalls (100 ms) nach der Erhöhung der Token-Rate stattfindet. Der Aufstieg von Priorität 2 zu Priorität 0 findet dagegen erst zum Zeitpunkt 16,1 s statt. Bei dieser Verzögerung handelt es sich um die Aufstiegsverzögerung, welche $k \cdot T = 11 \cdot 100$ ms $= 1{,}1$ s beträgt. Sie stellt eine Bewährungszeit dar, mit welcher verhindert wird, dass ein Strom ständig zwischen den Prioritäten alterniert. Die Aufstiegsverzögerung wird im nächsten Abschnitt genauer untersucht.

6.3 Einfluss der Aufstiegsverzögerung

Das Ziel dieser Messung ist die Überprüfung der korrekten Funktionsweise der beiden Aufstiegsverzögerungs-Varianten. Die Aufstiegsverzögerung soll verhindern, dass Ströme mit niedriger Priorität und großen Bursts in den Pausen zwischen den Bursts direkt wieder aufsteigen. Die Länge der Aufstiegsverzögerung ist ein einstellbarer Parameter, der Auswirkung auf die Reaktionszeit des Verfahrens hat. Eine zu lang gewählte Aufstiegsverzögerung lässt Ströme mit positiv verändertem Sendeverhalten erst nach einer langen Bewährungszeit aufsteigen. Diese Auswirkung schränkt die Dynamik des Verfahrens ein, ist jedoch nicht so gravierend wie eine zu kurz gewählte Aufstiegsverzögerung. Diese hat zur Folge, dass Ströme in den Pausen zwischen den Bursts in der Priorität aufsteigen, obwohl sie ihr Sendeverhalten nicht geändert haben. Um dies zu verhindern,

muss die Länge Aufstiegsverzögerung mindestens dem zeitlichen Abstand zwischen zwei Bursts entsprechen.
In Abschnitt 4.2 wurden zwei Varianten für die Implementierung der Aufstiegsverzögerung vorgestellt. Zum einen die individuelle Aufstiegsverzögerung, welche einen eigenen Zähler pro Strom benötigt und den Start der Verzögerung individuell steuert und zurücksetzt. Und zum anderen die allgemeine Aufstiegsverzögerung, welche nur einen allgemeinen Zähler verwendet und den Prioritätsaufstieg alle $2k + 1$ Messintervalle für alle Ströme ermöglicht. Beide Varianten wurden für diese Evaluierung implementiert.
Für diese Messung wird das Single-Router-Setup verwendet. Es wird der Warteschlangenfüllstand der drei Priorität im MLFQ-Router gemessen, womit gezeigt wird, wie viele Bits in den jeweiligen Warteschlangen gepuffert werden. Da es nur einen Sender gibt, kann daran auch direkt die Priorität des Senders ermittelt werden. Somit kann mit dieser Metrik die korrekte Funktionsweise evaluiert werden. Zusätzlich kann anhand des Warteschlangenfüllstands erkannt werden, wie viele Pakete eines Bursts in die verschiedenen Prioritäten eingeordnet werden. Dadurch wird sichtbar, wie schnell das Verfahren mit der jeweiligen Parametrisierung eine Abstufung durchführt. Die Messung wurde als erstes für das Messintervall $T = 1$ s durchgeführt. Die Messergebnisse der Warteschlangenfüllstände beider Implementierungen werden in Abbildung 6.4 visualisiert.

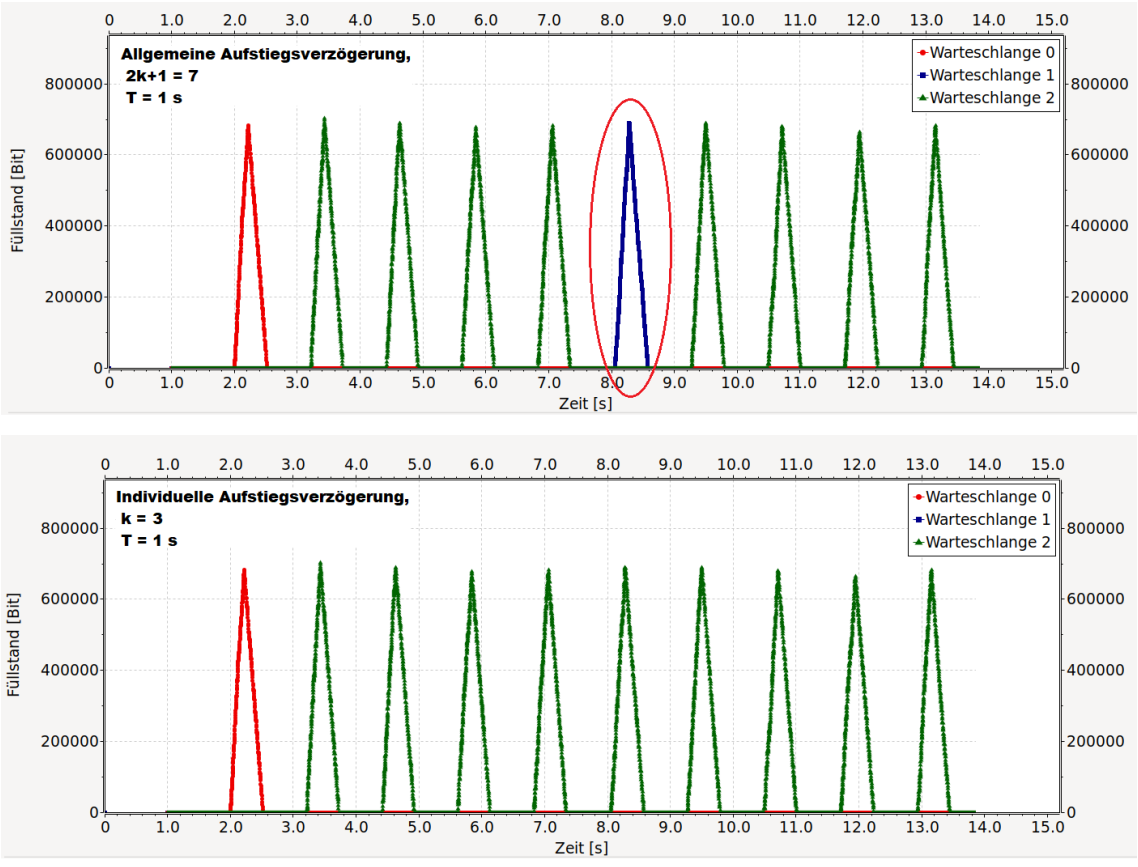

Abbildung 6.4: Es wird eine Visualisierung der Warteschlangenfüllstände während der Simulation gezeigt. Der Fälschliche Prioritätsaufstieg in der allgemeinen Variante ist durch eine rote Ellipsen hervorgehoben.

Der zeitliche Abstand zwischen zwei Bursts beträgt ca. 1,1 s, wodurch die Länge der Auf-

stiegsverzögerung mit 7 s bei der allgemeinen Variante und 3 s bei der individuellen Variante korrekt parametrisiert ist. Dennoch gibt es bei den Warteschlangenfüllständen zum Zeitpunkt 8 s eine Anomalie bei der allgemeinen Variante. Obwohl keine Änderung im Sendeverhalten bzw. der Burstgröße stattgefunden hat, wird der Burst zum Zeitpunkt 8 s komplett in die Priorität 1 eingeordnet. Erst ab Zeitpunkt 9 s befindet sich der Strom wieder in der niedrigsten Priorität. Bei der individuellen Variante kommt es nicht zu dieser Anomalie und der Strom verbleibt in Priorität 2. Bei beiden Varianten wird der erste Burst komplett in die Warteschlange 0 eingeordnet, da eine Änderung der Priorität erst zum Ende eines Messintervalls, also zum Zeitpunkt 3 s möglich ist. Der Grund dafür ist das lange Messintervall von einer Sekunde. Um den Einfluss des Messintervalls T zu verdeutlichen, wurden beide Varianten erneut mit einem geringeren Messintervall von 100 ms evaluiert. Die Aufstiegsverzögerung wurde soweit erhöht, dass sie bei der individuellen Variante mit 1,1 s etwa dem zeitlichen Abstand zwischen zwei Bursts entspricht. Bei der allgemeinen Variante beträgt sie entsprechend 2,3 s. Die Topologie bleibt unverändert. Die Warteschlangenfüllstände sind in Abbildung 6.5 visualisiert.

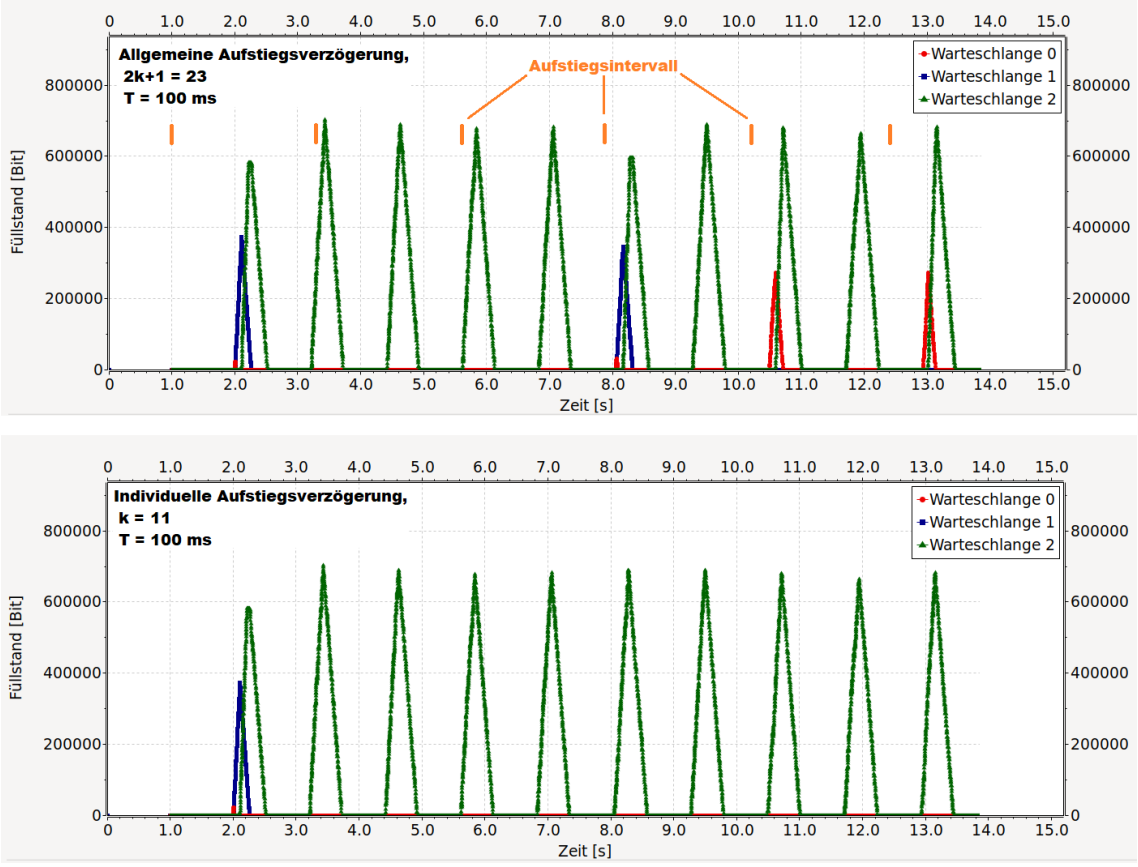

Abbildung 6.5: Die Warteschlangenfüllstände für T = 100 ms werden gezeigt. Die falsche Priorisierung bei der allgemeinen Variante bleibt bestehen. Die Zeitpunkte an denen ein Prioritätsaufstieg möglich ist (Aufstiegsintervalle), sind orange gekennzeichnet.

Durch das geringere Messintervall wird bei beiden Varianten bereits ein Teil des ersten Bursts in die Priorität 1 und 2 eingeordnet, da Änderungen der Priorität nun alle 100 ms stattfinden. Obwohl die Aufstiegsverzögerung mit 23 T = 2,3 s deutlich länger als der Abstand zwischen den Bursts

ist, findet bei der allgemeinen Variante auch mit kürzerem Messintervall ein Prioritätsaufstieg statt. Auch das weitere Erhöhen der Aufstiegsverzögerung löst das Problem nicht vollständig. Mit dem allgemeinen Verfahren kann die Entscheidung über einen Aufstieg nur vom Verhalten während *einem* Messintervall abhängig gemacht werden. Fällt dieses „Aufstiegsintervall“ in die Pause zwischen den Bursts (wie z.B. zum Zeitpunkt 7,9 s in Abbildung 6.5), so wird der Strom fälschlicherweise hochgestuft. Bei der individuellen Aufstiegsverzögerung kommt es dagegen zu keinem falschen Prioritätsaufstieg. Durch den individuellen Zähler steigt ein Strom erst nach 11 *aufeinanderfolgenden* Messintervallen mit nicht überschrittenem Schwellwert auf.
Es kann somit festgehalten werden, dass beide Varianten von einem kürzeren Messintervall profitieren. Dadurch können Bursts früher in der Priorität abgestuft werden, wodurch weniger Pakete in die Warteschlangen höherer Priorität eingeordnet werden. Außerdem kann festgehalten werden, dass die allgemeine Aufstiegsverzögerung ungewollte Prioritätsaufstiege nicht komplett verhindern kann. Dies ist nur mit der individuellen Variante möglich, unter der Voraussetzung, dass k größer als der zeitliche Abstand zwischen zwei Bursts ist. Aus diesem Grund wird für alle folgenden Evaluierungen die individuelle Aufstiegsverzögerung verwendet.

6.4 Garantierte Ende-zu-Ende-Verzögerung

Ziel dieser Messung ist die Untersuchung der garantierten maximalen Ende-zu-Ende-Verzögerung eines Netzwerks unter Last. In Abschnitt 4.2 wurden die Formeln vorgestellt, mit welchen die garantierten maximalen Verzögerung der Prioritäten berechnet werden können. Nun soll überprüft werden, ob die berechneten Garantien in der Praxis eingehalten werden. Dazu wird das Cross-Traffic-Setup verwendet, welches zu Beginn des Kapitels vorgestellt wurde. Zunächst wird für jeden Router die maximal mögliche Anzahl der zugelassenen Ströme M berechnet. Anschließend werden für verschiedene Token-Bucket-Spezifikationen die garantierten maximalen Ende-zu-Ende-Verzögerungen berechnet. Schließlich wird das Netzwerk mit den jeweiligen Token-Bucket-Spezifikationen und variierendem Cross-Traffic simuliert.

Berechnung von M Zur Berechnung der maximal möglichen Anzahl der zugelassenen Ströme M genügt es, den Flaschenhals R_1 zu betrachten. Dabei gilt:

$$\begin{aligned} \frac{|W_0|}{C_0^{max}} &= 10000 \\ \frac{mr_0 \cdot T}{C_0^{max}} &= 120 \\ \frac{|W_1|}{C_1^{max}} &= 1000 \\ \frac{4 \cdot h_0 \cdot mr_1 \cdot T}{C_1^{max}} &= 11,52 \\ \Rightarrow M &= 11 \end{aligned} \tag{6.1}$$

Es können somit bei R_1 maximal 11 Ströme zugelassen werden, damit die garantierten Verzögerungen für Priorität 0 und 1 erhalten bleiben. Theoretisch könnte R_2 mehr Ströme aufnehmen, da er über eine höhere Ausgangsbandbreite verfügt. Für die gegebene Topologie ist allerdings die

maximale Anzahl der Ströme durch den Flaschenhals begrenzt, weshalb auch für R_2 angenommen wird, dass $M = 11$ gilt.

Berechnung der maximalen Verzögerung Für F_1 werden die folgenden Spezifikationen (r_1, b_1) betrachtet:

- $Spez_0$: (50 kbit/s, 50 kbit) im Konstant-, Poisson- und Max-Burst-Modus.
- $Spez_1$: (500 kbit/s, 500 kbit) im Konstant-, Poisson- und Max-Burst-Modus.
- $Spez_2$: (5000 kbit/s, 5000 kbit) im Konstant-, Poisson- und Max-Burst-Modus.

Mit der Spezifikation $Spez_0$ erhält F_1 die Priorität 0 bei beiden Routern, unabhängig vom verwendeten Sendemodus. Spezifikation $Spez_1$ erhält ebenfalls die Priorität 0, solange innerhalb von T keine Bursts größer als 100 kbit gesendet werden. Dies ist im Konstant-Modus sichergestellt, in den anderen beiden Modi können allerdings auch größere Bursts bis zu 500 kbit gesendet werden. Diese Bursts fallen in die Priorität 1. Für die Priorität 0 kann die garantierte maximale Verzögerung wie folgt berechnet werden:

$$\begin{aligned} d^{max}_{Prio_0} &= d^{max}_{TokenBucket} + d^A + d^V + d^{max}_{R_1} + d^A + d^V + d^{max}_{R_2} + d^A \\ &= \frac{b_1}{R^{in}_1} + \frac{M \cdot C^{max}_0}{mr_{0,R_1}} + \frac{M \cdot C^{max}_0}{mr_{0,R_2}} + 3 \cdot d^A + 2 \cdot d^V \\ &= \frac{100 \cdot 10^3\ Bit}{1 \cdot 10^9\ Bit/s} + \frac{11 \cdot 100 \cdot 10^3\ Bit}{0,6 \cdot 200 \cdot 10^6\ Bit/s} + \frac{11 \cdot 100 \cdot 10^3\ Bit}{0,6 \cdot 1 \cdot 10^9\ Bit/s} + 0,25\ ms \\ &= 0,1\ ms + 9,1\overline{6}\ ms + 1,8\overline{3}\ ms + 0,25\ ms \\ &= 11,35\ ms \end{aligned} \tag{6.2}$$

Für die Priorität 1 sind Bursts mit bis zu 1000 kbit innerhalb von T möglich. Diese Priorität erhält $Spez_1$ im Poisson- und Max-Burst-Modus sowie die Spezifikation $Spez_2$ (5000 kbit/s, 5000 kbit) im Konstant-Modus. Die maximale Verzögerung für Priorität 1 kann wie folgt berechnet werden:

$$\begin{aligned} d^{max}_{Prio_1} &= d^{max}_{TokenBucket} + d^A + d^V + d^{max}_{R_1} + d^A + d^V + d^{max}_{R_2} + d^A \\ &= \frac{b_1}{R^{in}_1} + \frac{M \cdot C^{max}_1}{4 \cdot h_0 \cdot mr_{1,R_1}} + \frac{M \cdot C^{max}_1}{4 \cdot h_0 \cdot mr_{1,R_2}} + 3 \cdot d^A + 2 \cdot d^V \\ &= \frac{1 \cdot 10^6\ Bit}{1 \cdot 10^9\ Bit/s} + \frac{11 \cdot 1 \cdot 10^6\ Bit}{4 \cdot 0,6 \cdot 0,4 \cdot 0,6 \cdot 200 \cdot 10^6\ Bit/s} + \frac{11 \cdot 1 \cdot 10^6\ Bit}{4 \cdot 0,6 \cdot 0,4 \cdot 0,6 \cdot 1 \cdot 10^9\ Bit/s} \\ &\quad + 0,25\ ms \\ &= 1\ ms + 95,49\ ms + 19,1\ ms + 0,25\ ms \\ &= 115,84\ ms \end{aligned} \tag{6.3}$$

Der Poisson- und Max-Burst-Modus von $Spez_2$ fällt aufgrund der möglichen Bursts von 5000 kbit in Priorität 2. Hier kann es bei Überlast zu Paketverlusten kommen, wenn die Warteschlangen überlaufen.

Die verwendeten Cross-Traffic-Szenarien Für den Cross-Traffic werden folgende Szenarien untersucht, bei welchen alle Cross-Traffic-Sender im Konstant-Modus senden, um ihre Bandbreite vollständig auszunutzen:

- **M0:** $M-1$ Sender mit $r = 980$ kbit/s, sodass sie dauerhaft in Priorität 0 eingeordnet werden. Theoretisch wäre $r = 1$ Mbit/s möglich, durch die randomisierten Sendeereignisse würde es jedoch vereinzelt zu Überschreitungen des Schwellwerts und einer Abstufung in Priorität 1 kommen.
- **M1:** $M-1$ Sender mit $r = 9,8$ Mbit/s, sodass sie dauerhaft in Priorität 1 eingeordnet werden.
- **M01:** $M-1$ Sender, die zusammen mit F_1 die Belegung im schlechtesten Fall ergeben. Dabei befinden sich $\frac{M}{2 \cdot h_0}$ (≈ 9) Ströme in Priorität 1 und die restlichen Ströme in Priorität 0. Außerdem befindet sich ein zusätzlicher Sender mit $r = 200$ Mbit/s in Priorität 2, um die Verwendung der Platzhalter-Pakete in allen Prioritäten notwendig zu machen und eine Überlast zu simulieren. Damit soll gezeigt werden, dass die Garantien der Priorität 0 und 1 auch bei der schlechtesten Belegung und Überlast erhalten bleiben.

Messergebnisse Für jede Kombination aus Spezifikation ($Spez_0, Spez_1, Spez_2$), Sende-Modus (Konstant, Poisson, Max-Burst) und Cross-Traffic-Szenario wurde eine separate Simulation durchgeführt. Bei allen Simulationen betrug die simulierte Zeit 30 Minuten (1800 s). Die durchschnittliche Paketgröße betrug bei den Sende-Modi mit zufälliger Paketgröße 5500 Bits. Für $Spez_0$ kann die Anzahl der übermittelten Pakete N vom untersuchen Sender daher wie folgt berechnet werden:

$$N = \frac{1800\ s \cdot 50000\ Bit/s}{5500\ Bit} \approx 16363\ Pakete \tag{6.4}$$

Durch die höhere Token-Rate von $Spez_1$ wurden damit ca. 163630 Pakete übermittelt und mit $Spez_2$ ca. 1,6 Millionen Pakete. Im Konstant-Modus betrug die Paketgröße 1000 Bits, wodurch eine entsprechend größere Anzahl an Paketen verschickt wurde. Da die Verzögerung pro Paket gemessen wurde, entspricht N der Anzahl der Messwerte.
Um die Ergebnisse aller Kombinationen möglichst übersichtlich darstellen zu können, wird für jede Kombination nur die maximale gemessene Verzögerung sowie das Minimum tabellarisch dargestellt. Außerdem wurden gemäß der Arbeit von Le Boudec [Le 10] die beiden Grenzen des 99% Confidence-Intervalls des Medians bestimmt. Der Wert an der unteren Grenze des Confidence-Intervalls wird Median u. bezeichnet, der Wert an der oberen Grenze entsprechend Median o. Es wurde eine separate Tabelle pro Cross-Traffic-Szenario erstellt. Die folgenden Tabellen 6.1, 6.2 und 6.3 zeigen die Ergebnisse der Cross-Traffic-Szenarion M0, M1 und M01.
Da bei der Messung von Szenario M01 mit $Spez_2$ im Max-Burst-Modus Pakete verloren gegangen sind, wurden die maximale Verzögerung und der Median nur anhand der zugestellten Pakete berechnet (durch * markiert).

Tabelle 6.1: Die Messergebnisse des Szenarios M0 zur Evaluierung der garantierten maximalen Verzögerung. Alle Cross-Traffic-Ströme befinden sich dabei in Priorität 0.

Sender-Spezifikation	Berech. Maximum	Maximum	Minimum	Median u.	Median o.	Verlust
$Spez_0$, Konstant	11,35 ms	0,28 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_0$, Poisson	11,35 ms	0,34 ms	0,26 ms	0,29 ms	0,29 ms	0 Bit
$Spez_0$, Max-Burst	11,35 ms	0,49 ms	0,26 ms	0,40 ms	0,40 ms	0 Bit
$Spez_1$, Konstant	11,35 ms	0,28 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_1$, Poisson	115,84 ms	0,37 ms	0,26 ms	0,29 ms	0,29 ms	0 Bit
$Spez_1$, Max-Burst	115,84 ms	2,44 ms	0,26 ms	1,36 ms	1,37 ms	0 Bit
$Spez_2$, Konstant	115,84 ms	0,30 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_2$, Poisson	-	0,39 ms	0,26 ms	0,29 ms	0,29 ms	0 Bit
$Spez_2$, Max-Burst	-	21,73 ms	0,26 ms	10,97 ms	11,01 ms	0 Bit

Tabelle 6.2: Die Messergebnisse des Szenarios M1 zur Evaluierung der garantierten maximalen Verzögerung. Alle Cross-Traffic-Ströme befinden sich dabei in Priorität 1.

Sender-Spezifikation	Berech. Maximum	Maximum	Minimum	Median u.	Median o.	Verlust
$Spez_0$, Konstant	11,35 ms	0,36 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_0$, Poisson	11,35 ms	0,41 ms	0,26 ms	0,30 ms	0,30 ms	0 Bit
$Spez_0$, Max-Burst	11,35 ms	0,60 ms	0,26 ms	0,41 ms	0,42 ms	0 Bit
$Spez_1$, Konstant	11,35 ms	0,43 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_1$, Poisson	115,84 ms	0,73 ms	0,26 ms	0,30 ms	0,30 ms	0 Bit
$Spez_1$, Max-Burst	115,84 ms	2,99 ms	0,26 ms	1,45 ms	1,46 ms	0 Bit
$Spez_2$, Konstant	115,84 ms	0,83 ms	0,26 ms	0,26 ms	0,26 ms	0 Bit
$Spez_2$, Poisson	-	0,88 ms	0,26 ms	0,31 ms	0,31 ms	0 Bit
$Spez_2$, Max-Burst	-	45,18 ms	0,26 ms	22,37 ms	22,46 ms	0 Bit

Bei allen durchgeführten Messungen wurden die berechneten Garantien eingehalten. Selbst bei einer Überlastsituation wurden die maximalen Verzögerungen nicht überschritten und es gingen keine Pakete in den Prioritäten 0 und 1 verloren. Die höchsten Verzögerungen sind wie erwartet mit dem Max-Burst-Modus entstanden, besonders deutlich ist der Unterschied zum Konstant- und Poisson-Modus bei den Spezifikationen $Spez_1$ und $Spez_2$. Bei $Spez_0$ fallen die Unterschiede nicht gravierend aus, da die Senderate und Burstgröße sehr gering sind.
An den Messergebnissen von Szenario M0 und M1 (Tabelle 6.1 und 6.2) wird deutlich, dass die Verzögerungen für alle Ströme geringer ausfallen, wenn sich alle Ströme an die Spezifikation der hohen Prioritäten halten. Im Vergleich dazu fallen die Verzögerungen höher aus, wenn sich nicht-konforme Sender in Priorität 2 befinden, wie es bei Szenario M01 der Fall ist. Die kurzen Verzögerungen ohne Sender in Priorität 2 können auf die Eigenschaften der Ströme in Priorität 0 und 1 zurückgeführt werden. Diese Ströme dürfen nur geringe Bursts senden um in den hohen Prioritäten zu bleiben, wodurch die Warteschlangen hier die meiste Zeit wenig gefüllt sind und kurze Wartezeiten entstehen. Die höheren Verzögerungen mit einem Sender in Priorität 2 können auf dessen Sendeverhalten und die abgegebene Bandbreite durch die Platzhalter-Pakete zurückgeführt werden. Ein Sender in Priorität 2 sendet typischerweise große Bursts, wodurch in Warteschlange 2

Tabelle 6.3: Die Messergebnisse des Szenarios M01 zur Evaluierung der garantierten maximalen Verzögerung. Mit dem Sender-Strom befinden sich 9 Ströme in Priorität 0 und 2 Ströme in Priorität 1. Außerdem befindet sich ein zusätzlicher Strom in Priorität 2, der das Netzwerk überlastet.

Sender-Spezifikation	Berech. Maximum	Maximum	Minimum	Median u.	Median o.	Verlust
$Spez_0$, Konstant	11,35 ms	3,40 ms	0,26 ms	0,99 ms	1,01 ms	0 Bit
$Spez_0$, Poisson	11,35 ms	3,41 ms	0,26 ms	1,01 ms	1,04 ms	0 Bit
$Spez_0$, Max-Burst	11,35 ms	3,39 ms	0,27 ms	1,11 ms	1,13 ms	0 Bit
$Spez_1$, Konstant	11,35 ms	4,30 ms	0,26 ms	1,00 ms	1,01 ms	0 Bit
$Spez_1$, Poisson	115,84 ms	12,14 ms	0,26 ms	1,02 ms	1,03 ms	0 Bit
$Spez_1$, Max-Burst	115,84 ms	29,40 ms	0,31 ms	6,78 ms	6,85 ms	0 Bit
$Spez_2$, Konstant	115,84 ms	4,65 ms	0,26 ms	1,19 ms	1,19 ms	0 Bit
$Spez_2$, Poisson	-	14,99 ms	0,27 ms	1,37 ms	1,38 ms	0 Bit
$Spez_2$, Max-Burst	-	∞ / 5,80* s	1,03 ms	5,51* s	5,51* s	6,96 GBit $\approx$ 77%

viele Pakete angestaut werden. Solange sich keine Pakete in den Warteschlangen niedriger Priorität befinden, werden keine Platzhalter-Pakete benötigt und die volle Bandbreite steht für die verwendete Priorität zur Verfügung. Befinden sich allerdings Pakete in einer niedrigeren Priorität, wie es bei einem Sender in Priorität 2 der Fall ist, wird ein prozentualer Anteil der Bandbreite durch die Platzhalter-Pakete abgegeben. Dadurch kommt es zu häufigeren und längeren Staus in den Warteschlangen der Priorität 0 und 1, wodurch die höheren Verzögerungen entstehen. Außerdem kommt es durch zusätzliche Ströme mit hohem Paketaufkommen häufiger zu Verzögerungen, die aufgrund des belegten Sendemediums entstehen. Da das Verfahren nicht-präemptiv arbeitet, müssen neu ankommende Pakete unabhängig von ihrer Priorität warten, bis eine bereits begonnen Übertragung abgeschlossen ist.
Selbst mit der Belegung im schlechtesten Fall und einem überlasteten Netzwerk wurden die berechneten Garantien deutlich unterschritten. Dies kann an den sehr konservativen Annahmen bei den Formeln zur Berechnung der Garantien liegen. Dort wurde beispielsweise angenommen, dass ein kompletter Burst eines Stroms ohne Verzögerung beim Router ankommt und sich komplett in der Warteschlange befindet (siehe Abschnitt Admission-Control in Kapitel 4). In der Praxis und im Simulator werden die Bursts in Form von einzelnen Paketen übertragen, welche verzögert beim Router ankommen.
Abschließend kann festgehalten werden, dass die garantierten Verzögerungen beim konzipierten Verfahren zuverlässig eingehalten werden. Bei den vorgestellten Szenarien wurden sie sogar zu mehr als 50% unterschritten.

6.5 Effiziente Nutzung der Ausgangsbandbreite

Beim entworfenen Verfahren soll überschüssige Bandbreite, die nicht von Sendern in den hohen Prioritäten benötigt wird, für den Best-Effort-Verkehr in Priorität 2 zur Verfügung stehen. Bei dieser Messung soll darum die effiziente Nutzung der Ausgangsbandbreite für Ströme in Priorität 2 evaluiert werden. Dazu werden die Warteschlangenfüllstände von Priorität 2 sowie der Paketverlust gemessen. Es wird das Cross-Traffic-Setup verwendet, welches zu Beginn des Kapitels vorgestellt wurde. Dabei

wird ein Cross-Traffic-Strom mit verschiedenen Sende-Modi und verschiedenen Spezifikationen verwendet, sodass er sich in Priorität 0 oder 1 befindet. Dazu werden für den Cross-Traffic-Sender die Spezifikationen der vorherigen Evaluierung erneut verwendet:

- $Spez_0$: (50 kbit/s, 50 kbit) im Konstant-Modus.
- $Spez_1$: (500 kbit/s, 500 kbit) im Max-Burst-Modus.

Der gemessene Strom F_1 verwendet einen Token-Bucket der Größe $b_1 = 200$ Mbit und arbeitet im Max-Burst-Modus. Dessen Token-Rate r_1 wird in Abhängigkeit des Cross-Traffic-Stroms spezifiziert, sodass die Bandbreite beider Ströme zusammen immer der Ausgangsbandbreite des Flaschenhalses (200 Mbit/s) entspricht:

$$r_1 = \begin{cases} 199950\text{kbit/s}, & \text{wenn Cross-Traffic mit } Spez_0 \\ 199500\text{kbit/s}, & \text{wenn Cross-Traffic mit } Spez_1 \end{cases} \tag{6.5}$$

Wenn die Bandbreite effizient genutzt wird, sollten somit wenige bzw. bestenfalls keine Pakete verloren gehen und die Warteschlangenfüllstände nicht kontinuierlich zunehmen oder überlaufen.

Messergebnisse Bei allen Simulationen betrug die simulierte Zeit 30 Minuten (1800 s). Die Ergebnisse werden tabellarisch in Tabelle 6.4 dargestellt.

Tabelle 6.4: Die Messergebnisse zur Evaluierung der effizienten Bandbreitennutzung. Bei jeder Kombination beträgt die eingehende Datenrate beim Flaschenhals dessen Ausgangsbandbreite (200 Mbit/s).

Cross-Traffic-Sender	max. Füllstand W_2	Paketverlust W_2
$Spez_0$, Konstant	199,98 Mbit	0 Bit
$Spez_0$, Poisson	199,99 Mbit	0 Bit
$Spez_0$, Max-Burst	200,01 Mbit	0 Bit
$Spez_1$, Konstant	199,97 Mbit	0 Bit
$Spez_1$, Poisson	200,02 Mbit	0 Bit
$Spez_1$, Max-Burst	200,34 Mbit	0 Bit

Der maximale Warteschlangenfüllstand betrug jeweils ca. 200 Mbit, was der Burst-Größe entspricht, die F_1 sendet. Dennoch kam es zu keinem Paketverlust. Dies ist nur möglich, wenn die Ausgangsbandbreite zu 100% genutzt wird. Denn nur dann können alle gepufferten Pakete weitergeleitet werden, bevor der nächste Burst gesendet wird. Andernfalls würde der Warteschlangenfüllstand kontinuierlich steigen und schließlich Pakete verloren gehen. Tatsächlich können alle Pakete weitergeleitet werden, und die Warteschlange leert sich nach den Bursts wieder, wie in Abbildung 6.6 visualisiert wird.

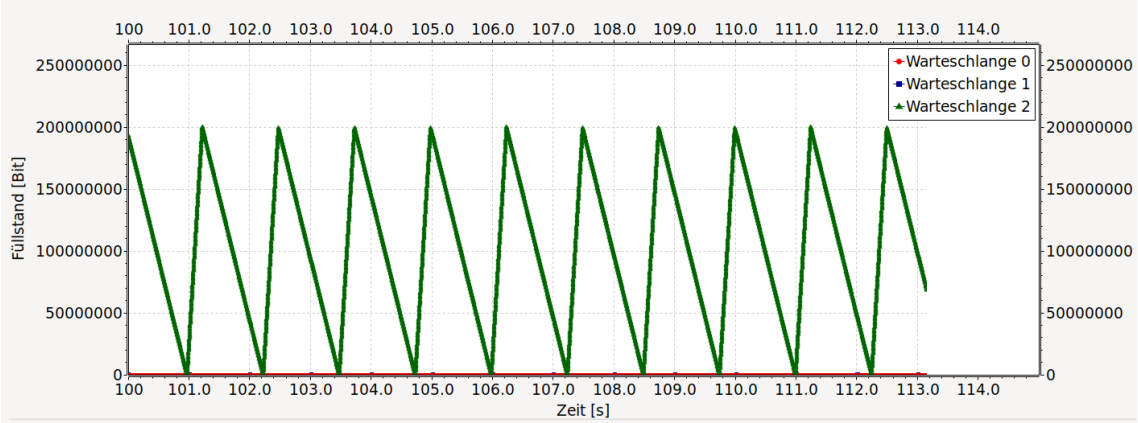

Abbildung 6.6: Für die Kombination $Spez_1$ mit Max-Burst-Modus wird ein Ausschnitt der Warteschlangenfüllstände bei R_1 visualisiert. Es wird der Zeitraum von 100 s bis 114 s abgebildet, das periodische Verhalten setzt sich über den kompletten Messzeitraum fort.

Es kann somit festgehalten werden, dass das Verfahren die verfügbare Bandbreite effizient für Ströme in Priorität 2 nutzbar macht. Es kann die Ausgangsbandbreite eines Router zu 100% genutzt werden.

7 Zusammenfassung & Ausblick

In dieser Arbeit wurden flexible Quality-of-Service-Modelle untersucht, die zusätzliche Abstufungen zwischen den bekannten beiden Klassen „Echtzeit“ und „Best-Effort“ ermöglichen. Diese abgestuften QoS-Klassen, bezeichnet als Performance-Klassen, können beispielsweise die Regelgüte von vernetzten Regelungssystemen verbessern, ohne teure Reservierungen für Echtzeitgarantien zu belegen.
Bei diesem flexiblen QoS-Modell werden die einzelnen Ströme dynamisch in eine Performance-Klasse eingeordnet, abhängig von ihrem Sendeverhalten. Dabei repräsentieren die Performance-Klassen eine Sendepriorität, die mit jeder Klasse abnimmt. Je höher die Performance-Klasse eines Stroms ist, desto besser ist die Service-Qualität für ihn. Diese Qualität reicht von garantierten niedrigen Verzögerungen in den hohen Klassen bis zu Best-Effort-Service ohne Garantien in der niedrigsten Performance-Klasse. Die Zuordnung in die einzelnen Klassen wird dynamisch an das Sendeverhalten angepasst, sodass sich die Priorität der Ströme verbessern und verschlechtern kann. Es ist für die Anwendung berechenbar, in welche Performance-Klasse ihr Strom fällt. Dadurch wird ihr ein Anreiz zur Selbstregulierung ihres Sendeverhaltens gegeben. Zugleich profitiert das Netzwerk von den kooperativen Strömen, da sie einen geringeren Ressourcenverbrauch und eine bessere Ressourcennutzung ermöglichen. Anwendungen, die sich aus Sicht des Netzwerks kooperativ verhalten und wenige, kleine Bursts senden, werden mit einer hohen Performance-Klasse belohnt. Darin erhalten sie eine hohen Priorität und einen besseren Quality-of-Service.
Um ein flexibles QoS-Modell zu entwerfen, wurde zunächst das bekannte IntServ-Modell mit Weighted Fair Queueing untersucht und Formeln zur Berechnung der maximalen Ende-zu-Ende-Verzögerung in Abhängigkeit von der Burst-Größe aufgestellt. Es konnte dabei gezeigt werden, dass die Verzögerung maßgeblich von der Burst-Größe und der reservierten Bandbreite am Flaschenhals des Kommunikationspfades abhängig ist. Insbesondere bedeutet dies, dass IntServ nicht nur die maximale Latenz eines Bursts maximaler Größe beschränkt. Auch kleinere Bursts, welche kleiner als die Kapazität des Token-Buckets sind, werden mit Latenzgarantien transportiert. Diese Garantien sind umso besser, je kleiner der Burst ist.
Anschließend wurde eine Modifikation für IntServ vorgestellt, mit welcher zusätzliche Performance-Klassen realisiert werden können, für die noch immer maximale Verzögerungen definiert sind. Ein allgemeiner Nachteil von IntServ ist allerdings die bekanntlich schlechte Skalierbarkeit, da bei den Routern pro Strom eine separate Warteschlange benötigt wird.
Deshalb wurde ein neues flexibles Scheduling-Verfahren entworfen, das zusätzliche Performance-Klassen unterstützt und keine separaten Warteschlangen pro Strom benötigt. Stattdessen wird nur eine feste Anzahl von Warteschlangen benötigt (eine pro Performance-Klasse). Dadurch ist es insbesondere besser skalierbar als IntServ. Das Verfahren basiert auf dem Multi-Level-Feedback-Queue-Scheduling-Algorithmus, welcher vom Prozess-Scheduling bekannt ist. Für jede Performance-Klasse wird eine Warteschlange des MLFQ-Verfahrens verwendet. Das Verfahren unterscheidet sich vom klassischen MLFQ-Verfahren in der Möglichkeit, dass Ströme nach einer gewissen Aufstiegsverzögerung (Bewährungszeit) wieder in der Priorität bzw. Warteschlange aufsteigen können. Außerdem wurde eine Erweiterung vorgestellt, die das Aushungern der niedrigen

Prioritäten verhindert. Dazu werden sogenannte Platzhalter-Pakete in die Warteschlangen eingefügt, die stellvertretend für Pakete in der darunterliegenden Warteschlange Bandbreite reservieren. Des weiteren wurde ein Admission-Control-Verfahren vorgestellt, mit dem die Überlastung des Netzwerks und die Verletzung von QoS-Garantien verhindert werden kann.
Zur Demonstration und Evaluierung wurde das konzipierte Modell für einen Netzwerksimulator implementiert. Damit konnte die Effektivität des Verfahrens gezeigt werden. Es wurde die dynamische Zuordnung der Priorität überprüft und zwei verschiedene Varianten der Aufstiegsverzögerung evaluiert. Außerdem wurden die berechneten garantierten maximalen Verzögerungen geprüft, indem verschiedene Last- und Sendeverhalten bei variierenden Cross-Traffic-Szenarien simuliert und mit den berechneten garantierten Verzögerungen verglichen wurden. Dadurch konnte gezeigt werden, dass die berechneten Garantien eingehalten werden und in den meisten Szenarien die Verzögerung sogar um bis zu 50% geringer ausfällt. Des Weiteren konnte die effiziente Bandbreitennutzung des Verfahrens verdeutlicht werden. Da es sich beim MLFQ-Verfahren um ein sogenanntes „work conserving“ Scheduling-Verfahren handelt, werden bei vorhandenem Verkehr dabei immer Pakete weitergeleitet. Dies ist ein Vorteil im Vergleich zu einem Zeitslot basierten Verfahren, welches nicht work conserving arbeitet, sondern eine feste Zuordnung von Strömen zu Zeitslots vornimmt.
Das vorgestellte Verfahren lieferte in den Simulationen vielversprechende Ergebnisse. Dabei wurden verschiedene Arten von Sendeverhalten untersucht, um mögliche Anwendungen zu modellieren. Ein weiterer Schritt kann die tatsächliche Integration des Ansatzes mit einer Regelungsanwendung sein. Damit könnten die Auswirkungen auf die anwendungsspezifische Metrik der Regelgüte evaluiert werden.
Außerdem verwendet das konzipierte Modell ein Admission-Control-Verfahren, das im Voraus die maximal mögliche Anzahl der zugelassenen Ströme berechnet. Dadurch können deterministische Garantien für die maximale Verzögerung gegeben werden. Ein möglicher Ansatz, der in Zukunft untersucht werden kann, ist die Verwendung eines probabilistischen Ansatzes. Damit könnte die Anzahl der Ströme weiter erhöht werden. Beispielsweise könnte anhand der Warteschlangenfüllstände die Entscheidung getroffen werden, ob zusätzliche Ströme zugelassen werden.

Literaturverzeichnis

[ATMT04] M. Angulo, D. Torres-Roman, D. Muñoz-Rodriguez, M. Turrubiartes. „IP networks Quality of Service: Overview and open issues“. In: *International Workshop on Innovative Internet Community Systems*. Springer. 2004, S. 28–37 (zitiert auf S. 20).

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. *RFC 2475: An architecture for differentiated services*. 1998 (zitiert auf S. 24).

[BCS94] R. Braden, D. Clark, S. Shenker. *RFC1633: Integrated services in the internet architecture: an overview*. 1994 (zitiert auf S. 16, 17, 23).

[BDHL16] A. Bechtolsheim, L. Dale, H. Holbrook, A. Li. „Why big data needs big buffer switches“. In: *Arista White Paper* 10 (2016) (zitiert auf S. 63).

[Bre98] E. C. Brewer. *”Round RobinDictionary of Phrase and Fable*. Philadelphia: Henry Altemus, Bartleby.com, 1898. URL: https://www.bartleby.com/81/14554.html (zitiert auf S. 19).

[BS98] L. Breslau, S. Shenker. „Best-effort versus reservations: A simple comparative analysis“. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'98)*. 1998, S. 3–16 (zitiert auf S. 23).

[BZ96] J. C. Bennett, H. Zhang. „WF2Q: worst-case fair weighted fair queueing“. In: *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'96)*. Bd. 1. IEEE. 1996, S. 120–128 (zitiert auf S. 20).

[CMD62] F. J. Corbató, M. Merwin-Daggett, R. C. Daley. „An experimental time-sharing system“. In: *Proceedings of the Joint Computer Conference May 1-3*. 1962, S. 335–344 (zitiert auf S. 21).

[DCB+02] B. Davie, A. Charny, J. C. Bennett, K. Benson, J.-Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis. *RFC 3246: An expedited forwarding PHB (per-hop behavior)*. 2002 (zitiert auf S. 24).

[Den68] P. J. Denning. „Thrashing: Its causes and prevention“. In: *Proceedings of the Joint Computer Conference Part I December 9-11*. 1968, S. 915–922 (zitiert auf S. 60).

[DIN19] DIN-EN-60204-1. *Sicherheit von Maschinen - Elektrische Ausrüstung von Maschinen - Teil 1: Allgemeine Anforderungen (IEC 60204-1:2016, modifiziert); Deutsche Fassung EN 60204-1:2018*. Norm. Juni 2019 (zitiert auf S. 15).

[DK+14] F. Dürr, T. Kohler et al. „Comparing the forwarding latency of OpenFlow hardware and software switches“. In: *Fakultät Informatik, Elektrotechnik Informationstechnik, Univ. Stuttgart, Stuttgart, Germany, Tech. Rep. TR* 4 (2014), S. 2014 (zitiert auf S. 75).

[DKS89] A. Demers, S. Keshav, S. Shenker. „Analysis and simulation of a fair queueing algorithm“. In: *ACM SIGCOMM Computer Communication Review* 19.4 (1989), S. 1–12 (zitiert auf S. 17, 20).

[DN16] F. Dürr, N. G. Nayak. „No-wait packet scheduling for IEEE time-sensitive networks (TSN)“. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 2016, S. 203–212 (zitiert auf S. 15, 25, 26).

[Edg20] Edge-Core. *Edge-Core Wedge100BF-32QS Data Center Switch Datasheet*. 2020. URL: https://www.edge-core.com/_upload/images/Wedge_100BF-32QS_DS_R01_20200428.pdf (zitiert auf S. 63).

[FHC+19] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, K. Rothermel. „NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++“. In: *Proceedings of the 2019 International Conference on Networked Systems (NetSys)*. Garching b. München, Germany, März 2019 (zitiert auf S. 17, 27).

[FJ93] S. Floyd, V. Jacobson. „Random early detection gateways for congestion avoidance“. In: *IEEE/ACM Transactions on networking* 1.4 (1993), S. 397–413 (zitiert auf S. 16).

[HBWW99] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski. *RFC 2597: Assured forwarding PHB group*. 1999 (zitiert auf S. 24).

[Hen98] R. Hendrickson. *QPB Encyclopedia of Word and Phrase Origins*. Facts on File, 1998, S. 580. ISBN: 9780965379458. URL: https://books.google.de/books?id=BgtZAAAAYAAJ (zitiert auf S. 19).

[Heu13] J. Heuer. *Das Multiprocessor Scheduling-Problem mit reihenfolgeabhängigen Rüstzeiten: Heuristische Lösungsverfahren*. Gabler Edition Wissenschaft. Deutscher Universitätsverlag, 2013, S. 4. ISBN: 9783322818867. URL: https://books.google.de/books?id=esj1BQAAQBAJ (zitiert auf S. 19).

[HG99] J. Heinanen, R. Guérin. *RFC 2698: A two rate three color marker*. 1999 (zitiert auf S. 24).

[HK00] J. Harju, P. Kivimaki. „Co-operation and comparison of DiffServ and IntServ: performance measurements“. In: *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks (LCN 2000)*. IEEE. 2000, S. 177–186 (zitiert auf S. 17, 24).

[HT11] A. Haseeb, V. Tralli. „QoS performance analysis for VoIP traffic in heterogeneous networks with WiMAX access“. In: *Proceedings of the 13th International Conference on Advanced Communication Technology (ICACT2011)*. IEEE. 2011, S. 960–965 (zitiert auf S. 20).

[IEE16a] IEEE. „IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic“. In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)* (2016), S. 1–57. DOI: 10.1109/IEEESTD.2016.8613095 (zitiert auf S. 16, 25).

[IEE16b] IEEE. „IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption“. In: *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)* (2016), S. 1–52. DOI: 10.1109/IEEESTD.2016.7553415 (zitiert auf S. 27).

[IEE18a] IEEE. „IEEE Standard for Ethernet“. In: *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)* (2018), S. 1–5600. DOI: 10.1109/IEEESTD.2018.8457469 (zitiert auf S. 15).

[IEE18b] IEEE. „IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area network–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Preassociation Discovery“. In: *IEEE Std 802.11aq-2018 (Amendment to IEEE Std 802.11-2016 as amended by IEEE Std 802.11ai-2016, IEEE Std 802.11ah-2016, IEEE Std 802.11aj-2018, and IEEE Std 802.11ak-2018)* (2018), S. 1–69. DOI: 10.1109/IEEESTD.2018.8457463 (zitiert auf S. 15).

[IEE20a] IEEE. „IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems“. In: *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)* (2020), S. 1–499. DOI: 10.1109/IEEESTD.2020.9120376 (zitiert auf S. 25).

[IEE20b] IEEE. „IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34:Asynchronous Traffic Shaping“. In: *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)* (2020), S. 1–151. DOI: 10.1109/IEEESTD.2020.9253013 (zitiert auf S. 27).

[IIHA12] M. Z. Islam, M. S. Islam, A. F. Haque, M. Ahmed. „A comparative analysis of different real time applications over various queuing techniques“. In: *Proceedings of the International Conference on Informatics, Electronics and Vision (ICIEV)*. 2012, S. 1118–1123. DOI: 10.1109/ICIEV.2012.6317525 (zitiert auf S. 20).

[Jac88] V. Jacobson. „Congestion avoidance and control“. In: *ACM SIGCOMM Computer Communication Review* 18.4 (1988), S. 314–329 (zitiert auf S. 37).

[Jun20] Juniper. *Juniper Networks PTX1000 Router Datasheet*. 2020. URL: https://www.juniper.net/us/en/products/routers/ptx-series/ptx1000-ptx10001-ptx10002-ptx10003-fixed-configuration-packet-transport-routers-datasheet.html (zitiert auf S. 64).

[LCD+19] S. Linsenmayer, B. W. Carabelli, F. Dürr, J. Falk, F. Allgöwer, K. Rothermel. „Integration of Communication Networks and Control Systems Using a Slotted Transmission Classification Model“. In: *Proceedings of the 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2019, S. 1–6. DOI: 10.1109/CCNC.2019.8651811 (zitiert auf S. 16, 27).

[Le 10] J.-Y. Le Boudec. *Performance evaluation of computer and communication systems*. Bd. 2. 2010 (zitiert auf S. 83).

[ML89] R. A. Milito, H. Levy. „Modeling and dynamic scheduling of a queueing system with blocking and starvation“. In: *IEEE transactions on communications* 37.12 (1989), S. 1318–1329 (zitiert auf S. 20).

[MP02] A. Mascis, D. Pacciarelli. „Job-shop scheduling with blocking and no-wait constraints“. In: *European Journal of Operational Research* 143.3 (2002), S. 498–517 (zitiert auf S. 26).

[MS99] I. Mahadevan, K. M. Sivalingam. „Quality of service architectures for wireless networks: IntServ and DiffServ models“. In: *Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*. IEEE. 1999, S. 420–425 (zitiert auf S. 23).

[Nag87] J. Nagle. „On packet switches with infinite storage“. In: *IEEE transactions on communications* 35.4 (1987), S. 435–438 (zitiert auf S. 19).

[NBBB98] K. Nichols, S. Blake, F. Baker, D. Black. *RFC 2474: Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers*. 1998 (zitiert auf S. 24).

[Oku15] A. M. Okun. *Equality and efficiency: The big tradeoff*. Brookings Institution Press, 2015 (zitiert auf S. 22).

[Pin12] M. Pinedo. *Scheduling*. Bd. 29. Springer, 2012 (zitiert auf S. 19).

[RAL04] D. Raz, B. Avi-Itzhak, H. Levy. „Classes, priorities and fairness in queueing systems“. In: *RUTCOR, Rutgers University, Tech. Rep. RRR-21-2004* (2004) (zitiert auf S. 20).

[Sch04] E. Schemm. „SERCOS to link with ethernet for its third generation“. In: *Computing and Control Engineering* 15.2 (2004), S. 30–33 (zitiert auf S. 25).

[Sch13] C. Schneeweiss. *Einführung in die Produktionswirtschaft*. Heidelberger Taschenbücher. Springer Berlin Heidelberg, 2013, S. 244. ISBN: 9783662068748. URL: https://books.google.de/books?id=sQ%5C_RBgAAQBAJ (zitiert auf S. 19).

[SDT+17] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, G. Mühl. „ILP-Based Joint Routing and Scheduling for Time-Triggered Networks“. In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. RTNS '17. Grenoble, France: Association for Computing Machinery, 2017, S. 8–17. ISBN: 9781450352864. DOI: 10.1145/3139258.3139289. URL: https://doi.org/10.1145/3139258.3139289 (zitiert auf S. 26).

[SS16] J. Specht, S. Samii. „Urgency-based scheduler for time-sensitive switched ethernet networks“. In: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE. 2016, S. 75–85 (zitiert auf S. 27).

[SV96] M. Shreedhar, G. Varghese. „Efficient fair queuing using deficit round-robin“. In: *IEEE/ACM Transactions on networking* 4.3 (1996), S. 375–385 (zitiert auf S. 19).

[SW97] S. Shenker, J. Wroclawski. *RFC 2215: General characterization parameters for integrated service network elements*. 1997 (zitiert auf S. 17, 22).

[TT99] P. P. Tang, T.-Y. Tai. „Network traffic characterization using token bucket model“. In: *Proceedings of the 18th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99). The Future is Now (Cat. No. 99CH36320)*. Bd. 1. IEEE. 1999, S. 51–62 (zitiert auf S. 23).

[Tur86] J. Turner. „New directions in communications (or which way to the information age?)“ In: *IEEE communications Magazine* 24.10 (1986), S. 8–15 (zitiert auf S. 22).

[TV99] E. Tovar, F. Vasques. „Real-time fieldbus communications using Profibus networks“. In: *IEEE transactions on Industrial Electronics* 46.6 (1999), S. 1241–1251 (zitiert auf S. 25).

[VH08] A. Varga, R. Hornig. „An overview of the OMNeT++ simulation environment“. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. 2008, S. 1–10 (zitiert auf S. 17, 27, 71).

[WM03] M. Welzl, M. Muhlhauser. „Scalability and quality of service: a trade-off?“ In: *IEEE Communications Magazine* 41.6 (2003), S. 32–36 (zitiert auf S. 17, 24).

[Wol20] L. A. Wolsey. *Integer programming*. John Wiley & Sons, 2020 (zitiert auf S. 26).

[Wro+97] J. Wroclawski et al. *RFC 2210: The use of RSVP with IETF integrated services*. 1997 (zitiert auf S. 23).

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala. „RSVP: A new resource reservation protocol“. In: *IEEE network* 7.5 (1993), S. 8–18 (zitiert auf S. 16, 23).

Alle URLs wurden zuletzt am 03. 11. 2021 geprüft.

Erklärung

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

Ort, Datum, Unterschrift