Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Simon Sorgenfrei

Evaluation der WebRTC Congestion Control SCReAM

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Simon Sorgenfrei

Evaluation der WebRTC Congestion Control SCReAM

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr.-Ing Martin Hübner

Eingereicht am: 22. November 2017

Simon Sorgenfrei

Thema der Arbeit
Evaluation der WebRTC Congestion Control SCReAM

Stichworte
SCReAM, Congestion Control, WebRTC, OMNet++, INET

Kurzzusammenfassung
Diese Bachelorarbeit befasst sich mit der Evaluation der WebRTC *Congestion Control* SCReAM. Für diesen Zweck wurde die Referenzimplementierung in das INET Framework des diskreten Eventsimulators OMNet++ integriert. Mit OMNet++ als vollständig kontrollierbare Simulationsumgebung, wurde eine Auswahl, aus den der *IETF RTP Media Congestion Avoidance Techniques* (RMCAT WG) definierten Evaluationstests, die die Basisfunktionalitäten einer WebRTC *Congestion Control* abdecken, durchgeführt. Im Anschluss erfolgt eine Vorstellung der eigenen Messergebnisse und eine Gegenüberstellung mit den Messergebnissen aus der Evaluation durch Ericsson Research, mit anschließender Diskussion.
Simon Sorgenfrei

Title of the paper
Evaluation of the WebRTC Congestion Control SCReAM

Keywords
SCReAM, Congestion Control, WebRTC, OMNet++, INET

Abstract
This bachelor thesis deals with the evaluation of the WebRTC Congestion Control SCReAM. For this purpose, the reference implementation has been integrated into the INET framework of the discrete event simulator OMNet++. A suitable selection was made from the IETF RTP Media Congestion Avoidance Techniques (RMCAT WG) defined evaluation tests that cover the basic functionalities of a WebRTC Congestion Control. With OMNet++ as a fully controllable simulation environment, the test cases were performed. Subsequently, the own measurement results were presented and compared with the measurement results from the evaluation by Ericsson Research, followed by a discussion.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Relevanz des Themas . . . 1
- 1.2 Problemstellung . . . 2
- 1.3 Anforderungen . . . 7
- 1.4 Vorgehensweise . . . 8

2 WebRTC **9**
- 2.1 Einleitung . . . 9
- 2.2 Entwicklung . . . 10
- 2.3 Audio- und Video-Engines . . . 10
- 2.4 WebRTC Protokolle . . . 11

3 SCReAM **18**
- 3.1 Einleitung . . . 18
- 3.2 Übersicht über das SCReAM Framework . . . 18
- 3.3 Beschreibung der Hauptkomponenten . . . 22
  - 3.3.1 RTP Queue . . . 22
  - 3.3.2 Congestion Control . . . 22
  - 3.3.3 Transmission Scheduler . . . 28
  - 3.3.4 Media Rate Control . . . 29

4 Evaluationsplattform **30**
- 4.1 OMNeT++/INET . . . 30
- 4.2 Evaluationseinstellungen . . . 31
  - 4.2.1 Metriken . . . 31
  - 4.2.2 Pfad Eigenschaften . . . 31
- 4.3 Struktur der Experimente . . . 33
- 4.4 Messung und Visualisierung . . . 34
- 4.5 Integration ins Simulationsmodell . . . 36
  - 4.5.1 Verwendete Codecs . . . 36
  - 4.5.2 Validierung . . . 37

5 Experimente **47**
- 5.1 Variable Pfadkapazität mit einem SCReAM-Strom . . . 47
- 5.2 Variable Pfadkapazität mit zwei SCReAM-Strömen . . . 52
- 5.3 Überlast im Feedback-Pfad . . . 56
- 5.4 Konkurrierende SCReAM-Ströme . . . 62

5.5 Konkurrierende SCReAM-Ströme mit verschiedenen RTTs 67
5.6 SCReAM-Strom konkurriert mit einem langlebigen TCP-Strom 70
5.7 Zwei SCReAM-Ströme konkurrieren mit mehreren kurzlebigen TCP-Strömen 75

6 Diskussion **80**

7 Fazit **85**

Tabellenverzeichnis

4.1 Ablauf: Variable Pfadkapazität mit einem SCReAM-Strom 39
4.2 Ablauf: Variable Pfadkapazität mit zwei SCReAM-Strömen 43

5.1 Ablauf: Variable Pfadkapazität in Pfadrichtung *Vorwärts* bei Überlast im Feedback-Pfad 57
5.2 Ablauf: Variable Pfadkapazität in Pfadrichtung *Rückwärts* bei Überlast im Feedback-Pfad 57

Abbildungsverzeichnis

1.1 Übertragung von WebRTC Medien . . . 4
1.2 Übertragung von WebRTC Medien und Daten über den gleichen Endpunkt . . 5
1.3 Konkurrierende TCP Verbindung . . . 5
1.4 Illustration: Medienstrom verliert gegen einen TCP-Strom . . . 6

2.1 WebRTC Audio- und Videoengines . . . 11
2.2 WebRTC Protokolle . . . 12

3.1 Illustration: SCReAM Sender Übersicht . . . 21
3.2 Beispiel: Berechnung der *bytes-in-flight* . . . 23
3.3 Beispiel: Messung *One-Way Delay* . . . 24
3.4 Aktivitätsdiagramm: *Congestion Window* Berechnung . . . 27

4.1 Basis Topologie . . . 32
4.2 Testbed-Topologie für variable Pfadkapazität mit einem SCReAM-Strom . . . 38
4.3 TC5.1 RTT20 Validierung . . . 41
4.4 Testbed-Topologie für variable Pfadkapazität mit zwei SCReAM-Strömen . . . 43
4.5 TC5.2 RTT20 Validierung . . . 45
4.6 Eigene Messung: TC5.2, verlängerte Versuchsdauer . . . 46

5.1 TC5.1 RTT100 . . . 50
5.2 TC5.1 RTT200 . . . 51
5.3 TC5.2 RTT100 . . . 54
5.4 TC5.2 RTT200 . . . 55
5.5 Testbed-Topologie für Überlast im Feedback-Pfad . . . 56
5.6 TC5.3 RTT100 . . . 60
5.7 TC5.3 RTT200 . . . 61
5.8 Testbed-Topologie für konkurrierende SCReAM-Ströme . . . 62
5.9 TC5.4 RTT100 . . . 65
5.10 TC5.4 RTT200 . . . 66
5.11 TC5.5 . . . 69
5.12 Testbed-Topologie für konkurrierenden langlebigen TCP-Strom . . . 70
5.13 TC5.6 RTT100 . . . 73
5.14 TC5.6 RTT200 . . . 74
5.15 TC5.7 RTT100 . . . 78
5.16 TC5.7 RTT200 . . . 79

Listings

3.1 Codebeispiel: Sendefenster Berechnung 28
3.2 Codebeispiel: *Packet Pacing* Intervallberechnung 28

1 Einleitung

1.1 Relevanz des Themas

Eine Gesellschaft ohne Internet kann man sich heutzutage kaum noch vorstellen. Das Internet ist ein weltweites Netzwerk, dessen Anforderungen ständig steigen. In den letzten Jahren haben sich insbesondere die Anforderungen, vom klassischen Transport der Daten, zur Echtzeitkommunikation gewandelt. Zurückzuführen ist dies auf immer beliebter werdende Echtzeitapplikationen wie Videokonferenzen oder Videospiele.
Die steigende Relevanz der Echtzeitkommunikation über eine einheitliche Plattform hat zu einer erstmaligen Zusammenarbeit der Standardisierungsgremien W3C (*World Wide Web Consortium*) [1] und IETF (*Internet Engineering Task Force*) [2] geführt. Das Ziel dieser Zusammenarbeit ist die Standardisierung der Protokolle und Programmierschnittstellen, um primär eine interoperable Echtzeitkommunikation über den Webbrowser zu ermöglichen.

Dabei entstandene Lösungen sind unter anderem *Websockets* [3] und *Web Real-Time Communication* (WebRTC) [4]. Seit 2011 wird die Entwicklung von WebRTC vorangetrieben. Innerhalb kürzester Zeit konnten bedeutende Ergebnisse erzielt werden. Mittlerweile wird WebRTC von vielen mobilen Plattformen und den großen Internet Browsern Google Chrome, Mozilla Firefox und Opera unterstützt und ermöglicht somit Millionen von Nutzern eine Ende-zu-Ende Echtzeitkommunikation. Trotz des bisherigen Erfolgs von WebRTC befinden sich noch einige wichtige Funktionen in der Entwicklung. Eine dieser Funktionen ist die Überlastkontrolle (*Congestion Control*) für die Echtzeitübertragung von WebRTC Medien. Sie dient dem Schutz des Netzwerkes und hat maßgeblich Einfluss auf die Dienstgüte und Nutzungserfahrung aller Beteiligten. Um den Anforderungen an WebRTC gerecht zu werden und um Überlast im Netzwerk zu verhindern, müssen WebRTC Medienströme eine geeignete *Congestion Control* implementieren, die sich von etablierten *Congestion Control* Algorithmen — wie TCP — unterscheiden.

Aus diesem Grund wird sich diese Arbeit mit der Evaluation der *Congestion Control* SCReAM [5] für WebRTC befassen. Für diesen Zweck wird die Referenzimplementierung [6] von SCReAM in das vorhandene *Real-Time Transport Protocol* (RTP) [7] Modell des *INET Frameworks* [8] integriert. Das *INET Framework* ist Bestandteil des diskreten Eventsimulators OMNeT++ [9] und ermöglicht das Simulieren von verkabelten, kabellosen und mobilen Netzwerken.
Um vergleichbare Ergebnisse zu erzielen, wird eine Auswahl, aus den der *IETF RTP Media Congestion Avoidance Techniques* (RMCAT WG) [10] definierten Evaluationstests [11], die die Basisfunktionalitäten einer *Congestion Control* für WebRTC Medien abdecken, durchgeführt. Anschließend werden die Ergebnisse vorgestellt und mit den Ergebnissen aus der Evaluation — einer alternativen Implementierung — von Ericsson Research verglichen. Zuletzt werden die Ergebnisse des Vergleichs diskutiert.

1.2 Problemstellung

Mit WebRTC wurde ein Standard geschaffen, der nahezu jedem eine einfache Echtzeitkommunikation von WebRTC Medien über den Webbrowser ermöglicht. Mit dem bisherigen Erfolg von WebRTC steigt auch die Verantwortung eine zuverlässige *Congestion Control* zu standardisieren, um den Millionen von Anwendern einen verlässlichen Dienst zu gewährleisten.

Ein großes Problem im Internet ist das *Delay* [12], welches einen direkten Einfluss auf die Qualität und Nutzungserfahrung von Echtzeitapplikationen hat. Das *Delay* einer Ende-zu-Ende Kommunikation ist die Dauer eines Signals vom Zeitpunkt des Absendens bis zum Zeitpunkt an dem das Signal beim Empfänger angekommen ist. Dabei setzt sich das *Delay* aus mehreren Arten von Verzögerungen zusammen. Die Basis bildet das *Propagation Delay*, also die Ausbreitungsgeschwindigkeit der Signale über ein physikalisches Übertragungsmedium. Dazu kommen unter anderem Verzögerungen durch Serialisierung, *Queuing Delay*, Jitter und durch den benutzten Codec.
Zum primären Problem für Echtzeitapplikationen wird das *Queuing Delay*. Grundlage bilden Netzwerkkomponenten, wie Router, die über einen Buffer verfügen, mit denen Datenpakete für den Transport zwischengespeichert werden können.
Das *Queuing Delay* entsteht, wenn die Anzahl an Datenpaketen, die über den selben Pfad gesendet werden, einen bestimmten Wert überschreiten. Kurzzeitige Lastspitzen können durch die Buffer in den jeweiligen Netzwerkkomponenten ausgeglichen werden. Halten diese Lastspitzen jedoch über einen längeren Zeitraum an, führt dies zum Überlauf der Buffer. Empfangende Netzwerkkomponenten können keine weiteren Datenpakete zwischenspeichern und beginnen

neu ankommende Datenpakete zu verwerfen. Die Überlastsituation hat eine deutliche Leistungsminderung des Netzwerks zur Folge. Denn mit ansteigender Anzahl an Datenpaketen im Buffer, benötigt das Netzwerk mehr Zeit um den Datenstau zu behandeln.
Ein zu hohes *Delay* wirkt sich negativ auf die Nutzungserfahrung für Echtzeitapplikationen aus. Aus diesem Grund müssen WebRTC Medienströme eine delay-based *Congestion Control* implementieren. Mit dieser Art von Algorithmus wird Überlast und ein Ansteigen von den Buffern der Router durch Änderungen in der *Round Trip Time* (RTT) erkannt. Sofern der Medienstrom die einzige Datenquelle im Netzwerk ist, kann mit einer Reduzierung der Senderate, frühzeitig ein weiteres Ansteigen der Buffer verhindert werden.

Mit dem Protokoll Portfolio von WebRTC ist neben der Übertragung von mehreren Medienströmen, auch ein Übertragen von normalen Daten über mehrere Datenkanäle möglich. Für den Transport der normalen Daten benutzt WebRTC das *Stream Control Transmission Protocol* (SCTP) [13], welches eine loss-based *Congestion Control* implementiert. Eine loss-based *Congestion Control* benutzt Paketverluste als Indikator für Überlast. Es werden die Buffer der Router im Netzwerk gefüllt, bis diese keine Pakete mehr zwischenspeichern können und neu ankommende Pakete verwerfen. Erst dann erfolgt eine Reduzierung der Senderate.
WebRTC multiplext mehrere Medienströme und Datenkanäle in eine Verbindung, somit ist es möglich, dass die Ströme über den gleichen Pfad im Netzwerk transportiert werden. Dabei sollte ein SCTP-Strom mit seiner loss-based *Congestion Control* einen Medienstrom nicht in seiner Übertragung beeinträchtigen. Um dies zu ermöglichen, sind bereits Mechanismen [14] in der Entwicklung. Dieser Ansatz sieht eine Kopplung zwischen RTP [7] und SCTP vor, um die Auswirkungen der loss-based *Congestion Control* von SCTP zu begrenzen.

In einem realen Netzwerk, wie es das Internet bietet, ist eine Überlappung verschiedenster externer Ströme nichts ungewöhnliches. Ein Großteil der externen Ströme wird mit dem Transportprotokoll *Transmission Control Protocol* (TCP) [15] übertragen. Es ist eines der am häufigsten genutzten Transportprotokolle im Internet. Mit TCP werden üblicherweise nicht zeitkritische Daten in einer vollständigen, in Reihenfolge gesicherten Übertragung transportiert. Aus diesem Grund benutzen die gängigsten TCP Implementationen, wie auch SCTP, eine loss-based *Congestion Control* [16]. Der Unterschied zum WebRTC Datenkanal ist, dass die externen Ströme nicht unter der Kontrolle der WebRTC Applikation stehen und somit keine Kopplung der Ströme möglich ist.

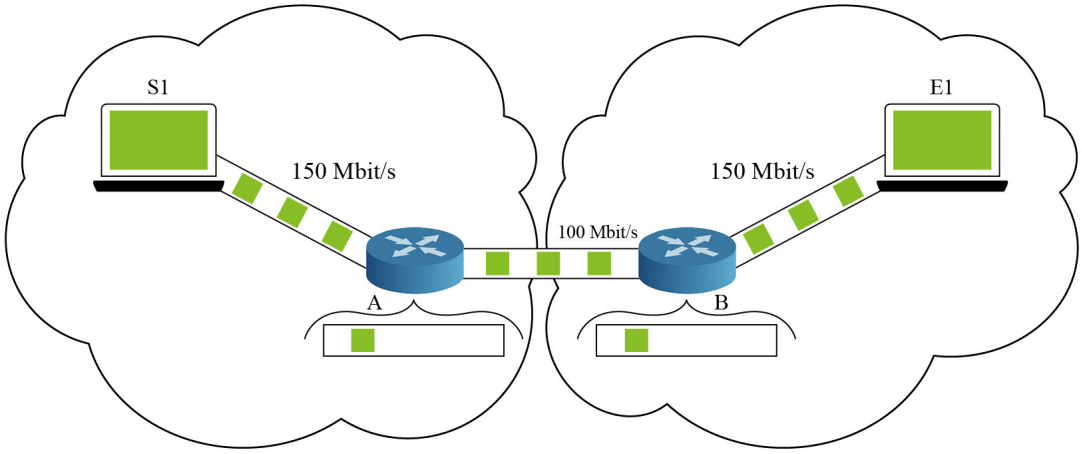


Abbildung 1.1: Übertragung von WebRTC Medien

Die Abbildung 1.1 zeigt eine Topologie, in der eine WebRTC Medienquelle S1 mit seinem zugehörigen Empfänger E1 verbunden ist. In dieser Topologie übersteigt die Senderate von S1 die Pfadkapazität zwischen den Routern. Aus diesem Grund wird dieser zum *Bottleneck*-Pfad. Außerdem besteht kein weiterer Datenverkehr neben der Übertragung des Medienstroms. Die vom WebRTC Medienstrom implementierte delay-based *Congestion Control* hat die volle Kontrolle über das *Queuing Delay* der Ende-zu-Ende Verbindung, indem der Algorithmus seine Senderate an die *Bottleneck*-Pfadkapazität anpasst, bevor das *Queuing Delay* einen Grenzwert erreicht und Verluste auftreten.

Aufbauend zur Abbildung 1.1, zeigt die Topologie in Abbildung 1.2 neben einen Medienstrom auch einen WebRTC Datenstrom. Die Ströme werden in einer WebRTC Applikation über den gleichen Sender S1 übertragen und vom zugehörigen Empfänger E1 angenommen. Auch in diesem Szenario wird der Pfad zwischen den Routern zum *Bottleneck*. Ohne den Einsatz einer Kopplung zwischen RTP und SCTP übersteigt die Senderate der beiden Ströme die *Bottleneck*-Pfadkapazität. Folglich wird der Buffer von Router A gefüllt. Die Reduzierung der Senderate des Datenkanals erfolgt erst mit Auftreten von Verlusten. Gegensätzlich versucht der delay-based *Congestion Control* Algorithmus des Medienstroms die Bufferbelegung so gering wie möglich zu halten, indem dieser seine Senderate reduziert.

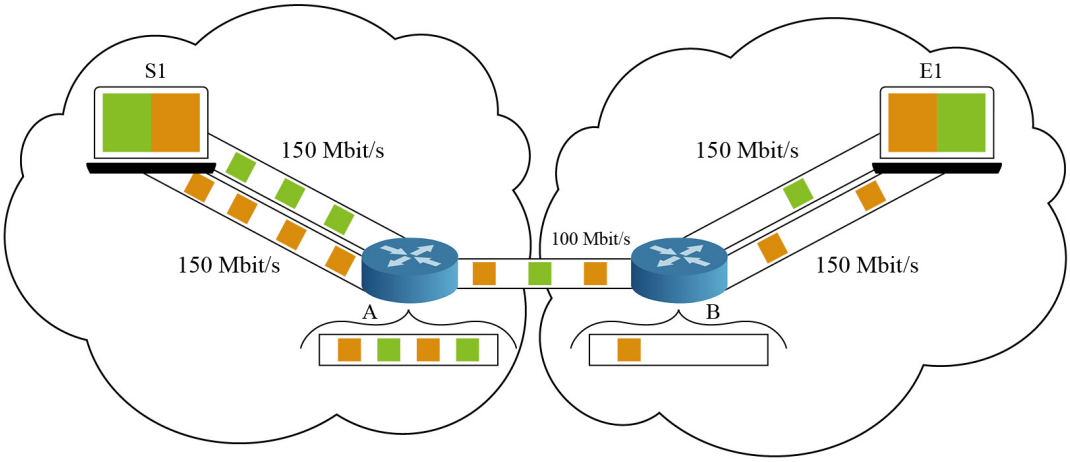


Abbildung 1.2: Übertragung von WebRTC Medien und Daten über den gleichen Endpunkt

Die Topologie in Abbildung 1.3 zeigt ein Szenario in dem ein externer TCP-Strom in Konkurrenz mit einem WebRTC Medienstrom steht. Der Aufbau ist ähnlich wie in der vorherigen Topologie 1.2, mit der Ausnahme, dass die Datenquelle des TCP-Stroms durch einen weiteren Endpunkt dargestellt wird. Der Sender des Medienstroms S1 ist mit dem Empfänger E1 verbunden. Der konkurrierende TCP-Strom S_TCP ist mit dem Empfänger E_TCP verbunden. Der Pfad zwischen den Routern bildet wiederum ein *Bottleneck*.

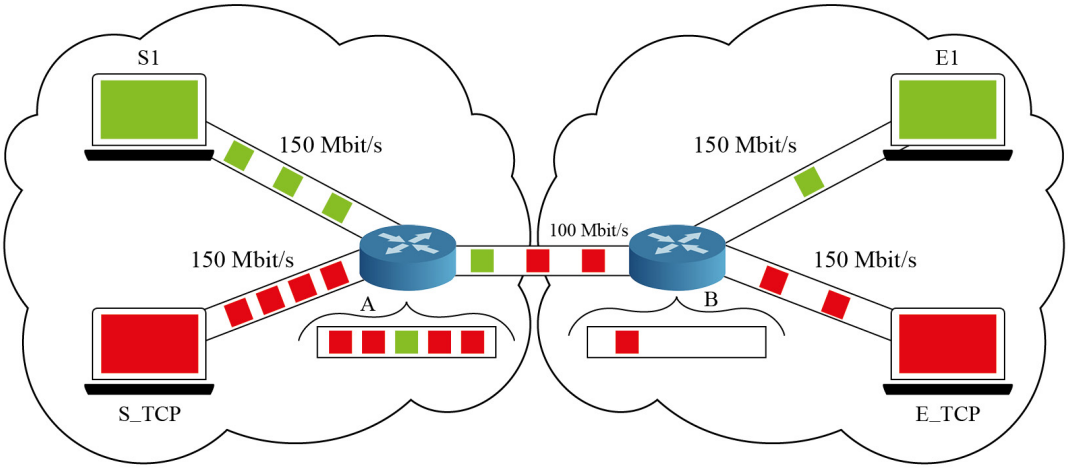


Abbildung 1.3: Konkurrierende TCP Verbindung

Durch die loss-based *Congestion Control* von S_TCP wird der Buffer von Router A gefüllt. Der *Congestion Control* Algorithmus des Medienstroms versucht die Bufferbelegung so gering wie möglich zu halten, indem dieser seine Senderate reduziert.
Erkennt die *Congestion Control* eines Medienstroms einen konkurrierenden TCP-Strom nicht, kann es dazu führen, dass dieser seine Senderate bis auf ein Minimum reduziert und die Dienstgüte der Anwendung nicht mehr gewährleisten kann.

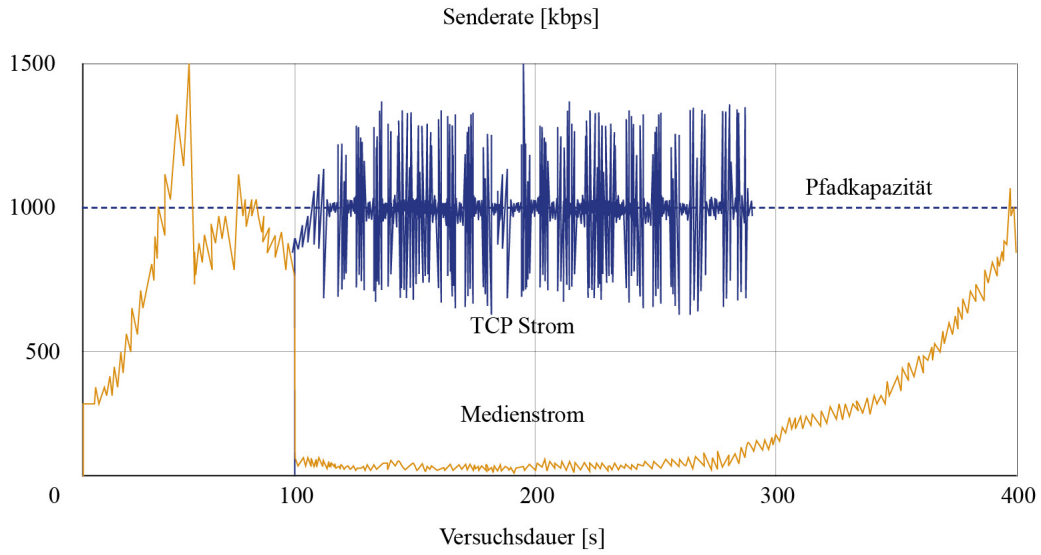


Abbildung 1.4: Illustration: Medienstrom verliert gegen einen TCP-Strom, aus [17]

Abbildung 1.4 zeigt ein Szenario, in dem ein Medienstrom mit einer delay-based *Congestion Control* gegen einen TCP-Strom mit einer loss-based *Congestion Control* verliert.
Die Abszissenachse des Plots stellt die Versuchsdauer in Sekunden dar und die Ordinatenachse den Durchsatz in Kilobits pro Sekunde. Der Medienstrom — in orange — startet bei Versuchsbeginn. Ab 100s beginnt ein TCP-Strom — in blau gekennzeichnet — mit seiner Übertragung. Die Senderate des TCP-Stroms nimmt zu Beginn annäherungsweise die komplette Pfadkapazität ein. Die Senderate übersteigt oftmals die verfügbare Pfadkapazität, gefolgt von einer Reduzierung der Senderate. Mit dem Überschreiten der Pfadkapazität werden die Netzwerkbuffer mit Datenpaketen gefüllt. Erreichen die Buffer eine Grenze, reduziert die loss-based *Congestion*

Control von TCP die Senderate.
Der Medienstrom hat seine Senderate bis auf ein Minimum abgegeben und kann seine Dienstgüte nicht mehr gewährleisten. Nach 290s hat der TCP-Strom seine Übertragung beendet und der Medienstrom beginnt wieder damit seine Senderate zu erhöhen.

Ein *Congestion Control* Algorithmus für die Übertragung von zeitkritischen Daten hat eine große Herausforderung, um gegen die vielseitigen Einflüsse von außen zu bestehen. Neben konkurrierenden rohen UDP-Strömen [18], bildet TCP, mit seinen loss-based *Congestion Control* Implementationen und der weiten Verbreitung [19], eine wesentliche Herausforderung.

1.3 Anforderungen

Die Anforderungen an eine WebRTC *Congestion Control* wurden von der RMCAT Arbeitsgruppe definiert und in [20] beschrieben. Im Folgenden werden die wichtigsten Anforderungen aus [20] und insbesondere der Sektion 1.2 zusammengefasst.

1. Ein *Congestion Control* Algorithmus für die Echtzeitübertragung von WebRTC Medien muss versuchen, ein möglichst geringes *Delay* in der Übertragung zu gewährleisten. Das Ende-zu-Ende *Delay* sollte den Grenzwert von 100ms nicht überschreiten. Der Grenzwert sollte auch in einem *Bottleneck*-Szenario oder bei der Präsenz von konkurrierenden Medien-/Datenströmen eingehalten werden.

2. Der Algorithmus muss eine stabile Senderate, die dicht an der verfügbaren Pfadkapazität liegt, bereitstellen. Häufige Schwankungen der Senderate sollten vermieden werden.

3. Der Algorithmus sollte widerstandsfähig gegenüber Events sein, die die verfügbare Pfadkapazität verändern, vor allem bei einer Reduzierung der Pfadkapazität oder Erhöhung des Ende-zu-Ende *Delays*. Der Mechanismus sollte schnell auf solche Änderungen reagieren, um ein Ansteigen des Ende-zu-Ende *Delays* zu vermeiden oder um neu verfügbar gewordene Pfadkapazität einzunehmen. Aufgrund der Trägheit von Codecs, sollte die Anpassung der Senderate innerhalb einer Sekunde erfolgen.

4. Der Algorithmus muss sich *Fair* gegenüber anderen Echtzeit-/ und Datenströmen, wie lang oder kurzlebigen TCP-Strömen, verhalten. Der Begriff *Fair* ist in diesem Kontext schwer zu definieren. Der Algorithmus sollte mit sich selbst *Fair* sein und einen *fairen* Anteil der Pfadkapazität an Ströme mit gleicher (RTT) und wenn möglich auch an Ströme mit unterschiedlicher RTT abgeben.

5. Der Algorithmus sollte nicht gegen konkurrierende TCP-Ströme oder SCTP-Ströme verlieren und er sollte dieses Szenario so gut wie möglich verhindern.
6. Der Algorithmus sollte sich zu Beginn so schnell wie möglich an die Netzwerkbedingungen anpassen.

Anzumerken ist, dass das Ziel dieser Bachelorarbeit die Evaluation eines *Congestion Control* Kandidaten für WebRTC Medien ist.

1.4 Vorgehensweise

Diese Bachelorarbeit gliedert sich in weitere sechs Teile:

Kapitel 2 - WebRTC - beschreibt kurz die wichtigsten Funktionen und Protokolle, die für die Übertragung von WebRTC Medien und Daten eingesetzt werden.

Kapitel 3 - SCReAM - gewährt einen Überblick über den *Congestion Control* Kandidaten für WebRTC, indem die wichtigsten Komponenten des Algorithmus beschrieben werden.

Kapitel 4 - Evaluationsplattform - beschreibt die Evaluationsumgebung, sowie die Einstellungen und Metriken die für diese Evaluation von Bedeutung sind. Außerdem wird die Grundstruktur der Experimente erläutert. Zuletzt wird die Implementation mit geeigneten Versuchen validiert.

Kapitel 5 - Experimente - beschreibt den Aufbau, Zweck und die Erwartungshaltung der Experimente mit anschließender Visualisierung und Beschreibung der Vergleichsmessungen und der eigenen Messwerte.

Kapitel 6 - Diskussion - diskutiert die Ergebnisse der Experimente in Bezug den vorgestellten Erwartungswert an die eigenen Messergebnisse.

Kapitel 7 - Fazit - fasst die Ergebnisse der Bachelorarbeit zusammen und gibt einen Ausblick für zukünftige Arbeiten.

2 WebRTC

Dieses Kapitel befasst sich mit den für diese Arbeit relevanten Möglichkeiten und Funktionen von WebRTC. Dabei liegt der Schwerpunkt bei den für die Übertragung von WebRTC Medien und Daten eingesetzten Funktionen und Protokollen.

2.1 Einleitung

Web Real-Time Communication (WebRTC) [4] und [21] ist ein offenes Framework, das aus einer Sammlung von Standards, Protokollen und Programmierschnittstellen besteht. In dessen Kombination ermöglicht WebRTC eine Ende-zu-Ende Echtzeitkommunikation über den Webbrowser und kommt dabei gänzlich ohne Plug-ins, proprietäre Treiber oder andere Software aus. Somit kann der Webbrowser nicht nur Daten von *Backend*-Servern laden, sondern auch Echtzeitinformationen von anderen Browsern.
Unterstützt wird WebRTC neben den großen Browsern Google Chrome, Mozilla Firefox und Opera, auch nativ von den Plattformen Android[1] und iOS[2] und kann somit auch für mobile Anwendungsszenarien eingesetzt werden.
Mit WebRTC ist eine Übertragung von Audio-/Videoströmen in Echtzeit und der Austausch von anderen beliebigen Daten in einer verschlüsselten Ende-zu-Ende Verbindung möglich. Auch eine Integration in vorhandene Kommunikationssysteme, wie *Voice Over IP* (VOIP) oder *Public Switched Telephone Network* (PSTN), sind mit WebRTC möglich. Videokonferenzen, Telefonie oder Spiele sind dabei nur ein Bruchteil der Anwendungsmöglichkeiten, die den Entwicklern durch diese Funktionalitäten gegeben sind.
Um den Anforderungen gerecht zu werden, muss der Webbrowser eine Vielzahl neuer Funktionalitäten bereitstellen. Er muss rohe Audio- und Videodaten verarbeiten können, neue Schnittstellen anbieten und etliche Netzwerkprotokolle, für eine sichere Ende-zu-Ende Verbindung über *Network Address Translator* (NAT) hinaus, implementieren.
Der Browser abstrahiert die komplexen Funktionalitäten hinter einer einfachen JavaScript

[1] https://webrtc.org/native-code/android/ (Aufgerufen am 28-8-17)
[2] https://webrtc.org/native-code/ios/ (Aufgerufen am 28-8-17)

Programmierschnittstelle. Mit dieser lässt sich eine Videokonferenz oder Datenübertragung in nur wenigen Zeilen Code realisieren.

2.2 Entwicklung

Im Mai 2011 wurde WebRTC als ein quelloffenes Projekt von Google veröffentlicht. Seitdem haben mehrere Firmen ihr Interesse an WebRTC bekundet und bei der Entwicklung mitgewirkt. Darunter sind unter anderem Mozilla, Opera und Apple.
Die Architektur von WebRTC umfasst mehrere verschiedene Standards für die Applikations- und Browser-Schnittstellen, sowie etliche verschiedene Protokolle und Datenformate.
Seit Mitte 2011 ist eine Arbeitsgruppe der W3C [1] für die Entwicklung der Browser-Schnittstellen verantwortlich.
Zur gleichen Zeit begann eine Arbeitsgruppe der IETF [2] mit der Definition von Protokollen, Datenformaten und anderen notwendigen Aspekten für eine Ende-zu-Ende Echtzeitkommunikation.
WebRTC wurde mit der letzten veröffentlichten Version [22], vom November 2017, als Standard verabschiedet. Dies betrifft aber nicht die Standardisierung einer *Congestion Control* für WebRTC Medien.

2.3 Audio- und Video-Engines

Für die Übertragung von Audio- und Videoströmen benötigt der Webbrowser Zugriff auf Kamera und Mikrofon und damit auf die Systemhardware. Um dabei den Anforderungen an WebRTC gerecht zu werden und keine zusätzlichen Treiber zu verwenden, muss der Webbrowser eine einfache Schnittstelle zum Erfassen von Audio- und Videoströmen anbieten.
Ein direktes Übertragen der rohen Ströme ist nicht effizient. Jeder Strom muss vor der Übertragung kodiert, synchronisiert und an die schwankende Bitrate und Latenz zwischen den Endpunkten angepasst werden.

WebRTC kann diese durchaus komplexen Anforderungen abdecken und bringt bereits fertige Engines mit in den Webbrowser (siehe Abbildung 2.1).
Mit der von der W3C spezifizierten *Web Application API getUserMedia* lässt sich einfach ein Audio- und Videostrom der darunterliegenden Plattform belegen.

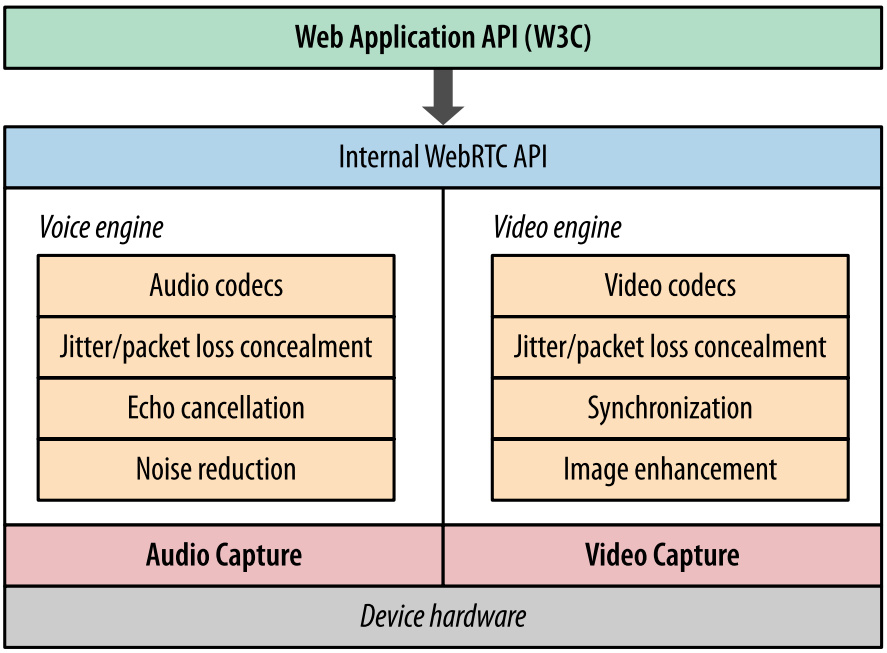


Abbildung 2.1: WebRTC Audio- und Videoengines, aus [21]

Der rohe Audiostrom wird zunächst durch *Noise Reduction* und *Echo Cancellation* optimiert, indem Störgrößen herausgefiltert werden. Anschließend wird der Strom automatisch kodiert. Zuletzt werden durch den *Concealment* Algorithmus Fehler überbrückt, die durch Netzwerk Jitter und Paketverluste auftreten.
Die Video Engine funktioniert ähnlich wie die Audio Engine. Zunächst wird der Videostrom durch die *Image Enhancement* optimiert. Anschließend findet eine Synchronisation und Kodierung der Ströme statt. Zuletzt werden wieder die negativen Effekte durch Netzwerk Jitter und Paketverluste überbrückt. Der Prozessablauf wird direkt durch den Webbrowser durchgeführt.

Diese Bachelorarbeit ordnet sich im Gesamtsystem unterhalb vom Applikationsinterface ein. Der *Congestion Control* Algorithmus nimmt bereits kodierte Pakete von Audio- und Videoströmen entgegen und prüft, ob diese übertragen werden dürfen.

2.4 WebRTC Protokolle

Wie bereits in dem Kapitel 1.2 erwähnt wurde, ist die Übertragung von Echtzeitdaten verzögerungsempfindlich. Aus diesem Grund sind die Audio-, und Videoströme dafür ausgelegt

Paketverluste in Grenzen zu tolerieren. Diese sind aber empfindlich gegenüber Verzögerungen.

Für den Transport von WebRTC Medien und Daten kommen nur die Transportprotokolle TCP und UDP in Frage. Denn WebRTC stellt die Anforderung an eine Ende-zu-Ende Verbindung über *Network Address Translation* (NAT) [23] hinweg. NAT-Boxen lassen nur Datenverkehr mit TCP und UDP als Transportprotokoll durch, da der Großteil der ausgelieferten NAT-Boxen lediglich eine TCP oder UDP Port Übersetzung vornehmen.

Abbildung 2.2 zeigt den aktuellen Protokollstack von WebRTC. Die auf der linken Seite abgebildeten Protokolle dienen der Signalisierung über einen externen Server und sind nicht im WebRTC Framework enthalten. Dadurch wird eine Interoperabilität mit anderen Signalisierungsprotokollen, die vorhandene Kommunikationsinfrastrukturen benutzen, ermöglicht.
Für eine Nutzung von WebRTC, über die Signalisierung hinweg, kommen die Protokolle auf der rechten Seite zum Einsatz. Grundlage hierfür bildet das Protokoll UDP auf der Transportschicht.

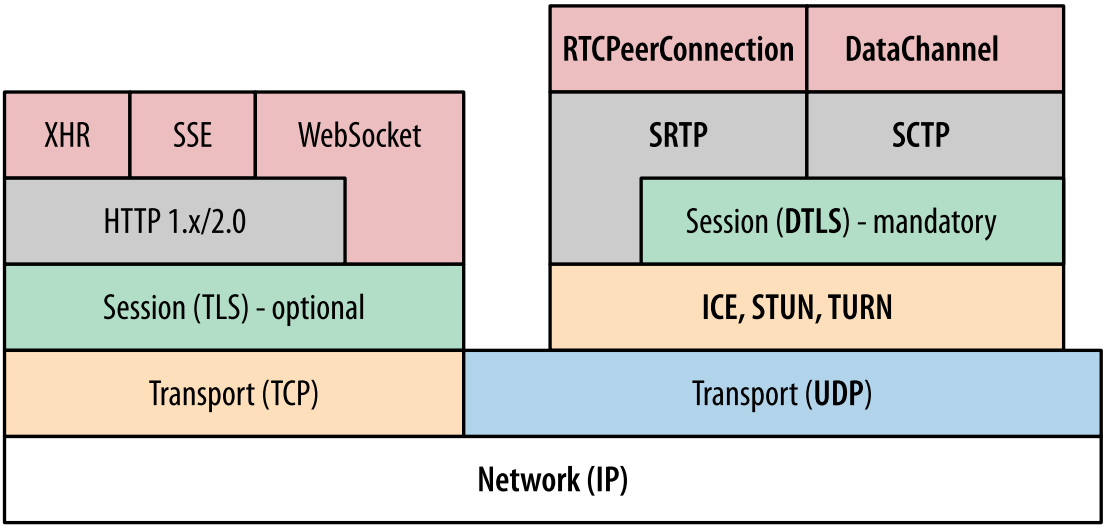


Abbildung 2.2: WebRTC Protokolle, aus [21]

UDP

Das *User Datagram Protocol* [18] ist eines der ältesten Netzwerkprotokolle. Es befindet sich auf der Transportschicht des OSI-Modells und gehört zum Kern des Internetprotokolls (IP). Im Gegensatz zu TCP ist UDP ein verbindungsloses Transportprotokoll mit der Dienstgüte *Best*

Effort und überträgt seine Daten in Datagrammen. UDP garantiert kein Ankommen der Daten, keine in Reihenfolge gesicherte Übertragung und keine *Congestion Control*. Dadurch versucht UDP, unter bester Ausnutzung der Ressourcen, die Daten zu versenden. Dabei entstehende Paketverluste werden toleriert, um eine vorzugsweise geringe Latenz zu erzielen.

UDP bildet die Grundlage für eine Echtzeitkommunikation über den Webbrowser. Alleinstehend kann UDP die Anforderungen an WebRTC nicht erfüllen. Es benötigt unter anderem Mechanismen, um NAT/PAT und Firewalls zu überqueren, zur Verschlüsselung und zur Datenratenanpassung des UDP-Stroms.

ICE

Interactive Connectivity Establishment (ICE) [24] ist ein Protokoll, das unter Verwendung der Protokolle STUN und TURN eine Ende-zu-Ende Verbindung über UDP herstellt. Für diesen Zweck kommt ein Offer-/Answer-Modell [25] zum Einsatz, das über einen Signalisierungskanal Verbindungsinformationen austauscht.
In einem realen Netzwerk sitzen Endgeräte meist hinter mehreren NAT-Schichten, einer Firewall oder beiden, in Form eines Netzwerkrouters. ICE versucht die komplexen Probleme einer Ende-zu-Ende Verbindung zu lösen, indem es versucht den besten Pfad zwischen zwei Endpunkten zu finden. Um dies zu erreichen, probiert ICE alle Verbindungsmöglichkeiten parallel aus und wählt die effizienteste Lösung.
Zunächst wird versucht eine Verbindung über die Adresse des Endgeräts herzustellen. Wenn das Endgerät hinter einer NAT-Schicht sitzt, wird dies nicht funktionieren. Denn NAT/PAT setzt in der Regel eine öffentliche Adresse auf mehrere private IP-Adressen um. Um eine Verbindung dennoch herstellen zu können, kommt ein STUN-Server zum Einsatz. Dieser versucht die öffentliche IP-Adresse des Endgeräts herauszufinden. Scheitert auch dieser Versuch, kommt ein TURN-Server zum Einsatz, der als vermittelnder Server dient. Diese Möglichkeit dient als letzter Ausweg, um eine gewisse Zuverlässigkeit beim Verbindungsaufbau zu gewährleisten.

STUN

Session Traversal Utilities for NAT (STUN) [26] ist ein Protokoll zur Überwindung von NAT. Das Protokoll läuft in der Regel auf einem externen Server. Dieser hat die Aufgabe eingehende Anfragen von Applikationen, die hinter einer NAT-Schicht sitzen, anzunehmen und deren öffentliche Adresse als Antwort zurückzusenden. ICE versucht den besten Pfad zu ermitteln und startet Anfragen über mehrere Ports.

TURN

Traversal Using Relays around NAT (TURN) [27] ist ein Protokoll, das zum Einsatz kommt, wenn keine direkte Ende-zu-Ende Verbindung über UDP hergestellt werden konnte. Ein TURN-Server wird dementsprechend als Backup eingesetzt und vermittelt Audio-, Video- und Datenströme zwischen den Endpunkten. Angesprochen werden die Server über eine öffentliche IP-Adresse.

DTLS

DTLS [28] steht für *Datagram Transport Layer Security* und ist ein Verschlüsselungsprotokoll, das zum Einsatz kommt bei der Verschlüsselung von Datagrammen einer UDP-Verbindung. DTLS setzt auf dem Protokoll TLS auf, welches eine vollständige Verschlüsselung mit asymmetrischen Kryptographiemethoden bietet.
Um DTLS für das unzuverlässige Transportprotokoll UDP benutzen zu können, mussten einige Veränderungen gegenüber TLS durchgeführt werden. Denn TLS ist für den zuverlässigen Transport durch TCP ausgelegt. Zu den Änderungen gehört das Wiederherstellen des *Handshakes*. Denn zu Beginn einer Kommunikation müssen, für eine erfolgreiche Authentifizierung und einen Schlüsselaustausch, alle Pakete vollständig übertragen worden sein. Fortführend musste eine explizite Paketnummerierung während der Übertragung eingeführt werden, um zu prüfen ob alle Pakete vorhanden sind und um gegebenenfalls eine gezielte Neuübertragung anzufordern.

Für die Spezifikation von WebRTC ist die Verschlüsselung der Medien- und Datenströme durch DTLS zwingend notwendig.
Alle Datenkanalströme werden über das Protokoll SCTP gemultiplext, welches aber keine Verschlüsselung anbietet. Aus diesem Grund werden alle über einen Datenkanal gesendeten Daten über einen DTLS Tunnel verschlüsselt. Audio- und Videoströme werden durch das Protokoll SRTP realisiert und verschlüsselt. Allerdings spezifiziert WebRTC, dass der SRTP-Schlüsselaustausch durch DTLS verschlüsselt wird, um unter anderem einen *Man in The Middle* Angriff frühzeitig erkennen zu können.

SRTP

Das *Secure Real-Time Transport Protocol* (SRTP) [29] ist eine Erweiterung zum *Real-Time Transport Protocol* (RTP) [7], mit verbesserten Sicherheitsmerkmalen wie Verschlüsselung und Authentifizierung.
SRTP ist ein Transportprotokoll und befindet sich auf der Applikationsschicht des OSI-Modells.

Es läuft größtenteils über UDP und ermöglicht eine Ende-zu-Ende Übertragung von Echtzeitdaten, wie Audio-, Video- oder Simulationsdaten.
Für diesen Zweck ist es auch im Protokollstack von WebRTC enthalten. Es multiplext mehrere eingehende Audio- oder Videoströme, packt RTP Pakete, bettet diese zur Verschlüsselung in SRTP Pakete und versendet Pakete über UDP.
RTP selbst bietet keine eigene *Congestion Control* — die aber zum Schutz des Netzwerkes, der Dienstgüte und für die Nutzungserfahrung mit Echtzeitapplikationen zwingend erforderlich ist. RTP benutzt das Kontrollprotokoll (RTCP) [30] als Feedback-System, um in Lastsituationen die Senderate anzupassen.
An diesem Punkt setzt diese Arbeit an und wird, mit der Vorstellung eines *Congestion Control* Kandidaten für WebRTC Medien, in Kapitel 3 noch näher erläutert.

SCTP

Das *Stream Control Transmission Protocol* (SCTP) [13] ist ein Transportprotokoll, das wie UDP und TCP auf der Transportschicht des OSI-Modells sitzt. SCTP gewährleistet — so wie TCP — einen zuverlässigen Transport der Daten über ein Netzwerk und implementiert eine loss-based *Congestion Control*. Im Gegensatz zu TCP erlaubt SCTP jedoch die Übertragung von mehreren unabhängigen logischen Strömen und stellt gleichzeitig die vollständige Übertragung dieser sicher.
Eine weitere wichtige Eigenschaft von SCTP ist *Multihoming*. Mit dieser kann ein verbundener Endpunkt alternative IP-Adressen haben und redundante Verbindungen halten. Fällt der primäre Pfad aus, wird automatisch der Datenverkehr auf den alternativen Pfad geroutet. Jedoch wird diese Funktion noch nicht vom WebRTC Framework berücksichtigt.
Aufgrund der Protokollerweiterung *Partial Reliability Extension* [31] wird SCTP noch flexibler und vereint die Vorteile von TCP und UDP. Mit dieser Eigenschaft lässt sich jeder Strom in Zuverlässigkeit und Transport frei konfigurieren.

Durch die flexiblen Anpassungsmöglichkeiten und den Eigenschaften von SCTP, wird es den folgenden Anforderungen an einen *DataChannel* [22] von WebRTC gerecht.

1. Das Protokoll muss das Multiplexen von mehreren unabhängigen Datenkanälen unterstützen.
2. Jeder Kanal muss eine in Reihenfolge oder nicht in Reihenfolge gesicherte Übertragung unterstützen.
3. Jeder Kanal muss eine zuverlässige oder unzuverlässige Übertragung unterstützen.

4. Das Protokoll muss eine *Congestion Control* implementieren.

5. Die Daten müssen verschlüsselt übertragen werden, um die Authentizität und Integrität der Daten zu gewährleisten.

Alle Anforderungen bis auf den letzten Punkt unterstützt SCTP. Die Verschlüsselung geschieht durch einen DTLS Tunnel zwischen den beiden Endpunkten.

RTCPeerConnection

Die *RTCPeerConnection* [32] ist eine Schnittstelle und stellt die eigentliche WebRTC-Verbindung zwischen zwei Endpunkten dar. Die Schnittstelle bietet Methoden, um eine Verbindung zu einem entfernten Endpunkt herzustellen, die Verbindung zu überwachen und um diese wieder zu beenden. Damit ist die *RTCPeerConnection* verantwortlich für die Verwaltung des gesamten Lebenszyklus einer Ende-zu-Ende Verbindung.
Wenn eine Verbindung zu einem entfernten Endpunkt hergestellt werden soll, beginnt der Webbrowser mit der Instanziierung eines *RTCPeerConnection* Objektes. Dafür wird eine selbst generierte *Session Description Protocol* (SDP) [33] -Beschreibung an den entfernten Endpunkt gesendet. Dieser antwortet wiederum mit einer eigenen SDP-Beschreibung. Die SDP-Beschreibungen sind notwendig für die Überwindung von NAT durch das ICE Framework.
Der Ablauf beim Verbindungsaufbau ist nicht relevant für diese Arbeit und soll nur ein Basisverständnis für diesen geben.
Mit einer etablierten Verbindung kann WebRTC, mit der *RTCPeerConnection*, seine vollen Möglichkeiten entfalten. Unter Verwendung von SRTP über UDP und der Verschlüsselung durch DTLS kann die *RTCPeerConnection* Audio- und Videoströme in Echtzeit an einen entfernten Endpunkt senden und eingehende Ströme verwalten und an die Applikation übergeben.

DataChannel

Der *DataChannel* [34] ist ebenfalls eine Schnittstelle, die von WebRTC angeboten wird. Dieser erlaubt einen direkten Austausch von beliebigen Daten zwischen zwei Endpunkten, ohne den Einsatz eines externen Servers. Eine Bedingung hierfür ist eine vorher etablierte Verbindung über eine *RTCPeerConnection*, damit ein Senden über diese möglich ist.
Mit dem unter dem *DataChannel* liegenden Protokoll SCTP, lassen sich die Übertragungseigenschaften frei konfigurieren. Wie zuvor schon genannt, kann die Übertragung zuverlässig oder unzuverlässig und in Reihenfolge oder nicht in Reihenfolge gesichert sein. Darüber hinaus kann der *DataChannel* auch als teilweise zuverlässig konfiguriert werden, indem eine maximale

Anzahl an Neuübertragungen definiert oder eine zeitliche Begrenzung der Neuübertragungen festgelegt wird.

3 SCReAM

Dieses Kapitel beschreibt den *Congestion Control* Kandidaten SCReAM für WebRTC. Nach einer kurzen Einführung, wird ein Überblick über den Algorithmus gegeben. Anschließend werden die wichtigsten Komponenten von SCReAM im Detail vorgestellt.

3.1 Einleitung

SCReAM [5] (**S**elf-**C**locked **R**ate **A**daption for **M**ultimedia) ist ein *Congestion Control* Algorithmus, entwickelt von Ericsson Research, der primär für den Echtzeitaustausch von Videodaten in mobilen Anwendungsszenarien entworfen wurde. Der Algorithmus befindet sich zurzeit in der Standardisierung der *IETF RTP Media Congestion Avoidance Techniques* (RMCAT) [10]. Die Arbeitsgruppe definiert Anforderungen mit dem Ziel einen *Congestion Control* Algorithmus zu standardisieren, der für den Echzeitransport von WebRTC Medien genutzt werden kann. Mit der Google *Congestion Control* (GCC) [35] und *Network Assisted Dynamic Adaption* (NADA) [36] von Cisco befinden sich zwei weitere Kandidaten neben SCReAM in der Standardisierung der RMCAT Arbeitsgruppe.

3.2 Übersicht über das SCReAM Framework

Das SCReAM Framework [37] basiert auf einer dynamischen Anpassung der Übertragungsrate nach den sich ständig wechselnden Netzwerkbedingungen. Dafür verfolgt es das Prinzip der *Packet Conservation* und implementiert einen hybriden loss- und delay-based *Congestion Control* Algorithmus.

Das Prinzip der *Packet Conservation* [38] wird beschrieben als ein wichtiger Faktor um Netzwerke vor Überlast zu schützen.
Das Ziel ist ein Gleichgewicht in der Verbindung herzustellen. Dies kann nur erreicht werden, wenn ein Sender, mit einem vollen Sendefenster, erst wieder ein neues Paket in das Netzwerk gibt, sobald ein altes Paket es verlassen hat.

SCReAM realisiert dies, indem der Empfänger eine Liste von Sequenznummern der empfangenen Pakete hält und diese als Feedback an den Sender schickt. Analog dazu führt der Sender eine Liste von gesendeten Paketen und deren jeweiligen Größe. Anhand von diesen Informationen kann die Anzahl an Bytes, die sich in der Übertragung befinden, festgestellt werden.
Die SCReAM *Congestion Control* basiert auf einem fensterbasierten und byteorientierten Algorithmus. Die Anzahl an übertragenen Bytes wird aus den Größen der RTP-Pakete berechnet. Ein *Congestion Window* begrenzt die Anzahl an Bytes, die sich zur gleichen Zeit in der Übertragung befinden dürfen.

Die primäre Aufgabe der *Congestion Control* von SCReAM ist die frühzeitige Erkennung von Überlast, um so ein weiteres Ansteigen des *Delays* im Netzwerk zu verhindern. Die Überlasterkennung geschieht auf Basis von Messungen des *One-Way Delays*, also der Zeit die ein RTP-Paket vom Sender bis zum Empfänger benötigt. Mit diesem Ansatz kann das durch die Buffer der Netzwerkkomponenten entstandene *Queuing Delay* abgeschätzt werden. Ein ansteigendes *Delay* ist ein Anzeichen dafür, dass die Buffer gefüllt werden und die Senderate die Pfadkapazität überschritten hat. Damit es nicht zu einem weiteren Anstieg des *Delays* kommt, muss der Datenstau abgearbeitet werden. Dies geschieht indem die *Congestion Control* das *Congestion Window* reduziert und damit verbunden die Senderate.
Sobald ein konkurrierender TCP-Strom erkannt wird, schaltet die *Congestion Control* von einer delay-based, auf eine loss-based *Congestion Control*. Mit diesem Ansatz soll ein Verlieren gegen einen TCP-Strom verhindert werden.

In der Architektur von SCReAM implementiert der Sender die komplette Logik. Der Empfänger hat lediglich die Aufgabe Messungen durchzuführen und Feedback-Nachrichten an den Sender zu übertragen. Für diesen Zweck wird das in der Sektion 2.4 angesprochene Kontrollprotokoll RTCP [30] als Feedback-System benutzt. Die Feedback-Nachrichten enthalten folgende Elemente.

- *Synchronization Source Identifier* (SSRC) [39]: Identifiziert eine Synchronisationsquelle und wird benötigt um verschiedene Quellen auseinanderzuhalten.
- Zeitstempel: Markiert den Empfang des letzten RTP-Pakets. Dieser wird für die Berechnung der Ende-zu-Ende Verzögerung benötigt.
- Sequenznummer: Kennzeichnet das letzte empfangene RTP-Paket und wird zur Berechnung der *bytes-in-flight* genutzt. Die *bytes-in-flight* setzen sich aus den abgeschickten, aber noch nicht bestätigten Paketen zusammen.

- Liste von Sequenznummern: Kennzeichnet die letzten N empfangenen Pakete. Die Liste wird benutzt um verlorengegangene RTP-Pakete zu bemerken und um die *bytes-in-flight* zu berechnen.
- *Explicit Congestion Notification* (ECN) [40]: Erlaubt eine Ende-zu-Ende Benachrichtigung von Überlast im Netzwerk. Die ECN wird in dieser Arbeit nicht verwendet, wird aber dennoch Default, als ausgeschaltet, mitübertragen.

Ein SCReAM Sender implementiert für jede Medienquelle eine *Media Rate Control* und eine RTP-*Queue*. Die Abbildung 3.1 zeigt einen Überblick über die Funktionalitäten eines SCReAM Senders.
Medien-Frames werden kodiert und an die RTP-*Queue* (1), in Abbildung 3.1, weitergeleitet. Die RTP-*Queue* speichert kodierte RTP-Pakete für den Transport zwischen. Die *Media Rate Control* berechnet auf Basis der Größe der RTP-*Queue* (2) eine Bitrate und stellt diese dem *Media Encoder* (3) zur Verfügung. Anschließend werden RTP-Pakete aus jeder RTP-*Queue* nach einer definierten Prioritätsreihenfolge oder nach einem Rundlauf-Verfahren vom *Transmission Scheduler* (4) ausgewählt. Bevor der *Transmission Scheduler* die RTP-Pakete an den UDP-Socket (5) übergibt, muss er abprüfen, ob die Anzahl an *bytes-in-flight* kleiner ist, als das aktuelle *Congestion Window*.
Nachdem das RTP-Paket an den UDP-Socket übergeben wurde, fügt der *Transmission Scheduler* folgende Elemente seiner Liste aus 4-Tupeln hinzu:

- *Synchronization Source Identifier* (SSRC) [39]: Der Wert wird zufällig ausgewählt und identifiziert eine Synchronisationsquelle. Der Identifier ist wichtig für die *Congestion Control*, um verschiedene Quellen auseinanderzuhalten.
- Einen Zeitstempel: Vermerkt den Zeitpunkt wann das RTP-Paket abgeschickt wurde.
- Eine Sequenznummer: Gibt die Nummer des abgeschickten RTP-Pakets an.
- Die Größe des abgeschickten RTP-Pakets.

Die *Congestion Control* Komponente nimmt eingehende RTCP-Feedback-Nachrichten entgegen (6) und bildet aus der SSRC, dem Zeitstempel und der Sequenznummer, des zuletzt empfangenen RTP-Pakets, ein 3-Tupel.
Anschließend berechnet die Komponente mit diesen Informationen das aktuelle *Congestion Window*, das *One-Way Delay* und die *bytes-in-flight* und tauscht (7) diese Informationen mit dem *Transmission Scheduler* aus.

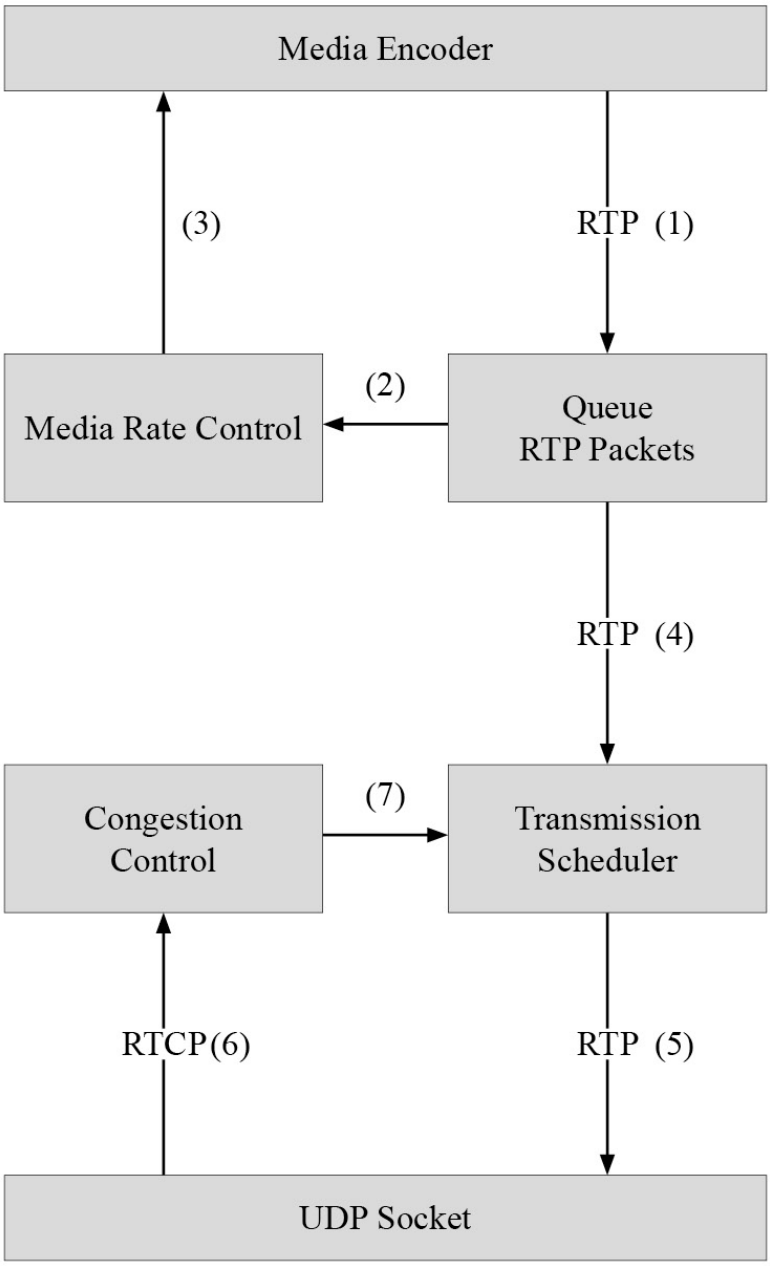


Abbildung 3.1: Illustration: SCReAM Sender Übersicht, aus [37]

3.3 Beschreibung der Hauptkomponenten

In dieser Sektion wird zunächst die Funktion einer *Queue* auf Seiten des Senders diskutiert. Anschließend werden die drei Hauptkomponenten *Transmission Scheduler*, *Congestion Control* und *Media Rate Control* näher betrachtet und erläutert.

3.3.1 RTP Queue

Alternativ, zu einer Queue beim Sender, können bereits kodierte RTP-Pakete direkt übertragen werden. Dieser Fall trifft für kleine Audio-Pakete zu, da diese mit einer geringen Bitrate übertragen werden und selten zur Überlast beitragen. Das Problem sind große Video-Pakete, die ab dem Zeitpunkt des Sendens nicht mehr unter Kontrolle des Senders sind und zu einem Anstieg des *Delays* im Netzwerk führen können. Mit einer *Queue* auf der Senderseite kann das Risiko minimiert werden, die Buffer im Netzwerk übermäßig mit Video-Paketen zu füllen.
Folglich kann es dazu führen, dass bereits kodierte RTP-Pakete zu einem Anstieg der RTP-*Queue* führen. Die Konsequenz daraus ist ein *Delay* zwischen dem Audiostrom und dem Videostrom. Um die Ströme wieder zu synchronisieren kann die RTP-*Queue* vollständig geleert werden, indem bereits kodierten RTP-Pakete verworfen werden.
Der Nachteil ist hinnehmbar, da bei Überlast mit der RTP-*Queue* ein übermäßiges Ansteigen des *Delays* und Paketverluste im Netzwerk verhindert werden können.

3.3.2 Congestion Control

Die *Congestion Control* Komponente hat zwei Hauptaufgaben:

- Die Berechnung vom *Congestion Window* beim Sender. Das *Congestion Window* legt eine Obergrenze der *bytes-in-flight* fest.
- Die Berechnung vom Sendefenster. Durch das Sendefenster kann kontrolliert werden, ob das Verhältnis zwischen den *bytes-in-flight* und dem *Congestion Window* stimmt und RTP-Pakete übertragen werden dürfen.

3.3.2.1 Grundlagen

- *Bytes-in-flight*: Werden berechnet aus der Summe der RTP-Paketgrößen, der bereits gesendeten, aber noch nicht bestätigten RTP-Pakete.
  Zur Verdeutlichung wurden in Abbildung 3.2 SN-RTP-Pakete bereits übertragen. Das zuletzt bestätigte RTP-Paket ist SN-3. Die Summe aus den RTP-Paket-Größen SN bis einschließlich SN-2 ergeben die *bytes-in-flight*. Bereits verlorengegangene Pakete werden

in der Berechnung berücksichtigt.

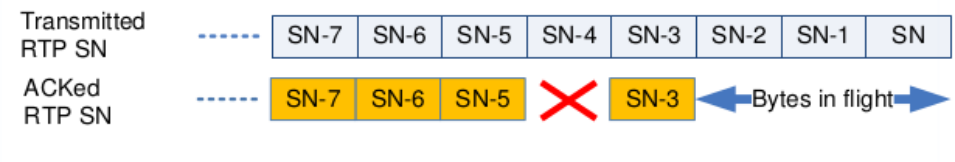


Abbildung 3.2: Beispiel: Berechnung der *bytes-in-flight*, aus [41]

- *Bytes-newly-acked*: Setzen sich aus der Summe der Paketgrößen und der seit dem letzten Feedback bestätigten RTP-Pakete zusammen. Verloren gegangene RTP-Pakete werden bei der Berechnung berücksichtigt. Die *bytes-newly-acked* spielen eine wichtige Rolle bei der Berechnung vom *Congestion Window*.

- Überlasterkennung: Um Überlast vorzeitig zu erkennen, wird mit jeder ankommenden RTCP-Feedback-Nachricht das *Queuing Delay* berechnet. Das *Queuing Delay* gibt die Zeit an, die ein RTP-Paket in den Buffern der Router im Netzwerk verweilt hat.
  Für diesen Zweck bietet die *Congestion Control* eine Methode, ähnlich der von LEDBAT [42], zum Messen des *One-Way Delays* (OWD). Berechnet wird dies aus der Differenz zwischen dem Zeitpunkt T_i, an dem das i-te Paket gesendet wurde und dem Zeitpunkt t_i, an dem das i-te Paket empfangen wurde (siehe Abbildung 3.3).
  Das *Queuing Delay* wird nun aus der Differenz des gemessenen *One-Way Delays* und dem Basis-*One-Way Delay*, also der minimalen Ende-zu-Ende Verzögerung die auf dem Pfad gemessen wurde, berechnet.

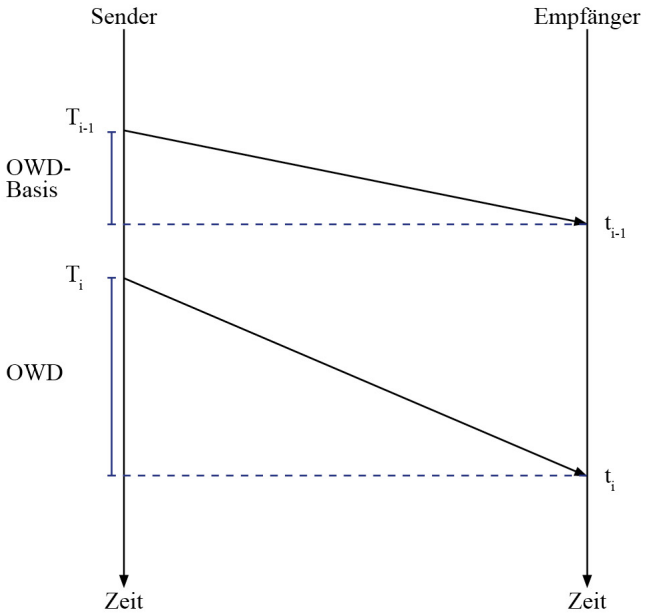


Abbildung 3.3: Beispiel: Messung *One-Way Delay*

- Verlusterkennung: Ein Verlust wird angezeigt, sobald ein oder mehrere RTP-Pakete als verlorengegangen deklariert wurden. Nachdem ein Verlust erkannt wurde, werden für eine RTT weitere als verloren erkannte RTP-Pakete ignoriert. Damit die Reduzierung des *Congestion Windows* auf eine RTT limitiert wird.
  Die Verlusterkennung basiert auf der Liste der empfangenen Sequenznummern. Mit dieser und der Liste der bereits übertragenen RTP-Pakete kann geprüft werden, ob Pakete unterwegs verloren gegangen sind.
  In einem Netzwerk kann es zu einem Szenario kommen, in dem RTP-Pakete mit einer kleineren Sequenznummer später beim Empfänger ankommen, als RTP-Pakete die zuletzt abgeschickt worden sind. Damit dies fälschlicherweise nicht zu einer Verlusterkennung führt, implementiert die *Congestion Control* ein *Reordering Window*. Das *Reordering Window* bildet eine Zeiteinheit ab und dient als Timer. Grundlage hierfür bildet die Liste von bereits empfangenen Sequenznummern. Sobald ein RTP-Paket als verloren erkannt wurde, wird in der Liste der übertragenen RTP-Pakete ein Verlustbit für das jeweilige RTP-Paket gesetzt. Wenn durch eine später ankommende Feedback-Nachricht, ein zuvor als verlorengegangenes RTP-Paket als eingetroffen markiert wurde, wird das *Reordering*

Window zurückgesetzt.
Ein Verlust wird erkannt und als dieser veröffentlicht, wenn ein RTP-Paket nicht innerhalb eines Zeitfensters (*Reordering Window*) bestätigt wurde, unter der Voraussetzung, dass zuvor ein RTP-Paket mit einer höheren Sequenznummer bestätigt wurde.

- *Fast Start*: Mit dem *Fast Start* erreicht der Durchsatz innerhalb weniger Sekunden die verfügbare Pfadkapazität. Die Methode besteht aus zwei Komponenten, eine funktioniert wie der *Slow-Start* von TCP und die andere Komponente lässt die Bitrate in der Anfangsphase schneller ansteigen.

3.3.2.2 Congestion Window Berechnung

Die *Congestion Window* Berechnung basiert auf dem *Queuing Delay* und der Verlusterkennung. Das *Congestion Window* darf erhöht werden, sofern das *Queuing Delay* unter einer vorher definierten Grenze liegt, andernfalls muss es reduziert werden. Diese Grenze nennt sich *Queuing Delay Target* und ist in SCReAM auf 100ms eingestellt.
Bei einem Verlust von RTP-Paketen, wird das *Congestion Window* sofort reduziert.

Die Abbildung 3.4 zeigt, in Form eines Aktivitätsdiagramm, den Ablauf der *Congestion Window* Berechnung. Nach jeder empfangenen RTCP-Feedback-Nachricht wird der dargestellte Ablauf durchgeführt.
Bevor es zur eigentlichen Berechnung des *Congestion Windows* kommt, müssen zuvor einige Variablen berechnet und aktualisiert werden. Zunächst wird aus der Differenz der Zeitstempel das *One-Way Delay* gebildet. Anschließend wird das *Queuing Delay* aus der Differenz vom *One-Way Delay* und der Basisverzögerung berechnet. Von den *bytes-in-flight* müssen die neu bestätigten Bytes abgezogen werden und es wird ein Faktor *offTarget* aus dem *Queuing Delay* und dem *Queuing Delay Target* gebildet, der später für die Berechnung vom *Congestion Window* benötigt wird.

Mit der ersten Entscheidung wird geprüft, ob bereits Verluste aufgetreten sind. Wurden Verluste bereits erkannt und als solche veröffentlicht, wird das *Congestion Window* mit einem Faktor multipliziert, der eine sofortige Reduzierung von diesem erwirkt. In SCReAM ist der *lossBeta* Faktor standardmäßig mit “0.8“ initialisiert. Mit Auftreten von Verlusten, wird auch der *Fast Start* Modus verlassen, indem die Variable auf *false* gesetzt wird.
Sind noch keine Verluste aufgetreten, folgt eine weitere Entscheidung. Sofern das *Queuing Delay* kleiner als die Hälfte des *Queuing Delay Targets* ist und die *Congestion Control* sich noch

im *Fast Start* befindet, wird das *Congestion Window* um die *bytes-newly-acked* inkrementiert. Trifft eine dieser Bedingungen nicht zu, sind bereits zuvor Verluste aufgetreten oder das *Queuing Delay* übersteigt die zulässige Grenze für ein aggressives Ansteigen des *Congestion Windows*. In diesem Fall folgt eine weitere Prüfung des zuvor berechneten *offTargets*. Ist das *offTarget* größer als "0", befindet sich das *Queuing Delay* unterhalb des *Queuing Delay Targets*. Dementsprechend wird das *Congestion Window* um einen sehr viel weniger aggressiven Faktor *increment* erhöht. Die für die Berechnung verwendete Variable *gainUp*, ist in SCReAM parametrierbar und ist standardmäßig mit "1" initialisiert und hat dementsprechend derzeit keinen Einfluss. Je nach Wunsch des Steigungsverhaltens in Lastsituation, kann der Wert beliebig angepasst werden.
Trifft die zuvor abgeprüfte Bedingung nicht zu, hat das *Queuing Delay* die Grenze des *Queuing Delay Targets* überschritten. Das *Congestion Window* wird um das Minimum, aus einem zuvor berechneten Faktor *delta* und einem Viertel des *Congestion Windows*, dekrementiert.
Nach jeder Aktualisierung des *Congestion Windows* erfolgt als letzter Schritt eine Limitierung, damit dieses in den zulässigen Grenzen bleibt. Für die Limitierung nach oben, wird das Minimum aus dem *Congestion Window* und den (*maxBytesInFlight* * 1.5) ermittelt. Die *maxBytesInFlight* setzen sich aus der maximalen Anzahl an *bytes-in-flight* über die letzten 5s zusammen. Für die Limitierung nach unten, wird das Maximum aus dem *Congestion Window* und dem minimalen *Congestion Window* (cwndMin) festgelegt.

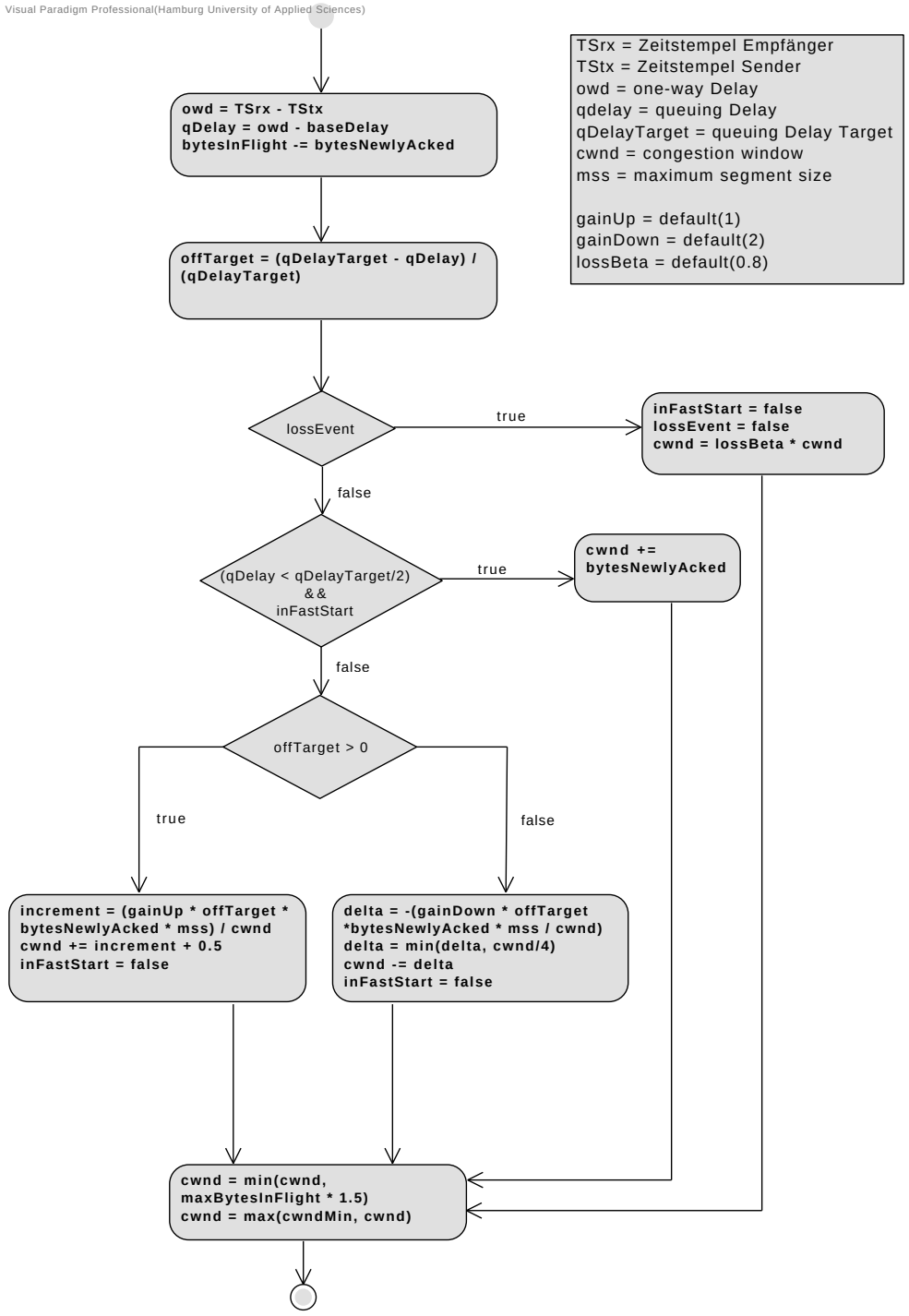


Abbildung 3.4: Aktivitätsdiagramm: *Congestion Window* Berechnung

3.3.2.3 Sendefenster Berechnung

Das Sendefenster wird aus der Relation zwischen dem *Congestion Window* und den *bytes-in-flight* gebildet und gibt an, wie viele Bytes noch in das Netzwerk gegeben werden dürfen.

```
calculate_send_window(qDelay, qDelayTarget)
  if (qDelay <= qDelayTarget)
    send_wnd = cwnd + MSS - bytesInFlight
  else
    send_wnd = cwnd - bytesInFlight
```

Listing 3.1: Codebeispiel: Sendefenster Berechnung

Die Berechnung (3.1) erfolgt abhängig vom aktuellen *qDelay* und dem *qDelayTarget*. Ist das *qDelay* kleiner als das *qDelayTarget*, wird der Relation aus cwnd und *bytes-in-flight* eine *Maximum Segment Size* (MSS) hinzugefügt. Dadurch kann die Bitrate schneller erhöht werden, wenn keine Überlast erkannt wurde.
Ist das *qDelay* größer als das *qDelayTarget*, wird eine strengere Regel angewandt, um Ansteigen vom *Delay* im Netzwerk zu verhindern.
Das Sendefenster wird, nach jeder Übertragung eines RTP-Pakets und sobald eine RTCP-Feedback-Nachricht empfangen wurde, aktualisiert.

3.3.3 Transmission Scheduler

Der *Transmission Scheduler* ist eine zentrale Komponente im SCReAM Framework und ist für das Senden von RTP-Paketen verschiedenster Datenquellen zuständig. Um dem Prinzip der *Packet Conservation* nachzukommen, limitiert der *Transmission Scheduler* die Ausgabe der Daten. Um dies zu realisieren nutzt er das von der *Congestion Control* bereitgestellte Sendefenster, die *bytes-in-flight* und das *Congestion Window*.
Außerdem implementiert der *Transmission Scheduler* zwei wichtige Features:

- *Packet Pacing*: Wird verwendet um Paketverluste und Netzwerk Jitter, ausgelöst durch burstartige Datenschübe, zu verringern.
  Auch wenn das Sendefenster die Übertragung von RTP-Paketen erlaubt, werden diese nicht sofort übertragen, sondern in Intervallen. Die *Packet Pacing* Intervalle werden durch das *Congestion Window* und der geschätzten *Round Trip Time* berechnet (3.2).

```
        pacingBitrate = max(RATE_PACE_MIN, cwnd * 8 / s_rtt)
        paceInterval = (MSS * 8) / pacingBitrate
```

Listing 3.2: Codebeispiel: *Packet Pacing* Intervallberechnung

- *Congestion Window* Überschreitung: Erlaubt es mehr Daten zu übertragen als das *Congestion Window* vorgibt. Voraussetzung ist ein geringes *One-Way Delay*. Die Anzahl an zusätzlich zu übertragenen Bytes wird berechnet aus der Relation des *One-Way Delays* und dem *Queuing Delay Target*. Diese Eigenschaft verbessert die Performance im Umgang mit Videocodecs, die eine variable Bitrate erzeugen, bricht aber mit dem Prinzip der *Packet Conservation*.

3.3.4 Media Rate Control

Die *Media Rate Control* passt die Medienbitrate vom *Media Encoder*, an die sich ständig wechselnden Netzwerkbedingungen, an. Das Ziel ist ein Gleichgewicht zwischen einer geringen Sender-*Queue* und der Menge an gesendeten Daten, die den Datenpfad auslasten, herzustellen. Eine zu vorsichtige Einstellung der Medienbitrate führt zu einer Unterauslastung des Datenpfades und zu einem möglichen Verhungern des Stroms. Eine zu aggressive Einstellung hingegen führt zu einem erhöhten Netzwerk Jitter und Paketverlusten.
Der Algorithmus wird in einem Intervall ausgeführt, damit schnell auf eine Reduzierung der verfügbaren Pfadkapazität reagiert werden kann, um so ein Ansteigen der Sender-*Queue* zu vermeiden.
Die Medienbitrate wird auf Basis der Größe der RTP-Queue, dem *Queuing Delay* und auftretenden Verlusten berechnet. Erreicht das *Queuing Delay* einen Grenzwert, wird die Medienbitrate um einen vorher berechneten Faktor reduziert. Der Faktor wird aus der aktuellen Verzögerung der RTP-Queue berechnet. Wurden Verluste erkannt oder hat das *One-Way Delay* das *Queuing Delay Target* überschritten, kann die Medienbitrate sofort reduziert werden.
Die *Media Rate Control* implementiert zwei wichtige Funktionen, um die RTP-Queue bei Bedarf schnell zu leeren. Zum einen können bereits kodierte RTP-Pakete aus der RTP-Queue verworfen werden. Oder es können Frames, bevor sie kodiert werden, ausgelassen werden. Die zweite Variante ist der ersten vorzuziehen.

Anzumerken ist, dass der SCReAM *Congestion Control* Algorithmus sich noch in der Standardisierung befindet und im vorgesehenen *Internet-Draft* den Status “experimentell“ hat. Die zuletzt veröffentlichte Version, zur SCReAM *Congestion Control*, ist vom Oktober 2017. Das heißt, dass die in diesem Kapitel vorgestellten Komponenten, Funktionen, sowie Variablen und Einstellungen sich jederzeit noch ändern können.

4 Evaluationsplattform

Um eine Evaluation durchführen zu können, bedarf es einer geeigneten Plattform. In diesem Kapitel wird die Simulationsumgebung OMNeT++ kurz vorgestellt.
Darüber hinaus werden die für diese Arbeit relevanten Evaluationseinstellungen beschrieben. Außerdem wird die Grundstruktur der Experimente vorgestellt. Anschließend wird die Art der Messungen und Darstellung erläutert. Zuletzt wird kurz auf die Implementation eingegangen, mit anschließender Validierung.

4.1 OMNeT++/INET

Eine geeignete Simulationsumgebung für die Evaluation einer *Congestion Control* von WebRTC sollte über eine bereits fertige Implementation der Protokolle IP, UDP, RTP und TCP verfügen. Des Weiteren sollten verschiedene Testaufbauten in einer vollständig kontrollierbaren Umgebung realisiert werden können.

OMNeT++ [9] ist ein diskreter Eventsimulator, der primär für die Netzwerksimulationen entwickelt wurde. Mit dieser Art der Simulation lässt sich das Verhalten und die Leistung von realen Prozessen nachbilden. Dies geschieht auf Basis von Nachrichten, Warteschlangen und Zufallszahlen. Objekte kommunizieren über Nachrichten mit anderen Objekten, aber auch mit sich selbst, um Timer zu realisieren. Die Nachrichten werden in Warteschlangen vorgehalten und zu bestimmten Zeiten ausgeliefert. Eine Variation der Messergebnisse werden durch Zufallszahlen realisiert.
OMNeT++ bietet vorzugsweise eine komponentenbasierte Architektur für Modelle. Die Komponenten werden in C++ entwickelt und mittels einer Netzwerkbeschreibungssprache (NED) zu größeren Komponenten zusammengefügt. Eine .ini Datei beschreibt und parametriert die Experimente. Mit dieser lassen sich mehrere Durchläufe mit verschiedenen Parametern, schnell und einfach, realisieren.
OMNeT++ selbst ist kein Netzwerksimulator und kann die am Anfang der Sektion genannten Anforderungen nicht abdecken. Erst mit dem INET [8] Framework kommen die, für diese Arbeit wichtigen Funktionen, hinzu. Das INET Framework ist eine quelloffene Bibliothek für

die OMNeT++ Simulationsumgebung. INET bietet Protokolle und eine Infrastruktur, die uns das Simulieren von verkabelten, kabellosen und mobilen Netzwerken erlaubt. Das Framework bietet sich besonders an, um neue Protokolle, oder wie in dieser Arbeit einen *Congestion Control* Algorithmus, zu evaluieren.

4.2 Evaluationseinstellungen

In dieser Sektion werden die empfohlenen Einstellungen für die Experimente [11] und die Validierung beschrieben. Außerdem werden die Eigenschaften, mit denen ein Pfad parametriert werden, kann vorgestellt.

4.2.1 Metriken

Für die Evaluation und Validierung der SCReAM *Congestion Control* müssen genug Informationen über die Zeit ermittelt werden, um diese anschließend zu visualisieren. Für diese Arbeit relevante Metriken sind:

- *One-Way Delay* vom Medienstrom unter der SCReAM *Congestion Control*
- Video-Verzögerung, entstanden durch die RTP-Queue
- Feedback-Nachrichten Overhead
- Eingangs-Bitrate vom Videocodec
- Sende-Bitrate und Durchsatz
- Verfügbare Pfadkapazität

4.2.2 Pfad Eigenschaften

Jedes Experiment benötigt eine eigene Testbed-Topologie. Die Abbildung 4.1 repräsentiert die Basis. In dieser Topologie sind S_1-S_n Sender. Diese generieren *Traffic* und sind unter Kontrolle der SCReAM *Congestion Control*. Analog dazu bilden E_1-E_n die dazugehörigen Empfänger.
Ein Experiment kann einen oder mehrere Sender haben und die dazugehörigen Empfänger. Wenn nicht anders spezifiziert, werden die Medienströme vom Sender über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den mit *Rückwärts* markierten Pfad.

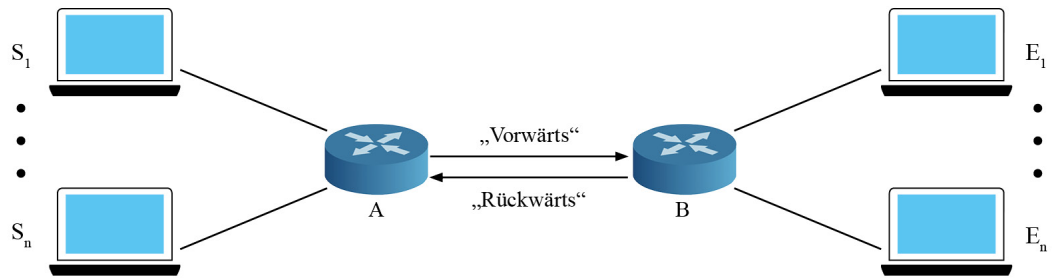


Abbildung 4.1: Beispiel: Einer Testbed-Topologie

Jeder Pfad zwischen einem Sender und einem Empfänger kann, die für die Validierung und Evaluation spezifizierten Eigenschaften, haben:

- *Propagation Delay*: 10ms, 50ms oder 100ms. Das *Propagation Delay* beschreibt das *One-Way Delay* auf dem Pfad, sofern die Buffer der Router leer sind.
- Maximum Ende-zu-Ende Jitter: 5ms.
- *Queue-Management*: *Drop Tail* und *Random Early Detection* (RED).
  - *Drop Tail* ist ein einfacher Mechanismus, der von Routern benutzt wird, um Pakete zu verwerfen. Sobald der Buffer die maximale Kapazität erreicht hat, werden neu ankommende Pakete verworfen, bis der Buffer wieder genügend Platz hat.
    Ein Problem von *Drop Tail* ist die Synchronisierung verschiedener Ströme. Ankommende Pakete aller Verbindungen werden verworfen, was eine gemeinsame Reduzierung des *Congestion Windows* zur Folge hat und somit zu einer Unterauslastung der Pfadkapazität führt.
  - RED ist ein Mechanismus, der zum *Active Queue Management* gehört und soll Überlast verhindern, indem Pakete vor Erreichen der Kapazitätsgrenze bereits verworfen werden. Der Mechanismus arbeitet auf Basis der durchschnittlichen Buffergröße und verwirft Pakete anhand von statistischen Informationen. Ist der Buffer leer, werden alle ankommenden Pakete akzeptiert. Mit ansteigender Buffergröße, erhöht sich auch die Wahrscheinlichkeit, dass Pakete verworfen werden. Erreicht die Buffergröße die Grenze, werden alle neu ankommenden Pakete verworfen.
- Puffergröße: 300ms oder 1000ms.

- Pfadverlust: Beschreibt das Verlustmodell, das eingesetzt wird, um Verluste auf dem Pfad zu erzeugen. Falls nicht anderes spezifiziert, wird für die Experimente der Evaluation kein Verlustmodell eingesetzt.

Für Experimente mit variabler Pfadkapazität über die Zeit, gibt es zwei Mechanismen, um diese zu realisieren:

1. Durch explizites Ändern des Wertes der physikalischen Pfadkapazität. Für diesen Anwendungsfall bietet das INET Framework den *ScenarioManager*. Dieser nimmt ein XML Skript entgegen und führt die Änderungen der Pfadkapazität zur Simulationszeit aus.

2. Durch einen nicht adaptiven UDP-Strom, der über die Zeit seine Datenrate ändert. Der UDP-Strom hat den Nachteil, dass er zusätzlich die Buffer im Netzwerk füllt.

Wenn es in den Experimenten nicht anders beschrieben wurde, wird die variable Pfadkapazität durch das explizite Ändern des Wertes der physikalischen Pfadkapazität, durch den *ScenarioManager*, realisiert.

4.3 Struktur der Experimente

Mit einer einheitlichen Struktur lassen sich neue Experimente definieren, die für andere reproduzierbar und vergleichbar sind. Außerdem wird ein besserer Überblick gewährt und Attribute müssen nicht für jedes Experiment neu erklärt werden. Aus diesen Gründen verfolgen alle Experimente dieser Evaluation, mit leichten Änderungen, die Basisstruktur aus [11]:

- Erläuterung des Experiments: Beschreibt die Motivation und das Ziel des Experiments.
- Beschreibung der Testbed-Topologie: siehe Sektion 4.2.2
- Definieren der Testbed-Attribute:
  - Versuchsdauer: Definiert die Dauer des Experiments in Sekunden.
  - Pfadeigenschaften: Definiert die Pfadeigenschaften des Testbeds für ein bestimmtes Experiment. Beschrieben werden, unabhängig voneinander, die Attribute für den *Vorwärts* Pfad und die vom *Rückwärts* Pfad. Wenn nur ein Pfad spezifiziert wird, gilt dies für beide Pfade. Zu den in Sektion 4.2.2 genannten Eigenschaften kommen noch drei weitere Attribute, die vorher spezifiziert werden.
    - Übertragungsrichtung: *Vorwärts* oder *Rückwärts*.

        * *Bottleneck*-Pfadkapazität: Gibt die Kapazität vom *Bottleneck*-Pfad an.
        * Referenz *Bottleneck*-Pfadkapazität: 1 Mbit/s. Definiert einen Referenzwert für die *Bottleneck*-Pfadkapazität für Experimente, in denen sich die Pfadkapazität zur Simulationszeit ändert. Alle *Bottleneck*-Pfadkapazitäten werden im Verhältnis zum Referenzwert angegeben.

- Anwendungsbezogene Informationen: Definiert das Verhalten der verschiedenen Quellen.
  - Medienquellen
    * Medientyp: Simulation von Videodaten durch synthetischen adaptiven Codec.
    * Medienstrom Richtung: *Vorwärts*, *Rückwärts* oder beides.
    * Anzahl Medienströme: Definiert die maximale Anzahl an Medienströmen für ein bestimmtes Experiment.
    * Medienstrom Zeitplan: Beschreibt die Zeitpunkte wann ein Medienstrom mit seiner Übertragung beginnt und wann er diese beendet.
  - Konkurrierende Datenquellen:
    * Datenstrom Richtung: *Vorwärts*, *Rückwärts* oder beides.
    * Typ der Datenquelle: Definiert den Typ der konkurrierenden Datenquelle.
    * Anzahl Datenströme: Definiert die maximale Anzahl an konkurrierenden Datenströmen für ein bestimmtes Experiment.
    * Datenstrom Zeitplan: Beschreibt die Zeitpunkte wann ein Datenstrom mit seiner Übertragung beginnt und wann er diese beendet.
- Erwartungswert an das Experiment: Beschreibt das erwartete Verhalten des Experiments.
- Erwartungswert an die Messergebnisse: Beschreibt das erwartete Verhalten an die eigenen Messergebnisse.
- Ist-Zustand: Vorstellung der eigenen Messergebnisse.

4.4 Messung und Visualisierung

Die Messung von Daten erfolgt mit dem von OMNeT++ bereitgestellten Objekt *cOutVector*. Mit diesem lassen sich Werte mit der aktuellen Simulationszeit als Zeitstempel, in der Applikation

selbst, aufzeichnen. Die Ausgabe der gesammelten Werte erfolgt in einer Vektordatei. Diese lässt sich nach beendeter Simulation betrachten, aber auch in andere Datenformate exportieren.

Die Messwertaufnahme, der zuvor vorgestellten Metriken, erfolgt mit dem von der RMCAT Arbeitsgruppe vorgeschlagenen [43] Intervall von 200ms. Mit dieser Granularität lassen sich auch kurze Schwankungen und Spitzen aufzeichnen. Für jedes Experiment werden zehn Durchläufe aufgezeichnet. Jeder Durchlauf wird mit einem anderen *Seed* durchgeführt. Ein *Seed* ist ein Wert, mit dem ein Zufallszahlengenerator initialisiert wird. Dieser erzeugt mit dem *Seed* als Startwert eine Folge von Zufallszahlen und ermöglicht uns in den Experimenten eine größere Variation der Messwerte unter verschiedenen Bedingungen.

OMNeT++ selbst bietet Funktionalitäten an Messwerte zu verarbeiten und in Diagrammen darzustellen. Diese sind aber eher rudimentär und decken die Anforderungen an eine geeignete Visualisierung von Messwerten für eine wissenschaftliche Arbeit nicht ab. Aus diesem Grund werden die Datenvorverarbeitung und die Visualisierung in Python realisiert.
Python ist eine Skriptsprache und eignet sich für die Bearbeitung und Darstellung von einer kleinen Anzahl an Messwerten. Für diesen Zweck werden die Vektordateien aus OMNeT++ in Python-Skripte exportiert. Innerhalb dieser, werden die Werte in einem *Numpy Array* gespeichert.
Die Metriken (Ausgangs-Bitrate, Sende-Bitrate, Durchsatz) werden als Variation über die Zeit visualisiert. Die Messungen in den Plots erfolgen in 1-Sekunden-Intervallen. Aus diesem Grund wird zuvor für jeden Durchlauf der gleitende Median über fünf Werte, also eine Sekunde berechnet. Anschließend wird der Mittelwert über die zehn Durchläufe berechnet und mit der Python Bibliothek *Matplotlib* geplottet. Damit keine wichtigen Spitzen verloren gehen, wird für die restlichen Metriken lediglich der Mittelwert über die Durchläufe berechnet.

Die Messwerte der zehn Durchläufe und der berechnete Mittelwert decken nur eine kleine Stichprobe von unendlich vielen Messwerten ab. Der Erwartungswert kann erheblich vom berechneten Mittelwert abweichen. Für diesen Zweck bietet sich die Berechnung von einem Konfidenzintervall, auch Vertrauensintervall genannt, an. Mit einem zuvor definierten Konfidenzniveau lässt sich eine Ober- und Untergrenze für einen Mittelwert berechnen. In den Grenzen liegt mit der Wahrscheinlichkeit des Konfidenzniveaus der wahre Mittelwert. Für diese Arbeit wird das Konfidenzintervall mit einem 95%-Konfidenzniveau berechnet. Das heißt, dass mit einer Wahrscheinlichkeit von 95%, bei einer Wiederholung der Messreihe, der neue Mittelwert zwischen den Grenzen liegt und mit einer Wahrscheinlichkeit von 5% tut er es

nicht. Die Berechnung vom Konfidenzintervall erfolgt für die Metriken (Ausgangs-Bitrate, Sende-Bitrate, Durchsatz). Die Grenzen werden in Form von Fehlerlinien, vom Ausgangspunkt aus, angegeben.

4.5 Integration ins Simulationsmodell

Das INET Framework implementiert die, für die Durchführung der Experimente, nötigen Protokolle und Infrastrukturen. Allerdings bietet sowohl OMNet++, als auch das INET Framework keine Implementation des SCReAM Algorithmus an. Insofern war es Bestandteil dieser Arbeit, die Referenzimplementierung [6] vom SCReAM Algorithmus in das von INET bereitgestellte RTP-Modell zu integrieren.
Für diesen Zweck mussten sämtliche Nachrichten und Nachrichtenflüsse aufgebrochen werden. Der Algorithmus musste daran angepasst werden, dass Objekte lediglich über Nachrichten kommunizieren und *Timer* ebenfalls in Nachrichten realisiert werden. Die genannten Probleme sind nur ein Bruchteil, die bei der Integration aufkamen.
Die Implementierung [44] liegt in einem INET *Fork* und ist frei zugänglich. Anzumerken ist, dass das INET Framework aus mehreren Teilen besteht, von denen jeder Teil eine eigene Lizenz hat. Die Lizenz besteht entweder aus einer *GNU General Public License* (GPL) oder einer *GNU Lesser General Public License* (LGPL).

4.5.1 Verwendete Codecs

Bevor rohe Audio- und Videoströme verschickt werden, müssen diese aus Effizienzgründen kodiert werden. OMNeT++ und das INET Framework bieten keine eigenen Codecs an. Aus diesem Grund wurden Syncodecs [45] von Cisco ebenfalls in das Projekt integriert. Diese wurden extra für die Evaluation von RMCAT *Congestion Control* Kandidaten entwickelt.
Die Syncodecs sind adaptive, synthetische Codecs und wurden in C++ geschrieben. Das besondere an synthetischen Codecs ist, dass sie keine echten Videodaten kodieren, sondern nur eine Sequenz von Paketen bereitstellen, die sich mit ihrer jeweiligen Größe an die zuvor eingestellte Bitrate anpassen. Sie simulieren einen echten Codec und dienen gleichzeitig als Datenquelle, dessen Bitrate über die Zeit frei einstellbar ist.
Für die Evaluation der SCReAM *Congestion Control* kommen zwei Codecs aus den Syncodecs zum Einsatz:

- *Perfect Codec.* Solange die Bitrate stabil ist, erzeugt der *Perfect Codec* keine Schwankungen und kein Rauschen in den Größen der Pakete. Die Sequenz aus Paketen wird genau

der Bitrate entsprechend generiert. Das Problem an diesem Codec ist, dass er keinen Zufall beinhaltet. Mit jedem Durchlauf würde der Codec die gleichen Daten liefern. Er bildet einen idealen Codec ab. Aus diesem Grund eignet er sich für die Validierung der Implementation in OMNeT und wird dafür auch eingesetzt.

- *Statistics Codec.* Der *Statistics Codec* ist durchaus komplexer. Er simuliert einen realen Codec, indem er ein statistisches Modell [46] implementiert.
  Bevor ein Paket versendet wird, wird die Größe des Pakets modifiziert, um ein Rauschen zu simulieren. Die Standardfunktion ändert die Größe von jedem Paket durch Vergrößern oder Verkleinern. Dies geschieht auf Basis einer zufälligen Laplace-Verteilung. Eine weitere Funktion fügt ein Rauschen, ebenfalls auf Basis der Laplace-Verteilung, dem konfigurierten Paketintervall hinzu.
  Aufgrund dieser Grundlage lassen sich mehrere Durchläufe, mit verschiedenen *Seeds*, realisieren. Um eine Variation von Messdaten zu erhalten, wird dieser Codec bei den eigentlich Experimenten eingesetzt.

4.5.2 Validierung

Bevor die eigentliche Evaluation durchgeführt werden kann, ist es notwendig zu prüfen, ob die Integration in das Simulationsmodell erfolgreich war. Dies geschieht auf Basis einer geeigneten Validierung. Dabei ist zu beachten, dass bei der Validierung, so wie bei der Evaluation, nur ein Bruchteil an Parameterkombination getestet werden können.
Für die Validierung der SCReAM Implementation in OMNeT++ werden die ersten beiden Experimente aus [11] durchgeführt und mit den Messerergebnisse aus der Evaluation [47] von Ericsson Research verglichen.

Die Plots der Experimente für die Validierung und für die eigentliche Evaluation sind jeweils unterteilt in drei weitere Subplots. Der erste Subplot zeigt den Durchsatz der Medien- und Datenströme in Kilobits pro Sekunde. Im zweiten Subplot wird die Video Verzögerung in Sekunden dargestellt. Die Video-Verzögerung setzt sich aus der Anzahl an RTP-Paketen aus der RTP-*Queue* zusammen und repräsentiert das Delay zwischen dem Audiostrom und dem Videostrom (siehe Sektion 3.3.1). Der dritte Subplot zeigt die Ende-zu-Ende Netzwerk-Verzögerung in Sekunden, die sich aus dem *Propagation Delay* und dem *Queuing Delay* zusammensetzt. Für jeden Subplot werden die Messwerte über die Versuchsdauer in Sekunden dargestellt.

4.5.2.1 Variable Pfadkapazität mit einem SCReAM-Strom

In diesem Experiment variiert die *Bottleneck*-Pfadkapazität über die Zeit. Entwickelt wurde das Experiment, um folgende Szenarien abzubilden:

- Ein *Bottleneck* in der Übertragung
- Änderungen in der verfügbaren Pfadkapazität (ausgelöst durch einen Interface-Wechsel, Routing-Änderungen oder durch einen nicht adaptiven UDP-Strom)
- Maximale Medien-Bitrate ist größer als die Pfadkapazität

Mit diesem Experiment soll versucht werden die Anforderungen 1, 2, 3 und 6 aus der Sektion 1.3 abzudecken. Das Ziel ist die Reaktionsfähigkeit des SCReAM *Congestion Control* Algorithmus zu messen.

Testbed-Topologie (siehe Abbbildung 4.2): Die Medienquelle S_1 ist mit dem zugehörigen Empfänger E_1 verbunden. Der Medienstrom wird über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den mit *Rückwärts* markierten Pfad.

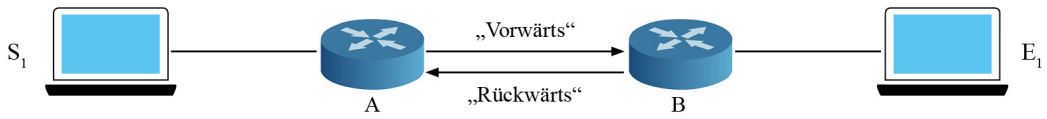


Abbildung 4.2: Testbed-Topologie für variable Pfadkapazität mit einem SCReAM-Strom

Testbed-Attribute:

- Versuchsdauer: 100s
- Pfad-Eigenschaften:
  - *Propagation Delay*: 10ms
  - Queue-Management: *Drop Tail* mit einer Buffergröße von 300ms

  - *Bottleneck*-Pfadkapazität: Variable Pfadkapazität (siehe Tabelle 4.1)

- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts*
    - Anzahl Medienströme: 1
    - Medienstrom Zeitplan:
      - Startzeit: 0s
      - Endzeit: 100s
  - Konkurrierende Datenquellen
    - Anzahl Datenströme: 0

Pfadkapazitätsänderung	Pfadrichtung	Startzeit [s]	Pfadkapazität [kbit/s]
Eins	*Vorwärts*	0	1000
Zwei	*Vorwärts*	20	2500
Drei	*Vorwärts*	40	600
Vier	*Vorwärts*	60	1000

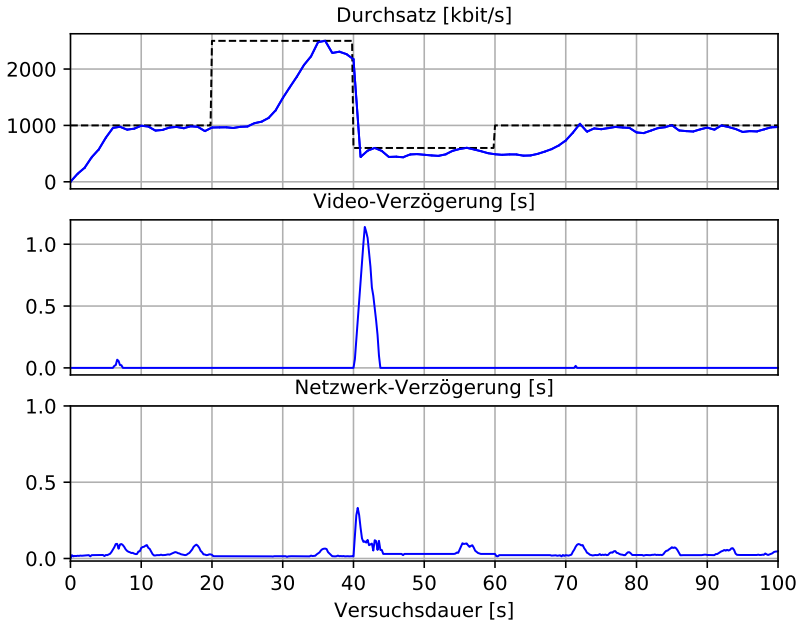
Tabelle 4.1: Ablauf: Variable Pfadkapazität mit einem SCReAM-Strom

Erwartungswert an das Experiment: Es wird erwartet das der SCReAM Algorithmus die maximale Pfadkapazität erkennt und seine Senderate an die verfügbare *Bottleneck*-Pfadkapazität anpasst. Zusätzlich wird erwartet, dass keine großen Schwankungen auftreten, sobald die Senderate die verfügbare Pfadkapazität erreicht.

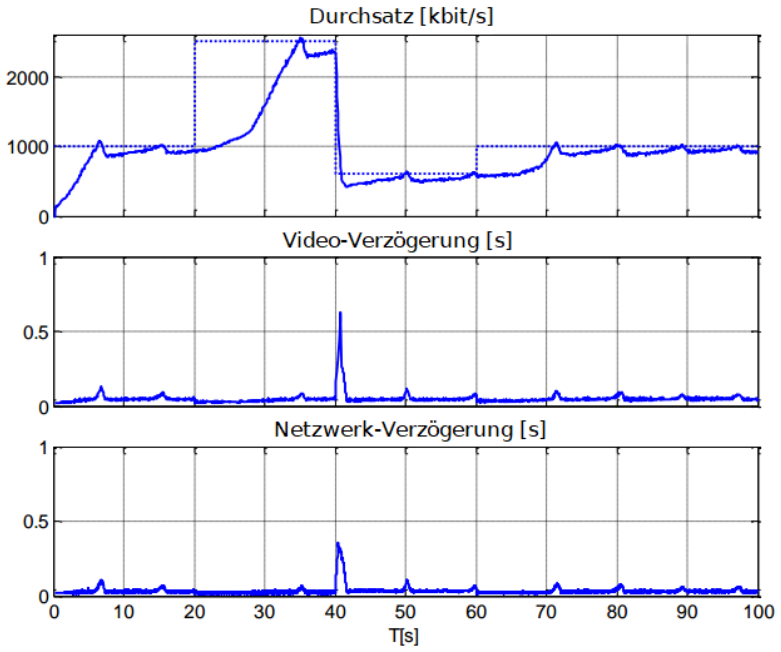
Erwartungswert an die Messergebnisse: Von den eigenen Messergebnissen wird erwartet, dass sie ein vergleichbares Verhalten bei einer variablen *Bottleneck*-Pfadkapazität zeigen, wie die visualisierten Messergebnisse 4.3(b), aus der Evaluation durch Ericsson Research.
Die gestrichelte Linie im ersten Subplot stellt die variable Pfadkapazität nach Tabelle 4.1 dar. Mit dem Beginn der Übertragung wird die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf 1000 kbit/s eingestellt. Der SCReAM-Strom erreicht, mit einem linearen Steigungsverhalten, innerhalb von 7s die verfügbare Pfadkapazität von 1000 kbit/s. Mit der Pfadkapazitätsänderung **Zwei**, nach Tabelle 4.1, ändert sich die verfügbare Pfadkapazität auf 2500 kbit/s. Der Änderung entsprechend zeigt der Graph im Durchsatz zunächst ein exponentielles Steigungsverhalten, welches ab 8s, in ein annähernd lineares Steigungsverhalten

übergeht.
Nach einer Reduzierung der Pfadkapazität, ausgelöst durch Änderung **Drei**, benötigt der Algorithmus Zeit, um sich an die neuen Netzwerkbedingungen anzupassen. Es werden kurzzeitig mehr RTP-Pakete auf den Pfad gegeben, als dieser gleichzeitig übertragen kann. Aus diesem Grund ist eine Spitze bei der Netzwerkverzögerung zu erkennen. Die Folge eines erhöhten *Queuing Delays*, ist die sofortige Reduzierung des *Congestion Windows*. Dieses Verhalten ist auch im Durchsatz zu erkennen. Der Graph fällt innerhalb einer Sekunde unterhalb der verfügbaren Pfadkapazität und passt sich dieser erneut an.
Die Spitze im zweiten Subplot zeigt den Einsatz der RTP-*Queue*. Der Grund für die Spitze ist das im Kapitel 3 angesprochene *Packet Pacing*, das burstartige Datenübertragungen verhindert, sodass die RTP-*Queue* mit bereits kodierten RTP-Paketen gefüllt wird. Nach der Anpassung des *Congestion Windows* und der Bitrate des *Media Encoders* wird der Datenstau abgearbeitet und die Spitzen aufgelöst.
Die letzte Anhebung der Pfadkapazität erfolgt mit der Änderung **Vier**, die der Graph innerhalb von 11s erreicht hat.
Bei konstanter Pfadkapazität ist der Durchsatz nah an der verfügbaren Pfadkapazität. Allerdings sind immer wieder kurze Spitzen im Durchsatz erkennbar, gefolgt von einer minimalen Reduzierung. Die Ursache für diese, ist die kurzzeitige Überschreitung der Pfadkapazität durch die Senderate und die darauffolgende Reduzierung, sobald die Netzwerk-Verzögerung ansteigt.

Ist-Zustand: Mit der Abbildung 4.3(a) erfolgt die Ergebnisvorstellung der eigenen Messung. Beim Steigungsverhalten, zu Beginn der Übertragung, und bei einem Anstieg der verfügbaren Pfadkapazität, zeigt der Graph das gleiche Verhalten wie der SCReAM-Strom in der Referenzabbildung 4.3(b). Auf eine Reduzierung der Pfadkapazität, durch Änderung **Drei**, fällt der Graph innerhalb einer Sekunde unter die neue Pfadkapazität. Die daraus resultierende Spitze in der Video-Verzögerung und der Netzwerk-Verzögerung treten zum gleichen Zeitpunkt auf, wobei die Spitze in der Video-Verzögerung sein Maximum bei über einer Sekunde hat und breiter ist, als die Spitze in der Referenzabbildung.
Bei einer konstanten Pfadkapazität sind im Durchsatz ebenfalls Spitzen, gefolgt von einem kleinen Einbruch des Graphen, erkennbar. Insgesamt sind die Spitzen nicht so ausgeprägt, wie die in der Referenzabbildung 4.3(b).



(a) Eigene Messung: TC5.1



(b) Referenz: TC5.1, aus [47]

Abbildung 4.3: TC5.1 RTT20 Validierung

Diskussion: Für eine Validierung wird in dieser Diskussion primär auf das Entstehen der Unterschiede zwischen der eigenen Messung und der Referenzmessung eingegangen.
Mit der Beschreibung der visualisierten, eigenen Messwerte wurden schon einige Gemeinsamkeiten identifiziert. Es wurden aber auch Unterschiede festgestellt, dessen Entstehungsgrund geklärt werden muss, bevor es zur eigentlichen Evaluation kommt.
Der erste Unterschied wurde im Subplot der Video-Verzögerung erkannt. Denn das Maximum der Spitze in der eigenen Messung ist breiter und mit einer halben Sekunde deutlich höher als in der Referenzabbildung 4.3(b). Der Grund hierfür liegt bei der Trägheit des verwendeten Codecs, der deutlich langsamer auf eine Anpassung der Bitrate an die neuen Netzwerkbedingungen reagiert. Außerdem passt sich die Senderate deutlich schneller an die neue verfügbare Pfadkapazität an, sodass es länger dauert, den Datenstau wieder abzuarbeiten.
Ein weiterer Unterschied wurde in der Ausprägung der Spitzen im Durchsatz identifiziert. Die Ursache liegt bei der Darstellung der Messwerte über die Zeit, die mit der Berechnung des gleitenden *Medians* erfolgt (siehe Sektion 4.4).

4.5.2.2 Variable Pfadkapazität mit zwei SCReAM-Strömen

Dieses Experiment ist ähnlich aufgebaut wie das vorherige. Um eine durchgängige Netzwerklast zu simulieren, erfolgt der Einsatz eines weiteren, konkurrierenden SCReAM-Stroms.

Testbed-Topologie (siehe Abbbildung 4.4): Die Medienquellen S_1 und S_2 sind mit ihren zugehörigen Empfängern E_1 und E_2 verbunden. Die Medienströme werden über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den mit *Rückwärts* markierten Pfad.

Testbed-Attribute: Die Testbed-Attribute sind die gleichen wie im vorherigen Experiment 4.5.2.1. Mit Ausnahme von:

- Versuchsdauer: 120s
- Anzahl Medienströme: 2
- Referenz *Bottleneck*-Pfadkapazität: 2 Mbit/s
- *Bottleneck*-Pfadkapazität: Variable Pfadkapazität (siehe Tabelle 4.2)

Erwartungswert an das Experiment: Es wird erwartet, dass die SCReAM-Ströme, die Veränderungen der verfügbaren Pfadkapazität erkennen und sich an diese anpassen. Zusätzlich

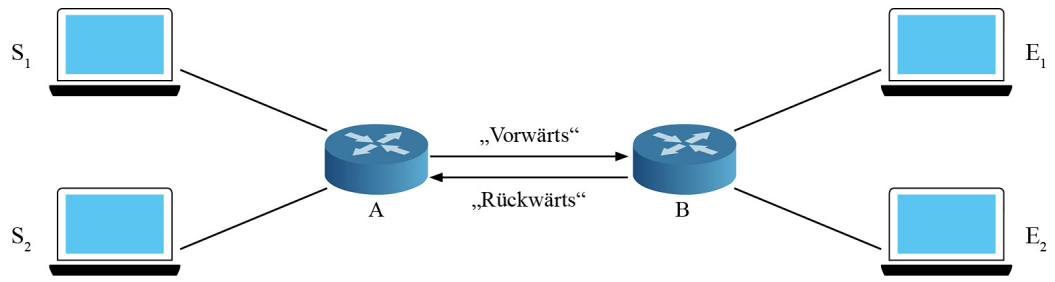


Abbildung 4.4: Testbed-Topologie für variable Pfadkapazität mit zwei SCReAM-Strömen

Pfadkapazitätsänderung	Pfadrichtung	Startzeit [s]	Pfadkapazität [kbit/s]
Eins	*Vorwärts*	0	2000
Zwei	*Vorwärts*	25	1000
Drei	*Vorwärts*	50	1750
Vier	*Vorwärts*	75	500
Fünf	*Vorwärts*	100	1000

Tabelle 4.2: Ablauf: Variable Pfadkapazität mit zwei SCReAM-Strömen

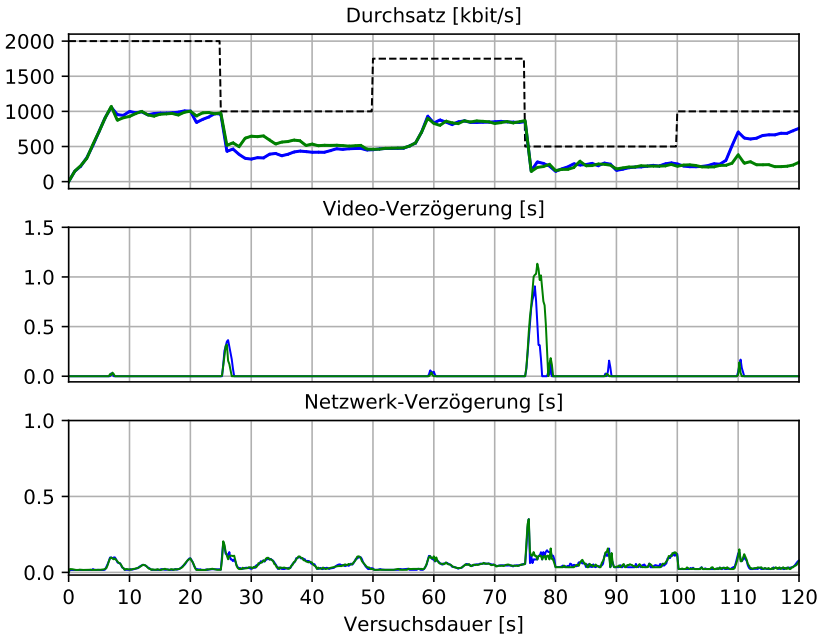
soll der Durchsatz der SCReAM-Ströme keine großen Schwankungen aufweisen. Außerdem sollten die konkurrierenden SCReAM-Ströme sich *Fair* die verfügbare Pfadkapazität teilen.

Erwartungswert an die Messergebnisse: Es wird von den eigenen Messergebnissen erwartet, dass sie ein vergleichbares Verhalten bei einer variablen *Bottleneck*-Pfadkapazität mit zwei konkurrierenden SCReAM-Strömen zeigen, wie die visualisierten Messergebnisse 4.5(b), aus der Evaluation — der alternativen Implementation — durch Ericsson Research.
Die gestrichelte rote Linie im ersten Subplot stellt wiederum die variable Pfadkapazität nach Tabelle 4.2 dar.
Mit dem Beginn der Übertragung wird die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf 2000 kbit/s eingestellt. Die beiden konkurrierenden SCReAM-Ströme zeigen im Durchsatz ein lineares Steigungsverhalten und nehmen nach 7s beide jeweils die Hälfte der verfügbaren Pfadkapazität ein. In Folge einer Reduzierung der verfügbaren Pfadkapazität, nach Änderung **Zwei** und **Vier**, fällt der Graph im Durchsatz, wie auch im vorherigen Experiment, innerhalb einer Sekunde unter die neu verfügbare Pfadkapazität. Bei den Subplots der Video-Verzögerung und der Netzwerk-Verzögerung sind, in Folge der Redu-

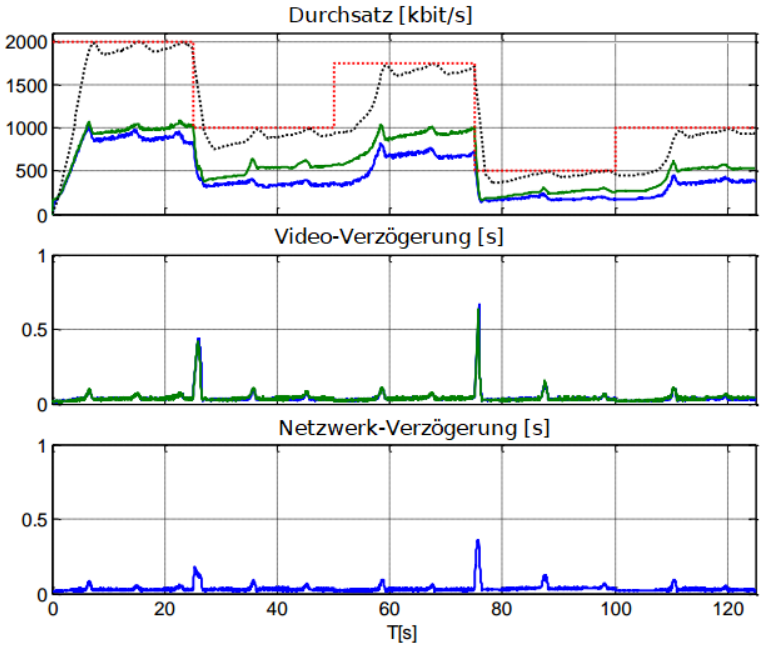
zierung der Pfadkapazität, jeweils zwei ausgeprägte Spitzen zu erkennen.
Bei zusätzlich frei gewordener Pfadkapazität, nach Änderung **Drei** und **Fünf**, zeigen die Graphen der beiden SCReAM-Ströme, jeweils ein exponentielles Steigungsverhalten im Durchsatz. Die SCReAM-Ströme teilen sich die Pfadkapazität nicht exakt 50:50, sondern divergieren bei konstanter Pfadkapazität und bei einem Anstieg nach Änderung **Drei** und **Fünf** .

Ist-Zustand: Die eigene Messung des zweiten Experiments wird mit Abbildung 4.5(a) dargestellt. Zu Beginn der Übertragung steigt der Durchsatz der beiden SCReAM-Ströme linear an und der Graph erreicht die Pfadkapazität nach 9s. Nach einer Reduzierung der verfügbaren Pfadkapazität, wird der Durchsatz ebenfalls innerhalb einer Sekunde an diese angepasst. Nach Änderung **Drei** steigt der Durchsatz exponentiell an.
Insgesamt divergieren die konkurrierenden SCReAM-Ströme weniger als in der Referenzabbildung 4.5(b). Mit Ausnahme der letzten 20s. In Folge der letzten Änderung **Fünf**, divergieren die Graphen im Durchsatz deutlich weiter, sodass die gleichgestellten SCReAM-Ströme sich die Pfadkapazität nicht mehr gleichmäßig aufteilen.

Diskussion: Neben den Unterschieden, die schon im ersten Experiment erläutert wurden, kommt mit dem zweiten Experiment ein weiterer Unterschied hinzu. Nach der Pfadkapazitätsänderung **Fünf** divergiert der Durchsatz der Graphen deutlich weiter als in der Referenzabbildung. Um die Entstehungsursache zu klären und um zu prüfen ob die Graphen noch weiter divergieren, wird das selbe Experiment mit einer Versuchsdauer von 200s erneut durchgeführt und mit der Abbildung 4.6 dargestellt. Der Plot unterscheidet sich von den bisher vorgestellten, im zweiten und dritten Subplot. Der zweite Subplot zeigt die Eingangsbitrate vom Codec und der dritte Subplot das Feedback-Intervall in Millisekunden.
Zu erkennen ist, dass die Graphen nicht weiter divergieren, aber sie erreichen erst nach 160s wieder eine Konvergenz.
Die Graphen der Eingangsbitrate vom Codec zeigen annähernd den gleichen Verlauf wie der Durchsatz. Jedoch die Graphen vom Feedback-Intervall zeigen, dass sie eine direkte Abhängigkeit zur Eingangsbitrate des Codecs haben. Mit einem Anstieg vom Durchsatz, wird das Feedback-Intervall erhöht und umgekehrt. Dennoch kann mit diesen Erkenntnissen keine genaue Ursache für die Divergenz der SCReAM-Ströme identifiziert werden.



(a) Eigene Messung: TC5.2



(b) Referenz: TC5.2, aus [47]

Abbildung 4.5: TC5.2 RTT20 Validierung

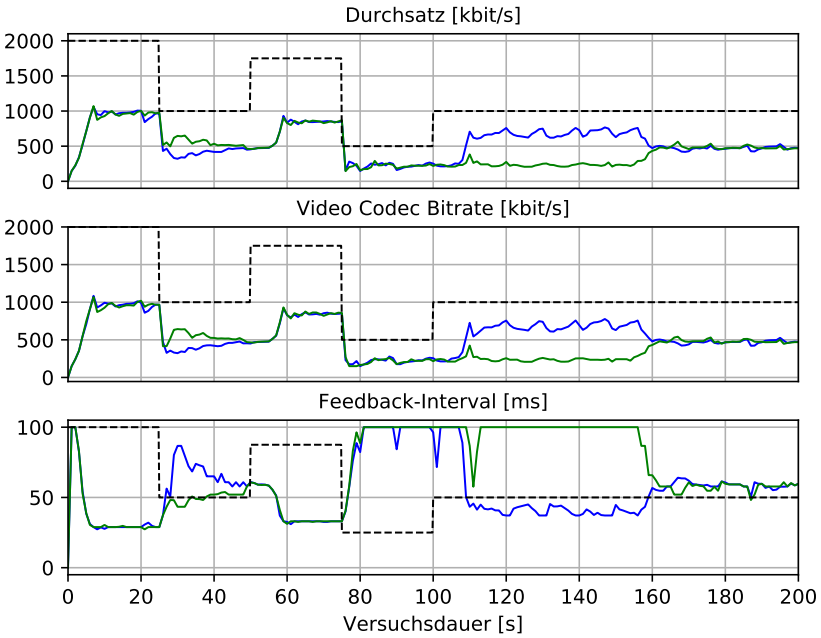


Abbildung 4.6: Eigene Messung: TC5.2, verlängerte Versuchsdauer

Da es das Ziel dieser Arbeit ist, eine geeignete Evaluation der SCReAM *Congestion Control* durchzuführen und keine Verbesserung des Algorithmus, ist die Validierung mit diesen Ergebnissen und Vergleichen der Integration des SCReAM Algorithmus in OMNeT++ abgeschlossen. Das Problem der Divergenz wird in den nachfolgenden Experimenten unter Verwendung eines *Statistic Codecs* mit verschiedenen *Seeds* weiter beobachtet.

5 Experimente

In diesem Kapitel werden die Experimente, die für diese Evaluation relevant sind, vorgestellt. Für jedes einzelne Experiment wird vorweg der Aufbau, der Zweck und die Erwartungshaltung an das Experiment und an die eigenen Messergebnisse beschrieben. Anschließend werden die eigenen Messergebnisse im Detail vorgestellt.

5.1 Variable Pfadkapazität mit einem SCReAM-Strom

Dieses Experiment wurde bereits bei der Validierung in Sektion 4.5.2.1 vorgestellt. Für dieses Experiment ändert sich in der Evaluation lediglich der Parameter des *Propagation Delays*. Das Experiment wurde jeweils mit einem *Propagation Delay* von 50ms und 100ms erneut durchgeführt.

Aufgrund der Änderungen im *Propagation Delay*, ändert sich auch die Erwartungshaltung an die Messergebnisse:
Es wird von den eigenen Messergebnissen erwartet, dass sie ein vergleichbares Verhalten aufweisen, wie die visualisierten Messergebnisse 5.1(b) und 5.2(b), aus der Evaluation durch Ericsson Research.
Die Abbildung 5.1(b) zeigt die Messergebnisse zum ersten Experiment, durchgeführt mit einem *Propagation Delay* von 50ms. Zum Start der Übertragung wird die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf 1000 kbit/s eingestellt. Der Durchsatz steigt zu Beginn annähernd linear an und erreicht nach etwa 8s die verfügbare Pfadkapazität von 1000 kbit/s. Mit der Pfadkapazitätsänderung **Zwei**, nach Tabelle 4.1, wird die Pfadkapazität auf 2500 kbit/s erhöht. Der Graph im Durchsatz zeigt — auf die Änderung — zunächst ein exponentielles Steigungsverhalten, das nach 10s in ein lineares Steigungsverhalten übergeht. Auf die Reduzierung der Pfadkapazität — nach Änderung **Drei** — reagiert der Graph monoton fallend. Der Grund für die sofortige Reduzierung im Durchsatz liegt beim Anstieg der Netzwerk-Verzögerung. Denn der *Transmission Scheduler* sendet zunächst, mit gleichbleibender Senderate, RTP-Pakete an den Empfänger. Damit übersteigt die Senderate die Pfadkapazität,

sodass Router A zum *Bottleneck*-Router wird und RTP-Pakete für den Transport zwischenspeichern muss. Mit dem Überschreiten des *Queuing Delay Targets*, wird das *Congestion Window* reduziert und durch die *Congestion Control* Komponente an die neuen Netzwerkbedingungen angepasst.
In Folge der Reduzierung der Pfadkapazität ist ebenfalls in dem Subplot der Video-Verzögerung ein Anstieg, in Form einer Spitze, zu erkennen. Diese zeigt den Einsatz der RTP-*Queue*, die versucht den *Bottleneck*-Pfad zu entlasten, indem sie RTP-Pakete noch beim Sender zwischenspeichert. Auf Basis der Größe der RTP-*Queue* wird die Bitrate für den *Media Encoder* berechnet, sodass auch dieser an die neuen Netzwerkbedingungen angepasst wird.
Mit der letzten Pfadkapazitätsänderung **Vier** wird diese wieder auf 1000 kbit/s erhöht. Auf die Änderung reagiert der Graph im Durchsatz mit einem annähernd linearen Steigungsverhalten. Insgesamt sind, bei einer konstanten Pfadkapazität, keine großen Schwankungen im Durchsatz zu erkennen.

Da sich bei bei der Versuchsdurchführung, der visualisierten Messergebnissen in Abbildung 5.2(b), nur der Parameter des *Propagation Delays* von 50ms auf 100ms ändert, ähneln sich die Ergebnisse. Aus diesem Grund werden nur die Unterschiede, die zwischen den beiden Referenzabbildungen auftreten, vorgestellt:

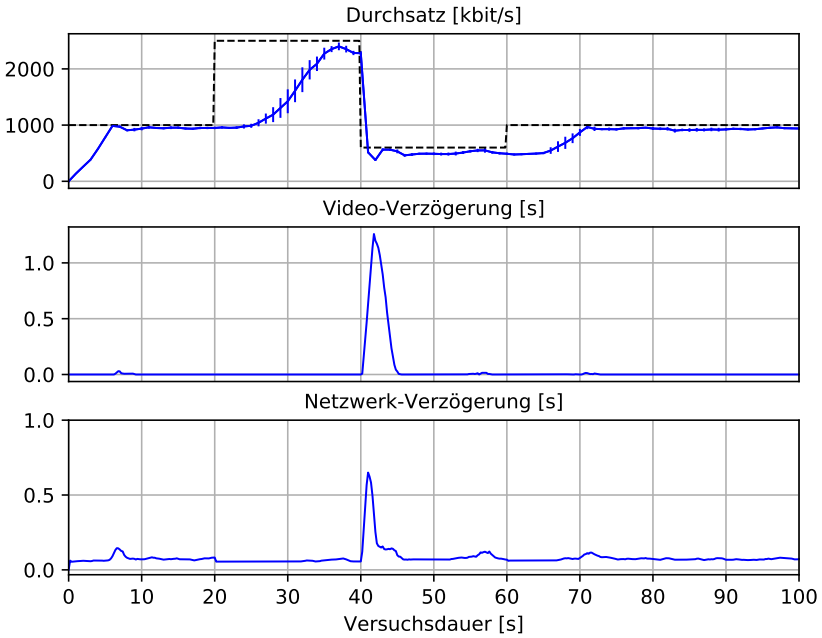
- Der Graph vom Durchsatz erreicht, nach Pfadkapazitätsänderung **Zwei**, etwas verzögert bei 40s die verfügbare Pfadkapazität von 2500 kbit/s.
- Die Rumpf der Spitzen in der Video-Verzögerung und in der Netzwerk-Verzögerung ist deutlich breiter.

Ist-Zustand: Die Abbildung 5.1(a) zeigt die visualisierten Messergebnisse der eigenen Messung mit einem *Propagation Delay* von 50ms.
Auch hier wird zum Start die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf 1000 kbit/s eingestellt. Zu Beginn der Übertragung steigt der Graph im Durchsatz linear an und erreicht die verfügbare Pfadkapazität nach 6s. Auf die Erhöhung der Pfadkapazität, durch Änderung **Zwei**, zeigt der Graph im Durchsatz anfänglich ein exponentielles Steigungsverhalten, welches nach 5s in eine annähernd lineare Steigung übergeht. Nach der Reduzierung der Pfadkapazität — durch Änderung **Drei** — fällt der Graph im Durchsatz, innerhalb einer Sekunde unterhalb der verfügbaren Pfadkapazität und passt sich dieser nach 5s wieder an. Die durch die Reduzierung der Pfadkapazität bedingten Spitzen in der Video/- und Netzwerk-Verzögerung halten jeweils 5s an und haben ihr Maximum bei über einer Sekunde und einer halben Sekunde.

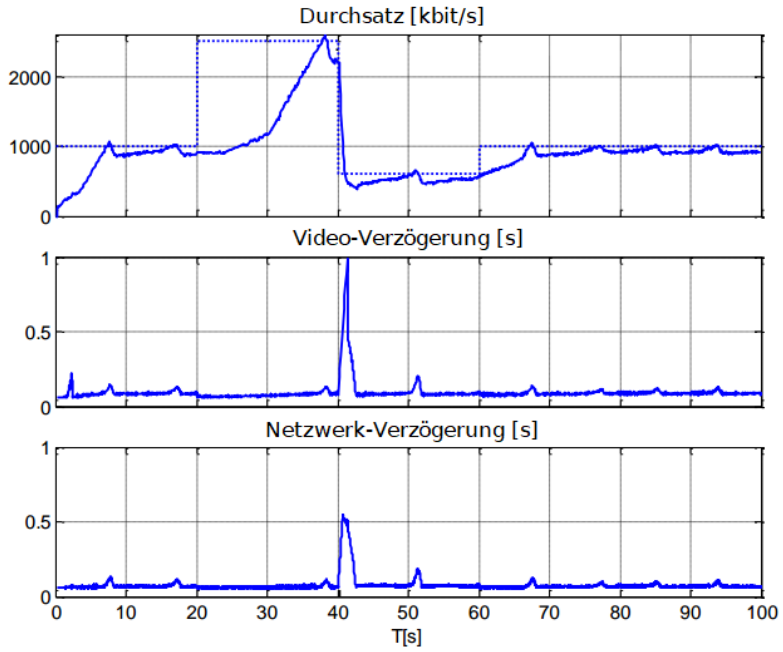
Auf die letzte Pfadkapazitätsänderung, zeigt der Graph nach 5s ein lineares Steigungsverhalten und erreicht nach 11s die neue Pfadkapazität.
In dem Subplot der Netzwerk-Verzögerung ist neben der Spitze eine durchgängige Verzögerung zu erkennen, die das *Propagation Delay* zeigt.

Aufgrund der Ähnlichkeit bei der Versuchsdurchführung werden für die Abbildung 5.2(a) lediglich die Unterschiede zur Abbildung 5.1(a) vorgestellt:

- Auf die Pfadkapazitätsänderung **Vier** reagiert der Graph im Durchsatz direkt mit einem linearen Steigungsverhalten.
- Die Spitze in der Netzwerk-Verzögerung hat ihr Maximum bei über einer Sekunde.

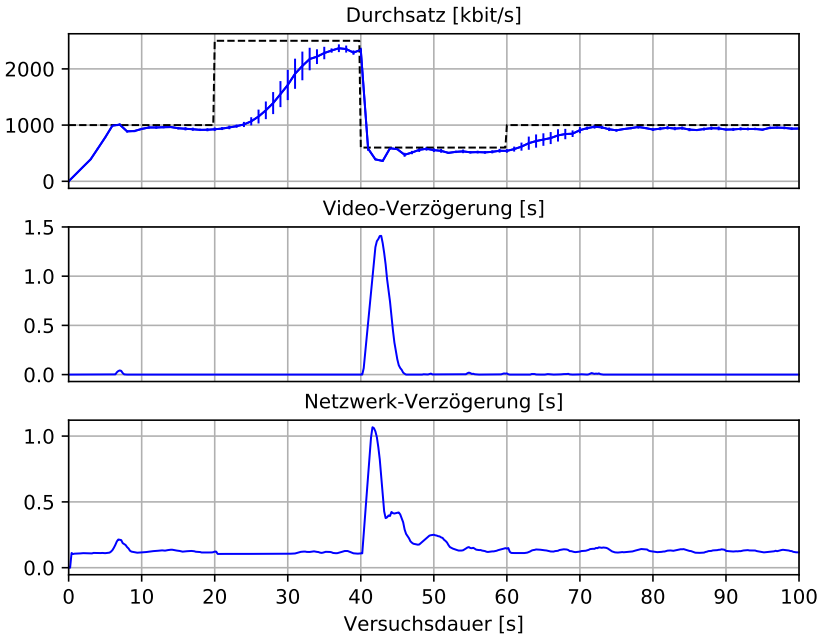


(a) Eigene Messung: TC5.1

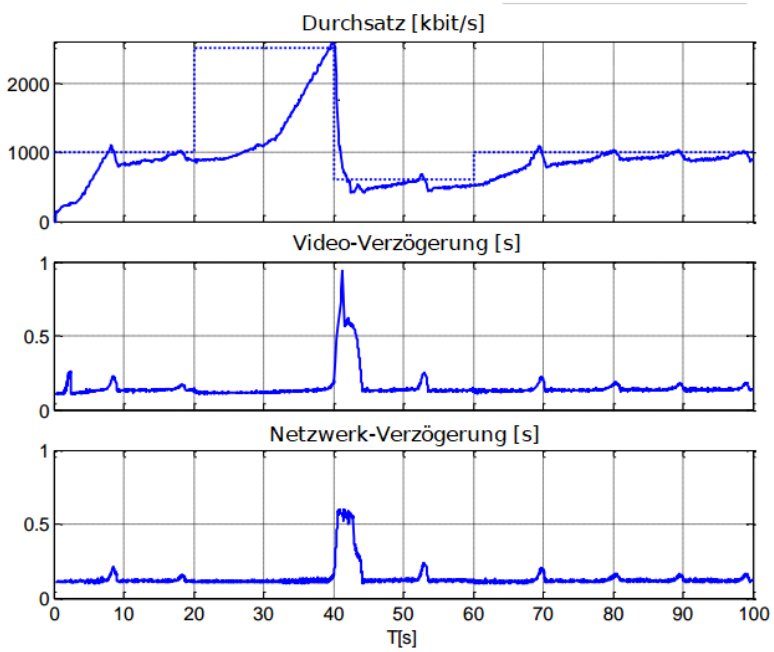


(b) Referenz: TC5.1, aus [47]

Abbildung 5.1: TC5.1 RTT100



(a) Eigene Messung: TC5.1



(b) Referenz: TC5.1, aus [47]

Abbildung 5.2: TC5.1 RTT200

5.2 Variable Pfadkapazität mit zwei SCReAM-Strömen

Im Sinne der Validierung wurde auch dieses Experiment bereits vorgestellt (siehe Sektion 4.5.2.2). Für die Versuchsdurchführung in der Evaluation ändert sich lediglich der Parameter des *Propagation Delays*. Das Experiment wurde jeweils mit einem *Propagation Delay* von 50ms und 100ms durchgeführt.

Folglich ändert sich auch in diesem Experiment der Erwartungswert an die Messergebnisse: Es wird von den eigenen Messergebnissen erwartet, dass sie ein vergleichbares Verhalten aufweisen, wie die visualisierten Messergebnisse 5.3(b) und 5.4(b), aus der Evaluation durch Ericsson Research.
Anhand der Abbildung 5.3(b) werden zunächst die Messergebnisse mit einem *Propagation Delay* von 50ms vorgestellt. Der gepunktete, schwarze Graph im Durchsatz zeigt das Experiment, bei einer Durchführung mit einem SCReAM-Strom. Dieser dient nur als Marker und ist für diese Evaluation nicht von Bedeutung, da für das Experiment zwei SCReAM-Ströme vorgesehen sind.
Zum Start wird die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf 2000 kbit/s eingestellt. Mit dem Beginn der Übertragung steigt der Graph im Durchsatz der beiden konkurrierenden SCReAM-Ströme linear an, sodass nach 8s beide Ströme sich die Hälfte der verfügbaren Pfadkapazität teilen. Nach einer Reduzierung der Pfadkapazität, nach Änderung **Eins** und **Drei**, zeigen beide Graphen das gleiche Verhalten, indem sie beide innerhalb einer Sekunde unterhalb der Pfadkapazität fallen und sich dieser neu anpassen, indem sie sich diese gleichmäßig aufteilen. In Folge der Reduzierung der Pfadkapazität steigt auch kurzzeitig die Video-Verzögerung und die Netzwerk-Verzögerung. Bemerkbar macht sich dies in Form von jeweils zwei ausgeprägten Spitzen.
Nach einer Erhöhung der Pfadkapazität, nach Änderung **Zwei**, steigen beide Graphen gleichmäßig exponentiell an. Lediglich nach Änderung **Vier** divergieren die beiden Graphen, indem einer der Graphen linear ansteigt, während der andere seinen Durchsatz nahezu beibehält. Insgesamt teilen sich die Graphen aber die Pfadkapazität gleichmäßig auf.

Aufgrund der Ähnlichkeit der beiden Referenzabbildungen, werden für die Abbildung 5.4(b) lediglich die Unterschiede, zu der zuvor beschriebenen Referenzabbildung, vorgestellt:

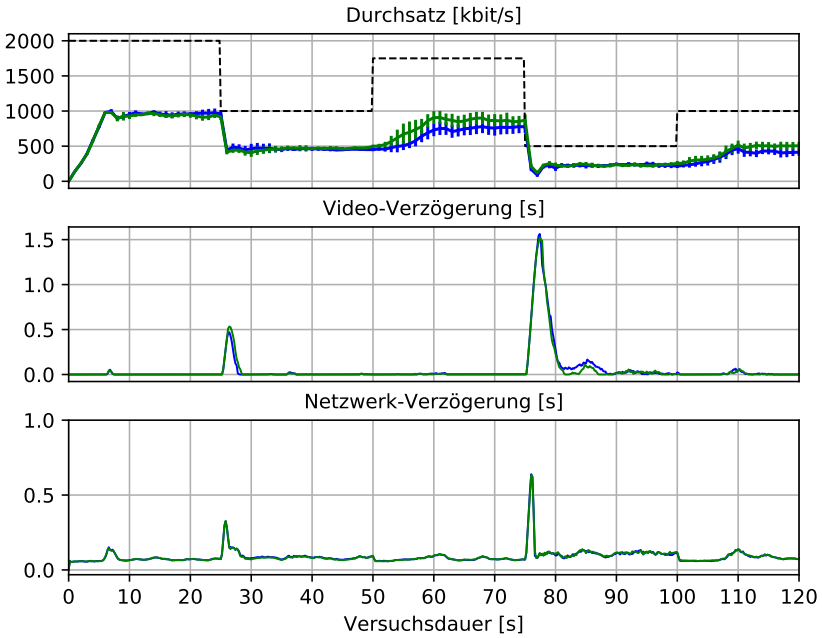
- Die beiden Graphen im Durchsatz divergieren schon ab Beginn der Übertragung, erreichen aber, nach der ersten Reduzierung der Pfadkapazität, wieder eine Konvergenz.

- Auf die letzte Anhebung der Pfadkapazität reagieren die Graphen früher. Das Problem der Divergenz besteht dennoch weiterhin.

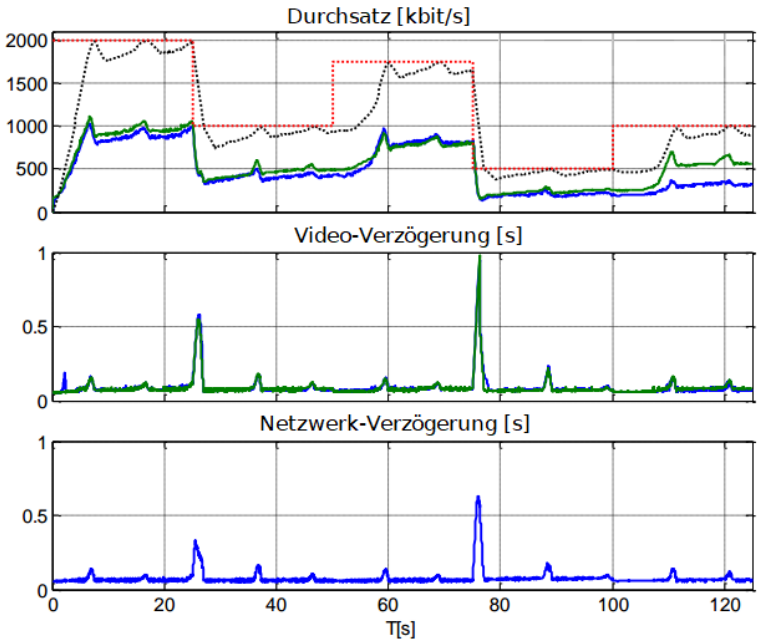
Ist-Zustand: Mit der Abbildung 5.3(a) erfolgt die Ergebnisvorstellung der eigenen Messung mit einem *Propagation Delay* von 50ms.
Am Anfang steigen beide Graphen im Durchsatz linear an und teilen sich nach 7s die verfügbare Pfadkapazität von 2000 kbit/s. In Folge der Reduzierung der Pfadkapazität — durch Änderung **Zwei** — fallen beide Graphen streng monoton.
Bis zur Pfadkapazitätsänderung **Drei** haben beide SCReAM-Ströme jeweils einen Durchsatz von 1000 kbit/s. Mit Eintreten der Änderung divergieren die Graphen im Durchsatz, dies mach sich auch in einem erhöhten Konfidenzintervall bemerkbar. Nach der Pfadkapazitätsänderung **Vier** fallen beide Graphen im Durchsatz, passen sich wieder der Pfadkapazität an und erreichen eine Konvergenz.
Auf die letzte Änderung der Pfadkapazität reagieren beide Ströme sofort und nehmen beide nach 11s jeweils die Hälfte der Pfadkapazität ein. In den Subplots der Video-Verzögerung und der Netzwerk-Verzögerung sind, in Folge einer Reduzierung der Pfadkapazität, die typischen Spitzen zu erkennen.

Aufgrund der Ähnlichkeit bei der Versuchsdurchführung werden für die Abbildung 5.4(a) lediglich die Unterschiede zur Abbildung 5.3(a) vorgestellt:

- Nach der Pfadkapazitätsänderung **Drei** divergieren die Graphen im Durchsatz nicht, zeigen aber eine große Streuung der Messwerte.
- Die Streuung der Messwerte besteht bis zum Ende der Versuchsdauer.
- Die Spitzen in der Video- und Netzwerk-Verzögerung haben, in Folge der Pfadkapazität-Änderung **Vier**, ein höheres Maximum. Außerdem wird die Spitze in der Video Verzögerung nicht nach kurzer Zeit aufgelöst, sondern zeigt bis zum Versuchsende ein unruhiges Verhalten.

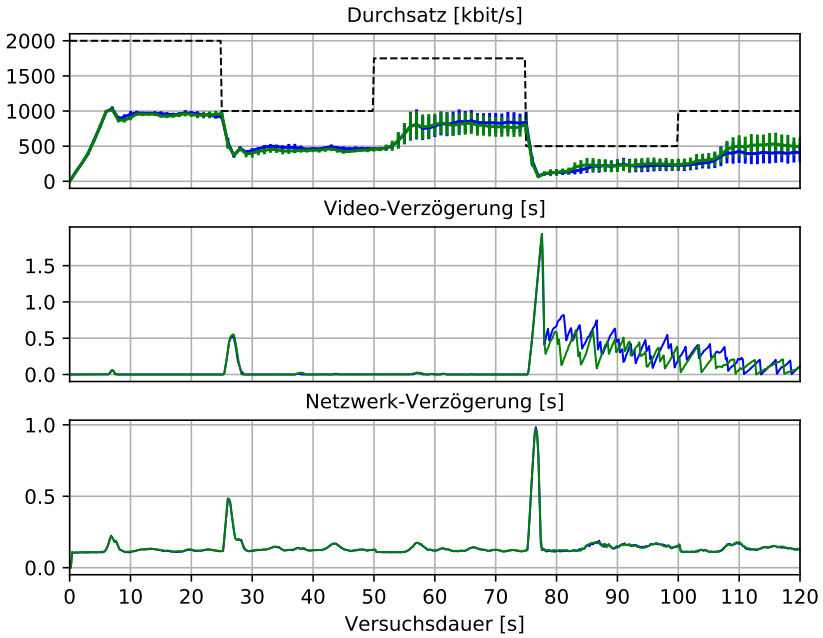


(a) Eigene Messung: TC5.2

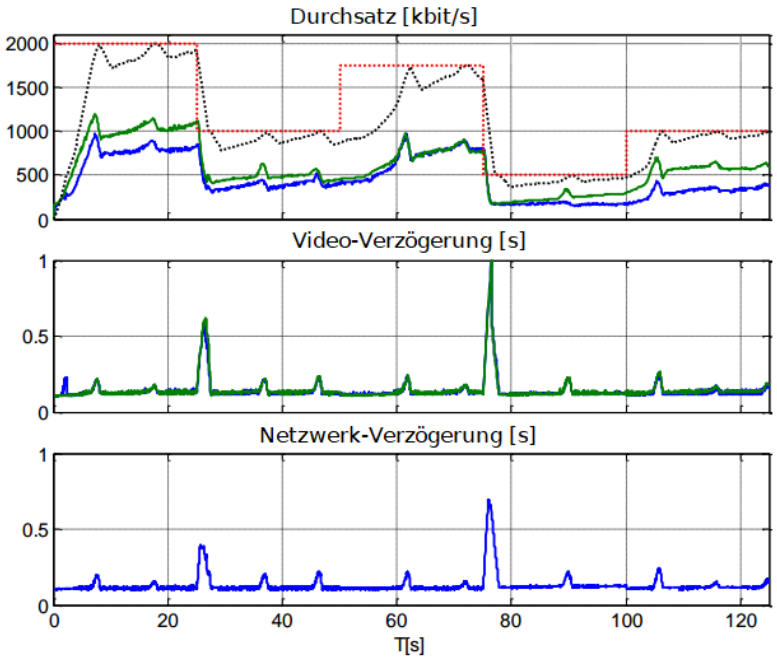


(b) Referenz: TC5.2, aus [47]

Abbildung 5.3: TC5.2 RTT100



(a) Eigene Messung: TC5.2



(b) Referenz: TC5.2, aus [47]

Abbildung 5.4: TC5.2 RTT200

5.3 Überlast im Feedback-Pfad

Aufgrund der asynchronen Beschaffenheit des Pfades einer Ende-zu-Ende Verbindung, kann es dazu führen, dass einer der Endpunkte keine Feedback-Nachrichten erhält. Der Grund hierfür ist eine Überlast im Feedback-Pfad. In diesem Experiment soll das Verhalten der SCReAM *Congestion Control*, in solch einem Szenario, geprüft werden.

Testbed-Topologie (siehe Abbbildung 5.5): Die Medienquellen S_1 und S_2 sind mit ihren zugehörigen Empfängern E_1 und E_2 verbunden. Der Medienstrom (S_1->E_1) wird über den *Vorwärts* Pfad übertragen und die Feedback-Nachrichten über den *Rückwärts* Pfad. Analog dazu wird der Medienstrom (S_2-E_2) über den *Rückwärts* Pfad übertragen und die Feedback-Nachrichten über den *Vorwärts* Pfad.

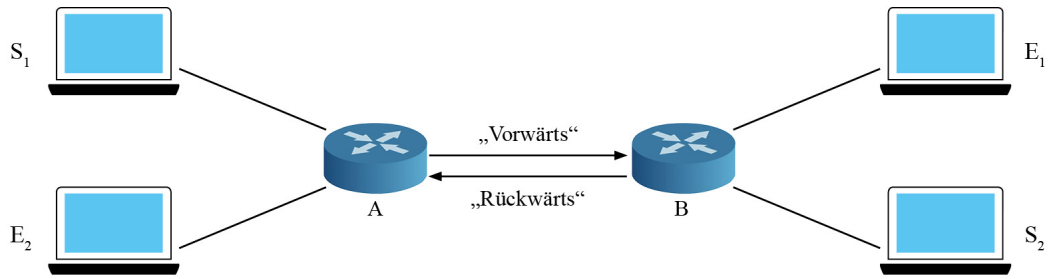


Abbildung 5.5: Testbed-Topologie für Überlast im Feedback-Pfad

Testbed-Attribute:

- Versuchsdauer: 100s
- Pfad-Eigenschaften:
  - *Propagation Delay*: 50ms und 100ms
  - Queue-Management: *Drop Tail* mit einer Buffergröße von 300ms
  - *Bottleneck*-Pfadkapazität: Für beide Pfade wird unabhängig voneinander die Pfadkapazität zur Simulationszeit verändert.
    Variable Pfadkapazität in Pfadrichtung *Vorwärts* (siehe Tabelle 5.1).
    Variable Pfadkapazität in Pfadrichtung *Rückwärts* (siehe Tabelle 5.2).

- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts* und *Rückwärts*
    - Anzahl Medienströme: 2
    - Medienstrom Zeitplan:
      - Startzeit: 0s
      - Endzeit: 100s
  - Konkurrierende Datenquellen
    - Anzahl Datenströme: 0

Pfadkapazitätsänderung	Pfadrichtung	Startzeit [s]	Pfadkapazität [kbit/s]
Eins	*Vorwärts*	0	2000
Zwei	*Vorwärts*	20	1000
Drei	*Vorwärts*	40	500
Vier	*Vorwärts*	60	2000

Tabelle 5.1: Ablauf: Variable Pfadkapazität in Pfadrichtung *Vorwärts*

Pfadkapazitätsänderung	Pfadrichtung	Startzeit [s]	Pfadkapazität [kbit/s]
Eins	*Rückwärts*	0	2000
Zwei	*Rückwärts*	35	800
Drei	*Rückwärts*	70	2000

Tabelle 5.2: Ablauf: Variable Pfadkapazität in Pfadrichtung *Rückwärts*

Erwartungswert an das Experiment: Es wird erwartet, dass der SCReAM Algorithmus mit dem Ausbleiben von Feedback-Nachrichten zurechtkommt und seine Sendebitrate so anpasst, dass kein zu hoher Performanz-Verlust auftritt.

Erwartungswert an die Messergebnisse: Von den eigenen Messwerten zum Experiment, **Überlast im Feedback-Pfad**, wird erwartet, dass sie ein vergleichbares Verhalten aufweisen, wie die visualisierten Messergebnisse 5.6(b) und 5.7(b).
Die Abbildung 5.6(b) zeigt die Messergebnisse zur Durchführung mit einem *Propagation Delay* von 50ms. Der SCReAM-Strom mit der Pfadrichtung *Vorwärts* wird mit dem grünen Graphen dargestellt und der SCReAM-Strom mit der Pfadrichtung *Rückwärts* durch den blauen Graphen.

Bevor die SCReAM-Ströme mit ihrer Übertragung beginnen, wird die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität auf jeweils 2000 kbit/s eingestellt. Beide SCReAM-Ströme zeigen im Durchsatz, zu Beginn der Übertragung, ein lineares Steigungsverhalten und passen sich beide der verfügbaren Pfadkapazität von 2000 kbit/s an.
In Folge der Pfadkapazitätsänderung **Zwei** in Pfadrichtung *Vorwärts*, fällt der Durchsatz des grünen Graphen weit unter die neue eingestellte Pfadkapazität. Der blaue Graph zeigt im Durchsatz keine Reaktion auf die Änderung und behält diesen bei.
Auf die Pfadkapazitätsänderung **Zwei** in Pfadrichtung *Rückwärts* reagieren sowohl der blaue Graph, als auch der grüne Graph streng monoton fallend. Der Grund für das Fallen des grünen Graphen ist die, durch die Änderung **Zwei** in Pfadrichtung *Rückwärts*, ausgelöste Überlast in seinem Feedback-Pfad. Das Ausbleiben oder verzögerte Eintreffen von Feedback-Nachrichten hat eine Reduzierung der Sendebitrate zur Folge.
Auf die Pfadkapazitätsänderung **Drei** in Pfadrichtung *Vorwärts* ist keine direkte Reaktion der Graphen zu erkennen. Erst auf die Anhebung der verfügbaren Pfadkapazität — durch Änderung **Vier** in Pfadrichtung *Vorwärts* — reagiert der grüne Graph mit einem linearen Steigungsverhalten. Der blaue Graph zeigt bei der letzten Anhebung, die seinen *Rückwärts* Pfad betrifft, ein logarithmisches Steigungsverhalten.
In den Subplots der Video-Verzögerung und der Netzwerk-Verzögerung ist das typische Verhalten bei einer Reduzierung der verfügbaren Pfadkapazität, in Form von Spitzen zu erkennen. Das Ausbleiben der Spitze des grünen Graphen in der Netzwerk-Verzögerung, zum Zeitpunkt der Änderung **Zwei** in Pfadrichtung *Rückwärts*, bestätigt, dass der Einbruch im Durchsatz durch das Ausbleiben von Feedback-Nachrichten erfolgte.

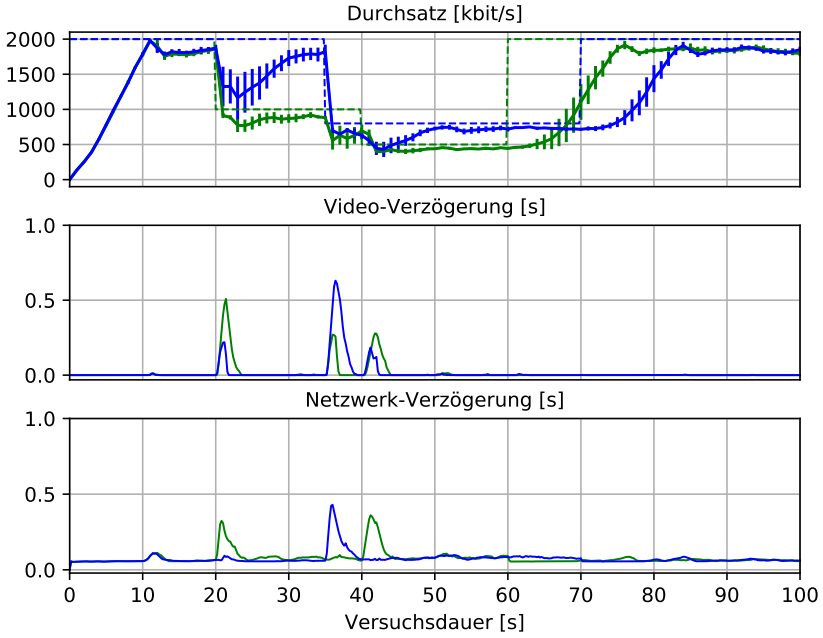
Da sich die Messwerte der Referenzabbildung nur in dem Parameter *Propagation Delay* unterscheiden, werden für die Vorstellung der Erwartungswerte für die Abbildung 5.7(b), lediglich die Unterschiede zur Abbildung 5.6(b) beschrieben:

- Die Graphen im Durchsatz benötigen mehr Zeit, um nach einer Erhöhung der verfügbaren Pfadkapazität, dass Maximum zu erreichen. Der Grund hierfür ist das doppelt so hoch eingestellte *Propagation Delay*. Abgeschickte RTP-Pakete brauchen dementsprechend doppelt so lange, um vom Sender zum Empfänger übertragen zu werden.
- In der Video-Verzögerung und der Netzwerk-Verzögerung ist zwischen 40s und 60s eine durchgängige Erhöhung der Verzögerung zu erkennen.

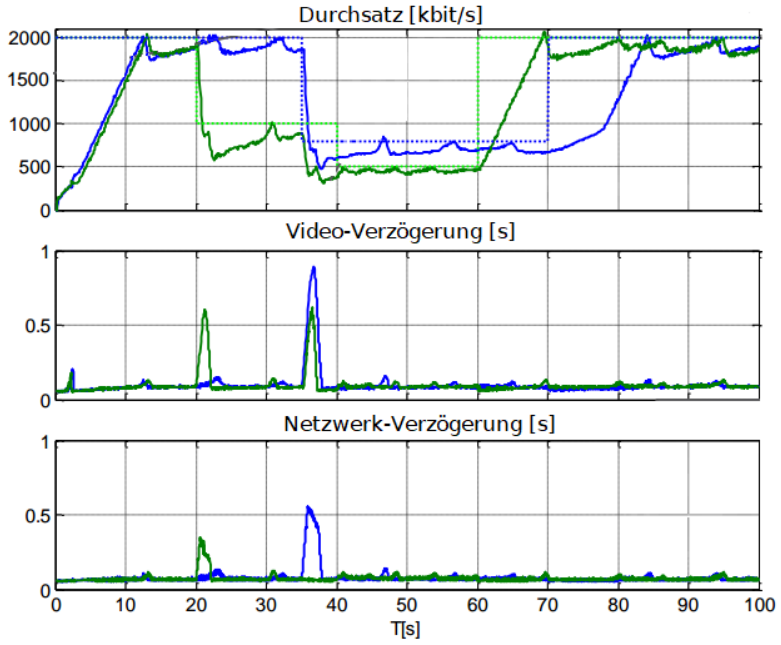
Ist-Zustand: Mit der Abbildung 5.6(a) werden die eigenen Messwerte zur Durchführung des Experiments, mit einem *Propagation Delay* von 50ms, vorgestellt. Der SCReAM-Strom

mit der Pfadrichtung *Vorwärts* wird ebenfalls durch den grünen Graphen dargestellt und der SCReAM-Strom in Pfadrichtung *Rückwärts* durch den blauen Graphen. Initial wird für beide Pfade die Pfadkapazitätsänderung **Eins** ausgeführt und die verfügbare Pfadkapazität jeweils auf 2000 kbit/s eingestellt. Zum Start der Übertragung steigen beide Graphen im Durchsatz linear an, bis sie nach 11s, die verfügbare Pfadkapazität von 2000 kbit/s erreicht haben.
In Folge der Pfadkapazitätsänderung **Zwei**, in Pfadrichtung *Vorwärts*, fällt der Durchsatz des grünen Graphen unterhalb der neuen Pfadkapazität und passt sich dieser schnell wieder an. Der blaue Graph ist ebenfalls von der Änderung betroffen. Denn mit der Überlast in seinem Feedback-Pfad, bleiben Feedback-Nachrichten aus oder kommen verzögert an. Aus diesem Grund fällt der blaue Graph im Durchsatz unter 1500 kbit/s.
In der Video-Verzögerung ist zu dem Zeitpunkt der Änderung jeweils eine Spitze zu erkennen. Die Verzögerung entsteht aufgrund der Trägheit des verwendeten Codecs, der Zeit benötigt, um sich an die neuen Netzwerkbedingungen anzupassen. In dem Subplot der Netzwerk Verzögerung ist nur eine Spitze des grünen Graphen zu erkennen. Dies bestätigt, dass Überlast in Pfadrichtung *Vorwärts* besteht und nicht in Pfadrichtung *Rückwärts*.
Durch die Pfadkapazitätsänderung **Zwei** in Pfadrichtung *Rückwärts* sind wiederum beide Graphen direkt betroffen. Der grüne Graph, dessen Feedback-Pfad betroffen ist, fällt linear auf 500 kbit/s. Erst mit Abhandlung des Datenstaus, in der Video- und Netzwerk-Verzögerung, steigt der Graph wieder in seinem Durchsatz.
Auch in Folge der Pfadreduzierung **Drei** in Pfadrichtung *Vorwärts* sind beide Graphen betroffen. Der blaue Graph fällt im Durchsatz kurzzeitig unter 500 kbit/s. Nachdem der Datenstau im Netzwerk und der RTP-*Queue* abgearbeitet wurde, beginnt der blaue Graph wieder Pfadkapazität zu beanspruchen.
Auf die letzten Anhebungen der Pfadkapazität reagieren beide Graphen nach 5s mit einem annähernd linearen Steigungsverhalten.

Die eigenen Messergebnisse aus Abbildung 5.7(a) unterscheiden sich, zu den zuvor vorgestellten aus Abbildung 5.6(a), nur durch ein erhöhtes Maximum der Spitzen in der Video-Verzögerung und der Netzwerk-Verzögerung. Der Grund hierfür ist, dass für diese Versuchsdurchführung, eingestellte *Propagation Delay* von 100ms. Durch das erhöhte *Propagation Delay*, brauchen RTP-Pakete länger, um vom Sender zum Empfänger übertragen zu werden. Dementsprechend wird mehr Zeit benötigt, um den Datenstau in einem *Bottleneck*-Szenario zu behandeln.

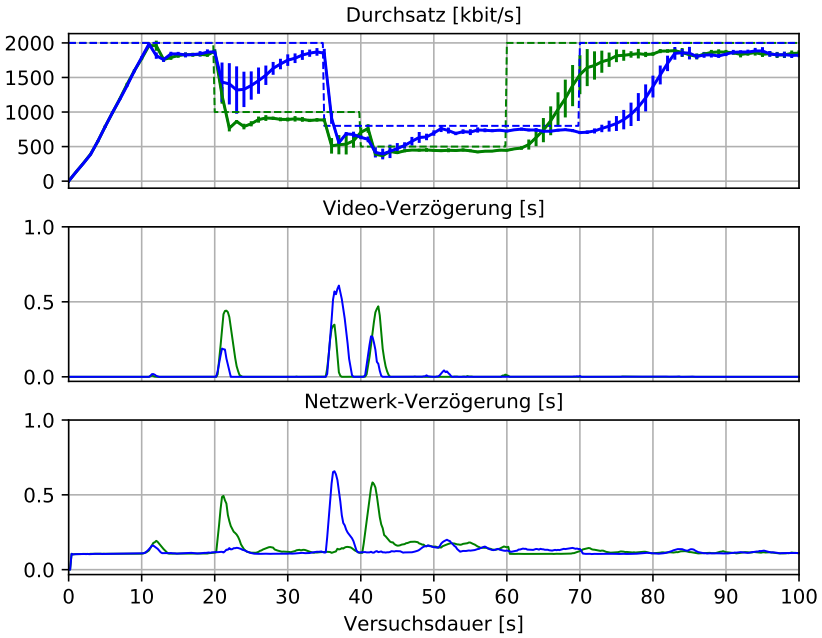


(a) Eigene Messung: TC5.3

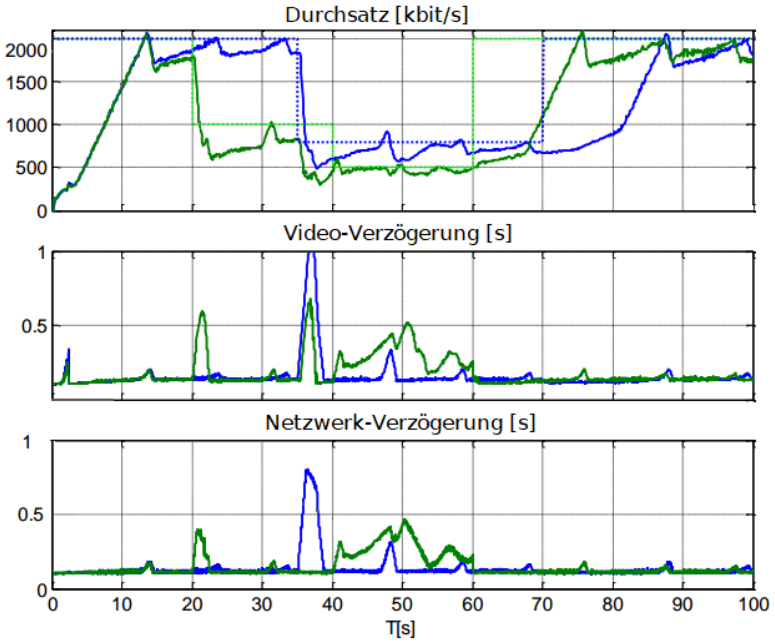


(b) Referenz: TC5.3, aus [47]

Abbildung 5.6: TC5.3 RTT100



(a) Eigene Messung: TC5.3



(b) Referenz: TC5.3, aus [47]

Abbildung 5.7: TC5.3 RTT200

5.4 Konkurrierende SCReAM-Ströme

In diesem Experiment teilen sich drei Medienströme einen *Bottleneck*-Pfad. Jeder Medienstrom implementiert den SCReAM *Congestion Control* Algorithmus.
Dieses Experiment zeigt ein typisches Szenario, in dem eine WebRTC Applikation mehrere Medienströme an den gleichen Endpunkt überträgt.
Wie in der Sektion 1.2 bereits erwähnt, werden alle Medienströme über den gleichen Port gemultiplext. Dementsprechend ist es möglich, dass die Medienströme über den gleichen Pfad übertragen werden und sie die Pfadkapazität unter sich aufteilen müssen.

Testbed-Topologie (siehe Abbbildung 5.8): Die Medienquellen S_1, S_2 und S_3 sind mit ihren zugehörigen Empfängern E_1, E_2 und E_3 verbunden. Der Medienstrom wird über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den mit *Rückwärts* markierten Pfad.

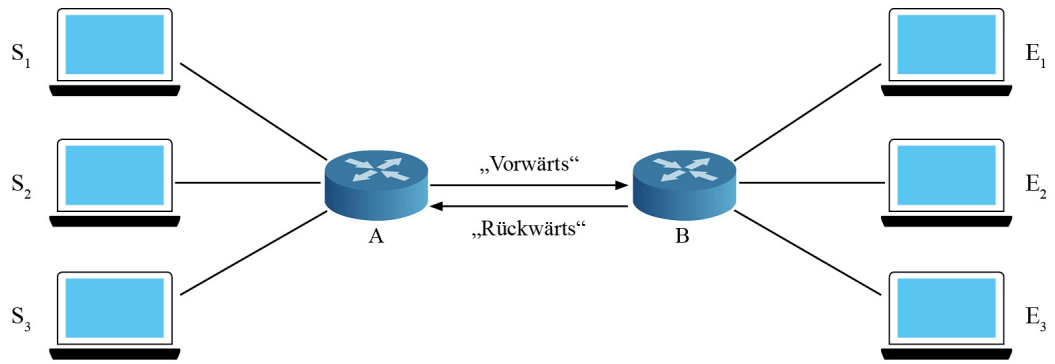


Abbildung 5.8: Testbed-Topologie für konkurrierende SCReAM-Ströme

Testbed-Attribute:

- Versuchsdauer: 120s
- Pfad-Eigenschaften:
  - *Propagation Delay*: 50ms und 100ms
  - Queue-Management: *Drop Tail* mit einer Buffergröße von 300ms
  - *Bottleneck*-Pfadkapazität: 3.5 Mbit/s

- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts*
    - Anzahl Medienströme: 3
    - Medienstrom Zeitplan: Der erste Medienstrom startet bei 0s, der zweite ab 20s und der dritte startet seine Übertragung nach 40s. Beendet werden alle Medienströme nach 120s.
  - Konkurrierende Datenquellen
    - Anzahl Datenströme: 0

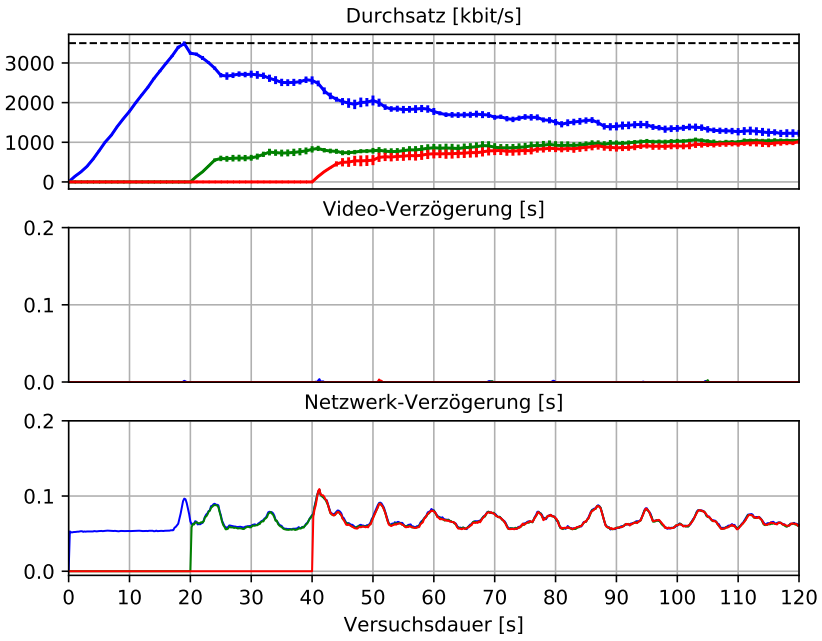
Erwartungswert an das Experiment: Es wird erwartet, dass die Medienströme eine optimale Bitrate beanspruchen und sich dabei nicht gegenseitig behindern. In diesem Experiment werden alle drei Medienströme zu verschiedenen Zeitpunkten gestartet. Sobald der zweite Medienstrom mit seiner Übertragung beginnt, sollte der zuvor gestartete Medienstrom einen fairen Anteil der Pfadkapazität an den zweiten Medienstrom abgeben. Das gleiche Verhalten wird auch beim Start des dritten Medienstroms erwartet. Außerdem sollte die *Bottleneck*-Pfadkapazität vollständig gesättigt sein.

Erwartungswert an die Messergebnisse: Von den eigenen Messergebnissen zum Experiment mit mehreren konkurrierenden SCReAM-Strömen wird erwartet, dass sie ein vergleichbares Verhalten zeigen, wie die visualisierten Messergebnisse 5.9(b) und 5.10(b). Für die Durchführung und Visualisierung wurde die Versuchsdauer, durch Ericsson Research, von 120s auf 200s erhöht.
Zu Beginn startet ein SCReAM-Strom mit seiner Übertragung. Der Graph zeigt im Durchsatz ein lineares Steigungsverhalten. Mit Erreichen der verfügbaren Pfadkapazität, nach 20 Sekunden, startet der zweite SCReAM-Strom mit seiner Übertragung. In Folge dessen, gibt der erste SCReAM-Strom einen Teil seiner Pfadkapazität an den zweiten SCReAM-Strom ab. Nach 40s startet wiederum der dritte SCReAM-Strom seine Übertragung.
Bis zum Ende der Versuchsdauer gibt der erste SCReAM-Strom kontinuierlich Pfadkapazität an die beiden später gestarteten Ströme ab. Sodass nach 200s der erste Strom noch etwa 1500 kbit/s Pfadkapazität beansprucht, der zweite Strom etwa 1000 kbit/s und der zuletzt gestartete etwa 900 kbit/s.
In den Subplots der Video-Verzögerung und der Netzwerk-Verzögerung sind, neben der Basis Verzögerung durch das *Propagation Delay*, immer wieder kleine Spitzen erkennbar. Ausgelöst

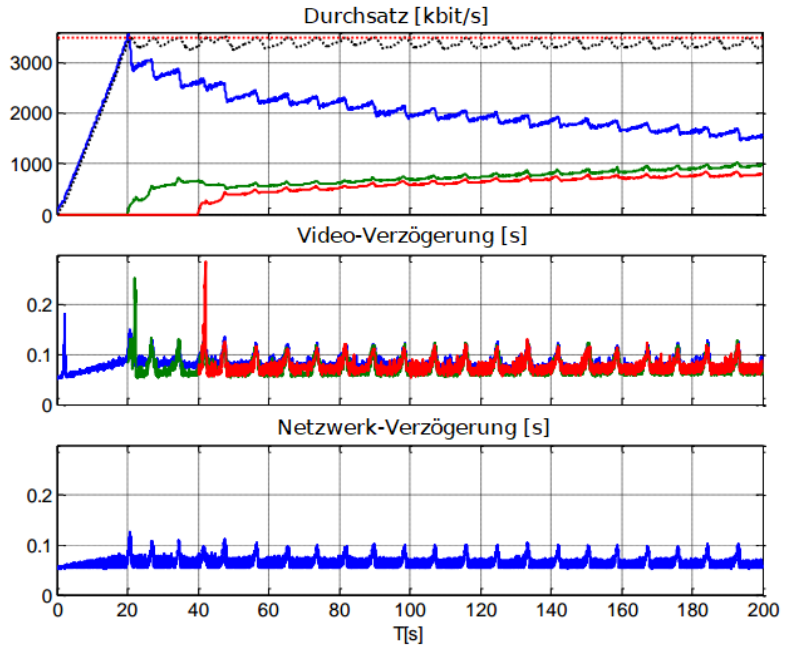
werden diese durch den Versuch des ersten SCReAM-Stroms wieder Pfadkapazität zu beanspruchen, mit der Folge einer Reduzierung der Datenrate.

In der Referenzabbildung 5.10(b) ist eine, durch das eingestellte *Propagation Delay*, erhöhte Basis-Verzögerung in den Subplot der Video-Verzögerung und der Netzwerk-Verzögerung zu erkennen. Sonst weisen die Referenzabbildungen keine gravierenden Unterschiede zueinander auf.

Ist-Zustand: Mit der Abbildung 5.9(a) erfolgt die Ergebnisvorstellung der eigenen Messwerte zur Durchführung des Experiments mit einem *Propagation Delay* von 50ms. Damit die eigenen Messergebnisse für Dritte vergleichbar sind, wird die Struktur der Experimente aus [11] beibehalten. Somit beträgt die Versuchsdauer, im Gegensatz zur Referenzabbildung, 120s.
Der zuerst gestartete SCReAM-Strom steigt im Durchsatz linear an und erreicht innerhalb von 20s die verfügbare Pfadkapazität von 3500 kbit/s.
Nach 20s startet der zweite SCReAM-Strom seine Übertragung. Aus diesem Grund reduziert der erste SCReAM-Strom seine Senderate und gibt einen Teil seiner beanspruchten Pfadkapazität an diesen ab.
Mit dem Start des dritten SCReAM-Stroms, gibt der zuerst gestartete, wiederum einen Teil seiner beanspruchten Pfadkapazität ab.
Bis zum Versuchsende von 120s gibt der erste SCReAM-Strom kontinuierlich Kapazität an die beiden anderen SCReAM-Ströme ab. Sodass nach 120s annähernd eine Konvergenz der Graphen im Durchsatz besteht.
Über die gesamte Versuchsdauer hinweg besteht keine Video-Verzögerung. Der Subplot der Netzwerk-Verzögerung zeigt neben dem *Propagation Delay* immer wieder kleine Schwankungen. Der Grund hierfür sind die SCReAM-Ströme, die versuchen Pfadkapazität einzunehmen. Mit ansteigender Verzögerung, reduzieren diese ihre Senderate wieder.
In der Abbildung 5.10(a), zur Versuchsdurchführung mit einem *Propagation Delay* von 100ms, ist im Vergleich nur eine erhöhte Netzwerk-Verzögerung zu erkennen.

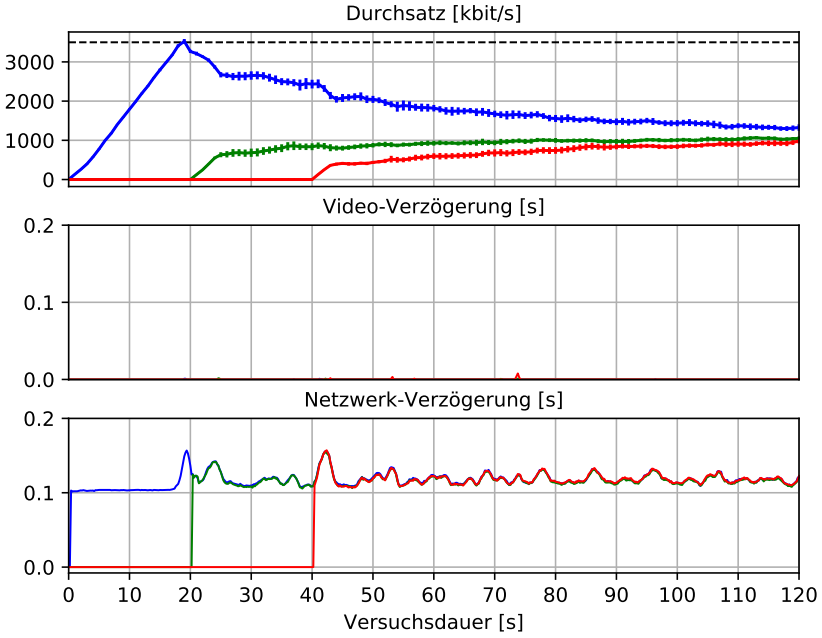


(a) Eigene Messung: TC5.4

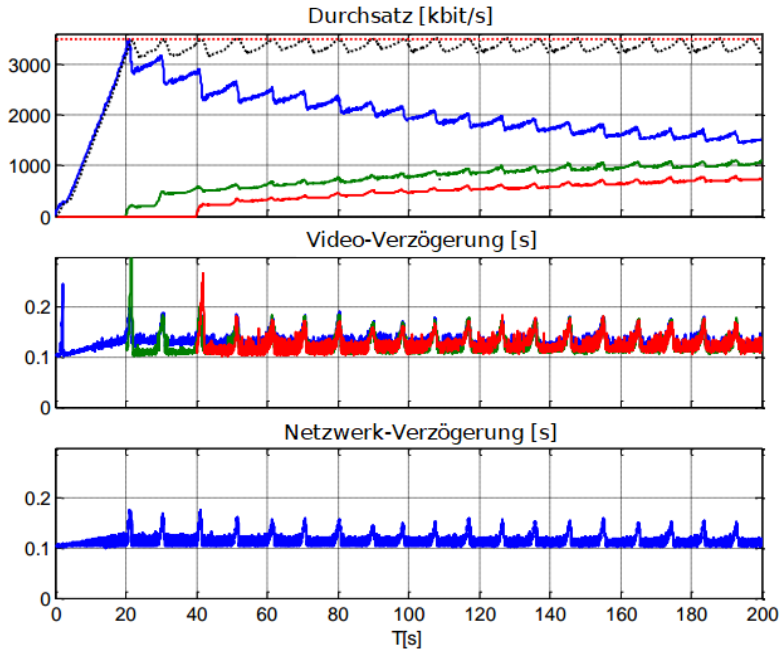


(b) Referenz: TC5.4, aus [47]

Abbildung 5.9: TC5.4 RTT100



(a) Eigene Messung: TC5.4



(b) Referenz: TC5.4, aus [47]

Abbildung 5.10: TC5.4 RTT200

5.5 Konkurrierende SCReAM-Ströme mit verschiedenen RTTs

In diesem Experiment teilen sich mehrere SCReAM-Ströme ein *Bottleneck*. Das *Propagation Delay* wird für jeden SCReAM-Strom unterschiedlich eingestellt. Mit diesem Experiment soll geprüft werden, ob mehrere konkurrierende SCReAM-Ströme mit unterschiedlicher RTT sich gleichmäßig die verfügbare Pfadkapazität aufteilen.

Testbed-Topologie: Die Topologie ist die gleiche wie in Sektion 5.4. Die Medienquellen S_1, S_2, S_3, S_4 und S_5 sind mit ihren zugehörigen Empfängern E_1, E_2, E_3, E_4 und E_5 verbunden. Der Medienstrom wird über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den mit *Rückwärts* markierten Pfad.

Testbed-Attribute:

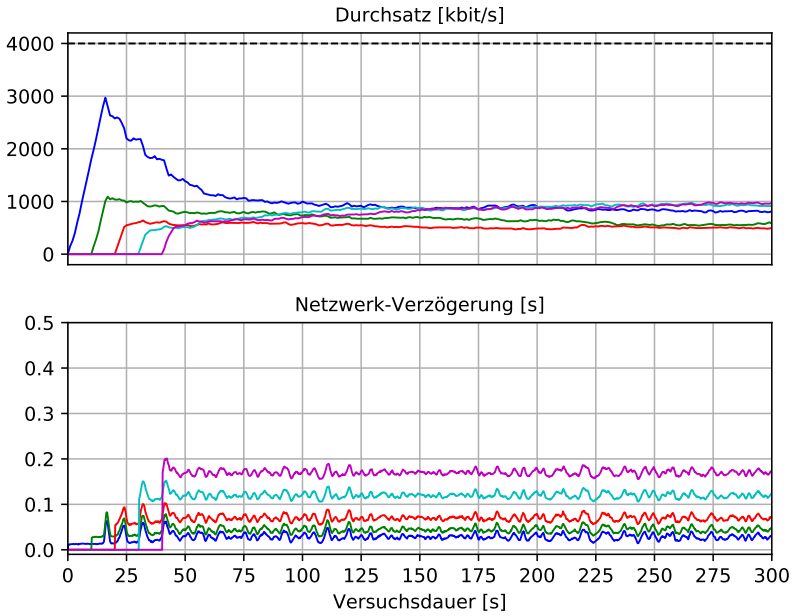
- Versuchsdauer: 300s
- Pfad-Eigenschaften:
  - *Propagation Delay* für jeden Medienstrom unterschiedlich: 10ms, 25ms, 50ms, 100ms und 150ms.
  - Queue-Management: *Drop Tail* mit einer Buffergröße von 300ms
  - *Bottleneck*-Pfadkapazität: 4.0 Mbit/s
- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts*
    - Anzahl Medienströme: 5
    - Medienstrom Zeitplan: Der erste Medienstrom startet bei 0s. Nachfolgend starten die einzelnen Medienströme in einem zehn Sekunden Intervall. Somit startet der letzte Medienstrom bei 40s. Beendet werden alle Medienströme nach 300s.
  - Konkurrierende Datenquellen
    - Anzahl Datenströme: 0

Erwartungswert an das Experiment: Es wird erwartet, dass alle konkurrierenden Medienströme sich die *Bottleneck*-Pfadkapazität gleichmäßig aufteilen und zu einem gemeinsamen

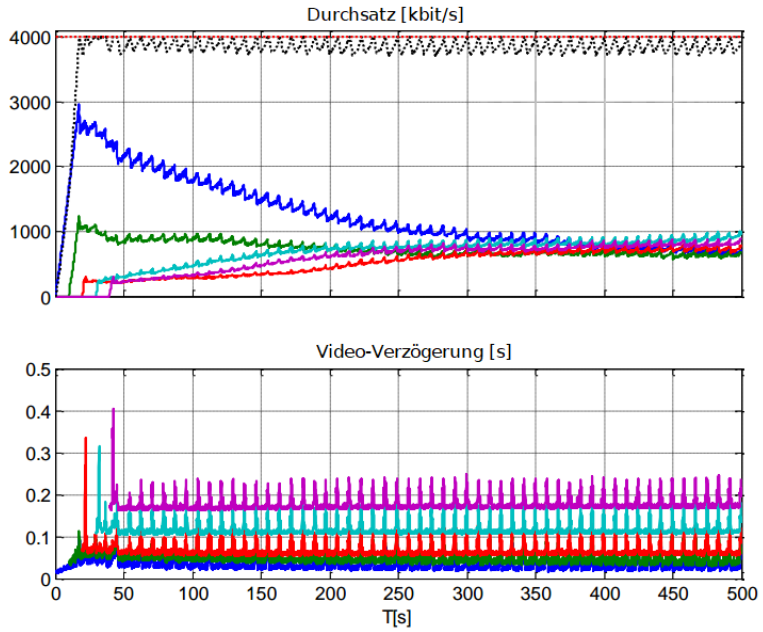
Durchsatz konvergieren. Dabei sollte die Netzwerk-Verzögerung so gering wie möglich sein.

Erwartungswert an die eigenen Messergebnisse: Es wird erwartet, dass die eigenen Messergebnissen ein vergleichbares Verhalten aufweisen, wie die visualisierten Messergebnisse aus Abbildung 5.11(b). Auch in diesem Experiment wurde die Versuchsdauer, in der Evaluation durch Ericsson, von den vorgeschlagenen 300s auf 500s erhöht.
Der Graph, des zu Beginn gestarteten SCReAM-Stroms, zeigt ein lineares Steigungsverhalten. Bevor dieser die maximale verfügbare Pfadkapazität beanspruchen kann, startet bei 10s, der zweite SCReAM-Strom seine Übertragung. Nachfolgend starten die restlichen SCReAM-Ströme ihre Übertragung. Ab etwa 400s erreichen die Ströme eine Konvergenz im Durchsatz. Bis zu diesem Zeitpunkt gibt der zuerst gestartete SCReAM-Strom kontinuierlich Pfadkapazität an die restlichen Ströme ab. Der zweite gestartete SCReAM-Strom hingegen behält seinen Durchsatz bis zum Versuchsende nahezu bei, während die später gestarteten SCReAM-Ströme ihren Durchsatz, bis zum Zeitpunkt der Konvergenz, stetig erhöhen.

Ist-Zustand: Mit der Abbildung 5.11(a) erfolgt die Ergebnisvorstellung der eigenen Messergebnisse. Anzumerken ist, dass auch hier die Versuchsdauer von 300s beibehalten wurde.
Zu Beginn startet der erste SCReAM-Strom mit einem linearen Steigungsverhalten im Durchsatz und nimmt kurzzeitig 3000 kbit/s ein. Mit den nachfolgend startenden SCReAM-Strömen, gibt der zuerst gestartete einen großen Teil seiner eingenommen Pfadkapazität ab. Nach 100s befinden sich bereits alle Graphen unterhalb von 1000 kbit/s.
Gegen Ende des Versuchs nehmen zwei SCReAM-Ströme 1000 kbit/s ein, ein SCReAM-Strom nimmt 800 kbit/s ein und die anderen beiden 600 kbit/s. Der zuletzt gestartete SCReAM-Strom mit dem höchsten *Propagation Delay*, hat gegen Ende der Versuchsdauer den höchsten Durchsatz.



(a) Eigene Messung: TC5.5



(b) Referenz: TC5.5, aus [47]

Abbildung 5.11: TC5.5

5.6 SCReAM-Strom konkurriert mit einem langlebigen TCP-Strom

In diesem Experiment teilen sich ein SCReAM-Strom und ein langlebiger TCP-Strom den *Bottleneck*-Pfad. Der TCP-Strom simuliert einen Download von Daten über die ganze Verbindung hinweg. Das Experiment misst die Anpassungsfähigkeit, des SCReAM Algorithmus, gegenüber einem konkurrierenden TCP-Strom.

Testbed-Topologie (siehe Abbbildung 5.12): Die Medienquelle S_1 ist mit ihren zugehörigen Empfänger E_1 verbunden. Der konkurrierende TCP-Strom S_{tcp} ist mit dem Empfänger E_{tcp} verbunden. Beide Ströme teilen sich den *Bottleneck*-Pfad. Der SCReAM-Strom wird über den mit *Vorwärts* markierten Pfad übertragen und die Feedback-Nachrichten über den *Rückwärts* Pfad. Die Daten von TCP werden über den *Vorwärts* Pfad übertragen und die *Acknowledgment* Pakete über den *Rückwärts* Pfad.

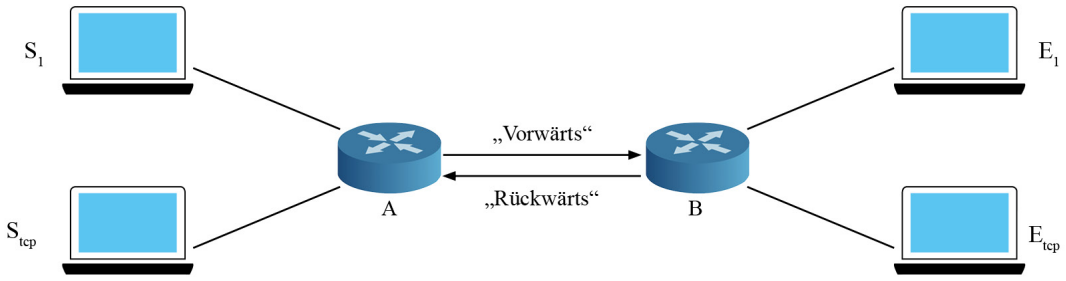


Abbildung 5.12: Testbed-Topologie für konkurrierenden langlebigen TCP-Strom

Testbed-Attribute:

- Versuchsdauer: 120s
- Pfad-Eigenschaften:
  - *Propagation Delay*: 50ms und 100ms.
  - Queue-Management:
    - *Drop Tail*: mit einer Buffergröße von 300ms.
    - *REDQueue*

  - *Bottleneck*-Pfadkapazität: 2.0 Mbit/s

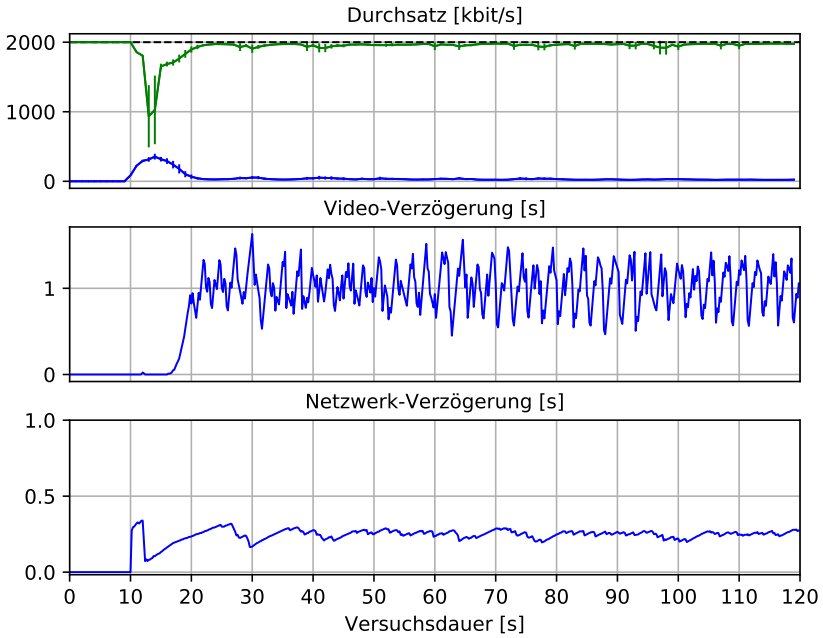
- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts*
    - Anzahl Medienströme: 1
    - Medienstrom Zeitplan:
      - Startzeit: 10s
      - Endzeit: 120s
  - Konkurrierende Datenquellen
    - Datenstrom Richtung: *Vorwärts*
    - Typ der Datenquelle: Langlebiger TCP-Strom mit einer loss-based *Congestion Control*.
    - Anzahl Datenströme: 1
    - Datenstrom Zeitplan:
      - Startzeit: 0s
      - Endzeit: 120s
- Experiment spezifische Informationen: Neben den in Sektion 4.2.1 genannten Metriken, wird in diesem Experiment, der Durchsatz vom TCP-Strom ermittelt und visualisiert.

Erwartungswert an das Experiment: Nach der Anforderung 5, ist die Erwartungshaltung, dass der SCReAM-Strom gegenüber dem konkurrierenden TCP-Strom nicht verliert. Im schlimmsten Fall fällt der SCReAM-Strom auf seine minimale Medienbitrate.

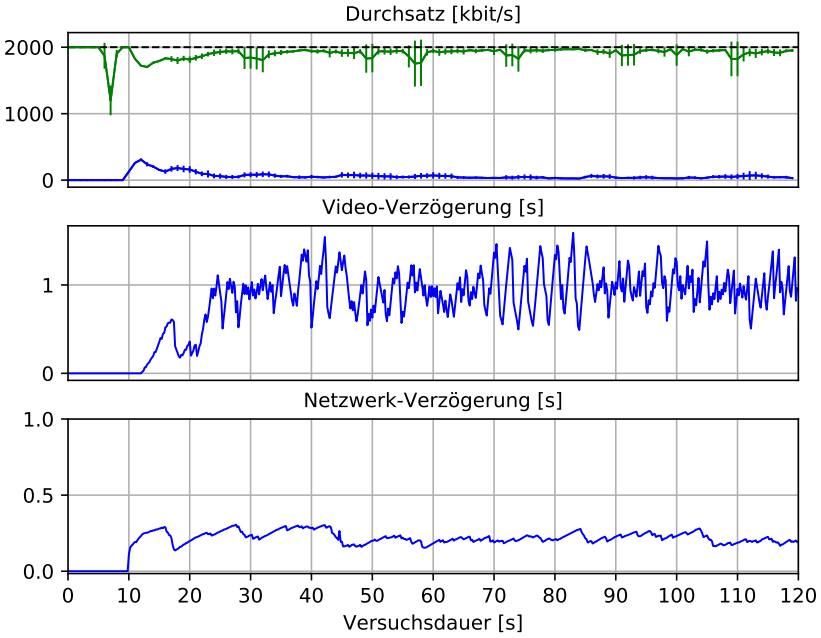
Erwartungswert an die Messergebnisse: Aus den Referenzabbildungen, zu diesem Experiment, wird nicht ersichtlich, wie der TCP-Strom konfiguriert wurde. Aus diesem Grund werden diese nicht als Erwartungswert vorgestellt und auch nicht zum Vergleich herangezogen.

Ist-Zustand: Die Abbildung 5.13(a) zeigt die Messergebnisse zur Durchführung mit einem *Propagation Delay* von 50ms und einer eingestellten *Drop Tail Queue*.
Der TCP-Strom hat zu Beginn des Plots, den Pfad bereits saturiert. Nach 10s startet der SCReAM-Strom mit seiner Übertragung und kann zu Beginn etwa 250 kbit/s beanspruchen.

Dementsprechend hat der TCP-Strom einen Teil seiner Pfadkapazität an den SCReAM-Strom abgegeben.
Ab 20s saturiert der TCP-Strom hingegen wieder nahezu die komplette Pfadkapazität. Der SCReAM-Strom ist im Durchsatz, bis zum Versuchsende, auf sein Minimum begrenzt. Mit Erreichen des minimalen Durchsatzes, schwingt die Video-Verzögerung durchgängig um eine Sekunde. Der Graph in der Netzwerk-Verzögerung zeigt bis zum Versuchsende eine durchgängige Verzögerung.
Die Versuchsdurchführung mit einer *REDQueue* (siehe Abbildung 5.13(b)) zeigt in den Graphen das gleiche Verhalten, wie mit dem Einsatz einer *Drop Tail Queue*.
Die visualisierten Messergebnisse, aus der Durchführung mit einem *Propagation Delay* mit 100ms, zeigen in den Graphen beide das gleiche Verhalten, wie die zuvor beschriebenen. Unterschiede bestehen lediglich zum Start des jeweiligen SCReAM-Stroms. Doch sobald der TCP-Strom den Pfad wieder saturiert hat, fallen alle SCReAM-Ströme auf ihren minimalen Durchsatz.

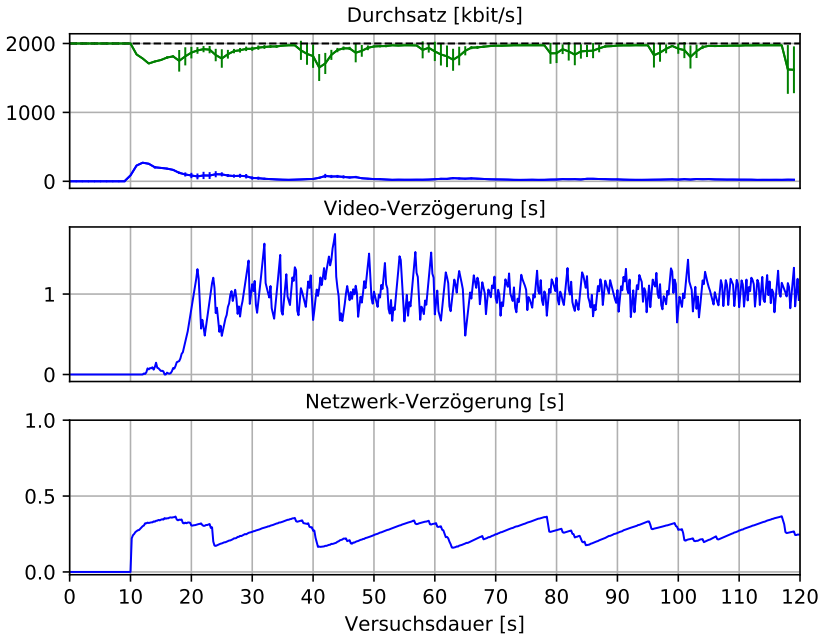


(a) Eigene Messung: TC5.6, 300ms *Drop Tail*

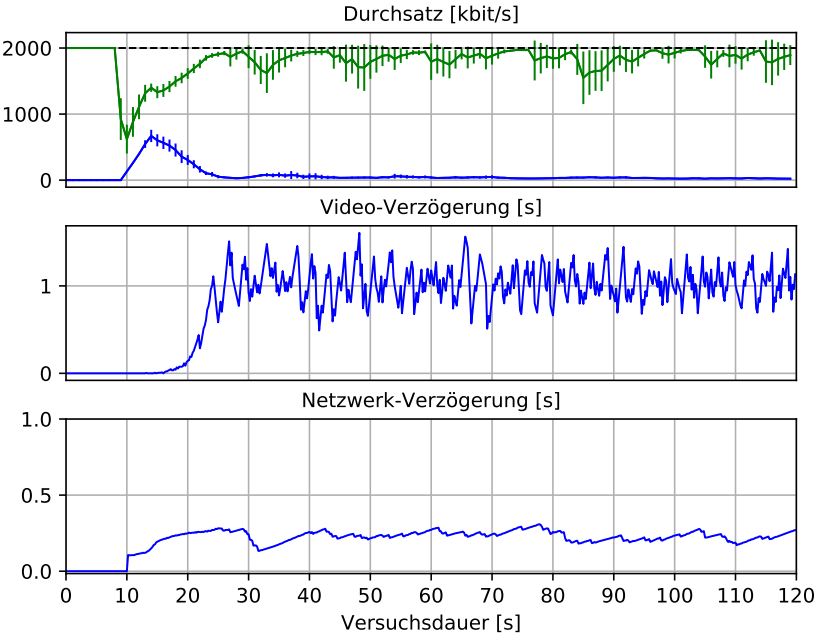


(b) Eigene Messung: TC5.6, *REDQueue*

Abbildung 5.13: TC5.6 RTT100



(a) Eigene Messung: TC5.6, 300ms *Drop Tail*



(b) Eigene Messung: TC5.6, *REDQueue*

Abbildung 5.14: TC5.6 RTT200

5.7 Zwei SCReAM-Ströme konkurrieren mit mehreren kurzlebigen TCP-Strömen

In diesem Experiment teilen sie zwei SCReAM-Ströme ein *Bottleneck*-Pfad mit mehreren kurzlebigen TCP-Strömen. Mit den kurzlebigen TCP-Strömen soll das typische Verhalten im Web simuliert werden. Ein Webbrowser verbindet sich mit einem Server und lädt mit mehreren TCP-Verbindungen eine Webseite, Bilder und Textdaten. Das Ziel von diesem Experiment ist es, die Anpassungsfähigkeit des SCReAM Algorithmus gegenüber konkurrierendem Datenverkehr aus dem Web zu messen. Das Experiment wurde entwickelt um primär die Anforderungen 4 und 5 abzudecken.

Testbed-Topologie: ist die gleiche wie in Sektion 5.6.

Testbed-Attribute:

- Versuchsdauer: 300s
- Pfad-Eigenschaften:
  - *Propagation Delay*: 50ms und 100ms.
  - Queue-Management: *Drop Tail* mit einer Buffergröße von 300ms
  - *Bottleneck*-Pfadkapazität: 2.0 Mbit/s
- Anwendungsbezogene Informationen:
  - Medienquellen
    - Medienstrom Richtung: *Vorwärts*
    - Anzahl Medienströme: 2
    - Medienstrom Zeitplan:
      - Startzeit: 10s
      - Endzeit: 300s
  - Konkurrierende Datenquellen
    - Datenstrom Richtung: *Vorwärts*
    - Typ der Datenquelle: Kurzlebige TCP-Ströme mit einer loss-based *Congestion Control*.

    - Anzahl Datenströme: 10
    - Datenstrom Zeitplan: Zwei TCP-Ströme starten im *Off-State*: 0s und 3s.
      Die restlichen TCP-Ströme starten im *On-State*: Ab 15s bis 50s in 5-Sekunden-Intervallen.

- Experiment spezifische Informationen: Neben den in Sektion 4.2.1 genannten Metriken, wird in diesem Experiment, der Durchsatz vom TCP-Strom ermittelt und visualisiert.

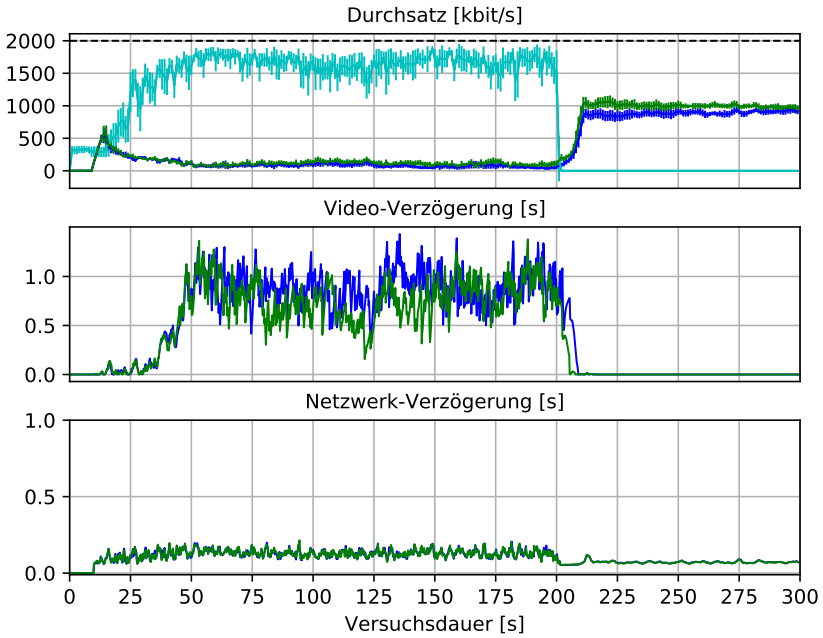
Erwartungswert an das Experiment: Es wird erwartet, dass die SCReAM-Ströme nicht gegenüber den konkurrierenden TCP-Strömen verlieren.

Erwartungswert an die Messergebnisse: Aus den Referenzabbildungen, zu diesem Experiment, wird nicht ersichtlich, wie die verschiedenen TCP-Ströme konfiguriert wurden und zu welchen Zeitpunkten diese starten. Aus diesem Grund werden diese nicht als Erwartungswert vorgestellt und auch nicht zum Vergleich herangezogen.

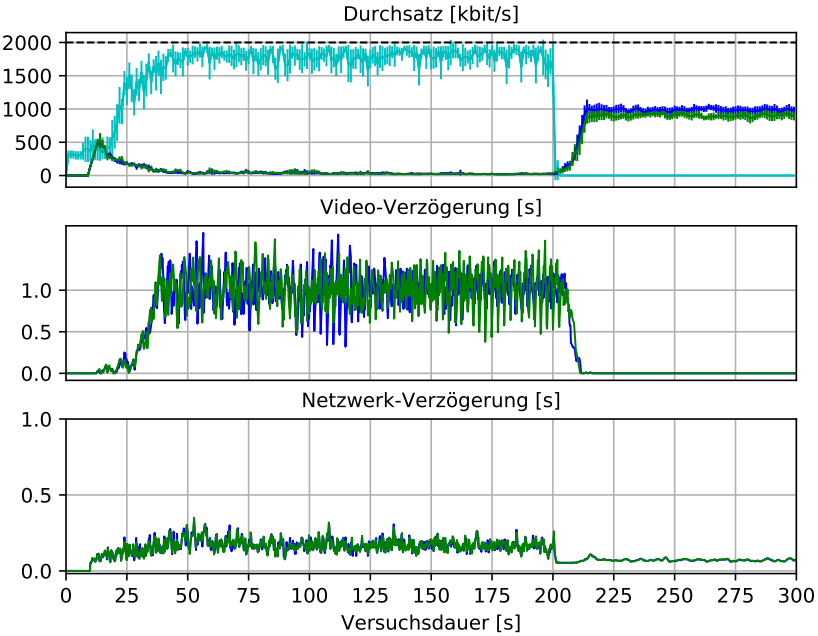
Ist-Zustand: Mit der Abbildung 5.15(a) erfolgt die Vorstellung der eigenen Messergebnisse zur Durchführung des Experiments mit einer *Drop Tail Queue* und einem *Propagation Delay* von 50ms.
Im *On-State* starten bereits zwei TCP-Ströme ihre Übertragungen. Zusammen beanspruchen sie eine Pfadkapazität von 350 kbit/s.
Nach 10s starten beide SCReAM-Ströme und erreichen kurzzeitig einen Durchsatz von jeweils 500 kbit/s. Mit den restlich startenden TCP-Strömen, geben die SCReAM-Ströme mehr und mehr von ihrer beanspruchten Pfadkapazität ab. Sodass mit dem Start des letzten TCP-Stroms bei 50s, die Graphen der SCReAM-Ströme im Durchsatz auf ein Minimum fallen. Mit dem Start des letzten TCP-Stroms schwankt die Video-Verzögerung bis zum Ende der TCP-Ströme. Auch die Netzwerk-Verzögerungen zeigt, im Dasein der TCP-Ströme, eine konstante Erhöhung.
Erst mit dem Übertragungsende der TCP-Ströme steigen die SCReAM-Ströme wieder exponentiell im Durchsatz und erreichen gegen Versuchsende eine Konvergenz.
Der Einsatz einer *REDQueue* (siehe Abbildung 5.15(b)) zeigt keine gravierenden Unterschiede zur Durchführung des Experiments mit einer *Drop Tail Queue*.
Der einzige Unterschied zur Durchführung mit einem *Propgation Delay* von 100ms, ist die Streuung der Messwerte, nachdem die TCP-Ströme ihre Übertragungen beendet haben (siehe Abbildung 5.16(a) und 5.16(b)). Der Grund für die Streuung der Messwerte, die sich im Konfidenzintervall bemerkbar macht, ist das erhöhte *Propagation Delay*. Dies führt unter Verwendung

von mehreren *Seeds*, in diesem Szenario, zu einer breiteren Werteverteilung.

Nachdem die einzelnen Experimente, Erwartungswerte an die eigenen Messergebnisse und schließlich die eigenen Messergebnisse im Detail vorgestellt wurden, folgt eine ausführliche Diskussion.

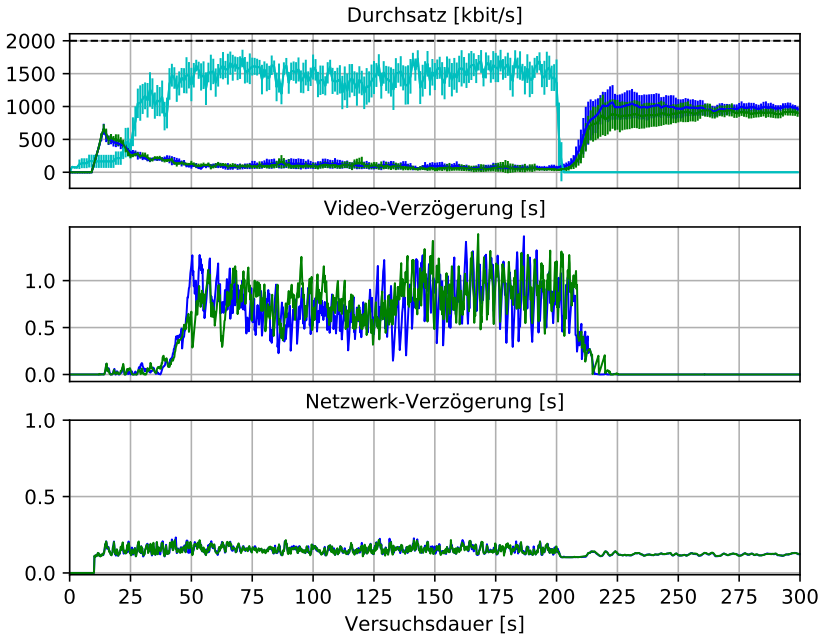


(a) Eigene Messung: TC5.7, 300ms *Drop Tail*

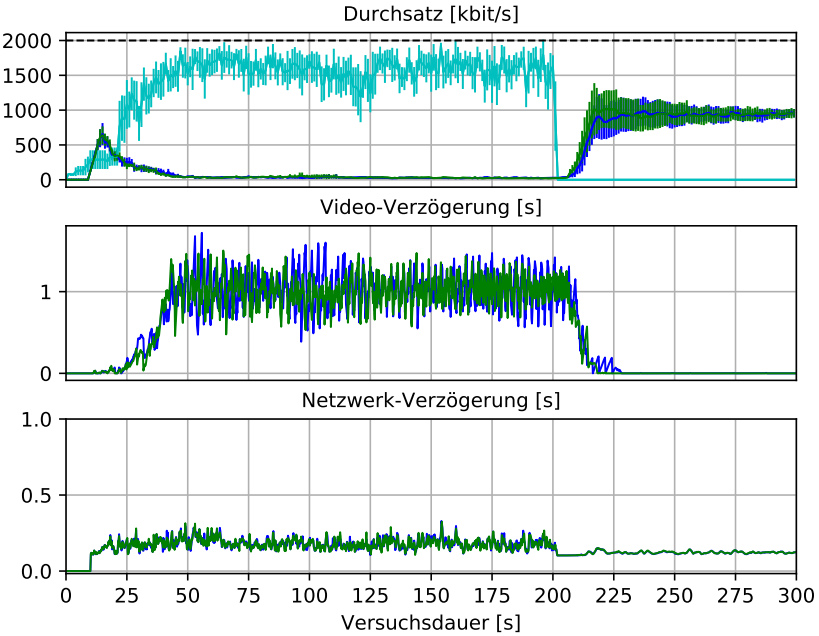


(b) Eigene Messung: TC5.7, *REDQueue*

Abbildung 5.15: TC5.7 RTT100



(a) Eigene Messung: TC5.7, 300ms *Drop Tail*



(b) Eigene Messung: TC5.7, *REDQueue*

Abbildung 5.16: TC5.7 RTT200

6 Diskussion

Mit jedem einzelnen der sieben Experimente wurde die SCReAM *Congestion Control* in seinen grundlegenden Funktionen analysiert. Anhand der vor jedem Experiment beschriebenen Erwartungswerte an die eigenen Messergebnisse, wird nun diskutiert, ob die eigenen Messungen geeignete Ergebnisse anbieten. Dabei werden Unterschiede in den Messergebnissen identifiziert und es wird eine potentielle Entstehungsursache diskutiert.

Variable Pfadkapazität mit einem SCReAM-Strom

In diesem Experiment wurde das Verhalten von einem SCReAM-Strom bei einer variablen Pfadkapazität untersucht. Die Messergebnisse, zur Durchführung mit einem *Propagation Delay* von 50ms, ähneln sich sehr. Doch sind bei einer genaueren Betrachtung der Ergebnisse, Unterschiede zu erkennen.
Nach Pfadkapazitätsänderung **Drei** fallen beide Graphen im Vergleich gleichmäßig, doch passt sich der Durchsatz der eigenen Messung, schneller wieder an die Pfadkapazität an. Daraus resultiert ein etwas breiterer Rumpf der Spitzen in der Video- und Netzwerk-Verzögerung als in der Referenzabbildung. Aufgrund der sofortigen Anpassung im Durchsatz, bedarf es mehr Zeit bis der Datenstau im Netzwerk und der RTP-*Queue* abgearbeitet ist.
Ein weiterer grundlegender Unterschied ist das Steigungsverhalten nach Pfadkapazitätsänderung **Vier**. Der Graph vom Durchsatz der eigenen Messung reagiert deutlich langsamer auf die Änderung und erreicht damit verbunden auch später die verfügbare Pfadkapazität.
Auffällig in der Video-Verzögerung der eigenen Messung ist, dass bis auf der ausgeprägten Spitze, keine weitere Verzögerung vorhanden ist. Es werden also genauso schnell RTP-Pakete aus der *Queue* genommen, wie sie kodiert und *gequeued* werden.
In der Referenzmessung hingegen besteht eine geringe Verzögerung. Aus der Messung wird nicht ersichtlich wie die Video-Verzögerung zustande kommt. Doch ähnelt die durchgängige Verzögerung stark dem des *Propagation Delays*.
Die dargestellte Video-Verzögerung der eigenen Messung und der noch folgenden Messungen setzt sich aus der Anzahl der RTP-Pakete in der RTP-*Queue* zusammen. Denn die RTP-*Queue* wird beschrieben als Stoßdämpfer und als die meiste Zeit leer [5].

Zu den bereits vorgestellten Unterschieden kommt mit der Diskussion, zur Durchführung des Experiments mit einem *Propagation Delay* von 100ms, eine weitere Diskrepanz hinzu. Die Maxima der Spitzen, der eigenen Messergebnisse, sind deutlich höher als die in der Referenzmessung.

Obwohl das Experiment unter Verwendung von verschiedenen Codecs durchgeführt wurde, zeigen die eigenen Messergebnisse insgesamt ein vergleichbares Verhalten zu den Referenzmessungen.

Variable Pfadkapazität mit zwei SCReAM-Strömen

In diesem Experiment wurde das Verhalten von zwei SCReAM-Strömen bei einer variablen Pfadkapazität untersucht. Zunächst werden die Unterschiede, zur Durchführung mit einem eingestellten *Propagation Delay* von 50ms vorgestellt.
In Folge der Pfadkapazitätsänderung **Drei** divergieren die Graphen im Durchsatz, während sich die Graphen der Referenzabbildung die Pfadkapazität gleichmäßig aufteilen.
Gegensätzlich teilen sich die SCReAM-Ströme den Durchsatz nach der Pfadkapazitätsänderung **Vier** nahezu gleich auf, während die SCReAM-Ströme der Referenzabbildung deutlich divergieren.
Ein weiterer Unterschied ist in Folge der Pfadkapazitätsänderung **Drei** in der Video-Verzögerung zu erkennen. Denn das Maximum der Spitze ist mit einer halben Sekunde deutlich höher als das in der Referenzabbildung. Der Grund hierfür ist die Trägheit des verwendeten Codecs, der sich deutlich langsamer an die neuen Netzwerkbedingungen anpasst.
Bei der Durchführung des Experiments mit einem *Propagation Delay* von 100ms, teilen sich die Graphen im Durchsatz die Pfadkapazität sichtlich fairer als die Graphen in der Referenzabbildung. Dennoch ist in Folge der Pfadkapazitätsänderung **Drei**, bis zum Ende des Versuchs, eine deutliche Streuung der eigenen Messwerte zu beobachten, was sich auch in der Video-Verzögerung bemerkbar macht.
Wie auch in dem vorherigen Experiment, sind die Maxima der Spitzen in Folge der zweiten Reduzierung der Pfadkapazität, deutlich höher als die in der Referenzabbildung.

Abgesehen von der erhöhten Video- und Netzwerk-Verzögerung, ähneln sich die eigenen Messergebnisse und die Referenzmessungen. Insgesamt teilen sich die SCReAM-Ströme, der eigenen Messergebnisse, die Pfadkapazität deutlich fairer, während die der Referenzmessung immer mal wieder divergieren.

Überlast im Feedback-Pfad

Mit diesem Experiment wurde der Einfluss, von Überlast im Feedback-Pfad, auf den Durchsatz der SCReAM-Ströme geprüft. Zuerst werden die Messergebnisse zur Durchführung mit einem *Propagation Delay* von 50ms diskutiert.
In Folge der Pfadkapazitätsänderung **Zwei** in Pfadrichtung *Vorwärts* sind, im Gegensatz zur Referenzabbildung, beide SCReAM-Ströme betroffen. Während der Durchsatz des SCReAM-Stroms in Pfadrichtung *Rückwärts* ebenfalls einbricht, zeigt der SCReAM-Strom in der Referenzabbildung keine Reaktion und behält seinen Durchsatz bei.
Auch in Folge der Pfadkapazitätsänderung **Drei** in Pfadrichtung *Vorwärts*, zeigt der Graph, dessen Feedback-Pfad betroffen ist, keine Reaktion, während der Graph der eigenen Messergebnisse seinen Durchsatz reduziert. Der Grund hierfür ist der SCReAM-Strom, der in Folge der Pfadkapazitätsänderung **Zwei** in Pfadrichtung *Rückwärts*, bereits im Durchsatz gefallen ist. Dementsprechend befindet er sich zum Zeitpunkt der Änderung bereits unterhalb der verfügbaren Pfadkapazität, sodass es zu keiner Überlast kommt.
Mit der Durchführung des Experiments mit einem *Propagation Delay* von 100ms, kommt ein wesentlicher Unterschied hinzu. Die erhöhte durchgängige Netzwerk-Verzögerung des SCReAM-Stroms in Pfadrichtung *Vorwärts*, zwischen 40s und 60s.

Im Steigungsverhalten und bei der Beanspruchung der Pfadkapazität zeigen die eigenen Messergebnisse ein vergleichbares Verhalten zu den Referenzmessungen. Doch ist es die Motivation von diesem Experiment, den Einfluss von Überlast auf den Feedback-Pfad zu prüfen. Und unter diesem Gesichtspunkt unterscheiden sich die eigenen Messergebnisse erheblich von denen der Referenzmessungen. Denn sie zeigen in jedem Überlast-Szenario des Feedback-Pfades den Einbruch des dazugehörigen SCReAM-Stroms.

Konkurrierende SCReAM-Ströme

In diesem Experiment teilten sich drei SCReAM-Ströme, mit gleicher *Round Trip Time* und unterschiedlichen Startzeiten, einen *Bottleneck*-Pfad. Die Motivation ist es die *Intra-Protocol-Fairness* auf die Probe zu stellen.
Der Vergleich der Messergebnisse offenbart beträchtliche Unterschiede. Während die SCReAM-Ströme der eigenen Messung nach 120s annähernd eine Konvergenz im Durchsatz aufweisen, beansprucht der zuerst gestartete SCReAM-Strom der Referenzmessung, nach 120s, immer noch einen großen Teil der Pfadkapazität.
Des Weiteren kommt die, in der Diskussion vom ersten Experiment, angesprochene durchgän-

gige Video-Verzögerung hinzu.

Insgesamt ist festzuhalten, dass die SCReAM-Ströme der eigenen Messung deutlich schneller eine Konvergenz erzielen als die SCReAM-Ströme der Referenzmessung, die selbst mit verlängerter Versuchsdauer nach 200s keine gemeinsame Konvergenz erreichen.

Konkurrierende SCReAM-Ströme mit verschiedenen RTTs

Bei diesem Experiment wurde das Verhalten von fünf SCReAM-Strömen, mit unterschiedlicher *Round Trip Time* und unterschiedlichen Startzeiten, in einem *Bottleneck*-Szenario untersucht. In beiden Abbildungen erreicht der zuerst gestartete SCReAM-Strom eine Pfadkapazität von 3000 kbit/s. Doch in Folge der später startenden SCReAM-Ströme gibt der zu Beginn gestartete SCReAM-Strom, in der eigenen Messung, wesentlich schneller Pfadkapazität an die anderen ab. Nach 100s beansprucht dieser noch 1000 kbit/s, während der SCReAM-Strom der Referenzmessung erst 200s später, im Durchsatz unterhalb von 1000 kbit/s fällt.

Alles in allem zeigen die Messergebnisse, bis auf den ersten SCReAM-Strom, aber ein vergleichbares Verhalten. In beiden Messungen erreichen die SCReAM-Ströme keine Konvergenz, sind aber nahe beieinander.

SCReAM-Strom konkurriert mit einem langlebigen TCP-Strom

In diesem Experiment wurde das Verhalten eines SCReAM-Stroms mit einem konkurrierenden langlebigen TCP-Stroms geprüft. Aufgrund der nicht vergleichbaren Referenzen, wird der Erwartungswert an das Experiment diskutiert. Die Erwartungshaltung an das Experiment ist, dass der SCReAM-Strom gegen den TCP-Strom nicht verliert.
Alle visualisierten Messergebnisse zeigen einen SCReAM-Strom der im Durchsatz auf sein Minimum beschränkt ist und einem TCP-Strom, der nahezu die komplette Pfadkapazität einnimmt. Der Grund hierfür ist das schon in der Sektion 1.2 angesprochene Problem der loss-based *Congestion Control* von TCP, die dafür sorgt, dass die Buffer im Netzwerk gefüllt werden.

Dementsprechend kann die Erwartungshaltung an das Experiment nicht bestätigt werden, denn in allen Durchführungen, verliert der SCReAM-Strom gegen den TCP-Strom.

Zwei SCReAM-Ströme konkurrieren mit mehreren kurzlebigen TCP-Strömen

In dem letzten Experiment konkurrieren zwei SCReAM-Ströme mit mehreren kurzlebigen TCP-Strömen, die das Verhalten im Web simulieren. Auch in diesem Experiment konnten keine Referenzabbildungen zum Vergleich herangezogen werden. Aus diesem Grund wird wiederum die Erwartungshaltung an das Experiment diskutiert. Diese ist die gleiche wie im vorherigen Experiment. Es wird erwartet, dass die SCReAM-Ströme nicht gegenüber den konkurrierenden TCP-Strömen verlieren.
Auch in diesem Experiment zeigt sich der Einfluss der loss-based *Congestion Control*, in allen Versuchsdurchführungen. Mit dem Einsatz einer *Drop Tail Queue* oder einer *REDQueue*, in allen visualisierten Messergebnissen fallen die SCReAM-Ströme im Durchsatz auf ihr Minimum, während die TCP-Ströme zusammen nahezu die komplette Pfadkapazität einnehmen.
Zumindest nach Beendigung der Übertragungen der TCP-Ströme, steigen die SCReAM-Ströme wieder schnell im Durchsatz und erreichen eine Konvergenz.
Festzuhalten ist, dass zumindest der TCP-Strom seinen Dienst sicherstellen kann, jedoch ist der Echtzeit-Strom in einem *Bottleneck*-Szenario mit einem konkurrierenden TCP-Strom nicht nutzbar.

7 Fazit

Das Ziel dieser Arbeit bestand darin, die Referenzimplementierung des *Congestion Control* Algorithmus SCReAM, in das vorhandene RTP-Modell des *INET Frameworks* zu integrieren und anhand von Vergleichen mit Referenzmessungen, aus einer alternativen Implementation, zu evaluieren.
Die Experimente mit einer **variablen Pfadkapazität** haben gezeigt, dass die Video- und Netzwerk-Verzögerung der eigenen Messung in Lastsituation höher ist als die in den Referenzmessungen. Insgesamt wurde der Erwartungswert an die eigenen Messergebnisse aber erfüllt.
Mit dem Experiment **Überlast im Feedback-Pfad** hingegen konnte der Erwartungswert nicht erfüllt werden. Denn in den eigenen Messergebnissen konnte, in jedem Überlast Szenario des Feedback-Pfades, der Einbruch des dazugehörigen SCReAM-Stroms beobachtet werden.
Das Experiment mit mehreren **konkurrierenden SCReAM-Strömen** und unterschiedlichen Startzeiten hat offenbart, dass in den eigenen Messungen der zuerst gestartete SCReAM-Strom deutlich schneller seinen Durchsatz reduziert. Dadurch wird erheblich schneller eine Konvergenz der SCReAM-Ströme erreicht.
Das Experiment mit mehreren **konkurrierenden SCReAM-Strömen** und zusätzlich unterschiedlichen RTTs, hat ebenfalls eine deutlich schnellere Reduzierung vom Durchsatz des zuerst gestarteten SCReAM-Stroms offenbart. Abgesehen davon, wird dennoch der Erwartungswert an die Messergebnisse erfüllt, da die SCReAM-Ströme, nach erfolgter Anpassung der Senderate vom ersten SCReAM-Strom, ein vergleichbares Verhalten vorweisen.
Die Experimente mit **konkurrierenden TCP-Strömen** haben die Probleme des SCReAM *Congestion Control* Algorithmus aufgezeigt. Denn die SCReAM-Ströme mit ihrer delay-based *Congestion Control* verlieren in einem *Bottleneck*-Szenario gegenüber einer loss-based *Congestion Control*. Die SCReAM-Ströme können ihre Dienstgüte und Nutzungserfahrung nicht mehr gewährleisten.

Dahingehend liegt der Ausblick für weitere Arbeiten, in der Verbesserung der SCReAM *Congestion Control*, um in einem *Bottleneck*-Szenario gegenüber einem TCP-Strom seine Dienstgüte durchzusetzen. Ein erster Ansatz ist die dynamische Anpassung des *Queuing Delay Targets* bei

der Präsenz von konkurrierenden TCP-Strömen, sowie es bereits in den *Congestion Control* Algorithmen GCC und NADA geschieht [48].

Literaturverzeichnis

[1] W3C. *World Wide Web Consortium.* URL: www.w3.org (aufgerufen am 15.08.2017).

[2] IETF. *Internet Engineering Task Force (IETF).* URL: www.ietf.org (aufgerufen am 15.08.2017).

[3] Ian Fette and Alexey Melnikov. *The WebSocket Protocol.* RFC 6455, Dezember 2011. URL: https://tools.ietf.org/html/rfc6455 (aufgerufen am 23.08.2017).

[4] WebRTC. *Web Real-Time Communication.* URL: www.webrtc.org (aufgerufen am 15.08.2017).

[5] I. Johansson. Self-clocked rate adaptation for conversational video in lte. *Proc. 2014 ACM SIGCOMM Wksp. Capacity Sharing Workshop*, August 2014. Chicago, USA, pp. 51 - 56.

[6] C++ Implementation of SCReAM. URL: https://github.com/EricssonResearch/openwebrtc-gst-plugins.

[7] R. Frederick H. Schulzrinne, S. Casner and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550, Juli 2003. URL: https://www.rfc-editor.org/std/std64.txt (aufgerufen am 30.08.2017).

[8] INET Framework. URL: https://inet.omnetpp.org/.

[9] OMNeT++. URL: https://omnetpp.org/.

[10] RMCAT. *RTP Media Congestion Avoidance Techniques.* URL: https://datatracker.ietf.org/wg/rmcat/about/ (aufgerufen am 15.08.2017).

[11] Z. Sarker et al. *Test Cases for Evaluating RMCAT Proposals.* Internet-Draft, Oktober 2016. URL: https://tools.ietf.org/html/draft-ietf-rmcat-eval-test-04 (work in progress).

[12] Ana Hernandez and Eduardo Magana. One-way delay measurement and characterization. pages 114–114, June 2007. doi: 10.1109/ICNS.2007.87.

[13] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960, September 2007. URL: https://tools.ietf.org/html/rfc4960 (aufgerufen am 30.08.2017).

[14] M. Welzl S. Islam and S. Gjessing. *Coupled congestion control for RTP media*. Internet-Draft, März 2017. URL: https://tools.ietf.org/html/draft-ietf-rmcat-coupled-cc-06 (work in progress).

[15] J. Postel. *Transmission control protocol*. RFC 793, September 1981. URL: https://tools.ietf.org/html/rfc793 (aufgerufen am 24.08.2017).

[16] V. Paxson M. Allman and E. Blanton. *TCP Congestion Control*. RFC 5681, September 2009. URL: https://tools.ietf.org/html/rfc5681 (aufgerufen am 11.09.2017).

[17] Congestion Control for Real-time Communication. URL: http://c3lab.poliba.it/MultimediaCC.

[18] J. Postel. *User Datagram Protocol*. RFC 768, August 1980. URL: https://tools.ietf.org/html/rfc768 (aufgerufen am 24.08.2017).

[19] Analyzing UDP usage in Internet traffic. URL: https://www.caida.org/research/traffic-analysis/tcpudpratio/.

[20] J Randell and Z. Sarker. *Congestion Control Requirements for RMCAT*. Internet-Draft, Dezember 2014. URL: https://tools.ietf.org/html/draft-ietf-rmcat-cc-requirements-09 (work in progress).

[21] Ilya Grigorik. High performance browser networking. *O'Reilly Media*, Sept. 2013.

[22] Cullen Jennings Anant Narayanan Adam Bergkvist, Daniel C. Burnett. *WebRTC 1.0: Real-time Communication Between Browsers*. W3C Working Draft, August 2017. URL: https://www.w3.org/TR/webrtc/ (work in progress).

[23] P. Srisuresh and M. Holdrege. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663, August 1999. URL: https://tools.ietf.org/html/rfc2663 (aufgerufen am 10.09.2017).

[24] J. Rosenberg. *Interactive Connectivity Establishment (ICE)*. RFC 5245, April 2010. URL: https://tools.ietf.org/html/rfc5245 (aufgerufen am 29.08.2017).

[25] J. Rosenberg and H. Schulzrinne. *An Offer/Answer Model with the Session Description Protocol (SDP)*. RFC 3264, Juni 2002. URL: https://tools.ietf.org/html/rfc3264 (aufgerufen am 30.08.2017).

[26] R. Mahy J. Rosenberg and P. Matthews. *Session Traversal Utilities for NAT (STUN).* RFC 5389, Oktober 2008. URL: https://tools.ietf.org/html/rfc5389 (aufgerufen am 29.08.2017).

[27] P. Matthews R. Mahy and J. Rosenberg. *Traversal Using Relays around NAT (TURN).* RFC 5766, April 2010. URL: https://tools.ietf.org/html/rfc5766 (aufgerufen am 29.08.2017).

[28] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security*. RFC 6347, Januar 2012. URL: https://tools.ietf.org/html/rfc6347 (aufgerufen am 30.08.2017).

[29] M. Naslund E. Carrara M. Baugher, D. McGrew and K. Norrman. *The Secure Real-time Transport Protocol (SRTP).* RFC 3711, März 2004. URL: https://www.rfc-editor.org/rfc/rfc3711.txt (aufgerufen am 30.08.2017).

[30] N. Sato C. Burmeister J. Ott, S. Wenger and J. Rey. *Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)* . RFC 4585, Juli 2006. URL: https://tools.ietf.org/html/rfc4585 (aufgerufen am 03.09.2017).

[31] Q. Xie M. Tuexen R. Stewart, M. Ramalho and P. Conrad. *Stream Control Transmission Protocol (SCTP) - Partial Reliability Extension.* RFC 3758, Mai 2004. URL: https://tools.ietf.org/html/rfc3758 (aufgerufen am 30.08.2017).

[32] RTCPeerConnection. *RTCPeerConnection API*. URL: https://www.w3.org/TR/webrtc/#peer-to-peer-connections (aufgerufen am 15.08.2017).

[33] V. Jacobson M. Handley and C. Perkins. *SDP: Session Description Protocol.* RFC 4566, July 2006. URL: https://tools.ietf.org/html/rfc4566 (aufgerufen am 25.08.2017).

[34] RTCdatachannel. *RTCdatachannel API*. URL: https://www.w3.org/TR/webrtc/#rtcdatachannel (aufgerufen am 15.08.2017).

[35] G. Carlucci et al. Analysis and design of the google congestion control for web real-time communication (webrtc). *Proc. ACM Multimedia Systems Conf.*, Mai 2016. doi: 10.1145/2910017.2910605.

[36] X. Zhu et al. *NADA: A Unified Congestion Control Scheme for Real-Time Media.* Internet-Draft, März 2017. URL: https://tools.ietf.org/html/draft-ietf-rmcat-nada-04 (work in progress).

[37] I. Johansson and Z. Sarker. *Self-Clocked Rate Adaptation for Multimedia*. Internet-Draft, September 2016. URL: https://tools.ietf.org/html/draft-ietf-rmcat-scream-cc-09 (work in progress) (aufgerufen am 12.06.2017).

[38] Van Jacobson and Michael J. Karels. *Congestion Avoidance and Control*. URL: http://ee.lbl.gov/papers/congavoid.pdf (aufgerufen am 03.09.2017).

[39] B. Burman M. Westerlund and F. Jansson. *Multiple Synchronization sources (SSRC) in RTP Session Signaling*. Internet-Draft, April 2012. URL: https://tools.ietf.org/html/draft-westerlund-avtcore-max-ssrc-00 (work in progress).

[40] C. Perkins P. O'Hanlon M. Westerlund, I. Johansson and K. Carlberg. *Explicit Congestion Notification (ECN) for RTP over UDP*. RFC 6679, August 2012. URL: https://tools.ietf.org/html/rfc6679 (aufgerufen am 03.09.2017).

[41] I. Johansson. *Self-clocked Rate Adaptation for Conversational Video in LTE*. URL: http://conferences.sigcomm.org/sigcomm/2014/doc/slides/150.pdf (aufgerufen am 25.07.2017).

[42] J. Iyengar S. Shalunov, G. Hazel and M. Kuehlewind. *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817, Dezember 2012. URL: https://tools.ietf.org/pdf/rfc6817.pdf (aufgerufen am 05.09.2017).

[43] V. Singh and J. Ott. *Evaluating Congestion Control for Interactive Real-time Media*. Internet-Draft, September 2016. URL: https://tools.ietf.org/html/draft-ietf-rmcat-eval-criteria-06 (work in progress).

[44] INET Fork with SCReAM implementation. URL: https://github.com/Simse92/inet/tree/master/src/inet/transportlayer/rtp.

[45] Synthetic codecs for evaluation of RMCAT work. URL: https://github.com/cisco/syncodecs.

[46] Peter McCullagh. What is a statistical model? *Ann. Statist.*, 30(5):1225–1310, 10 2002. doi: 10.1214/aos/1035844977. URL https://doi.org/10.1214/aos/1035844977.

[47] I. Johansson and Z. Sarker. *IETF RMCAT presentation (final for WGLC)*. URL: https://www.ietf.org/proceedings/96/slides/slides-96-rmcat-0.pdf (aufgerufen am 25.07.2017).

[48] Gaetano Carlucci und Saverio Mascolo Luca De Cicco. Congestion control for webrtc: Standardization status and open issues. *IEEE Communications Standards Magazine*, June 2017.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. November 2017 Simon Sorgenfrei