UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Forwarding Loops

Diplomarbeit

Zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Mario Wendling und Holger Breitbach

Erstgutachter: Prof. Dr. Hannes Frey
(Institut für Informatik, AG Rechnernetze)

Zweitgutachter: Dipl.-Inform. Frank Bohdanowicz
(Institut für Informatik, AG Rechnernetze)

Koblenz, im Juli 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

Ort, Datum Unterschrift

Ort, Datum Unterschrift

Inhaltsverzeichnis

[1] Bearbeitet von Holger Breitbach
[2] Bearbeitet von Mario Wendling

ABBILDUNGSVERZEICHNIS

1 EINLEITUNG[1]

Das Internet unterliegt einem rapiden Wachstum. Neben immer schnelleren Zugängen und ausgefeilteren Protokollen führen nicht zuletzt auch neue „Nutzungsmodelle“ wie Social Networking zu einem immer stärkeren Vordringen in den Alltag der Gesellschaft. Stetig wachsendes Datenvolumen führt zu einer stärkeren Vernetzung des Internet. Messungen haben ergeben, dass alleine in den Jahren von 2007 bis 2009 der Datenverkehr jährlich um 44,5% wuchs [1]. Die traditionelle Topologie des Internet weicht stetig einer immer besseren Vernetzung, um Inhalte möglichst schnell und ausfallfrei zum Nutzer transportieren zu können. Heutzutage fließt der meiste Verkehr von wenigen (Google, Facebook) großen Content Providern (CDN) direkt zu den Nutzern. Dabei geraten einige vor ein paar Jahren noch weit verbreitete Protokolle (z.B. P2P) zunehmend in den Hintergrund. TCP/UDP inklusive Flash und Video Over HTTP dominieren weitgehend [1].

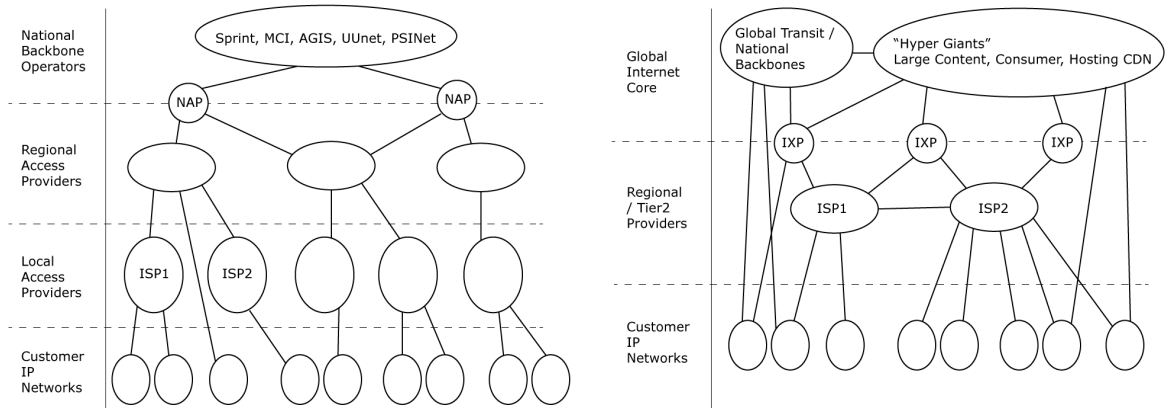


Abbildung 1: Traditional & New Internet Logical Topology [1]

Das Internet ist in Autonome Systeme (AS) untergliedert. Routing-Informationen werden über EGP (Exterior-Gateway-Protokoll) zwischen Autonomen Systemen und innerhalb über IGP (Interior-Gateway-Protokoll) ausgetauscht. Der Vorgänger von BGP, das heute nicht mehr verwendete Exterior-Gateway-Protokoll, diente als Namensgeber für die Oberklasse der EGP und wurde vollständig durch BGP verdrängt [2]. EGPs bauen auf die durch die IGPs sichergestellten Informationen auf, d.h. es wird pro Netzwerk im Normalfall ein EGP und ein IGP parallel verwendet. Je nach Topologie kommen heutzutage meistens OSPF, EIGRP oder

IS-IS als IGP zum Einsatz. RIP und IGRP wurden aufgrund ihrer einfachen Struktur weitgehend verdrängt [3].

Die dichtere Vernetzung ermöglicht eine höhere Redundanz im Fehlerfall (Ausfall eines Links, Stau/Überlastung). Weitgehend alle modernen und in der Praxis verwendeten IGPs und EGPs (in BGP ist dieses Feature herstellerspezifisch) unterstützen mehrere konkurrierende Pfade zum gleichen Ziel. Konkurrierende Pfade und eine bessere Vernetzung bergen das Risiko der Entstehung von Schleifen (Forwarding Loops).

Die Zeit, die ein Routing Protokoll benötigt, um Änderungen/Fehler durch das gesamte Netz zu propagieren, nennt man Konvergenzzeit. In der Konvergenzzeit besteht eine Inkonsistenz an Routing-Informationen und es kann zur Bildung von temporären Schleifen (Transient Loops) kommen. Eine längere Konvergenzzeit hat maßgeblichen Einfluss auf die Dauer von temporären Schleifen.

Schleifen können auf der anderen Seite dauerhaft (Persistent Loops) durch Administrationsfehler und Unterschiede in der Implementierung von IGP (RIP, OSPF, IS-IS etc.) oder EGP (BGP) entstehen. Dabei spielt das stetige Wachstum und die dichtere Vernetzung des Internet eine unterstützende Rolle für die Schleifenbildung. Techniken wie Aggregation in Verbindung mit CIDR fassen gemeinsame Routen zusammen und versuchen die Routing-Tabellen zu minimieren. Dabei werden neue schwer überschaubare Risiken geschaffen, die letztlich auch die Schleifenbildung unterstützen. Dadurch entstandene persistente Schleifen schwächen die Stabilität und Robustheit des Internet. Weiterhin intensivieren sie DDOS-Angriffe für Hacker [4].

Im zweiten Kapitel analysieren wir den aktuellen Datenverkehr und suchen nach Möglichkeiten, diesen zu reproduzieren. Das dritte Kapitel beschäftigt sich mit Grundlagen zu Routing-Protokollen und den Entstehungsgründen und der Verteilung von Schleifen im heutigen Internet. Im vierten Kapitel werden in einem Praxisversuch Schleifen erzeugt und die Auswirkung auf das Netzwerk abhängig vom verwendeten Protokoll untersucht.

Der Verfasser des Kapitels wird mit einer Fußnote hinter der jeweiligen Überschrift markiert. [1] bedeutet dabei, dass die Bearbeitung von Holger Breitbach erfolgte, bei [2] war der Verfasser Mario Wendling.

2 ANALYSE DES DATENVERKEHRS[2]

Bevor wir uns der Aufgabe stellen, das Verhalten von Datenpaketen und deren Auswirkungen in Forwarding-Schleifen zu untersuchen, wollen wir uns in diesem Kapitel damit beschäftigen, was Internetverkehr beinhaltet und welche Möglichkeiten und Messmethoden zur Verfügung stehen.

Neben einem Rückblick auf die Entwicklung des Internets in Bezug auf dessen Nutzung und den damit verbundenen wachsenden Ansprüchen wollen wir dabei auch die möglichen zukünftigen Entwicklungen betrachten. Dabei sind neben den technischen Anforderungen und Problemen auch die rechtlichen sowie auch kommerziellen Aspekte zu beachten.

Nachdem wir die einzelnen Bestandteile des Internetverkehr erforscht haben, werden wir auch die einzelnen Protokolle auf deren Routing-Eigenschaften untersuchen und beschreiben, welche möglichen Probleme eine Schleife in einer Netzwerktopologie hervorrufen könnte.

2.1 ENTWICKLUNG DES INTERNET[2]

Das Internet stellt heutzutage die Schlüsselkomponente zur privaten und kommerziellen Kommunikation dar und aufgrund seiner Vielseitigkeit und Flexibilität ist ein Ende der stetigen Erweiterung nicht absehbar. Nahezu jedes elektronische Gerät, vom gewöhnlichen Heimcomputer, über Serversysteme und Supercomputer, aber auch immer mehr mobile Endgeräte oder einfache Sensoren übermitteln heute ihre Daten über das Internet. Dementsprechend hat sich die Nutzung dieser stetig wachsenden Infrastruktur erheblich seit ihrer Einführung in den frühen 80er Jahren geändert, als noch eine überschaubare Anzahl an Geräten und Einrichtungen sowie verschiedene Dienste über das Internet miteinander kommunizierten.

Der Erfolg und das schnelle Wachstum des Internet ist auf seinem zentralen Element, dem Internetprotokoll (IP) begründet. Obwohl sich die benutzten Protokolle oberhalb und unterhalb dieses zentralen Protokolls im OSI-Referenzmodell signifikant geändert und weiterentwickelt haben, ist das Internetprotokoll seit über 30 Jahren in seinen Grundfunktionen nicht verändert worden.

Unterhalb des IP-Protokolls haben sich die Übertragungstechniken von einfachen Kupferleitungen hin zu Glasfasertechniken entwickelt und auch die kabellosen Übertragungswege transportieren heutzutage statt Kbit/s schon Bandbreiten im mehrstelligen Gigabit-Bereich pro Sekunde. Zusätzlich wurde der Einsatz sog. Middleware (NAT-Systeme, Firewalls etc.) immer größer.

Oberhalb der Transportschicht, auf welcher IP einzuordnen ist, sind zusätzlich ständig neue Anwendungsprotokolle hinzugekommen, welche sich wiederum den Transporteigenschaften des Internetprotokolls bedienen. Neben den grundsätzlichen Funktionen wie dem Domain Name System (DNS) oder dem Hypertext Transfer Protokol (HTTP) sind auch sehr komplexe Ende-zu-Ende (P2P) Protokolle entwickelt worden, welche IP Telefonie, Videostreaming und Datenaustausch ermöglichen.

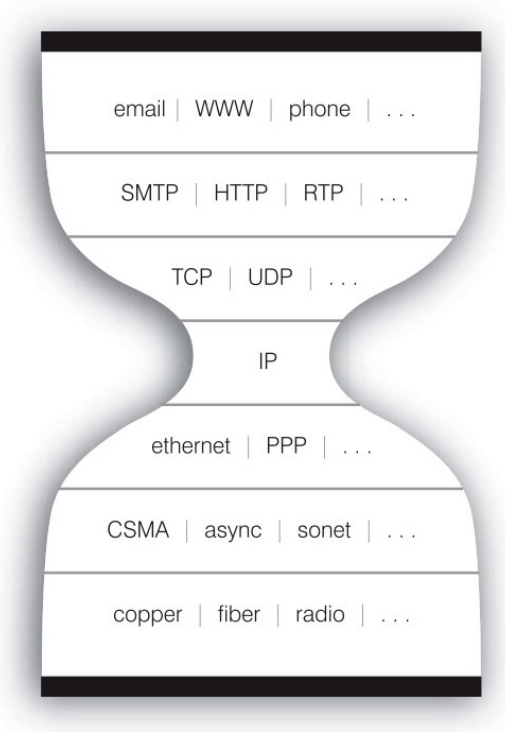


Abbildung 2: IP-Hourglass Schema

Diese ständig wachsende Anzahl an Anwendungsprotokollen und Technologien zur Datenübertragung führt zu einer immer größer werdenden Komplexität des Internet im Gesamten. Einzelne Technologien und Protokolle werden bei der Entwicklung oft nur isoliert getestet und auf ihr Verhalten hin untersucht. Die Auswirkungen in der Interaktion mit der Vielzahl anderer Protokolle oder Übertragungstechniken sind auch nur schwer zu testen,

besonders im weltweit umfassenden Maßstab des Internet. Denn das heutige Internet ist das Ergebnis eines unkontrollierten Wachstumsprozesses, bei dem unabhängige und in sich abgeschlossene Netzwerke zusammengeführt worden sind (INTERconnected-NETworks). Dabei hat jedes dieser einzelnen Netzwerke oder auch Autonomen-Systeme (AS) seine ganz eigenen Routing-Strategien, nicht zuletzt auf Grund verschiedener Transitgebühren für das jeweilige Netz. So wurde von Nelson et al. [13] von einer recht ungewöhnlichen Mischung aus Anwendungsprotokollen an einem Neuseeländischen Campus berichtet, welcher wohl auf höhere Transitkosten für den transpazifischen Netzwerkverkehr in dieser Zeit zurückzuführen war. Daraus ergibt sich, dass nicht nur die fortschreitenden technischen Entwicklungen sondern auch geographische und ökonomische Faktoren Auswirkungen auf den Datenverkehr im Internet haben.

Mit der zunehmenden Anzahl an Endgeräten stößt das Internetprotokoll jedoch an seine Grenzen, so dass mit der Einführung von IPv6 erstmals Änderungen am grundlegenden Protokoll des Internet vorgenommen wurden. Durch die damit verbundene Erweiterung des Adressraums ist es wieder möglich jedes Endgerät eindeutig zu adressieren und auf das heute übliche Verfahren der Network-Address-Translation zu verzichten. Dadurch wird auch eine direkte Erreichbarkeit jedes Teilnehmers wieder möglich, was ursprünglich ein zentrales Designelement bei der Entwicklung des Internet darstellte. Durch dieses Prinzip soll sichergestellt werden, dass nur die Endpunkte aktive Protokolloperationen ausführen und dass dazwischen liegende Netzwerk, in diesem Fall das Internet, einzig und allein als Transportnetz dient.

Ein weiteres Problem des bisher verwendeten Standard IPv4 ist ein exponentielles Wachstum der Routing-Tabellen. Dies ist darauf zurückzuführen, dass die Vergabestrategien der verfügbaren IP-Adressen mehrmals geändert wurde, was eine inzwischen starke Fragmentierung des Adressraums zur Folge hat. Dadurch führt die Routingstrategie des Internet, das Classless-Interdomain-Routing (CIDR), zu stetig wachsenden Routingtabellen, da immer mehr kleine, nicht zusammenhängende Adressbereiche zur gleichen Instanz im Internet führen und die Adressbereiche nicht mehr in den Routingtabellen aggregiert werden können (Route Aggregation).

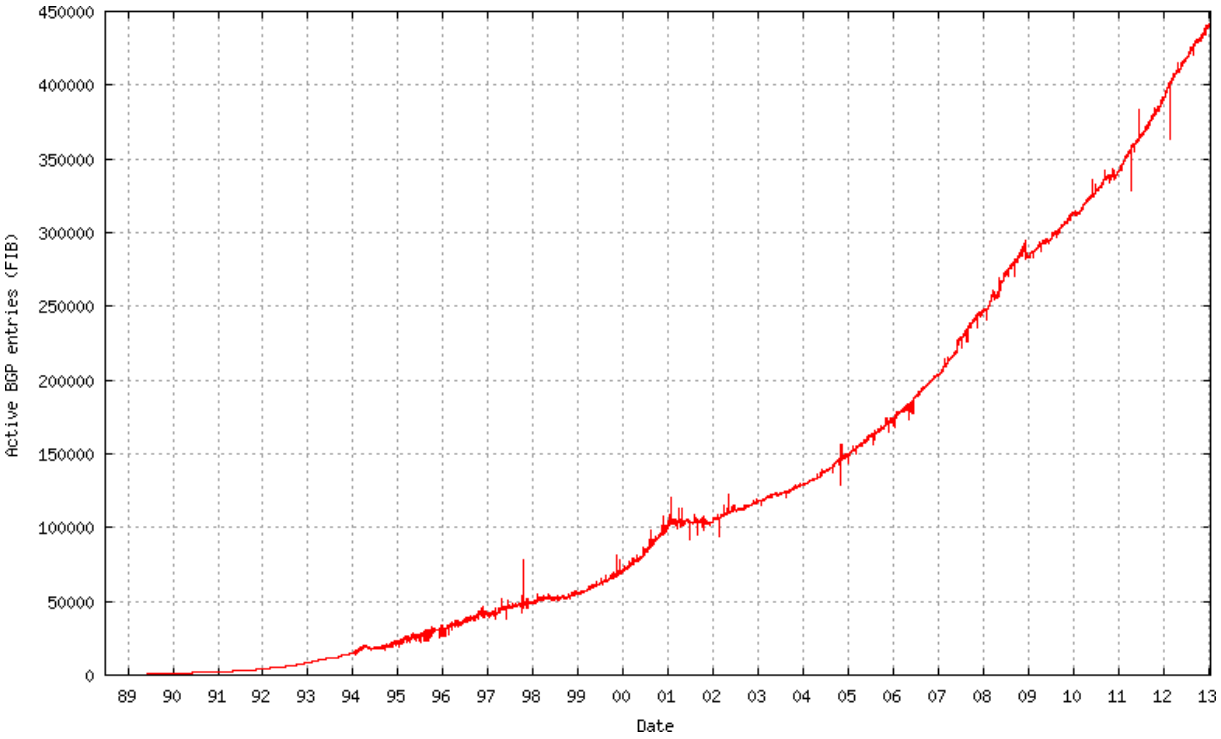


Abbildung 3: Entwicklung der BGP-Routingtabellen [5]

Über 50% der Routingeinträge zählen dabei Netzwerke mit einem /24 Präfix. Daraus resultiert neben dem höheren Speicheraufwand auch eine größere Belastung des Prozessors eines Routers, da jede Zieladresse eines Datenpakets genauer betrachtet werden muss. So sind heute für etwa 43.000 Autonome Systeme mehr als die zehnfache Anzahl an Routingeinträgen notwendig. Aufgrund einer durchdachten Strategie bei der Vergabe von IPv6 Adressen kann die Anzahl der Routingeinträge in Zukunft wieder auf ein Bruchteil reduziert werden.

2.2 ENTWICKLUNG DES DATENVOLUMENS[2]

Durch die dezentrale Organisation und Verwaltung des Internet und der verschiedenen Autonomen-Systeme ist es heute unmöglich, genaue Statistiken zum eigentlichen Transfervolumen im Internet zu erstellen. Auch unterscheiden sich die verschiedenen Voraussagen zum weiteren Wachstum verschiedener Organisationen und Unternehmen sehr stark. 1999 wurde noch von einer Verdoppelung des Datenvolumen alle 3 Monate

ausgegangen, was aber alleine schon durch das Moor‘sche Gesetzt beschränkt gewesen wäre [6]. Dies führte zu jener Zeit aber auch zu gravierenden Fehlplanungen beim Ausbau der Netzwerkstrukturen und hatte auch erhebliche wirtschaftliche Auswirkungen. Das wirtschaftliche Wachstum vieler Telekommunikationsanbieter, Hersteller von Netzwerk- und Leitungskomponenten sowie Internetdienstleister wurde massiv überschätzt, die Dotcom-Blase Anfang 2000 war das Resultat [7].

Einer der wichtigsten Hersteller für Netzwerkkomponenten, CISCO Systems, fasst jährlich verschiedene Messpunkte zusammen und wertet diese unter Berücksichtigung des durchschnittlichen Nutzerverhaltens aus. Die daraus resultierenden Daten geben demnach eine verifizierbare und realistische Aussage zum jährlichen Wachstum des Internetverkehrs.

Year	IP Traffic	Fixed Internet Traffic	Mobile Internet Traffic
	(PB/month)	(PB/month)	(PB/month)
1990	0,001	0,001	n/a
1991	0,002	0,002	n/a
1992	0,005	0,004	n/a
1993	0,01	0,01	n/a
1994	0,02	0,02	n/a
1995	0,18	0,17	n/a
1996	1,9	1,8	n/a
1997	5,4	5	n/a
1998	12	11	n/a
1999	28	26	n/a
2000	84	75	n/a
2001	197	175	n/a
2002	405	356	n/a
2003	784	681	n/a
2004	1.477	1.267	n/a
2005	2.426	2.055	0,9
2006	3.992	3.339	4
2007	6.430	5.219	15
2008	9.927	7.639	38
2009	14.414	10.676	92
2010	20.197	14.929	256
2011	27.483	20.634	597

Abbildung 4: CISCO Report - Entwicklung des Internetverkehr [8]

Zusätzlich zu den Messungen wird im VNI-Forecast eine Vorhersage über die zukünftige Entwicklung des Datenverkehrs gemacht.

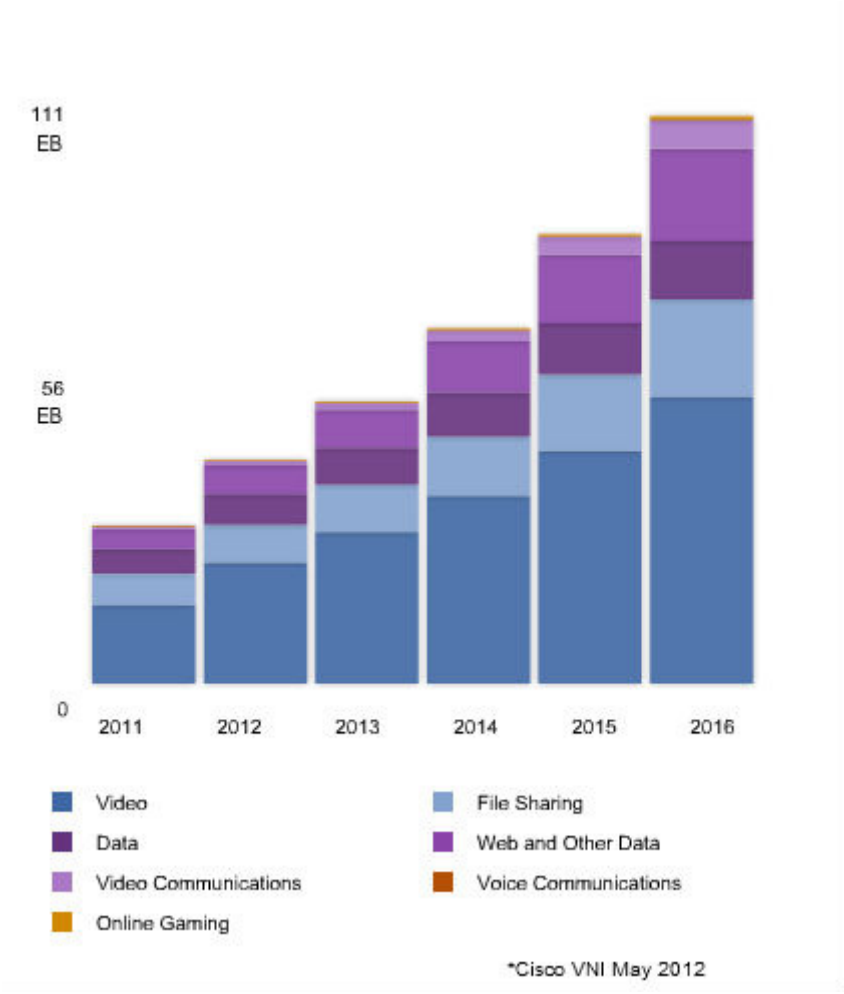


Abbildung 5: CISCO VNI-Report Mai 2012 [9]

Besonders deutlich zu erkennen ist der große Anteil an Videodaten die über das Internet versendet werden. Wie auch schon aus der VNI-Usage Statistik zum jetzigen Zeitpunkt zu entnehmen ist, ist der Anteil an Videodaten bei über 50% und wird auch in Zukunft weiter steigen. Zusätzlich werden auch Videokonferenzen einen immer größeren Anteil des Datenvolumens ausmachen.

2.3 MESSMETHODEN[2]

Die Messmethoden um Datenverkehr oder auch die Topologie eines Netzwerks zu untersuchen richten sich immer nach den gewünschten Ergebnissen und Anwendungsfällen. Je nach Zielsetzung sind dabei der Grad an Genauigkeit oder auch einfache statistische Ergebnisse erwünscht. Im Folgenden werden wir uns die fünf wichtigsten Dimensionen, in welche man diese Messmethoden unterteilen kann genauer ansehen.

2.3.1 AKTIVE UND PASSIVE MESSUNGEN[2]

Eine grundsätzliche Unterscheidung ist zwischen der aktiven und passiven Messung vorzunehmen. Während passive Messmethoden sich darauf beschränken, nur den tatsächlich vorhandenen Datenverkehr zu untersuchen, werden beim aktiven Messen Datenpakete dem Netzwerk zugefügt um die Erreichbarkeit eines Gerätes zu überprüfen (ping) oder um verschiedene Netzwerkeigenschaften zu untersuchen wie etwa die Round-Trip-Time (RTT), Bandbreite oder one-way Verzögerungen. Eine weitere aktive Messmethode ist auch *Traceroute*, welches die Untersuchung der Topologie eines Netzwerks ermöglicht.

Passive Messung, auch Monitoring genannt, basiert hingegeben auf der reinen Betrachtung des vorhandenen Datenverkehrs, ohne dass dieser von außerhalb in irgendeiner Form beeinflusst wird. Dazu wird der Netzwerkverkehr an einer bestimmten Stelle im Netzwerk abgeleitet und in einer definierten Granularität aufgezeichnet.

2.3.2 GRANULARITÄT[2]

Je nach gewünschtem Messergebnis ist eine bestimmte Genauigkeit der aufgezeichneten Daten notwendig. Mittels SNMP (Simple Network Management Protokoll) können grobe statistische Messungen auf einem Netzwerkgerät vorgenommen werden. Dabei werden z.B. die Anzahl der Pakete oder die Auslastung der einzelnen Schnittstellen erfasst. Diese Technik ermöglicht jedoch keine Analyse des Datenverkehrs bezüglich der verwendeten Transportprotokolle und liefert auch keine Informationen über die kommunizierenden Endpunkte. Der Einsatz von SNMP beschränkt sich deshalb auf den administrativen Betrieb eines Netzwerks um die Kapazitäten zu überwachen und zu planen.

Eine etwas detailliertere Messmethode ermöglicht Netflow bzw. IPFIX. Diese Technik wurde ursprünglich von Cisco entwickelt, aber schnell von anderen Herstellern wie Juniper oder Huawei adaptiert. Inzwischen existiert mit Netflow v9 ein offener Standard, am meisten Verwendung findet jedoch Version 5. Darüber hinaus wird unter dem Namen IPFIX ein herstellerunabhängiger Standard von der IETF entwickelt.

Mit Netflow werden Informationen über einen unidirektionalen IP-Datenstrom an einer Netzwerkschnittstelle gesammelt und an einen Kollektor im Netzwerk gesendet. Der Export erfolgt per UDP, so dass der Kollektor in der Lage sein muss, die anfallenden Daten schnell genug zu verarbeiten. Die dabei gesammelten Informationen werden als Flows bezeichnet und bestehen aus:

- Versionsnummer und Sequenznummer
- Zeitstempel
- Byte- und Paketzähler
- Quell- und Ziel-IP-Adressen
- Quell- und Ziel-IP-Ports
- Ingress- und Egress-Port-Nummern
- TOS-Informationen
- AS-Nummern (BGP 4)
- TCP-Flags
- Protokoll-Typ (z. B. TCP, UDP oder ICMP)

Durch die Analyse der Flows ist es daher möglich, detailliertere Aussagen über den Datenverkehr selbst zu machen. So können etwa die Teilnehmer, welche das größte Datenvolumen im Netzwerk verursachen, identifiziert werden, oder an Hand der Port Nummern die meistgenutzten Anwendungen aufgezeigt werden. Da die Mehrzahl der heutigen Internetapplikationen jedoch nur beim Verbindungsaufbau die sog. Well-Known-Ports verwenden und Netflow nur unidirektionale Flows generiert, müssen hierfür bei der Analyse stochastische Klassifizierungsmethoden angewandt werden [10].

Die detaillierteste passive Messmethode ist durch das Aufzeichnen aller einzelnen Pakete an einer Netzwerkschnittstelle gegeben. Hierbei werden die Messdaten nicht auf statistische Werte wie Datendurchsatz oder Flows zusammengefasst, sondern es wird ein exaktes Abbild

des gesamten Netzwerkverkehrs erstellt. Diese lückenlose Messung ist jedoch sehr ressourcenaufwändig und erfordert eine an die Linkgeschwindigkeit angepasste Verarbeitungsmethode. Da die Nutzdaten der einzelnen Pakete in den meisten Fällen nicht von großer Bedeutung sind wird bei diesem Verfahren auch oftmals das Paket nur bis zu einer bestimmten Länge aufgezeichnet. Die dabei abgespeicherten Informationen in den Header-Feldern eines Paketes reichen jedoch aus, um alle wichtigen Informationen aus dem so aufgezeichneten Datenstrom zu analysieren. Darüber hinaus ist das Mitschneiden der Paketdaten auch ein rechtliches Problem, da hier persönliche Daten eingesehen werden könnten, deren Überwachung nicht gestattet ist.

2.3.3 SOFTWARE- UND HARDWAREBASIERTE MESSUNGEN[2]

Passive Messmethoden, welche auf Softwarelösungen basieren, eignen sich besonders für kleine Netzwerke oder Schnittstellen mit einem geringen Datendurchsatz. Dabei werden das Betriebssystem des Host sowie dessen Netzwerktreiber modifiziert um Kopien der entsprechenden Datenpakete an dem gewünschten Messpunkt zu erhalten. Die bekanntesten Programme hierfür sind Wireshark und tcpdump. Diese Art der Messung ist leicht und auf jedem Hostsystem anzuwenden, belastet aber die Hardware des Systems zusätzlich und ist daher nicht für Messungen auf Hochgeschwindigkeitsverbindungen anwendbar. Einsatz finden solche softwarebasierten Messungen vor allem bei der Fehlersuche in kleinen lokalen Netzwerken. Dabei werden sog. Sniffer an einen Netzwerkport angeschlossen und der dort anfallende Datenverkehr auf mögliche Probleme untersucht.

Hardwarebasierte Messmethoden sind hingegen speziell für das Monitoring an High-Speed-Links entwickelt. Hierbei wird dem jeweiligen Gerät spezielle Hardware hinzugefügt, welche dann alle Datenpakete an einen aufzeichnenden Host weiterleitet. Diese Systeme benutzen in der Regel sog. optische Splitter um das Signal einer lichtbasierten Übertragung zu duplizieren oder gespiegelte Ports an einem Router und leiten diese Signale dann an eine spezielle Aufzeichnungskarte weiter. Hierbei wird eine 100% Aufzeichnung aller Pakete bis zu einer Geschwindigkeit von 10Gbit/s garantiert.

2.3.4 ONLINE- VS. OFFLINE VERARBEITUNG[2]

Die Überwachung und Analyse von Internetverkehr und Paketinhalten kann verschiedene Absichten verfolgen. Im wissenschaftlichen Bereich ist die Offline-Verarbeitung der gesammelten Daten von größerer Bedeutung, während administrative und sicherheitsrelevante Institutionen eher an einer Analyse des Datenverkehr in Echtzeit, also online, interessiert sind.

Eine Verarbeitung einzelner Datenpakete in Echtzeit wird vor allem in Firewalls vorgenommen um gefährliche Anomalien aufzuspüren oder vordefinierte Verbindungen erst gar nicht zuzulassen. Sog. Paketfilter analysieren die Pakete auf bestimmte Eigenschaften wie Quell- oder Zieladressen und entscheiden anhand vordefinierter Regeln, ob ein Paket die Schnittstelle passieren darf. Über diese statischen Regeln hinaus werden auch dynamische Filtersysteme in Firewalls verwendet, die sog. Stateful-Paket-Inspection (SPI), welche beim Verbindungsaufbau Regeln definieren, welche Pakete zwischen den Kommunikationspartnern ausgetauscht werden dürfen. Durch diese Technik wird das Risiko von Man-in-the-Middle Attacken verringert.

Die Analyse von abgespeicherten Daten im Offline-Betrieb ist weniger zeitkritisch und erlaubt daher eine genauere und tiefere Analyse von aufgezeichnetem Netzwerkverkehr. Außerdem ist es dabei möglich, die Ergebnisse mit früheren Aufzeichnungen zu vergleichen und die Untersuchungen auf verschiedene Anwendungsfälle zu untersuchen.

Wir werden uns bei der Frage, aus welcher Zusammensetzung Internetverkehr besteht, an Ergebnissen orientieren die aus solchen Offline-Analysen von abgespeicherten Webtraces entstanden sind.

2.3.5 STICHPROBEN UND VOLLSTÄNDIGE MESSUNGEN[2]

Wie wir bereits festgestellt haben, fallen bei passiven Messverfahren, besonders wenn diese vollständig und auf Paket- oder Flow-Level durchgeführt werden, schnell sehr große Datenmengen an. Um dieses Problem abzuschwächen werden oftmals nur Stichprobenartige Messungen durchgeführt, die auch als Sampling bezeichnet werden.

Grundsätzlich können dabei 3 verschiedene Techniken unterschieden werden:

- intervall-basiertes systematisches Sampling
  es wird nur jedes x-te Paket bzw. jeder x-te Flow aufgezeichnet
- intervall-basiertes zeitliches Sampling
  es werden in bestimmten zeitlichen Intervallen Pakete/Flows über einen definierten Zeitraum aufgezeichnet. (z.B. alle 3 Stunden über einen Zeitraum von 5 Minuten)
- Sampling über vordefinierte Kriterien
  Pakete oder Flows werden nur aufgezeichnet wenn vorher festgelegte Kriterien erfüllt sind. (z.B. nur Pakete in ein bestimmtes Zielnetz oder Flows welche TCP als Transportprotokoll benutzen)

Darüber hinaus existieren noch feinere Paket Sampling Techniken wie etwa das Adaptive Paket Sampling [11]. Eine gute Übersicht zu verschiedenen Sampling-Techniken im Bereich der passiven Messung von Internetverkehr findet sich bei Duffield [12].

2.3.6 RELEVANTE PROJEKTE ZUR MESSUNG DES INTERNETVERKEHR[2]

Um einen Einblick zu bekommen, wie das Internet funktioniert, ist es unerlässlich den tatsächlich anfallenden Datenverkehr zu beobachten und zu analysieren. Aus diesem Grund haben sich über die Jahre hinweg viele verschiedene Projekte entwickelt und es wurde sich darum bemüht die gesammelten Daten auszutauschen [13].

Nur mit einer genauen Analyse des anfallenden Internetverkehrs lässt sich die Notwendigkeit neuer Transportprotokolle voraussagen. Außerdem ist die Entwicklung neuer Anwendungen, besonders, wenn diese zeitkritische Anforderungen erfüllen muss, abhängig von den Transporteigenschaften des Internet mit den vorhandenen Protokollen.

In diesem Abschnitt wollen wir uns daher einen kurzen Überblick über die wichtigsten Projekte verschaffen, welche sich mit der passiven Messung von Internetverkehr beschäftigt haben.

2.3.7 NLANR PMA[2]

Das von der National Laboratory for Applied Network Research (NLANR) geführte Projekt PMA (Passive Measurement and Analysis) endete offiziell im Jahr 2006. Die Infrastruktur und Messpunkte dieses Projekts haben aber teilweise noch Bestand und werden heute in anderen Projekten weiterverwendet.

Das NLANR sammelte dabei tägliche Messungen auf Internet-Backbones, welche über die USA verteilt lagen. Gemessen wurde dabei auf Verbindungen mit einer Geschwindigkeit von bis zu OC48 (2.5Bbit/s), wozu die Messpunkte entsprechend mit den Systemen OC3MON und OC48MON ausgestattet wurden [14]. Diese Systeme basieren auf den bereits erwähnten DAG-Cards von Endace [15]. Das OC48MON System der NLANR war auch Grundlage der Entwicklung weiterer Monitoring Systeme, unter anderem des IPMON-Systems von Sprint [16].

Die Header-Informationen der so aufgezeichneten Daten wurden öffentlich zugänglich gemacht und führten so zu zahlreichen weiteren Untersuchungen auf der Grundlage des gemessenen Internetverkehrs auf einer Highspeed-Verbindung. Unter anderen wurde von Jiang und Dovrolis auf Grundlage dieser Daten ein Verfahren entwickelt, um aus passiven Messdaten die Round-Trip-Time (RTT) von TCP-Verbindungen zu bestimmen [17].

2.3.8 EU PROJEKTE SCAMPI, LOBSTER, MOMENT[2]

Die Europäische Kommission initiierte 2002 im Rahmen des Information Society Technologies (IST) Programm das Projekt SCAMPI. An diesem Projekt beteiligten sich 10 Partnerorganisationen in ganz Europa, mit dem Ziel eine Monitoring-Plattform aufzubauen, welche den Nutzen einer solchen Überwachung in Bezug auf eine verbesserte Entwicklung und Servicemöglichkeiten aufzeigt.

Das Projekt wurde anschließend unter dem Namen LOBSTER [18] von der IST weitergeführt und die Infrastruktur der Messpunkte sowie die Messungen der Linkgeschwindigkeiten auf bis zu 10Gbit/s erweitert. Im Rahmen dieser Projekte wurde auch MAPI entwickelt, ein Interface zur Entwicklung von Monitoring-Anwendungen.

Neben der Infrastruktur dieses verteilten Monitoring Projekts wurden auch verschiedene Anwendungen entwickelt, darunter das Überwachungs- und Visualisierungstool Stager [19]. Die teilnehmenden Organisationen waren auch aktiv an der Entwicklung des Flow-Standard IPFIX beteiligt [20].

Die Daten von SCAMPI und LOBSTER waren und sind nur von teilnehmenden Organisationen einzusehen, lediglich ein paar Mitschnitte von Attack-Traces sind öffentlich verfügbar.

Nachdem das Projekt LOBSTER am 30. Juni 2007 geschlossen wurde, blieben die eingerichteten Messpunkte aktiv und wurden in das neue MOMENT [21] Projekt übernommen. MOMENT integrierte die passiven Messdaten des LOBSTER-Projekts in die Datenbank der aktiven Messungen aus den Projekten ETOMIC [22] und DIMES [23]. Aktuell sind aus diesem Projekt jedoch keine Daten mehr verfügbar.

2.3.9 WAND NETWORK RESEARCH GROUP[2]

Die WAND Network Research Group stellt mit dem Waikato Internet Traffic Storage (WITS) Archiv eine detaillierte Sammlung von Traces im neuseeländischen Raum zur Verfügung. Neuere Messdaten werden jedoch nur als Statistiken zusammengefasst dargestellt.

Bekannt ist WAND vor allem für die Entwicklung der DAG Karten. Das Unternehmen Endace, welche dieses Equipment heute entwickelt und vermarktet, ist ein Ableger aus diesem ursprünglichen Forschungsprojekt.

Detaillierte Messdaten im WITS sind nur teilweise öffentlich, aber für jeden frei und ohne Registrierung, zugänglich. Leider sind diese Packet-Level-Traces sehr veraltet und nur bis zum Jahr 2008 als Download verfügbar.

2.3.10 CAIDA[2]

Das Projekt CAIDA (Cooperative Association for Internet Data Analysis) wurde 1997 an der University of California gestartet. Mit diesem Projekt ist auch die Entwicklung bekannter Mess- und Analysewerkzeuge verbunden wie etwa NeTraMet oder CoralReef [24].

An CAIDA teilnehmende Forscher haben eine Reihe wichtiger Studien im Zusammenhang mit passiven Messverfahren im Internet veröffentlich. Darunter unter anderem die Identifikation von P2P-Netzwerkverkehr [25] oder detaillierte Analyseverfahren des gesammelten Datenverkehrs [26], [27].

Die Messpunkte von CAIDA befinden sich an Backbones in Chicago und San Jose und zeichnen die Daten an Schnittstellen mit einer Geschwindigkeit von OC192 (10Gbit/s) auf. Zur Desensibilisierung des Datenverkehrs werden nur die Header bis Layer 3 & 4 der Netzwerkschicht aufgezeichnet und die IP Adressen mit dem Crypto-PAn [28] Verfahren verschlüsselt. Außerdem wird die gesamte Payload der Pakete abgeschnitten. Die Daten des CAIDA Projekt sind teilweise öffentlich und ohne Registrierung verfügbar, für detaillierte Trace-Files ist jedoch eine Anmeldung unter Angabe des Verwendungszwecks erforderlich.

2.3.11 PROJEKTÜBERSICHT[2]

Projekt	Datenart	Verfügbarkeit	Aktualität	Adresse
CAIDA (Cooperative Association for Internet Data Analysis)	Paket-Level-Traces & Statistiken	öffentlich (teilweise Registrierung erforderlich)	Januar 2013	[29]
WAND/WITS (Waikato Internet Traffic Storage)	Paket-Level-Traces	öffentlich	März 2008	[30]
MINTS (Minnesota Internet Traffic Studies)	Statistiken	öffentlich	September 2009	[31]
LOBSTER	Publikationen	öffentlich	Juni 2007	[18]
NLANR National Laboratory for Applied Network Research	Paket-Level-Traces	Registrierung erforderlich	2006	[32]

Abbildung 6: Projekte zu Passiven Messungen des Internetverkehrs

2.4 ÖFFENTLICH VERFÜGBARE MESSDATEN[2]

Im vorherigen Abschnitt wurden verschiedene Projekte vorgestellt welche sich mit passiven Messverfahren des Datenverkehrs im Internet beschäftigen. Über diese Projekte hinaus gab und gibt es noch zahlreiche weitere Bemühungen, Messpunkte für die Überwachung des Internetverkehrs einzurichten. Bis heute aber ist ein öffentlicher Zugang zu den daraus gewonnenen Messdaten, besonders zu detaillierten Paket-Traces, sehr eingeschränkt.

Aktuell stellt einzig das CAIDA Projekt anonymisierte aber detaillierte Trace-Daten von OC192 Schnittstellen eines Internet-Backbone zur Verfügung. Für die weiteren Untersuchungen zum Protokoll-Mix standen für diese Arbeit auch Trace-Dateien aus dem Jahr 2012 zur Verfügung.

Die ansonsten verfügbaren Traces sind überwiegend stark veraltet und alleine deshalb für aktuelle Forschungen nicht mehr repräsentativ. Darüber hinaus bestehen aufgrund rechtlicher Bestimmungen für die Aufzeichnung von Internetverkehr erhebliche Einschränkungen bei der Veröffentlichung detaillierter und forschungsrelevanter Trace-Daten.

2.4.1 JURISTISCHE BESCHRÄNKUNGEN[2]

In der Europäischen Union wird in der Richtlinie 95/24/EC zusammen mit der Richtlinie 2002/58/EC der Datenschutz im Telekommunikationswesen vorgegeben.

In Artikel 2a werden dabei persönliche Daten als solche deklariert, welche zur eindeutigen Identifikation einer Person beitragen oder aus denen Rückschlüsse auf deren Identität hergestellt werden können. Neben trivialen personenbezogenen Daten wie Namen, Adressen oder Telefonnummern zählen demnach auch IP Adressen zu diesen Daten. In der Richtlinie 2002/58/EC wird der Datenschutz besonders auf die Gegebenheiten des elektronischen Kommunikationssektors erweitert und neben natürlichen Personen auch auf juristische Personen, also Unternehmen und Organisationen, angewandt. In Artikel 5 ist dabei festgelegt, dass eine Speicherung oder Einsicht in die Kommunikationsdaten abseits der operativen Notwendigkeit (Abrechnungen, Provisionierung etc.) rechtswidrig ist. Ausgenommen davon sind nur Untersuchungen zur Vermeidung, Bekämpfung und Verfolgung von Straftaten die sich gegen die nationale Sicherheit richten. Abseits davon sind die Provider dazu verpflichtet

die gesammelten Daten nach der operativen Verarbeitung zu löschen oder einer Anonymisierung zu unterziehen.

In Folge der Terrorismusbekämpfung wurde 2006 eine Richtlinie zur Vorratsdaten-speicherung innerhalb der EU, die Richtlinie 2006/24/EC, erlassen. Danach sind Provider dazu verpflichtet, personenbezogene Daten mindestens 6 bis 24 Monate zu speichern um die Nachverfolgung von Straftaten zu unterstützen. In Deutschland scheiterte die Umsetzung dieser Richtlinie in nationale Rechtsprechung vor dem Bundesverfassungsgericht auf Grund der in der BRD geltenden Datenschutzbestimmungen.

Für die Analyse und Überwachung des Internetverkehrs innerhalb der EU bedeutet dies, dass Provider und forschende Organisationen Datenverkehr nur mit Zustimmung der Nutzer aufzeichnen und auswerten dürfen. Die Mitgliedstaaten dürfen jedoch Forschungs-organisationen das Speichern von Kommunikationsdaten erlauben, solange dabei datenschutzrechtliche Sicherheitsvorkehrungen eingehalten werden.

In der US-amerikanischen Gesetzgebung wird ein Unterschied zwischen der Echtzeitüberwachung und dem Aufzeichnen von Datenverkehr vorgenommen. Bis zur Verabschiedung des Patriot Act, in Folge der Terroranschläge in 2001, war im Wiretap Act [33] nur die Überwachung von Nutzerdaten untersagt. Erst mit den ergänzenden Regulierungen des Patriot Acts wurde auch die Überwachung von Verbindungsinformationen verboten. Ausgenommen sind dabei der Schutz vor Angriffen auf das Netzwerk und die Auswertung der Verbindungsdaten nach Zustimmung des Nutzers.

Das Aufzeichnen und der Austausch von aufgezeichnetem Internetverkehr ist im Electronic Communications Privacy Act festgelegt. Grundsätzlich ist es Providern darin verboten gespeicherte Verbindungsdaten aufzuzeichnen oder zu verbreiten. Ausgenommen sind davon der Austausch von Verbindungsdaten in nicht öffentlichen Netzwerken und reine Verbindungsinformationen (in diesem Fall Header-Informationen) ohne Nutzerdaten.

2.4.2 ANONYMISIERUNG DER MESSDATEN[2]

Um den im vorherigen Abschnitt beschriebenen rechtlichen Beschränkungen gerecht zu werden, wurden verschiedene Anonymisierungsmethoden entwickelt. Wenn der Datenverkehr

auf Paket-Level aufgezeichnet wird, werden die Pakete in den meisten Fällen nur in einer vorher bestimmten Länge aufgezeichnet. Dadurch wird gewährleistet, dass keine Nutzungsdaten, also Payload, abgespeichert werden. Die gängige Praxis dabei ist, dass Pakete nur bis auf die Länge der Transport-Schicht aufgezeichnet werden und alle weiteren Informationen und Daten abgeschnitten werden. Diese Handhabung garantiert, dass keine sensiblen Nutzdaten gespeichert werden, führt aber gleichzeitig dazu, dass Analysen auf Applikationsebene nicht mehr möglich sind. Um auch Untersuchungen auf Applikationsebene zu ermöglichen, wurde in manchen Studien (Karagiannis 18) auch zum Teil eine bestimmte Länge der Nutzdaten erhalten. Dadurch lässt sich der aufgezeichnete Datenverkehr an Hand von bestimmten Signaturen klassifizieren. Diese Methode ist jedoch höchst umstritten und solche Datensammlungen werden nur in einem sehr kleinen Kreis von vertrauenswürdigen Forschern ausgetauscht.

Eine weitere Maßnahme, um den gesammelten Datenverkehr zu anonymisieren, ist das Verändern der IP-Adressen. Unter der Anonymisierung der IP-Adresse versteht man dabei, dass die tatsächlichen Adressen gegen unechte ausgetauscht werden und das mit einer Methode, die keine Rückführung in die ursprüngliche echte Adresse zulässt. Die einfachste Methode um dies zu erreichen ist das Überschreiben aller IP-Adressen in einem Trace mit einer Konstanten. Dadurch jedoch geht jegliche verwertbare Information aus dem Trace verloren, da der ganze IP-Adressraum auf eine Adresse reduziert wird und man so nicht einmal mehr die Kommunikationsrichtung erkennen kann.

Eine etwas verfeinerte Form dieses Verfahrens verändert alle IP Adressen ab einer bestimmten Länge, was aber auch dazu führt, dass die Prefix Informationen verloren gehen. Um diese relevanten Informationen zu erhalten wurde die Prefix-erhaltende Anonymisierung (prefix-preserving-anonymization) entwickelt. Die erste Implementierung dieses Verfahrens war TCPdpriv, entwickelt von Minshall 1996 [34]. Nachteil dieses Programm ist jedoch, dass der Ansatz daraus besteht Adresspaare aus einer bestehenden Tabelle zu verwenden welche das Prefix mit der größten Übereinstimmung bietet. Dadurch erhält man bei der Übersetzung unterschiedlicher Traces inkonsistente IP-Adressen und eine Auswertung über mehrere Traces ist nicht möglich.

Dieses Problem wurde mit Crypto-PAn [28] behoben, was keine Übersetzung der IP-Adressen durch eine vorgegebene Tabelle vornimmt, sondern die Adressen auf der Basis

einer Verschlüsselung 1-zu-1 transformiert. Dadurch ist es möglich mit dem gleichen Schlüssel auf unterschiedlichen Datensätzen die gleiche Übersetzung der IP Adressen vorzunehmen und so alle relevanten Informationen beizubehalten. Crypto-PAn ist heute der Standard beim Anonymisieren von Trace-Dateien, musste aber aufgrund von verschiedenen Sicherheitslücken mehrmals nachgebessert werden. Zusammen mit anderen Anonymisierungstools findet man eine Crypto-PAn Implementierung in PktAnon.

2.5 ANALYSE DER MESSDATEN[2]

2 Die Analyse der so gesammelten Messdaten ist natürlich abhängig von der Messmethode. Wurde eine Messung auf Paket-Level-Ebene durchgeführt ist der Detailgrad der Untersuchung nur auf die Paketlänge beschränkt. Über die so aufgezeichneten Traces lassen sich mit Analysetools jegliche gewünschte Statistiken erstellen, was z.B. den unterschiedlichen Protokollanteil, Größe der Pakete usw. angeht. Für die Analyse von Trace-Daten, welche von einer Highspeed-Verbindung erstellt wurden, ist jedoch nicht die Verwendung von Wireshark [35] oder vergleichbaren Desktop-Analysetools zu empfehlen. So sind die von CADIA [29] oder WAND [30] bereitgestellten Aufzeichnungen auf den ersten Blick nicht gigantisch groß, jedoch ist die Anzahl der enthaltenen Pakete in diesen Aufzeichnungen um ein vielfaches höher als bei selbst erstellen Paket-Mitschnitten mit Wireshark. Ursächlich hierfür ist die bereits beschriebene Beschränkung der Paketlänge in den öffentlich bereitgestellten Messdaten. Um vertrauliche Informationen aus den Aufzeichnungen zu entfernen, werden in der Praxis Pakete nur bis zu einer Länge von 40 Byte aufgezeichnet. Führt man selbstständig einen Mitschnitt der Pakete durch, erhält man eine Aufzeichnung, welche auch die Nutzdaten der Pakete enthält. Das Resultat ist eine durchschnittliche Paketgröße von etwa 600Byte. Dadurch ergibt sich, dass eine anonymisierte und desensibilisierte Aufzeichnungs-Datei etwa 15mal mehr Pakete enthält als ein vollständiger Mittschnitt des Datenverkehrs.

3 Die durchschnittliche Dateigröße bei den verfügbaren Aufzeichnungen von CAIDA beträgt etwa 1 GByte, was einer Paketanzahlt von ca. 25 Mio entspricht. Beim Öffnen einer solchen Datei mit Wireshark war der Arbeitsspeicher von 16GByte auf einem leistungsstarken Rechensystem bereits nach 20% erschöpft. Das Analysetool NTOP öffnete die Dateien sehr schnell, verarbeitete dabei aber nur einen Bruchteil der enthaltenen Pakete. Von den knapp 25

Millionen Paketen wurden jeweils nur 1-3 Millionen verarbeitet, so dass die daraus resultierenden Ergebnisse nicht verwertbar waren.

4 Da eine 1Gbyte große PCAP-Datei von CAIDA nur den Zeitraum von einer Minute abdeckt, wäre eine Analyse von mindestens 20 aufeinanderfolgenden Aufzeichnungen notwendig, um eine repräsentative Aussage über die Verteilung der Transportprotokolle machen zu können. Aufgrund der Schwierigkeiten und der nicht vorhandenen Rechenkapazität beim Auswerten der Dateien, betrachten wir daher überwiegend die von CAIDA ausgewerteten Ergebnisse. Dass die Auswertung selbst für die dort vorhandene Rechenleistung ein aufwendiges Verfahren ist, zeigt auch die Verzögerung, mit welcher die Ergebnisse und anonymisierten Aufzeichnungen zur Verfügung gestellt werden. Es dauert etwa ein bis zwei Monate vom Zeitpunkt der Messung bis zur Veröffentlichung der gesammelten Daten. Ausgewertete Statistiken über den Anteil von Transportprotokollen, Portnummer etc. sind größtenteils über ein halbes Jahr alt.

2.6 INTERNET- UND TRANSPORTPROTOKOLLE[2]

Wie wir bereits festgestellt haben ist das Internetprotokoll (IP) das grundlegende Element des Internet. In den folgenden Abschnitten werden wir die Eigenschaften dieses Protokoll und besonders das Verhalten beim Routing, auch als Forwarding bezeichnet, näher betrachten. Darüber hinaus werden wir die zwei wichtigsten Transportprotokolle, TCP und UDP, genauer auf ihre Routingeigenschaften untersuchen um so deren Verhalten in Forwarding-Schleifen vorhersagen zu können.

2.6.1 DAS INTERNET PROTOKOLL (IP)[2]

Das Internet-Protokoll (IP) wurde im Jahr 1981 in RFC 791 [36] definiert. Dabei ist der Adressraum auf 32 Bit – Adressen definiert, was in der typischen Schreibweise 4 Blöcken mit je 8 Bit entspricht. Durch diese Festlegung sind maximal 4.294.967.296 eindeutige Adressen verfügbar, wovon einige Adressebereiche auch reserviert sind und nicht als öffentliche IP-Adressen im Internet verwendet werden dürfen [37]. Die Vergabe der IP-Adressen erfolgt dabei durch die IANA bzw. die regionalen RIR [38]. Seit der Einführung des Standard IPv6 wird das ursprüngliche Internetprotokoll als IPv4 bezeichnet. Die Festlegung des Standards

für IPv6 war dabei notwendig um den aufgebrauchten Adressraum von IPv4 an die heutigen Ansprüche aufgrund zunehmender Endgeräte anzupassen. Da jedoch IPv4 nach wie vor das vorherrschende Protokoll im Internet ist werden unsere Untersuchungen sich auch auf das Internetprotokoll in seiner ursprünglichen Version beschränken.

Das Internetprotokoll ist verbindungslos, d.h. Sender und Empfänger bauen keine gesicherte Verbindung auf. Dadurch übernimmt das IP keine Garantie dafür, dass ein gesendetes Paket auch den Empfänger erreicht oder in welcher Reihenfolge die Pakete beim Empfänger ankommen.

Die Adressen von IPv4 teilen sich dabei in einen Netz- und Hostanteil auf. Demnach befinden sich Geräte mit dem gleichen Netzanteil in einem Netzwerk und können direkt über ein Switch o.ä. miteinander kommunizieren. Unterscheiden sich die Netzanteile zweier Teilnehmer müssen die Pakete über einen oder mehrere Router in das entsprechende Zielnetz geleitet werden. Der Router überprüft dabei die im IP-Header eingetragene Zieladresse und entscheidet anhand seiner Routingtabelle, an welches ihm bekannte Netz er das Paket weiterleitet. Neben der Sender- und Zieladresse sind im Header eines IP Paket weitere wichtige Routing-Informationen enthalten.

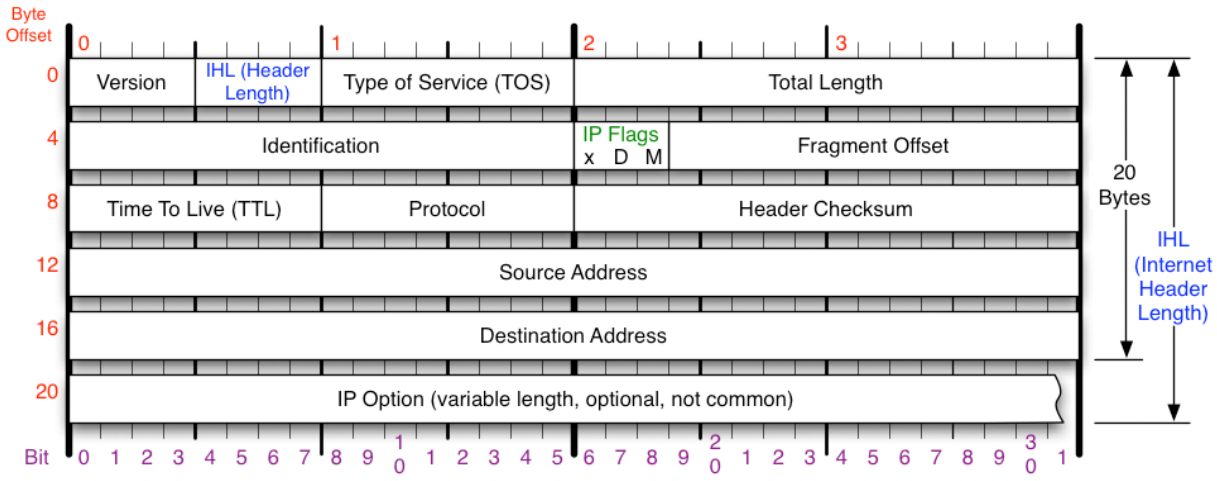


Abbildung 7: Aufbau Internetprotokoll-Header [39]

Standardmäßig ist der Header eines IPv4-Paket 20 Bytes lang, mit dem Wert der IHL (IP Header Length) kann die Länge jedoch bis auf 60Byte erweitert werden.

Die weiteren Angaben im Header sind:

- Version
  definiert die Version des IP-Paket, Optional IPv4 oder IPv6
- Type of Service / TOS
  dient zur Priorisierung von IP-Paketen oder wird zusammen mit TCP für Staukontrollmechanismen wie z.B. ECN genutzt.
- Total Length
  gibt die Gesamtlänge des IP-Pakets an. Aufgrund der Größe von 16Bit für dieses Feld ist eine Maximalgröße des gesamten Pakets von 65535Bytes (64Kbit) möglich. Der Standart sieht dabei vor, dass jedes Geräte Pakete mit mindestens 576 Byte Läge verarbeiten können muss. Im Internet werden jedoch nur Pakete <1500 Byte übertragen, da die MTU von Ethernet Pakete auf diese Länge beschränkt.
- Identification / IP Flags / Fragment Offset
  dienen zur De-/Fragmentierung eines Frames. Das Feld Identification dient dem Empfänger zusammen mit der Source-Address dazu, Fragmente zu erkennen und zu reassemblieren. Flags geben an, ob ein Paket fragmentiert werden darf und ob weitere Fragmente folgen. Das Offset gibt dabei an, ab welcher Länge ein Fragment innerhalb des Pakets beginnt.
- TTL / Time to Live
  gibt die „Lebenszeit“ eines Paketes an und ist sehr entscheidend für unsere Untersuchungen. Im Standard von 1981 wurde festgelegt, dass eine Station im Netzwerk diesen Wert um die Anzahl der Sekunden verringert, die das Paket dort verweilt, mindestens jedoch um 1. Heute wird es jedoch als Zähler der Hops gewertet und die TTL an jedem Router um den Wert 1 verringert. Erreicht die TTL den Wert Null, wird das Paket verworfen und die ICMP-Antwort: *time to live exceeded in transit* an den Absender des Pakets gesendet. Gesetzt wird dieser Wert vom Stack des jeweiligen Betriebssystems. Bei Windowssystemen ist die TTL standardmäßig 128, in Linuxsystemen 64. Der Maximalwert liegt aufgrund der Feldgröße für die TTL bei 255.
  Je größter dieser Wert ist, umso länger könnte also ein Paket in einer Forwarding-Schleife zirkulieren. Wie die Verteilung der TTL durchschnittlich im Internet aussieht, werden wir in den Untersuchungen zum Internetverkehr aufzeigen.
- Protocol

Dieser 8 Bit Wert teilt das Protokoll in den Nutzdaten des IP Paket mit. Die jeweiligen Protokolle werden anhand von Nummern identifiziert und von der IANA in einer Datenbank geführt [40].

- Header Checksum
  definiert die Kontrollnummer über die Daten des IP-Header. Diese muss an jeder Station neu berechnet werden, da sich die TTL wie beschrieben verringert.
- Source Address
  Die Absenderadresse des IP-Pakets in network-byte-order.
- Destination Address
  Die Zieladresse im gleichen Format wie die Absenderadresse. Dieser Wert wird vom Router mit seiner Routingtabelle verglichen und das Paket entsprechend weitergeleitet.
- IP Option
  Hier können Zusatzinformationen festgelegt werden wie z.B. Strict Routing was den kompletten Pfad eines Paket durch das Netzwerk beschreiben würde oder die Route durch das Netz aufzeichnet. Die Maximallänge ist aufgrund der IHL auf 40Byte begrenzt.

2.6.2 TCP EIGENSCHAFTEN[2]

Das Transmission-Control-Protocol baut direkt auf IP auf und ist ein Protokoll der Transportschicht. TCP wurde zusammen mit dem Internetprotokoll von Robert. E. Kahn und Vinton G. Cerf seit 1973 entwickelt und im Jahr 1981 erstmals im RFC 793 standardisiert.

TCP ist verbindungsorientiert und baut zu Beginn eine Ende-zu-Ende Verbindung auf, welche einen bidirektionalen Datenaustausch ermöglicht. Die kommunizierenden Endpunkte sind dabei eindeutig durch die IP-Adresse und einen Port identifizierbar.

Um einen Verbindungsaufbau zu initialisieren sendet der Client ein SYN-Paket mit einer Sequenznummer an den Server. Die Sequenznummer dient dabei zur Sicherstellung der korrekten Übertragung und auch zur Sicherstellung der korrekten Reihenfolge. Es handelt sich dabei um eine beliebige Zahl die aus Sicherheitsgründen jedoch zufällig gewählt werden sollte.

Wird das SYN-Paket vom Server empfangen und akzeptiert dieser den Verbindungsaufbau, bestätigt er den Erhalt dieses Paket mit einem SYN/ACK Paket. Ist der Port an welches das SYN-Paket vom Client gesendet wurde auf dem Server geschlossen, antwortet dieser mit einem TCP-RST und signalisiert so, dass kein Verbindungsaufbau möglich ist. Sendet der Server eine Bestätigung ist das ACK-Flag im Header des TCP-Paket gesetzt und als Acknowledgment-Number enthält dieses SYN/ACK-Paket die Sequenznummer x+1 des SYN-Pakets. Als neue Sequenznummer sendet der Server eine beliebige Nummer y die unabhängig von der Start-Sequenznummer im SYN-Paket ist.

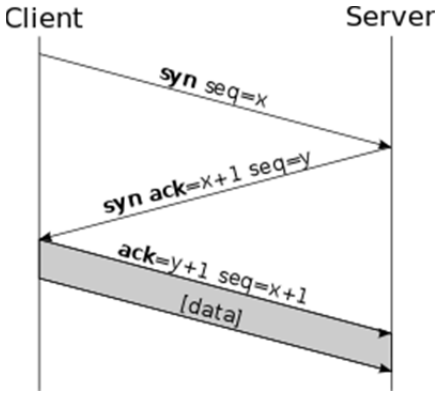


Abbildung 8: Verbindungsaufbau Transmission Control Protocol, Quelle [41]

Erhält der Client das SYN/ACK-Paket bestätigt er dies mit dem Senden eines eigenen ACK-Paket welches die Sequenznummer x+1 erhält. Dieser Ablauf wird auch als „Forward Acknowledgement bezeichnet. Die Acknowledgement-Number beträgt aus Sicherheitsgründen bei diesem Paket y+1, also die Sequenznummer des SYN/ACK-Paket des Servers + 1.

Nachdem die Verbindung so aufgebaut und die Sequenznummern damit synchronisiert wurden, ist es von außen nicht mehr ersichtlich welcher Teilnehmer die Verbindung initiiert hat und beide Teilnehmer sind dadurch gleichberechtigt. Eine einmal aufgebaute Verbindung gewährleistet jedoch nicht den reibungslosen und verlustfreien Datenaustausch.

Um die weiteren Eigenschaften von TCP zu beschreiben sehen wir uns zunächst den Header eines TCP-Pakets an:

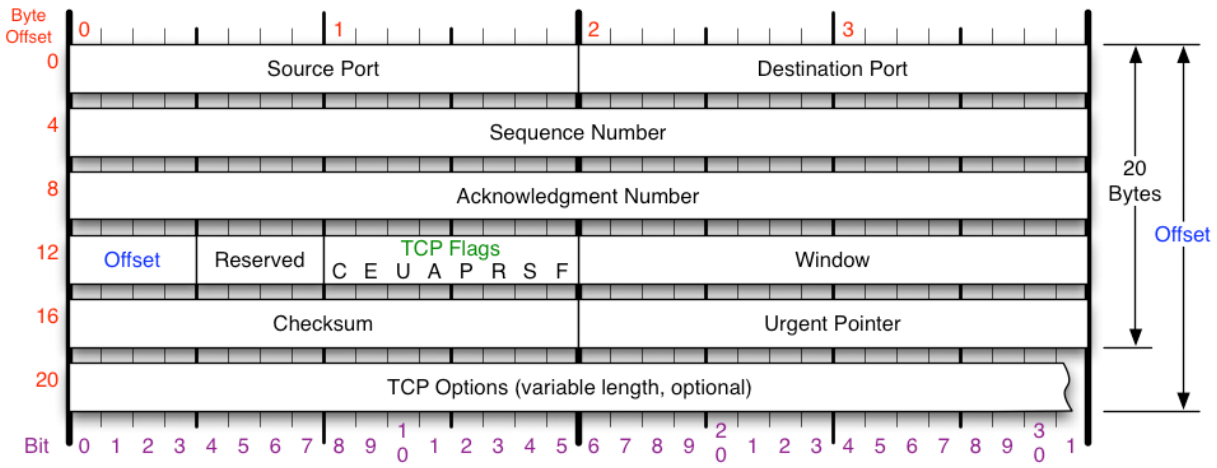


Abbildung 9: Aufbau TCP-Header [39]

- Source Port

  Der Quellport des TCP-Pakets, auf welchem der Sender auch die Antwort erwartet.
- Destination Port

  Der Zielport eines TCP-Pakets. Die Gesamtanzahl möglicher Ports beträgt dabei 65.535 (8 Byte Feldlänge) wovon die ersten 1024 reserviert sind und registrierten Anwendungen zugeordnet sind [40]. Die Zuteilung der Ports ist dabei nicht Anwendungsgebunden, d.h. ein Administrator kann einem bestimmten Dienst auch eine beliebige Portnummer vergeben, nur muss diese dem Client zum Verbindungsaufbau bekannt sein.
- Sequence Number

  Dieser Wert legt die Sequenznummer fest um segmentierte TCP-Pakete zu ordnen. Beim Verbindungsaufbau ist dies eine möglichst zufällige Nummer.
- Acknowledgement Number

  quittiert das letzte erhaltene Paket und enthält den Wert der empfangenen Sequenznummer + 1.
- Data Offset

  Dieser Wert legt die Länge des TCP-Headers in 32 Bit Blöcken fest und zeigt somit die Startadresse der Nutzdaten im Paket an.
- Reserved

  Ein reserviertes Feld dessen Wert Null sein muss. Die Länge des Feldes wurde zum Einführen neuer Flags auch in RFC 3168 von 6 auf 4 Bit reduziert.

- TCP Flags

  Hierbei handelt es sich um sog. Kontrollbits, welche entweder gesetzt oder nicht gesetzt sind. Benötigt werden sie um die Art der Nachricht und deren Weiterverarbeitung zu steuern.

  - CWR

    Wird für die explizite Staukontrolle verwendet (Congestion Window Reduced)
  - ECE

    wird ebenfalls zur expliziten Staukontrolle verwendet (Explizit Congestion Notification)
  - URG

    kennzeichnet, dass das Feld „Urgent Pointer“ gefüllt ist.
  - ACK

    Mit diesem Flag wird zusammen mit der ACK-Number der Empfang eines TCP-Segments bestätigt.
  - PSH

    Ist das PSH Flag gesetzt, wird der Puffer des Empfängers übergangen und die Segmente direkt an die Applikation weitergegeben, ohne dass diese vom TCP Stack zu größeren Einheiten zusammengesetzt werden. Die ist sinnvoll, wenn Befehle möglichst schnell Verarbeitet werden soll wie z.B. einer Telnet-Session.
  - RST

    Dieses Flag wird verwendet, wenn eine Verbindung abgebrochen werden soll (z.B. ein Teilnehmer ist abgestürzt), eine Verbindung abgelehnt wird (Port geschlossen) oder ein ungültiges Segment abgewiesen wird.
  - SYN

    TCP Pakete, in welchem dieses Flag gesetzt ist, initiieren eine Verbindung.
  - FIN

    Dient zur Freigabe einer Verbindung und übermittelt auch eine Sequenznummer, damit der Verbindungsabbau geregelt abläuft.
- Window

  Durch das Empfangsfenster wird dem Sender die maximale Datenmenge mitgeteilt, die er an den Empfänger senden kann, ohne auf eine Bestätigung warten zu müssen.
- Checksum

Dies ist eine Kontrollnummer welche über einen Pseudo-Header gebildet wird und Übertragungsfehler erkennen lässt. Der Header besteht aus der Ziel-IP, der Quell-IP, der TCP-Protokollkennung und der Länge des TCP-Headers inkl. Nutzdaten.

- Urgent Pointer
  Zusammen mit der Sequenz-Nummer gibt dieser Wert die genaue Position des ersten Bytes nach den Urgent-Daten im Datenstrom an. Die Urgent-Daten beginnen sofort nach dem Header. Der Wert ist nur gültig, wenn das URG-Flag gesetzt ist.

Beim Verbindungsaufbau einigen sich beide Kommunikationspartner auf eine MSS im Optionsfeld des TCP-Header. Diese Maximum Segment Size gibt die maximale Länge der Nutzdaten in einem Paket an und ist wie bei IP durch die MTU beschränkt. Bei einer MTU von 1500 Byte über Ethernet muss davon noch der Header von IP und TCP abgezogen werden, was mindestens 40 Byte (20 Byte IP-Header + 20 Byte TCP-Header) sind. Die Nutzdaten pro TCP-Paket würden sich dadurch also auf eine MSS von 1460 Byte beschränken.

Versucht nun ein Kommunikationspartner ein 8 KByte großen Datenblock zu versenden, muss dieser in mehrere Segmente unterteilt werden um eine Übertragung zu ermöglichen. Jedes Segment wird dabei in ein TCP-Paket mit entsprechendem Header aufgeteilt und einzeln an den Empfänger übertragen. Da die Pakete in unterschiedlicher Reihenfolge beim Empfänger eintreffen können, wird die Sequenznummer dazu verwendet um diese wieder korrekt sortieren zu können. Der Empfänger bestätigt dem Sender den Empfang eines Datenpakets mit einem leeren ACK-Paket und entsprechender Sequenznummer. Diese Bestätigung kann dabei kumulativ sein, d.h. es muss nicht jedes Paket einzeln bestätigt werden. Sinnvoll ist dies vor allem, wenn ein ACK-Paket verlorengeht, obwohl der Empfänger es korrekt erhalten hat. Erhält der Sender nun ein ACK-Paket eines später gesendeten Datenpakets, weiß er, dass auch das früher gesendete Paket korrekt beim Empfänger eingegangen ist und muss dieses nicht erneut versenden.

Um die Übertragung effizienter zu gestalten muss ein Sender nicht erst auf die Bestätigung des zuletzt übertragenen Pakets warten um das nächste zu versenden. Über die Window-Size teilt der Empfänger dem Sender mit, wie viele Datenpakete dieser versenden kann, ohne auf eine Bestätigung des Empfängers zu warten. Gehen beim Empfänger mehr Datenpakete ein

als dieser verarbeiten kann, reduziert sich die Window-Size entsprechend und wird erst wieder erhöht, wenn der Empfänger seinen Empfangspuffer abgearbeitet hat. Dieses Verfahren ist auch als Advertised-Window bekannt, da ein virtuelles Fenster über die Sequenznummern gelegt wird, das sich entsprechend der bestätigten Pakete und der mitgeteilten Window-Size verschiebt. Durch dieses Verfahren, was man auch als Flusskontrolle bezeichnet, wird verhindert, dass ein Sender den Empfänger überlastet und ihm mehr Pakete sendet als dieser verarbeiten kann.

Wird dem Sender der Empfang eines Datenpakets vom Empfänger nicht bestätigt, sendet er das nicht quittierte Paket nach dem Ablauf einer vorgegebenen Zeitspanne erneut. Diese Zeitspanne wird auch als Retransmission Timeout RTO bezeichnet und wird während der Übertragung an die gegebenen Laufzeiten im Netzwerk angepasst. Ist der Timeout bis der Sender eine Bestätigung erwartet zu kurz, sendet er das Paket erneut, obwohl das Paket den Empfänger erreicht hat. Wenn der Timeout dagegen zu lang gewählt ist, wartet der Sender unnötig lange bis er das verlorene Paket erneut überträgt. Die Berechnung des RTO erfolgt dabei wie folgt:

Initial wird ein RTO von 3 Sekunden angenommen. Anschließend wird der RTO für jedes Paket aus zwei beim Sender geführten Variablen ermittelt:

- Die geschätzte Round-Trip-Time SRTT (Smoothed RTT)
- Und die Variablität der Round-Trip-Time RTTVAR

Nach dem ersten bestätigten Paket wird aus dessen Umlaufzeit die sog. Round-Trip-Time RTT ermittelt und die RTO wie folgt berechnet:

- SRTT := RTT
- RTTVAR := 0,5 * RTT
- RTO := RTT + 4 * RTTVAR

Für die darauf folgenden Pakete wird die Berechnung der RTO durch geglättete Werte angepasst, d.h. es wird nicht eine neue RTT angenommen sondern aus der vorherigen SRTT und der aktuellen RTT ein angepasster Wert ermittelt. Somit wird verhindert, dass bei starken Schwankungen in der RTT der RTO nicht zu sehr variiert. Gleiches gilt auch für die Berechnung der variablen RTT. Die Berechnung der RTO nach dem zweiten bestätigten Paket sieht damit wie folgt aus:

- SRTT := (1-α) * SRTT + α * RTT' (empfohlen ist α = 1/8)
- RTTVAR := (1-β) * RTTVAR + β * |SRTT – RTT'| (empfohlen ist β = 1/4)
- RTO := SRTT + 4 * RTTVAR

Für die RTO kann auch ein Maximalwert angegeben werden, sofern dieser mindestens 60 Sekunden beträgt. Der Minimalwert der RTO darf jedoch, unabhängig von der obigen Berechnung, niemals weniger als eine Sekunde betragen.

Für das Verhalten von TCP-Paketen in Forwarding-Schleifen ergeben sich aus diesen Eigenschaften schon verschiedene Vorhersagen.

Besteht vor dem Verbindungsaufbau eine Forwarding-Schleife zwischen den Kommunikationspartnern, ist ein Verbindungsaufbau nicht möglich. Ein Client, der sich so versucht mit einem Server zu verbinden, sendet seine SYN-Pakete in die Schleife und enthält entsprechend kein SYN/ACK-Paket um die Verbindung zu initiieren. Wie oft der Client sein SYN-Paket dabei sendet, ist in den meisten Fällen anwendungsabhängig oder wird von der TCP-Implementierung des Betriebssystems verwaltet. Da die SYN-Pakete eine Größe von nur 48 Byte aufweisen ist die Belastung der Router in einer Forwarding schleife relativ gering.

Geraten TCP-Pakete bei einer bestehenden Verbindung in eine Forwarding -Schleife und erreichen somit nicht den Empfänger, wird der Sender nach Ablauf des RTO das Paket erneut versenden. Geschieht dies, wird der RTO für jedes nicht bestätigte Paket verdoppelt und steigt somit exponentiell an. Durch die vorgegebene Window-Size wird der Sender jedoch immer nur maximal so viele Pakete erneut versenden, wie der Empfänger auch bereit ist zu empfangen. Die Anzahl der Sendewiederholungen ist dabei auch von der Anwendung oder dem Betriebssystem abhängig. Da die RTO jedoch mit zunehmender Anzahl verworfener Pakete exponentiell steigt, wird die Belastung der Router in einer Forwarding-Schleife mit zunehmender Zeit immer geringer, da die Pakete in immer größeren Abständen versendet werden.

Eine weitere Eigenschaft des Transmission Control Protocol ist die Staukontrolle. Da beide Kommunikationspartner keine Information über die Ressourcen und Auslastung im Netzwerk haben, wurde mit der Staukontrolle ein Verfahren eingeführt, welches auf eine Überlast des Netzwerk reagieren soll. Um dies zu erreichen wurde das Congestion Window eingeführt (CWIN), welches beim Sender geführt wird. Beim Beginn der Übertragung entspricht das

Congestion Window dabei der MSS, welche der Empfänger mitgeteilt hat. Für jedes bestätigte Paket wird das Congestion Window anschließend verdoppelt. Dieses Verfahren wird auch als Slow-Start bezeichnet und wird fortgesetzt bis ein Schwellwert (Slow-Start-Threshold) erreicht wurde, ab welchem die Größe des Congestion Window für jedes bestätigte Paket nur noch um das einfache der MSS erhöht wird. Diese Phase der linearen Vergrößerung des Congestion Window wird fortgesetzt, bis entweder ein Timeout oder duplizierte ACK-Pakete beim Sender eintreffen. Im bestmöglichen Fall trifft keines dieser beiden Ereignisse ein und das Congestion-Window erreicht die Größe der vorgegebenen Window-Size des Empfängers.

Kommt es zu einem Timeout während einer Übertragung, wird das CWIN auf 1 reduziert und wieder mit dem Slow-Start Algorithmus begonnen. Der neue Start-Slow-Threshold wird nun aber auf die Hälfte des vorherigen CWIN reduziert, was einen langsameren Wachstum der Übertragungsgeschwindigkeit zur Folge hat. Dieses Verfahren wird auch als als TCP-Tahoe bezeichnet. Der Nachteil dieses Verfahrens ist, dass ein Paketverlust nur durch einen Timeout erkannt wird und das Congestion Window auf den initialen Wert gesetzt wird.

Eine Weiterentwicklung dieses Verfahrens ist TCP-Reno, welches sog. duplizierte Acknowledgements (dup-ack) berücksichtigt. Dabei bestätigt der Empfänger immer das zuletzt in richtiger Reihenfolge empfangene Paket. Geht also ein Paket verloren und der Empfänger erhält nachfolgende Pakete korrekt, quittiert er diese mit einem duplizierten ACK für das zuletzt in richtiger Reihenfolge empfangene Paket. Erhält der Sender 3 dieser DUB-ACK's, versendet er das fehlende Paket direkt ohne den RTO hierfür abzuwarten. Dieses Verfahren wird als Fast-Retransmit bezeichnet und bei dessen Eintritt wird das Congestion-Window nur auf die Hälfte des aktuellen Werts zurückgesetzt. Anschließend wird im Congestion-Avoidance Verfahren das CWIN wieder linear erhöht. Zusätzlich können dem CWIN die außer der Reihe bestätigten Pakete noch angerechnet werden. So wird schneller wieder ein größeres Sendefenster erreicht, weshalb diese Methode auch als Fast-Recovery bezeichnet wird.

Für das Verhalten von TCP-Pakten in Forwarding-Schleifen bedeutet dies, dass bei vollständigen Paketverlusten der Sender sein Congestion-Window auf die MSS reduziert und somit nur ein Paket Richtung Empfänger versendet. Da sich dieses Paket wiederum in der

Forwarding-Schleife verliert, wird der RTO verdoppelt und nur ein einziges Paket wiederholt gesendet, bis die Verbindung vollständig durch die Anwendung abgebrochen wird.

2.6.3 UDP EIGENSCHAFTEN[2]

Das User Datagram Protocol (UDP) ist wie IP ein verbindungsloses Protokoll und baut direkt auf dem Internetprotokoll auf. Dadurch ist wie bei IP selbst keine Absicherung der Pakete gegen Verlust, Duplizierung, Veränderung oder einer verfälschten Reihenfolge gegeben.

Das UDP wurde ab 1977 entwickelt und sein Standard ist in RFC 768 [42] definiert. Es entstand aus dem Bedürfnis eines einfachen Protokolls zur Übertragung von Sprache, da hier das bisher verwendete TCP-Protokoll aufgrund seiner verbindungsorientierten Übertragung zu Verzögerungen führte. UDP ist daher optimal für den Austausch von kleinen Datenmengen und Anfragen wo keine gesicherte Verbindung notwendig ist. Die kommunizierenden Prozesse können direkt mit dem Datenaustausch beginnen, ohne dass eine Verbindung ausgehandelt werden muss. Neben Multimediaanwendungen wieVoIP und IP-TV ist DNS ein typischer Frage-Antwort Dienst der auf UDP basiert. Weitere Dienste wie SNMP und Netflow benutzen ebenfalls das UDP als Transportprotokoll zum Exportieren ihrer gesammelten Informationen.

UDP ist ein minimales Protokoll, was IP nur um eine Portnummer erweitert, an der der Prozess identifiziert wird, welcher das Paket empfangen soll. Die Größe des Header ist daher auch nur 8 Byte groß und besteht aus 4 verschiedenen 16 Bit Feldern.

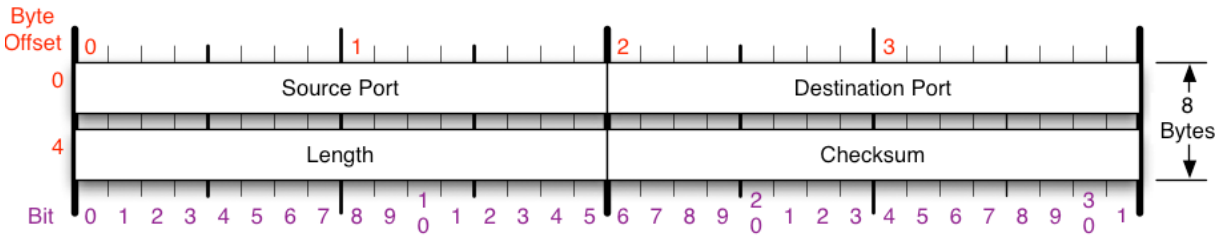


Abbildung 10: Aufbau des UDP-Header [39]

- Source Port
  Der Quellport eines UDP-Pakets. Möglich sind dabei gemäß 16 Bit Feldlänge 65.535 Portnummern. Da UDP verbindungslos ist und meist nur zum Senden von Daten

verwendet wird kann der Quellport auf 0 gesetzt werden, sofern keine Antwort erwartet wird.

- Destination Port
  Dieser Wert gibt den Zielport an, welcher zugleich auch den Prozess identifiziert, welcher das UDP-Paket empfangen soll. Die Portnummern sind dabei von der IANA standardisiert. So sind die Ports von 0-1023 als *well-known-ports* oder *system-ports* festgelegt und ordnen Portnummern feste Protokolle und Anwendungen zu. So ist zum Beispiel Port 80 für http und Port 25 für SMTP reserviert. Darüber gibt es von 1024-49151 die sog. *Userports,* die auf Antrag reserviert werden können. Der Bereich von 49152-65535 ist zur freien Verfügung gedacht.
- Length
  Gibt die Länge des UDP Paket an. Der Maximalwert von 65.535 Bit wird jedoch aufgrund der Längenbeschränkung von IP immer fragmentiert.
- Checksum
  Das Prüfsummenfeld, welches über den sog. Pseudo-Header gebildet wird. Dieser Pseudo Header setzt sich aus Teilen des IP Header, dem UDP Header und Teilen der Nutzdaten zusammen. Die Prüfsumme ist optional und kann auf 0 gesetzt werden, jedoch wird in der Praxis selten darauf verzichtet.

Wie bereits beschrieben wird UDP vor allem dort verwendet, wo ein schneller Datenaustausch Priorität gegenüber einer fehlerfreien Übertragung hat. In Multimedia-Streams wie etwa IP-TV oder VoIP ist eine gesicherte Übertragung nicht sinnvoll, vielmehr ist ein schneller Transport der Daten gewünscht. Auch ist es hier oft nicht erwünscht ein Datenpaket mit wenigen Fehlern zu verwerfen, da wenige Bitfehler von der Anwendung noch korrigiert werden können.

Aufgrund der verbindungslosen Übertragung steht mit UDP keine Stau- oder Flusskontrolle zur Verfügung. UDP-Pakete können auf dem Übertragungsweg verloren gehen, dupliziert oder in falscher Reihenfolge eintreffen, ohne dass der Sender hierüber informiert wird. Ein im Netzwerk verloren gegangenes Paket wird daher auch nicht erneut gesendet.

Ein UDP Paket, welches in eine Forwarding-Schleife gerät, wird dort einfach solange weitergeleitet, bis die TTL des IP-Paket den Wert 0 erreicht hat und wird anschließend verworfen. Der Sender wird in diesem Fall nur mit der üblichen ICMP Nachricht: *time to live*

exceeded benachrichtig, reagiert aber nicht mit einer erneuten Sendung des Pakets. Pakete, welche von einem Sender an einen Port gesendet werden, der nicht empfangsbereit ist, werden ebenfalls mit einer ICMP Nachricht beantwortet. In diesem Fall wird dem Sender mitgeteilt *port unreachable,* jedoch reagiert der Sender nicht auf diese Mitteilung und sendet weiter die Pakete aus.

Unter Linux ist Traceroute auch mit UDP realisiert. Hierbei werden UDP Pakete beginnend von 0 mit einer um 1 erhöhten TTL an garantiert geschlossene Ports versendet und entsprechend auf die ICMP Nachricht gewartet. Der Router, welcher das Paket auf Grund der abgelaufenen TTL verwirft, sendet dem Absender eine entsprechende Nachricht, wodurch er den Pfad schrittweise identifizieren kann.

2.6.4 WEITERE TRANSPORT-PROTOKOLLE[2]

Neben den bisher vorgestellten Transportprotokollen TCP und UDP gibt es schon seit langem Entwicklungen um die Vorteile beider Protokolle zu vereinen und deren negativen Effekte zu eliminieren.

Wie beschrieben ist der Vorteil von TCP, dass es eine gesicherte Verbindung aufbaut und Datenpakete garantiert und in geordneter Reihenfolge überträgt. Bei TCP wird dies durch die beschriebenen Acknowledgements erreicht, welche jedoch einen Overhead an Datenverkehr erzeugen. Zusätzlich ist es für manche Anwendungen irrelevant, dass ein Datenpaket erneut übertragen wird, wenn es im Netz verloren gegangen ist. Bei einem Live-Stream einer Audio- oder Videoübertragung ist es nicht sinnvoll ein verloren gegangenes Paket außer der Reihe erneut zu übermitteln, da dieses Paket von der Anwendung nicht mehr verarbeitet werden kann. Ein weiterer Vorteil von TCP ist die Staukontrolle, welche eine faire Nutzung der Netzinfrastruktur erlaubt und die Überlastung der Router verhindert.

Mit UDP steht daneben ein Transportprotokoll zur Verfügung, welches sich besonders gut für den Austausch von kleinen Datenmengen eignet. Bei einem Dienst wie dem DNS wäre ein vorheriger Verbindungsaufbau mit vollständigem 3-Way-Handshake völlig unnötig. Aber wie bereits erwähnt, verwenden auch viele Multimediaanwendungen zur Datenübertragung das UDP, da hier ein einzelner Paketverlust keine großen Auswirkungen mit sich bringt. Zusätzlich garantiert das UDP minimale Verzögerungen, da einzelne verlorene Pakete den

Datenfluss nicht unterbrechen. Der große Nachteil von UDP ist jedoch, dass Sender und Empfänger keinerlei Informationen über nicht empfangene Pakete austauschen und somit auch keine Mechanismen zum Erkennen von Stausituationen im Netzwerk, bei welchem die Puffer der Router oder der Empfänger selbst überlastet sind.

Für unsere Untersuchungen zum Verhalten von Internetverkehr in Forwarding-Schleifen sind diese unterschiedlichen Eigenschaften der Transportprotokolle ein entscheidender Faktor und daher ist eine Betrachtung der Entwicklungen auf dieser Protokollschicht durchaus interessant.

2.6.4.1 DCCP[2]

Mit dem Datagram Congestion Control Protocol, kurz DCCP, wurde in RFC 4340 [43] im Jahr 2006 von der IETF ein Standard definiert, welcher die beiden positiven Eigenschaften von UDP und TCP zusammenfügen soll.

Auf der einen Seite soll ein schneller Datenaustausch ohne viel Overhead ermöglicht werden und andererseits soll die Staukontrolle von TCP adaptiert werden. Dazu ist bereits beim Verbindungsaufbau ein Datenaustausch möglich, ohne dass dieser erst vollständig abgeschlossen sein muss. DCCP fordert auch nicht die korrekte Einhaltung der Reihenfolge eintreffender Daten. Zu spät eintreffende Pakete werden einfach verworfen. Da DCCP vor allem für die Anwendung bei zeitkritischen Applikationen wie Streaming-Medien, IP-Telefonie oder Onlinespielen gedacht ist, ist dies auch sehr sinnvoll, da hier die Aktualität der Daten gegenüber einer lückenlosen Übertragung Vorrang hat.

Zur Staukontrolle nutzt DCCP die im Internetprotokoll eingeführte Erweiterung ECN [44]. Nachteil dieses Verfahrens ist, dass alle auf dem Übertragungsweg befindlichen Geräte diese Erweiterung unterstützen müssen, da ansonsten das Paket verworfen wird. Als Staukontrollmechanismen sind zurzeit zwei verschiedene Varianten definiert, die sog. Congestion Control Identifier CCID 2 und CCID 3. Erstere Variante eignet sich vor allem für einen hohen Durchsatz, sorgt aber in Stausituationen für einen starken Einbruch der Übertragungsgeschwindigkeiten, ähnlich dem Staukontrollverfahren Reno bei TCP. CCID 3 ist für Anwendungen optimal, wenn sie einen gleichmäßigen Datendurchsatz ohne große

Schwankungen verlangen, sich dafür aber mit einem geringeren Datendurchsatz zufrieden geben.

In Linux ist DCCP seit Oktober 2005 in der Kernelversion 2.6.13 integriert, jedoch noch ohne eine vollständige Unterstützung der angesprochenen Staukontrollverfahren. In aktuellen Untersuchungen zum Internetverkehr wird eine Verwendung von DCCP auch noch nicht registriert [45].

2.6.4.2 GRUNDLAGEN DES DCCP-PROTOKOLLS[1]

Das Datagram Congestion Control Protocol (DCCP, Protokollnummer 33) ist ein Transportprotokoll für den bidirektionalen, unzuverlässigen Transport von Paketen über Unicast-Verbindungen und wird in [43] beschrieben. Im Unterschied zu UDP ist eine Staukontrolle implementiert. DCCP unterstützt [43]:

- Einen unzuverlässigen Transport von Paketen
- Einen zuverlässigen Verbindungsaufbau und -abbau
- Eine zuverlässige „Verabredung“ der Kommunikationsteilnehmer auf Verbindungsoptionen inklusive dem Aushandeln der zu verwendenden Staukontrolle
- Verschiedene Mechanismen erlauben dem Server, einen Wartezustand für unbeantwortete Verbindungsversuche und schon beendete Verbindungen zu vermeiden.
- Die Staukontrolle kann ECN (Explicit Congestion Notification) nutzen.
- Ein Bestätigungsmechanismus kommuniziert Paketverluste und ECN-Informationen. Bestätigungen (ACKS) können zuverlässig übermittelt werden.
- Der Sender kann erfahren, welche Pakete den Empfänger erreichen und ob diese einen Stau erfahren haben (ECN), korrupt sind oder vom Empfangsbuffer verworfen wurden. Damit bietet DCCP eine weitaus feinere Kategorisierung von Paketverlusten als TCP.
- Zwei Staukontrollalgorithmen stehen zur Wahl bereit: CCID2 und CCID3. CCID2 ist eine TCP-ähnliche Staukontrolle, die bei einem Paketverlust oder einem ECN-markierten Paket das Congestion Window halbiert (AIMD). CCID3 ist darauf fokussiert, einen gleichmäßigen Datenstrom zu erzeugen und zu erhalten.

Da TCP eine zuverlässige Zustellung der Pakete in korrekter Reihenfolge sicherstellt, kann dies zu längeren Verzögerungen führen. Applikationen, die zeitsensitive Daten übermitteln z.B. einen Video- oder Audiostream, nutzen daher oftmals UDP. UDP stellt einen schnellen und unzuverlässigen Transport der Daten sicher [43].

Eine typische DCCP-Verbindung durchläuft 3 Phasen: Verbindungsaufbau, Datentransfer und Verbindungsabbau. Daten können in beide Richtungen fließen. Besteht eine Datenverbindung zwischen zwei Knoten (A und B) in beide Richtungen, können Bestätigungen (ACKs) auch von Datenpaketen transportiert werden (DCCP-DataAck Packets). Hierzu sind jedoch zwei Verbindungen nötig, da jede DCCP-Datenverbindung exakt zwischen zwei Knoten in eine Richtung existiert. DCCP verhält sich wie UDP inklusive zusätzlichem Verbindungsaufbau, Bestätigungen (ACKs), Staukontrolle und einem Verbindungsabbau. Um dieses Verhalten zu realisieren, verwendet DCCP 10 Pakettypen [43]:

- **DCCP-Request:** Ein Client initiiert eine Verbindung (erster Teil des Three-Way Handshake). Dieser Pakettyp darf keine Nutzdaten enthalten. Der Server darf auf ein DCCP-Request-Paket mit einem DCCP-Reset antworten, wenn er die Verbindung schließen möchte.
- **DCCP-Response:** Ein DCCP-Server antwortet auf einen DCCP-Request (zweiter Teil des Three-Way Handshake). Ein DCCP-Response-Paket darf Nutzdaten enthalten. Zusätzlich enthält es die Sequenznummer des zuvor empfangenen DCCP-Request-Pakets als Acknowledgement Number. DCCP-Response-Pakete werden nicht erneut übertragen. Geht ein DCCP-Response-Paket verloren, versendet der Client einen neuen DCCP-Request.
- **DCCP-Data:** Der Datentransfer geschieht über DCCP-Datenpakete.
- Ein **DCCP-Ack**-Paket enthält eine Bestätigung für ein empfangenes Datenpaket. Die Sequenznummer des zu bestätigenden Pakets wird als Acknowledgement Number mitgeführt.
- **DCCP-DataAck:** Ein Datenpaket mit zusätzlichen Bestätigungsinformationen (ACK)
- **DCCP-CloseReq:** Der Server fordert den Client auf, die Verbindung zu schließen. Dabei teilt der Server dem Client mit, dass er keinen zusätzlichen TIMEWAIT-Zustand aufrechterhalten möchte.

- **DCCP-Close:** Der Server oder Client schließt die Verbindung und löst damit ein DCCP-Reset als Antwort aus.
- **DCCP-Reset:** Die Verbindung wird regulär oder irregulär terminiert. Dabei beschreibt ein mitgesendeter Reset Code den Grund. Neben einem regulären Verbindungsabbau können Fehler die Gründe für den Verbindungsabbau sein. Es stehen generell 12 Reset Codes zur Verfügung. Weitere werden spezifisch von der verwendeten Staukontrolle benutzt. Unter anderem stehen Reset Codes zur Verfügung, wenn ein ungültiger Pakettyp empfangen wurde und Optionen oder Checksummen fehlerhaft sind.
- **DCCP-Sync, DCCP-SyncAck:** Nach längeren aufeinanderfolgenden Paketverlusten, werden die Sequenznummern mit diesen beiden Pakettypen synchronisiert. Ein DCCP-Sync-Paket löst eine unmittelbare Antwort mit einem DCCP-SyncAck-Paket beim Empfänger aus.

Im Unterschied zu den bytebasierten TCP-Sequenznummern (siehe Kapitel 4.4.1.2), werden DCCP-Sequenznummern pro Paket erhöht. Jedes Paket wird mit einer inkrementierten Sequenznummer versendet, unabhängig ob es Daten enthält oder nicht. Dementsprechend erhöhen auch Ack-Pakete die Sequenznummer, was beide Endpunkte befähigt, einen Paketverlust (Daten und Bestätigungen) zu bemerken. Die Sequenznummern können nach mehreren aufeinanderfolgenden Paketverlusten unter Umständen nicht mehr harmonieren bzw. an beiden Endpunkten nicht synchron sein. Durch ein DCCP-Sync-Paket und ein DCCP-SyncAck-Paket kann die Sequenznummer wieder synchronisiert werden. Da DCCP im Unterschied zu TCP keine Retransmissions unterstützt, wird das Paket mit der höchsten Sequenznummer bestätigt und nicht das verlorengegangene Paket mit der niedrigsten Sequenznummer wie bei den von TCP verwendeten selektiven Bestätigungen (SACKs) (siehe Kapitel 2.6.2). Diese Verluste werden durch andere Optionen (wie Z.B. den Loss Intervals (CCID3) oder dem Ack Vector (CCID2)) mitgeteilt [43].

Die Kommunikationsteilnehmer einer DCCP-Verbindung durchlaufen 9 mögliche Zustände [43]:

- **CLOSED:** Es existieren keine DCCP-Verbindungen.
- **LISTEN:** Der Server lauscht passiv auf mögliche Verbindungsversuche.

- **REQUEST:** Ein Client wechselt von CLOSED in diesen Zustand, nachdem ein DCCP-Request-Paket versendet wurde.
- **RESPOND:** Ein Server wechselt vom LISTEN- in den RESPOND-Zustand, nachdem ein DCCP-Request-Paket empfangen wurde.
- **PARTOPEN:** Der Client wechselt in den PARTOPEN-Zustand, nachdem er ein DCCP-Respond-Paket vom Server erhalten hat. In diesem Zustand darf der Client Daten versenden, muss aber das DCCP-Respond-Paket bestätigen.
- **OPEN:** In diesem Zustand geschieht der eigentliche Datentransfer. Der Client wechselt von PARTOPEN zu OPEN und der Server von RESPOND zu OPEN.
- **CLOSEREQ:** Der Server weist den Client an, die Verbindung zu schließen und in den TIMEWAIT-Zustand zu wechseln.
- **CLOSING:** Server oder Client können in diesen Zustand wechseln, um die Verbindung zu schließen. In diesem Zustand wird ein DCCP-Close-Paket versendet und mit einem DCCP-Reset-Paket beantwortet. Der Empfänger des DCCP-Reset-Pakets wechselt in den Zustand TIMEWAIT.
- **TIMEWAIT:** Server oder Client verbleiben in diesen Zustand für 4 Minuten (2*MSL), um ggf. alte Pakete noch zu empfangen. Die MSL (Maximal Segment Lifetime) liegt wie bei TCP bei 2 Minuten. Sie beschreibt wie lange ein Paket im Netzwerk erhalten bleiben darf. Nur ein Kommunikationsteilnehmer muss nach dem Empfang eines DCCP-Reset im TIMEWAIT-Zustand verharren.

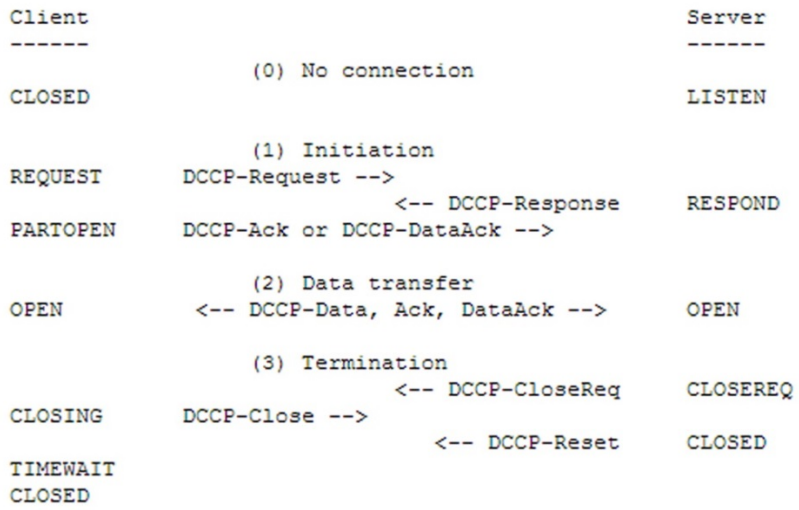


Abbildung 11: Zustände der Kommunikationsteilnehmer einer DCCP-Verbindung

DCCP-Verbindungen besitzen als Besonderheit eine wählbare Staukontrolle. Wie weiter oben beschrieben, steht eine TCP-ähnliche Staukontrolle bereit (CCID2) und eine Staukontrolle, die einen gleichmäßigen Durchsatz sicherstellt (CCID3) [43].

CCID2 reagiert ähnlich wie TCP sehr schnell auf Verzögerungen oder Paketverluste im Netzwerk. Mit ECN markierte Pakete oder Paketverluste signalisieren einen Stau im Netzwerk. Der Sender halbiert daraufhin sein Congestion Window (AIMD). Bestätigungen (ACKs) in CCID2 beinhalten alle Sequenznummern der empfangenen Pakete in einem bestimmten Fenster [43].

Die Staukontrolle von CCID3 basiert auf Gleichungen, die langsamer auf Stau im Netzwerk reagieren. Der Sender erhält vom Empfänger eine Einschätzung der Übertragungsrate und der Paketverluste und passt ggf. die Senderate an. Kurzzeitig reagiert CCID3 damit langsamer als TCP auf Stau und Paketverluste. Langfristig verhält sich CCID3 aber fair zu konkurrierenden TCP-Verbindungen [43].

Im Unterschied zu TCP besitzen DCCP-Verbindungen keine Flusskontrolle und damit kein Advertised Window. Je nach verwendetem Staualgorithmus stehen verschieden Verfahren zur Bestätigung einer erfolgreichen Paketübertragung zur Verfügung. In CCID2 wird ungefähr jedes zweite Paket bestätigt und jede Bestätigung beschreibt exakt, welche Pakete empfangen wurden. In CCID3 wird ungefähr jede RTT eine Bestätigung versendet. Diese muss mindestens die Länge der letzten Intervalle mit Paketverlusten (Loss Intervals) beinhalten [43].

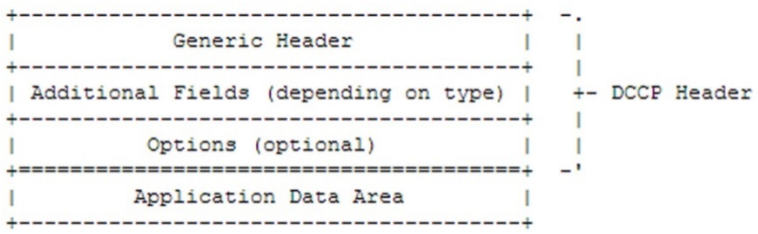


Abbildung 12: Der generische Header von DCCP-Paketen [43]

Der generische Header eines DCCP-Pakets (Abbildung 12) kann zwischen 12 und 1020 Bytes bestehen. Je nach Pakettyp folgen dem generischen Header die Felder des jeweiligen Pakettyps mit einer fixen Länge, gefolgt von den Optionen mit variabler Länge und den

eigentlichen Applikationsdaten, die aber je nach Typ des Pakets auch leer sein können (z.B. DCCP-Ack) [43].

Der generische Header eines DCCP-Pakets kann zwei verschiedene Formen annehmen, je nachdem ob lange (48 Bits) oder kurze Sequenznummern (24 Bits) verwendet werden [43].

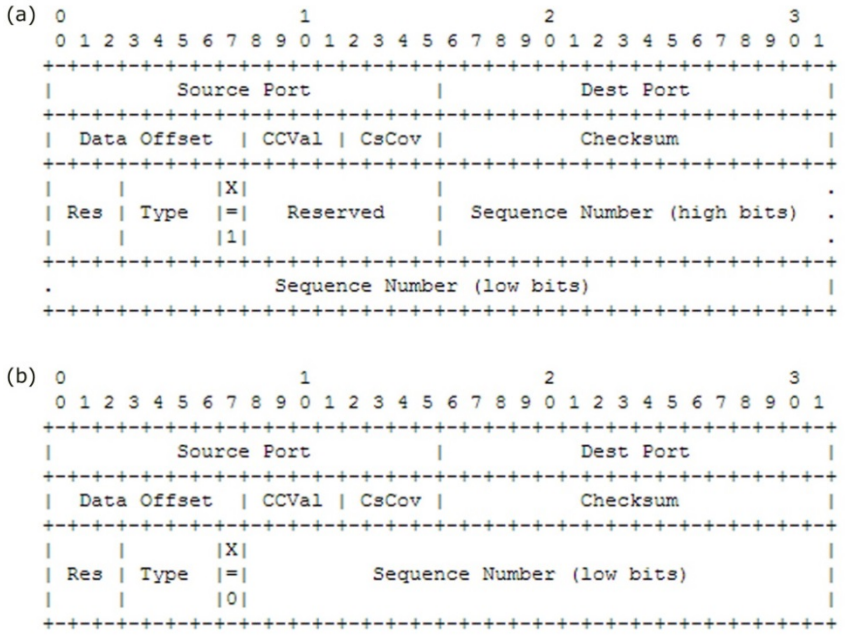


Abbildung 13: Der generische DCCP-Header mit langen (a) und kurzen (b) Sequenznummern [43]

Die Felder beschreiben im Einzelnen [43]:

- **Source Port** (16 Bits) enthält den verwendeten Port des Senders. Dieser sollte aus Sicherheitsgründen zufällig gewählt sein und wird wie bei TCP oder UDP verwendet.
- **Dest Port** (16 Bits) enthält den verwendeten Port des Empfängers.
- **Data Offset** (8 Bits) beschreibt den Beginn der eigentlichen Nutzdaten im Paket.
- **CCVal** (4 Bits) wird je nach verwendeter Staukontrolle vom Sender benutzt und besteht aus 4 Bits.
- **CsCov** (4 Bits) bestimmt, welche Teile des Pakets von der Checksumme abgedeckt werden. Der Header und die Optionen müssen von der Checksumme erfasst werden. Wahlweise können die Applikationsdaten miteinbezogen werden. Zu beachten ist hierbei, dass ein Paket mit inkorrekter Checksumme als Paketverlust gewertet wird und den Staualgorithmus beeinflusst.

- **Checksum** (16 Bits) beinhaltet die errechnete Checksumme über die von CsCov bestimmten Bereiche des Pakets.
- **Reserved** (3 Bits): Dieser Teil wird vom Sender mit Nullen aufgefüllt und der Empfänger muss dieses Feld ignorieren.
- **Extended Sequence Numbers (X=1)** (1 Bit) beschreibt, dass lange Sequenznummern (48 Bits) benutzt werden. Bei sehr schnellen Verbindungen bleiben die Pakete anhand der Sequenznummer eindeutig identifizierbar. DCCP-Data-, DCCP-Ack- und DCCP-DataAck-Pakete dürfen wahlweise auch kurze Sequenznummern verwenden (X=0). Alle anderen Pakettypen müssen lange Sequenznummern verwenden.
- Die **Sequence Number** (24 oder 48 Bits) identifiziert alle vom Sender übertragenen Pakete eindeutig. Auch Pakete ohne eigentliche Daten (z.B. DCCP-Ack) inkrementieren die Sequenznummer.

Üblicherweise handeln beide Teilnehmer beim Verbindungsaufbau die Optionen und Features der Verbindung aus und damit auch die Staukontrolle. Jedoch kann das Aushandeln zusätzlich auch zu jedem Verbindungszeitpunkt geschehen [43].

Type	Option Length	Meaning	DCCP-Data?
0	1	Padding	Y
1	1	Mandatory	N
2	1	Slow Receiver	Y
3-31	1	Reserved	
32	variable	Change L	N
33	variable	Confirm L	N
34	variable	Change R	N
35	variable	Confirm R	N
36	variable	Init Cookie	N
37	3-8	NDP Count	Y
38	variable	Ack Vector [Nonce 0]	N
39	variable	Ack Vector [Nonce 1]	N
40	variable	Data Dropped	N
41	6	Timestamp	Y
42	6/8/10	Timestamp Echo	Y
43	4/6	Elapsed Time	N
44	6	Data Checksum	Y
45-127	variable	Reserved	
128-255	variable	CCID-specific options	-

Abbildung 14: DCCP Optionen [43]

Die Spalte DCCP-Data gibt an, ob DCCP-Datenpakete diese Option beinhalten dürfen. Einzelne Optionen sollen hier näher betrachtet werden [43]:

- **Mandatory** gibt an, dass die unmittelbar folgende Option verpflichtend ist. Verpflichtend bedeutet, dass jede DCCP-Implementierung diese Option verstehen muss. Wird eine verpflichtende Option nicht verstanden, muss mit einem DCCP-Reset-Paket (Code 6: Mandatory Failure) geantwortet werden.
- **Change L, Change R, Commit L, Commit R** wird zum Aushandeln der Features benutzt (siehe folgenden Absatz).
- Die Sequenznummern von DCCP werden bei jedem Paket erhöht. Dadurch können Paketverluste festgestellt werden. Es kann jedoch nicht festgestellt werden, ob diese Pakete Applikationsdaten enthielten. Der **NDP Count** enthält die Anzahl von direkt aufeinanderfolgenden Paketen ohne Applikationsdaten. Der Client kann somit feststellen, ob vorangegangene Paketverluste auch Applikationsdaten enthielten.
- Ein **Ack Vector** übermittelt eine Historie der empfangenen Pakete und deren Zustand (ECN-markiert).
- Die Optionen **Timestamp**, **Timestamp Echo** und **Elapsed Time** werden benutzt, um die RTT einer Verbindung zu messen. Liegt keine Messung einer RTT vor, nimmt DCCP einen Initialwert an. Dieser darf nicht kleiner als 0,2 Sekunden an. Wird ein Paket mit der Option Timestamp empfangen, sollte mit einem Timestamp Echo im nächsten zu sendenden Paket geantwortet werden. Im Fall von CCID3 wird nicht jedes Paket bestätigt. Um hier eine akkurate Messung der RTT zu ermöglichen, wird die Option Elapsed Time verwendet. Sie beschreibt die Zeitspanne zwischen dem Empfang eines Pakets und dem Versand der Bestätigung. Die Option Elapsed Time darf demnach nur bei Acknowledgements verwendet werden.

Insgesamt stehen 4 Optionen für das Aushandeln der Features zur Verfügung: Change L, Confirm L, Change R, Confirm R. Das „L“ wird von dem Endpunkt benutzt, der das Feature besitzt und benutzen möchte (Feature Location). Der Empfänger (Feature Remote) benutzt das „R“. Ein „Change L“ oder „Change R“ initiiert das Aushandeln des Features und wird von einem „Commit R“ oder „Commit L“ bestätigt. Um das Aushandeln der Features sicherzustellen, werden diese ggf. bei einem Paketverlust erneut versendet bis eine Antwort eintrifft. Dabei wird die Sequenznummer bei jeder erneuten Übermittlung inkrementiert (keine Retransmission). Wie bei TCP wird ein Timer benutzt, der sich bei jeder Ausbleibenden Antwort verdoppelt (Exponential Backoff). Der initiale Wert darf eine RTT nicht unterbieten. DCCP-Datenpakete dürfen nicht zum Aushandeln der Features benutzt

werden. Beim Aushandeln der Features muss eine korrekte Paketreihenfolge mit inkrementierter Sequenznummer gewährleistet sein. Pakete in inkorrekter Reihenfolge müssen verworfen werden. Jeder Request (Change L oder Change R) muss mindestens einen Wert für ein Feature beinhalten. Wird ein ungültiges Feature verlangt, antwortet der Empfänger mit einem Commit ohne Wert. Um die Abstimmung in einem Konfliktfall zu gewährleisten, existieren zwei Regeln. Jeweils eine davon ist fest einem Feature zugeordnet und kommt im Konfliktfall zum Einsatz [43]:

1. **Server-Priority** (SP): Jede Change-Anfrage des Clients enthält eine Liste der zur Verfügung stehenden Werte, die nach Präferenz geordnet ist. Die Antwort des Servers (Commit) enthält den ggf. bestätigten Wert und eine nach Präferenz geordnete Liste der Werte des Servers. Um einen Abgleich im Konfliktfall zu gewährleisten, muss der Client den ersten mit seiner Liste übereinstimmenden Wert des Servers übernehmen. Existiert kein gleicher Wert müssen, beide Knoten die Option auf den vorherigen Wert zurücksetzen.
2. **Non-Negotiable** (NN): Der Sender versendet ein „Change L“ und signalisiert damit einen neuen Wert für ein Feature. Der Empfänger muss jeden gültigen Wert akzeptieren und sendet diesen durch ein „Commit R“ zurück. Ein „Change R“ und „Commit L“ ist für NN-Optionen nicht erlaubt.

Folgende Features stehen zur Verfügung. Required beschreibt, dass jede DCCP-Implementierung diese Option verstehen muss. Nicht erforderliche Features müssen eventuell von der gewählten Staukonrtolle (CCID) implementiert sein, z.B. erfordert CCID2 das Feature Send Ack Vector. Pro Feature kommt eine der oben beschriebenen Konfliktregeln zum Einsatz. Unter anderem stehen folgende Features stehen zur Verfügung [43]:

1. **CCID** (SP, Required): Diese Option beschreibt den zu verwendenden Staualgorithmus. Neue Verbindungen verwenden als Standardwert CCID2.
2. **Allow Short Seqnos** (SP, Required): Der Initialwert ist 0. Ein Wert von 1 beschreibt, dass kurze Sequenznummern (24 Bits) verwendet werden dürfen.
3. **Sequence Window** (NN, Required): Jeder DCCP-Knoten führt ein Fenster für gültige Sequenznummern und Bestätigungsnummern (Sequence And Acknowledgement Number Window). Der Initialwert des Fensters beträgt 100. Ein zu kleines Fenster birgt die Gefahr, dass die Synchronisation der Sequenznummern zwischen beiden

Knoten verloren geht oder kann die produktive Kommunikation behindern. Ein zu großes Fenster erhöht die Gefahr eines Angriffs (Connection Hijacking).

4. **ECN Incapable** (SP): Der Initialwert von 0 beschreibt die Verwendung von ECN, um Staus erkennen zu können. Ist ein Endknoten nicht „ECN-fähig“ muss er ein „Mandatory Change L(ECN Incapable, 1)“ senden. Bis das „Confirm R(ECN Incapable, 1)“ eintrifft, dürfen keine Daten versendet werden. Die Verwendung von ECN ist abhängig von der gewählten Staukontrolle.
5. **Ack Ratio** (NN): Durch die Ack Ratio kann ein Sender die Anzahl der zu versendenden Bestätigungen des Empfängers beeinflussen. Die Bestätigungen unterliegen demnach anders als bei TCP ebenfalls einer Staukontrolle. Durch die Ack Ratio wird versucht, eine unter Stau leidende Verbindung (in der Gegenrichtung) nicht komplett zu verstopfen.
6. **Send Ack Vector** (SP): Der Initialwert ist 0, d.h. der Empfänger versendet keinen Ack Vector.
7. **Send NDP Count** (SP): Der Initialwert beträgt 0 (kein Send NDP Count).

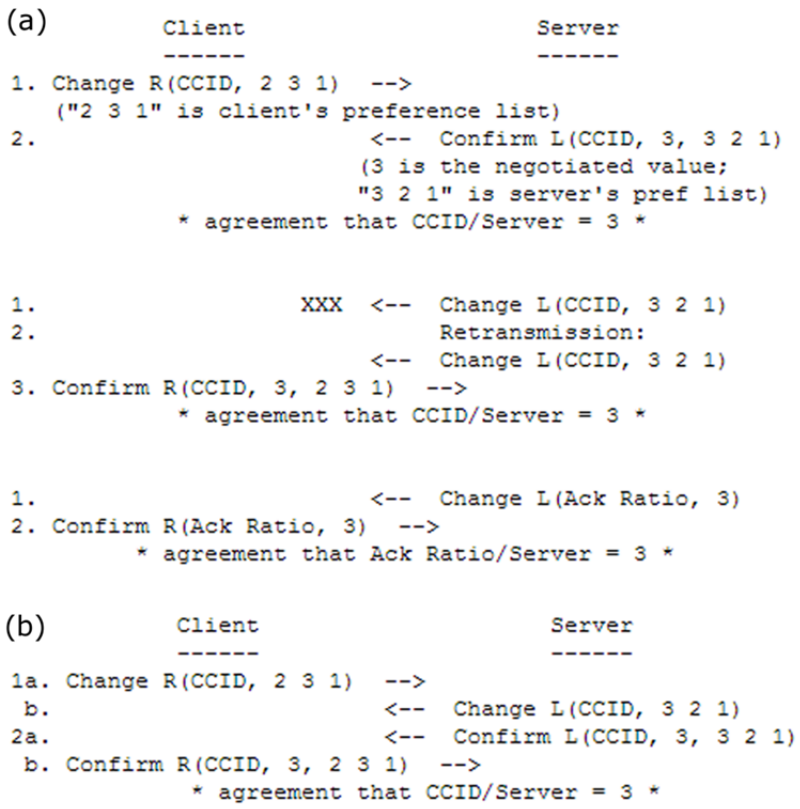


Abbildung 15: Beispielsequenzen für das Aushandeln der Features [43]

In Abbildung 15 werden kurze Beispielsequenzen für das Aushandeln der Features dargestellt [43]: Im ersten Beispiel in Abbildung 15 (a) möchte der Client den verwendeten Staualgorithmus des Servers ändern. Hierzu sendet er seine Liste der unterstützten Algorithmen nach Präferenz geordnet (CCID2, CCID3, CCID1) an den Server (Change R(CCID, 2 3 1)). Die Einigung auf die verwendete Staukontrolle wird nach der Konfliktregel Server-Priority abgearbeitet. Der Server darf seine präferierte Staukontrolle (CCID3) verwenden, wenn Sie vom Client unterstützt wird. Die verwendete Staukontrolle wird zusammen mit einer Liste der unterstützten Staukontrollen des Servers an den Client gesendet (Confirm L(CCID, 3, 3 2 1). Im zweiten Beispiel Abbildung 15 (a) geht die erste Anfrage verloren (Change L(CCID, 3 2 1)). Eine Retransmission kommt schließlich beim Empfänger an und wird bestätigt. Im dritten Beispiel in Abbildung 15 (a) wird das Feature Ack Ratio auf den Wert 3 gesetzt. Im letzten Beispiel in Abbildung 15 (b) gehen beide Change-Requests zeitgleich auf den Weg. Nach der gleichen Regel wie im ersten Beispiel (Server-Priority) wird sich auf einen Wert geeinigt und dieser mit einem Confirm bestätigt.

Bestätigungen können versendet werden (je nach Optionen) sobald ein DCCP-Header von dem Empfänger erfolgreich verarbeitet wurde. Die genaue Verwendung der Bestätigungen hängt von der verwendeten Staukontrolle ab.

2.6.4.2.1 DCCP STAUKONTROLLVERFAHREN CCID2[2]

Wie bereits in Kapitel 2 beschrieben, eignet sich das Staukontrollverfahren nach CCID2 besonders für Anwendungen, welche einen hohen Datendurchsatz erreichen möchten. Dieses Verfahren folgt dabei den Ansätzen aus den von TCP bekannten Verfahren mit selektiven Acknowledgements (SACK). Da DCCP jedoch ein nicht zuverlässiges und nachrichtenorientiertes Protokoll ist, ergeben sich zu TCP zwei wesentliche Unterschiede:

- verlorene Datenpakete werden nicht erneut übertragen. Informationen über den Verlust eines Datenpaket werden lediglich zur Anpassung der weiteren Übertragungsgeschwindigkeit ausgewertet
- die verwendeten Einheiten beziehen sich immer auf ganze Datenpakete, nicht auf Angaben in Byte-Größe

Darüber hinaus stellt CCID2 auch eine Staukontrolle für die vom Empfänger versendeten ACK-Pakete zur Verfügung, in entgegengesetzter Richtung der eigentlichen Datenübertragung. Um dies zu erreichen teilt der Sender dem Empfänger eine ACK-Ratio mit, also für wie viele Datenpakete der Sender ein ACK-Paket erwartet. Da Sender und Empfänger jedoch unterschiedliche Stauverfahren verwenden, kann der Empfänger sein eigenes Verfahren verwenden. Eine hohe ACK-Ratio korrespondiert dementsprechend mit einer niedrigen Rate an ACK-Paketen und indiziert auch gleichzeitig ein größeres Congestion-Window, da der Sender entsprechend viele Datenpakete versendet bevor er ein Acknowledgement erwartet.

Die ACK-Pakete des Empfängers übermitteln einen ACK-Vektor an den Sender, welcher genaue Informationen darüber gibt, welche Pakete korrekt übertragen wurden. Wie auch selektive Acknowledgements sind diese ACK-Pakete nicht kumulativ, das heißt es wird ein ACK für das Paket mit der höchsten zuletzt empfangenen Sequenznummer gesendet. Die ACK-Pakete selbst enthalten auch fortlaufend eindeutige Sequenznummer, woran der Sender verlorene ACK-Pakete identifizieren und so die ACK-Ratio anpassen kann.

Zur Staukontrolle verwaltet ein Sender bei DCCP-CCID2 drei Interger-Variablen:

- Das Congestion-Window (cwnd) gibt die Anzahl der maximal zulässigen Datenpakete im Netzwerk zu einem bestimmten Zeitpunkt an
- Die Pipe (pipe), definiert die Anzahl der im Netzwerk vorhanden Datenpakete, welche noch nicht bestätigt oder als verloren markiert wurden.
- Der Slow Start Threshold (ssthresh) gibt den Schwellenwert für die Steigerung der Übertragungsrate nach dem Slow-Start Verfahren wieder.

Daraus ergibt sich, dass ein Sender immer nur die Differenz aus Congestion-Window und Pipe an Datenpaketen ins Netzwerk senden darf. Für jedes bestätigte oder als verloren markierte Datenpaket wird dementsprechend die Pipe verkleinert und der Sender kann wieder mehr Datenpakete ins Netzwerk senden.

Zur Stauvermeidung arbeitet CCID2 nach dem additive-increase/mulitple-decrease (AIMD) Verfahren. Nach Überschreiten des Slow-Start-Threshold wird demnach pro RTT das Congestion-Window additiv um eine fest definierte Größe erweitert.

$$w(t+1) = \begin{cases} w(t) + a \\ w(t) \times b \end{cases}$$

Dabei ist w die Größe des Congestion-Window (cwnd) zu einem bestimmten Zeitpunkt t. Überschreitet der über den ACK-Vektor mitgeteilte Paketverlust einen definierten Wert, wird das Congestion-Window um einen vorbestimmten Faktor verkleinert (0<b<1). Stellt der Sender keinen Paketverlust fest, wird es um die Anzahl a an Paketen erhöht.

2.6.4.2.2 DCCP STAUKONTROLLVERFAHREN CCID3[1]

CCID3 ist ein Staukontrollverfahren, das den Fokus auf eine konstante Datenübertragungsrate legt. Die Mechanismen, um diesen gleichmäßigen Durchsatz zu erreichen, basieren auf TFRC (TCP-Friendly Rate Control) [46]. TFRC ist eine Staukontrolle, die eine faire Senderate gegenüber anderen TCP-Verbindungen ermittelt und die plötzliche Reaktion auf einen Paketverlust von TCP (Halbieren des Congestion Windows) verlangsamt. Fair bedeutet in diesem Zusammenhang, dass die Senderate sich innerhalb eines Faktors von 2 zu einer TCP-Verbindung unter den gleichen Rahmenbedingungen bewegt. CCID3 ist vor allem für Applikationen interessant, die einen gleichmäßigen Durchsatz benötigen wie z.B. bei Audio- oder Videostreams. Anwendungen mit einem relativ kleinen Empfangspuffer profitieren zusätzlich von einer gleichmäßigen Übertragungsrate. Ein Nachteil einer gleichmäßigen Übertragungsrate ist, dass CCID3 damit auch langsamer auf Änderungen in der zur Verfügung stehenden Bandbreite reagiert. Für Verbindungen mit einem maximalen Durchsatz sollte daher CCID2 verwendet werden. Weiterhin sollte CCID3 nicht für Applikationen verwendet werden, die ihre Senderate durch eine Veränderung der Paketgrößen beeinflussen. Bei CCID3 informiert der Empfänger den Sender über die Rate der Paketverluste (Loss Event Rate) und über die Empfangsrate (Receive Rate). Dieser kann daraufhin die Senderate entsprechend anpassen [43].

$$X = \frac{s}{R*\sqrt{2*b*p/3} + (t\_RTO * (3*\sqrt{3*b*p/8} * p * (1+32*p^2)))}$$

Abbildung 16: Gleichung zur Berechnung der Senderate in CCID3 [46]

Der Kern von TFRC basiert auf der Gleichung zur Berechnung der zulässigen Senderate (Abbildung 16). Diese ist eine leichte Abwandlung der von TCP-Reno verwendeten Gleichung [46].

Die Variablen haben folgende Bedeutung:

- **X** beschreibt die errechnete Senderate in Bytes/Sekunde
- **s** ist die Paketgröße in Bytes. CCID3 kann entweder einen Mittelwert über die letzten Paketgrößen berechnen oder es verwendet die MSS als fixe Größe für s. Im letzten Fall ist die errechnete Senderate X als Pakete pro Sekunde zu interpretieren.
- **R** steht für die RTT in Sekunden.
- **p** ist die Paketverlustrate (Loss Event Rate).
- **t_RTO** ist der Wert des Retransmission Timeouts. Da DCCP keine Retransmissions unterstützt, wird dieser Wert auf 4*RTT gesetzt.
- **b** ist die Anzahl der Pakete die von einem Acknowledgement bestätigt werden. Bei TCP ist dies meistens b=1. Bei CCID3 variiert dieser Wert, da nur ungefähr eine Bestätigung pro RTT gesendet wird.

Die initiale Senderate bei CCID3 liegt zwischen 2 und 4 Paketen innerhalb einer RTT. Je nach Paketgröße darf ein Wert von 4380 Bytes pro RTT jedoch nicht überschritten werden. Ein Sender startet wie bei TCP mit einer Verdoppelung der Senderate pro RTT (Slow Start). Die Slow-Start-Phase endet bei einem Paketverlust (oder ECN-markierten Paket). Der Sender misst die aktuelle RTT einer Verbindung mit den Optionen Elapsed Time oder Timestamp Echo, die vom Empfänger in jedem DCCP-Ack-Paket mitgesendet werden. Jedes DCCP-Ack-Paket muss mindestens gültige Loss Intervals, die Option Elapsed Time und die Option Receive Rate übermitteln, um als gültiges DCCP-Ack-Paket unter CCID3 gewertet zu werden. Die Verwendung von Timestamp Echo ist nicht zwingend vorgeschrieben. Ein Mittelwert der letzten Berechnungen der RTT wird beim Sender vorgehalten und für die Berechnung der Senderate verwendet. Die erlaubte Senderate darf das Zweifache der vom Empfänger übermittelten Empfangsrate nicht übersteigen. Der Empfänger darf weitere Optionen in die Bestätigungen einbinden wie z.B. die Loss Event Rate oder einen Ack Vector. Der Ack Vector kann nützlich für Applikationen sein, die genau feststellen möchten, welche Pakete angekommen sind und welche nicht. Auf zusätzliche Optionen wird sich beim Verbindungsaufbau geeinigt (siehe voriges Kapitel) [47].

Type	Option Length	Meaning	DCCP-Data?
128-191		Reserved	
192	6	Loss Event Rate	N
193	variable	Loss Intervals	N
194	6	Receive Rate	N
195-255		Reserved	

Abbildung 17: DCCP CCID3 Optionen [47]

CCID3 besitzt drei spezifische Optionen. Keine Option darf in einem DCCP-Data-Paket mitgesendet werden [47]:

- **Loss Event Rate**: Die Paketverlustrate wird in CCID3 mit Hilfe der Länge der letzten Intervalle mit Paketverlusten berechnet, die sogenannten Loss Intervals. Die Berechnung liefert einen Wert zwischen 0 und 1.
- **Loss Intervals** in CCID3 bestehen zum einen aus einem Intervall der maximalen Länge von einer RTT mit ECN-markierten oder verlorengegangenen Paketen, gefolgt von einem Intervall beliebiger Länge ohne Paketverluste oder ECN-markierte Pakete. Dementsprechend repräsentieren lange Loss Intervals wenig Paketverluste bzw. keinen Stau. Bis zum ersten Paketverlust oder ECN-markierten Paket der Verbindung existiert nur ein Loss Intervall.
- Die **Receive Rate** beschreibt die Empfangsrate seit der letzten versendeten Bestätigung. Die empfangenen Bytes seit der letzten Bestätigung wird durch max(geschätzte RTT, vergangene Zeit seit der letzten Bestätigung) dividiert.

Send Loss Event Rate (SP, Not Required) ist das einzige CCID3-spezifische Feature, welches von beiden Endknoten ausgehandelt wird. Der Initialwert ist 0 (Loss Event Rate wird nicht genutzt) [47].

Jedes DCCP-Data-Packet enthält neben der Sequenznummer einen sogenannten Window Counter (CCVal). Der Window Counter wird nach jedem Viertel einer RTT um eins inkrementiert. Der Empfänger nutzt CCVal, um die Grenzen der Loss Intervals festzulegen bzw. die Zugehörigkeit von Paketverlusten zu Loss Intervals, da eine verlustfreie Zeitspanne in einem Loss Interval beliebig lang sein kann. CCVal ist 4 Bits groß und kann dementsprechend 4 RTTs darstellen. Werden zwei aufeinanderfolgende Pakete in einem größeren Abstand als 4*RTT versendet, muss ein spezieller Algorithmus das CCVal

berechnen, sonst würde der Empfänger beide Pakete im Fehlerfall evtl. getrennten Loss Intervals zuordnen [47].

Jedes DCCP-Ack-Paket enthält eine Acknowledgement Number, die der größten Sequenznummer eines empfangenen Pakets entspricht. CCID3 versendet ungefähr jede RTT ein DCCP-Ack-Paket. Der Empfänger besitzt einen Feedback Timer. Läuft dieser ab und sind zwischenzeitlich neue Pakete nach der letzten Bestätigung eingetroffen, werden die Werte Loss Event Rate & Receive Rate berechnet und ein DCCP-Ack-Paket versendet. Der Feedback Timer wird daraufhin neu gestartet. Sind keine neuen Pakete seit der letzten Bestätigung eingetroffen, wird nur der Feedback Timer zurückgesetzt. CCID3 benötigt im Gegensatz zu CCID3 keine Bestätigung der DCCP-Ack-Pakete. Eine Ausnahme bildet die Verwendung des Ack Vectors. Wird dieser verwendet, müssen DCCP-Ack-Pakete bestätigt werden, damit der Empfänger den Ack Vector leeren kann. Da im Schnitt ein DCCP-Ack-Paket pro RTT gesendet wird, wird keine Staukontrolle für Bestätigungen im Gegensatz zu CCID2 verwendet.

Um im Falle von ausbleibenden Bestätigungen und der damit fehlenden Kenntnis über Paketverluste und Empfangsrate die Senderate drosseln zu können, wird ein Nofeedback Timer verwendet. Dieser liegt bei max(4*RTT, 2*(Zwischensendezeit)). Die Zwischensendezeit beschreibt die vergangene Zeitspanne zwischen zwei versendeten Paketen. Nach Ablauf des Nofeedback Timers halbiert der Sender seine Senderate und berechnet den Wert des NoFeedback Timers neu. Die erlaubte Senderate darf jedoch nie weniger als 1 Paket alle 64 Sekunden betragen.

2.6.4.3 SCTP[2]

Das Stream Control Transition Protocol, kurz SCTP, wurde von der Arbeitsgruppe Signaling Transport (SIGTRAN) der IETF entwickelt und ist im RFC 4960 definiert.

Wie auch das DCCP versucht SCTP die Eigenschaften von UDP und TCP zu verbinden und einen nachrichtenorientierten, aber zuverlässigen Transportweg mit Staukontrolle zu gewährleisten. Um dies zu erreichen werden die Daten- und Kontrollinformationen in sog. Chunks (dt. Brocken) übertragen. Ein SCTP Paket kann dabei Chunks von mehreren Nachrichten enthalten. Dadurch können verschiedene Inhalte, wie z.B. Bild und

Textinformationen einer Webseite, parallel in einem SCTP Paket übertragen werden. Gleichzeitig können mehrere Verbindungen zwischen den Kommunikationspartnern aufgebaut werden, auf welchen die Datenübertragung abläuft. Dadurch ist das SCTP besonders gut für Multihoming geeignet, da zwischen allen möglichen Endpunkten parallel Verbindungen aufgebaut werden können.

Die Datenübertragung läuft dabei wie bei TCP verbindungsorientiert ab und wird zu Beginn mit einem 4-Way-Handshake aufgebaut. Mit dem 3. Paket können jedoch schon Daten übertragen werden. Anders als bei TCP werden bei SCTP die Daten jedoch nicht byte- sondern nachrichtenorientiert versendet und jedes Paket wird auch nicht sequentiell bestätigt. Jedes SCTP Paket enthält eine TSN (Transmission-Sequence-Number), welche unabhängig der Reihenfolge vom Empfänger quittiert wird. Dadurch wird eine gesicherte Übertragung gewährleistet, auch wenn diese nicht sequentiell gesichert ist.

Die Staukontrolle funktioniert dabei vergleichbar mit den von TCP bekannten Verfahren, wird jedoch hauptsächlich von den selektiven Acknowledgements gesteuert. Ähnlich wie bei TCP werden dabei auch Variablen wie die Reciever-Window-Size (rwnd) und Congestion-Control-Window (cwnd) geführt, welche zum einen die Überlastung des Empfängers als auch des Netzes selbst verhindert. Im Falle eines Paketverlusts reagiert SCTP gleichermaßen wie TCP, was für unsere Schleifenproblematik entscheidend ist. Dadurch wird das Netz, besonders eine Forwarding-Schleife, nicht fortlaufend von weiteren Paketen belastet und der Sender bricht die Verbindung nach einer definierten Anzahl an wiederholten Sendungen, mit minimalem rwnd, (standardmäßig 4) ab.

2.7 VERTEILUNG DER TRANSPORTPROTOKOLLE IM INTERNET[2]

Um einen Überblick über die Verteilung verschiedener Transportprotokolle zu bekommen, müssen wir uns verschiedene Studien ansehen. Vorweg ist zu sagen, dass es keine eindeutige repräsentative Studie über den gesamten Internetverkehr geben kann und man einzelne Messergebnisse unter den gegebenen Bedienungen betrachten muss.

Protocol	Percent
TCP	84.35 %
UDP	13.92 %
ICMP	0.21 %
IGMP	0.01 %
IPv6 in IPv4	0.013 %
Other	1.49 %

Abbildung 18: Protokollverteilung im Internet [45]

In einer Studie aus 2012 [45] wird von einem Verhältnis von 84,35% TCP und 13,92% UDP Verkehr, was den Anteil an Paketen angeht, berichtet. Dabei ist die Größe der transportierten Daten in Byte noch TCP-behafteter. Hier wurde ein Verhältnis von 93,7% TCP- und 6,3% UDP-Datenverkehr gemessen. Diese Ergebnisse basieren jedoch auf einer Messung an einem Zugang zu einem Universitätsnetzwerk, was nicht repräsentativ für den allgemein durchschnittlichen Internetverkehr ist. Bei dieser Studie wurde auch ein Durchschnitt über einen Zeitraum von sieben Tagen erstellt, wobei die Verteilung der Transportprotokolle auch abhängig von der Tageszeit ist.

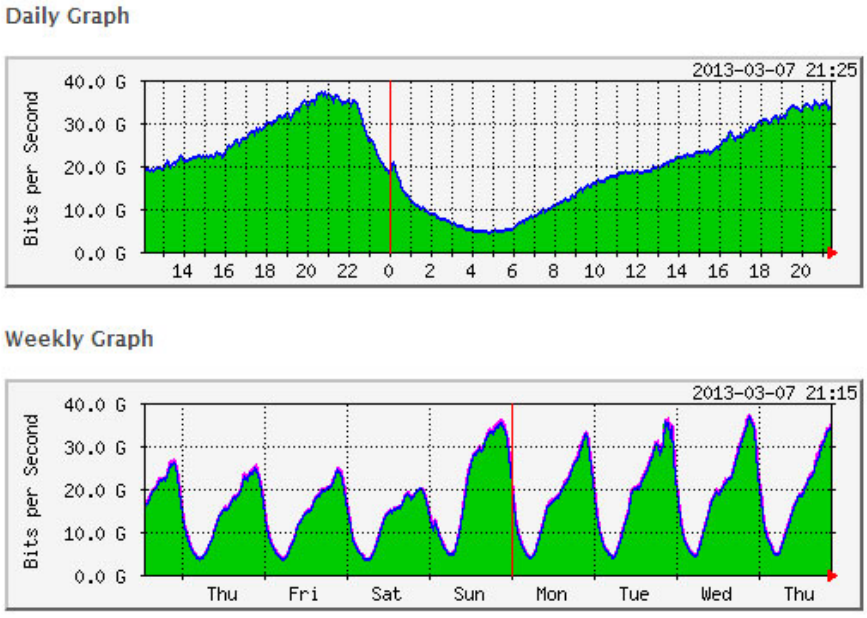


Abbildung 19: Verlauf Internetverkehr im Tages- bzw. Wochenzyklus [48]

Abbildung 19 zeigt den Internetverkehr am BCIX [48] und verdeutlicht die täglichen Schwankungen im anfallenden Internetverkehr. Mit Anlehnung an diese Schwankungen ist auch die Verteilung der Transportprotokolle zu betrachten.

	2AM		10AM		2PM		8PM	
	Pkts	Data	Pkts	Data	Pkts	Data	Pkts	Data
TCP	91.3	97.6	91.5	96.8	93.2	97.1	91.4	97.2
UDP	8.5	2.3	7.6	2.8	6.1	2.7	8.3	2.7
ICMP	0.2	0.02	0.19	0.02	0.20	0.02	0.12	0.01
ESP	0.01	0.00	0.47	0.19	0.35	0.14	0.02	0.02
GRE	0.01	0.01	0.08	0.08	0.04	0.03	0.06	0.04

Abbildung 20: Protokollverteilung im Tagesverlauf [49]

In [49] ist zu erkennen, dass der prozentuale Anteil an UDP-Paketen von 2:00 Uhr nachts zu 14:00 Uhr um 2% schwankt, was für die Verhältnismäßigkeit im Internetverkehr ein recht großer Wert ist. Zu erklären ist dies dadurch, dass zu Geschäftszeiten weniger Streaming-Applikationen und P2P Verkehr zu verzeichnen ist und dieser erst in den Abendstunden eine messbar größere Rolle spielt. Auffällig ist jedoch, dass die Größe der über das jeweilige Protokoll übertragenen Daten nicht so stark schwankt und auch die in [45] ermittelten Daten einen signifikant höheren Wert für UDP-Pakete und nicht für den übertragenen Inhalt aufweist. Die Ursache hierfür ist, dass selbst Applikationen für die UDP das „bessere“ Transportprotokoll wäre, TCP nutzen. Das wiederrum hängt damit zusammen, dass Streaming-Dienste oft in Webseiten integriert sind und diese eine Verbindung über http aufbauen, welches wiederum TCP als Transportprotokoll verwendet.

Vergleicht man aber die Studien [50] aus dem Jahr 2006 und [45] von 2012 ist eindeutig ein gestiegener UDP-Verkehr zu erkennen. Waren es 2006 im Mittel etwa 8,5% aller Datenpakete UDP Verkehr, so sind es in der Veröffentlichung von 2012 schon 13,9%.

Dies wird auch von Zang et. al. [51] in einer Langzeitstudie bestätigt, welche Datensätze im Zeitraum von 2002 – 2009 ausgewertet hat.

Trace	Sample	UDP/TCP Ratio		
		packets	bytes	flows
CAIDA-OC48	08-2002	0.11	0.03	0.11
	01-2003	0.12	0.05	0.27
GigaSUNET	04-2006	0.06	0.02	1.06
	11-2006	0.08	0.03	1.45
CAIDA-OC192	06-2008	0.14	0.05	1.43
	02-2009	0.19	0.07	2.34
OptoSUNET	01-2009	0.21	0.11	3.09
	02-2009	0.20	0.11	2.63

Abbildung 21: Entwicklung der Protokollverteilung 2002-2009 [51]

Hierbei wurden Datensätze des schwedischen Universitätsnetzwerks SUNET [52] aus dem Jahre 2006 und 2009 verglichen. Außerdem wurden repräsentative Datensätze aus den Aufzeichnungen von CAIDA untersucht, einmal auf einem OC48 Peering-Link eines Internet-Service-Provider aus den Jahren 2002 und 2003 sowie Traces eines OC192 Backbone Link in den Jahren 2008 und 2009.

Die Daten aus SUNET wurden dabei auf einem OC192 Link aufgezeichnet und umfassen jeweils einen Zeitraum von 20 Minuten. Die beiden Traces aus April und November umfassen zusammen 9M Flows, 422IP Pakete und 294GB an Daten. In Januar und Februar 2009 kamen am gleichen Messpunkt bei jeweils 2 x 20 Minuten Traces 41M Flows, 1100M IP-Pakete und 657GB zusammen.

Auf dem OC48 Peering Link in den USA wurden im August 2002 und Januar 2003 jeweils 60 Minuten Traces aufgezeichnet welche insgesamt 105M Flows, 1834M Pakete und 1105GB Daten zusammenbrachten. Die Aufzeichnungen am OC192 Backbone-Link liefen ebenfalls über einen Zeitraum von 60 Minuten und bestehen insgesamt aus 379M Flows, 8434M Paketen und 4446GB Daten.

Daraus lässt sich ableiten, dass der UDP Verkehr auf den amerikanischen Links von ca. 11% im August 2002 kontinuierlich auf etwa 19% im Februar 2009 gestiegen ist. Im SUNET war der Anstieg von etwa 6% aller Pakete im April 2006 auf über 20% im Januar 2009 sogar noch weitaus größer.

Sieht man sich nun neuere Statistiken an, bestätigt sich diese Tendenz und der Anteil an UDP Verkehr nimmt weiter zu. Folgende, von CAIDA ausgewertete Aufzeichnung zeigt den Internetverkehr an einem Backbone Link von Equinix in Chicago. Bis Dezember 2010

handelte es sich bei dieser Verbindung um einen OC192 Link welcher inzwischen zu einem 10Gbit/s Ethernet Link ausgebaut wurde. Die in [51] verwendeten Aufzeichnungen wurden dort nicht näher spezifiziert, es ist aber davon auszugehen, dass die Daten von dem gleichen Backbone-Link aufgezeichnet wurden.

July 21 2011 – July 21 2011 PDT ((1 hour)						
Protocol	bits/s		packets/s		tuples/s	
6	1214.108M	(75.03%)	186.621k	(69.53%)	4.837k	(34.17%)
17	377.684M	(23.34%)	77.437k	(28.85%)	9.040k	(63.86%)
47	22.741M	(1.41%)	2.791k	(1.04%)	1.922	(0.014%)
50	3.132M	(0.19%)	960.647	(0.36%)	1.214	(0.0086%)
1	490.650k	(0.030%)	587.227	(0.22%)	275.757	(1.95%)
41	17.394k	(0.0011%)	5.987	(0.0022%)	0.363	(0.0026%)
51	2.898k	(0.00018%)	1.015	(0.00038%)	0.010	(0.000071%)
4	531.301	(0.000033%)	0.326	(0.00012%)	0.003	(0.000024%)
46	0.476	(0.00000003%)	0.513m	(0.00000019%)	0.513m	(0.0000036%)
0	0.410	(0.00000003%)	0.001	(0.00000048%)	0.513m	(0.0000036%)
other	0.238	(0.00000001%)	0.255m	(0.00000010%)	0.257m	(0.0000018%)

Abbildung 22: Protokollverteilung 21. Juli 2011 [29]

Wie auch in [51] wurde ein Zeitfenster von 60 Minuten untersucht. Dabei ist zu erkennen, dass sich der Anteil der Pakete nochmals deutlich erhöht hat und nun bei 28,85% liegt. Deutlicher ist jedoch die Erhöhung des Datenvolumens, welches über das UDP transportiert wird. Aus [51] ist abzuleiten, dass sich der Datenanteil von 2002 von etwa 3% auf etwa 7% erhöht hatte. In Abbildung 22 ist jedoch nun ein Datenanteil von über 23% protokolliert, welcher über das UDP übermittelt wird. Dies würde bedeuten, dass sich das Datenvolumen für UDP Verkehr von 2009 bis 2011 mehr als verdreifacht hätte.

Wie man aber in den nachfolgenden Grafiken erkennen kann, unterliegen diese Verteilungen starken Schwankungen. So findet man die in Abbildung 23 angegebenen Verteilungen nur vereinzelt in einer Aufzeichnung über 24 Stunden auf dem gleichen Monitor wieder.

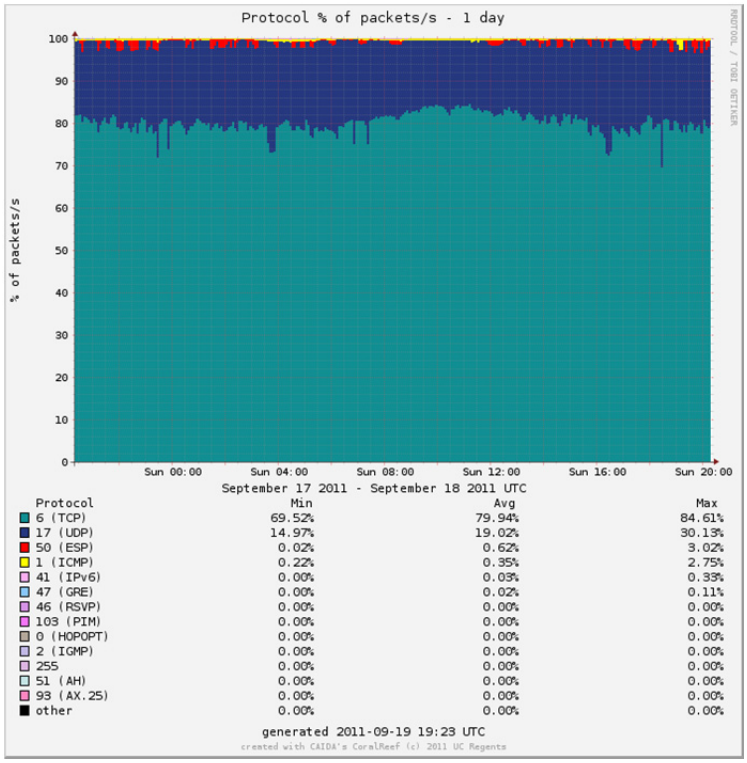


Abbildung 23: Protokollverteilung über 24 Stunden, 17.-18.09.2011 [29]

Ein Anteil von knapp 30% UDP Paketen ist hier nur an wenigen Zeitpunkten zu erkennen. Vielmehr bewegt sich die Verteilung von UDP Paketen um die 20% Marke, was auch so in anderen Studien berichtet wird [53]. Zu erkennen ist dies auch in der zugehörigen statistischen Zusammenfassung der ganztägigen Aufzeichnung.

September 17 2011 – September 18 2011 UTC (1 day)							
Protocol		bits/s		packets/s		tuples/s	
6	(TCP)	550.009M	(83.31%)	115.596k	(79.94%)	4.286k	(56.74%)
17	(UDP)	107.570M	(16.29%)	27.508k	(19.02%)	3.138k	(41.55%)
50	(ESP)	1.860M	(0.28%)	894.700	(0.62%)	0.345	(0.0046%)
1	(ICMP)	402.355k	(0.061%)	512.986	(0.35%)	128.552	(1.70%)
41	(IPv6)	210.697k	(0.032%)	43.401	(0.030%)	0.185	(0.0024%)
47	(GRE)	100.198k	(0.015%)	35.776	(0.025%)	0.026	(0.00034%)
46	(RSVP)	7.161k	(0.0011%)	4.193	(0.0029%)	0.253	(0.0034%)
103	(PIM)	12.013	(0.0000018%)	0.028	(0.000019%)	0.004	(0.000057%)
0	(HOPOPT)	3.012	(0.00000046%)	0.009	(0.0000065%)	0.004	(0.000053%)
2	(IGMP)	0.892	(0.00000014%)	0.003	(0.0000024%)	0.002	(0.000021%)
255		0.000	(0%)	0.000	(0%)	0.000	(0%)
51	(AH)	0.000	(0%)	0.000	(0%)	0.000	(0%)
93	(AX.25)	0.000	(0%)	0.000	(0%)	0.000	(0%)
other		4.634	(0.00000070%)	0.014	(0.0000098%)	0.000	(0%)

Abbildung 24: Protokollverteilung 18. September 2011 [29]

Man kann also davon ausgehen, dass sich der prozentuale Anteil an Paketen, welche mit UDP übertragen werden, nicht weiter drastisch erhöht hat. Deutlich zu erkennen ist aber das gestiegene Datenvolumen über das UDP. Auf einer Zusammenfassung der Messdaten über einen Zeitraum von 2 Jahren auf dem obigen Monitor ist dies in Abbildung 25 besonders gut zu erkennen.

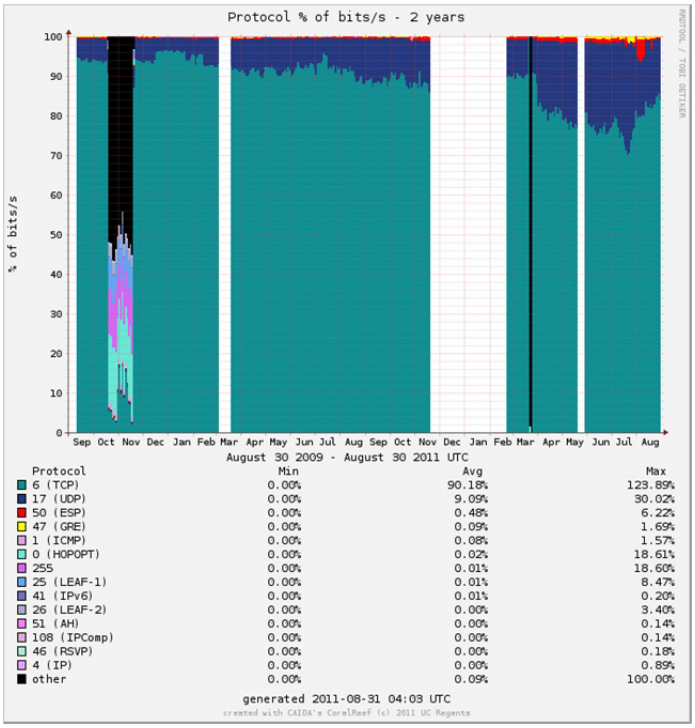


Abbildung 25: Entwicklung UDP Verkehr über 2 Jahre [29]

Zu erklären ist diese Entwicklung mit dem Aufschwung der P2P Filesharing Netzwerke wie etwa eDonkey, BitTorrent, Gnutella und WinMX. So waren in den Untersuchungen von 2002-2003 noch etwa 40% der verwendeten Ports in UDP Paketen <1024, also im Bereich der well-known-Ports. Diese Pakete lassen sich also Diensten wie DNS (Port 53), NTP (Port 123) und NetBios (Port 137) zuordnen. Untersuchungen nach 2003, als die oben genannten P2P Dienste vermehrt genutzt wurden, zeigen, dass nun über 95% der genutzten Ports beim Datenaustausch per UDP über 1024 liegen. Die angesprochenen Dienste nutzen dabei jedoch das UDP nicht als Transportprotokoll für den Austausch großer Datenmengen, sondern für das Signalisieren vorhandener Daten im Sharing-Netzwerk, was auch die geringe Größe der einzelnen Flows bestätigt.

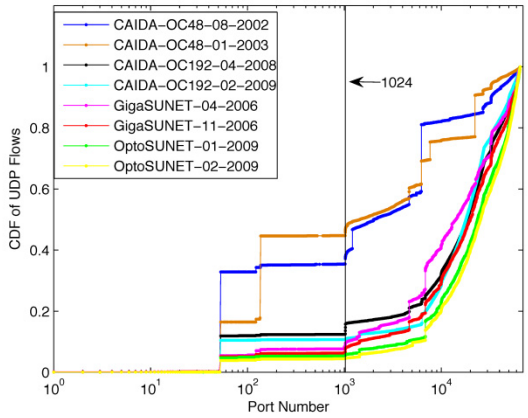


Abbildung 26: Entwicklung UDP Portnummern [29]

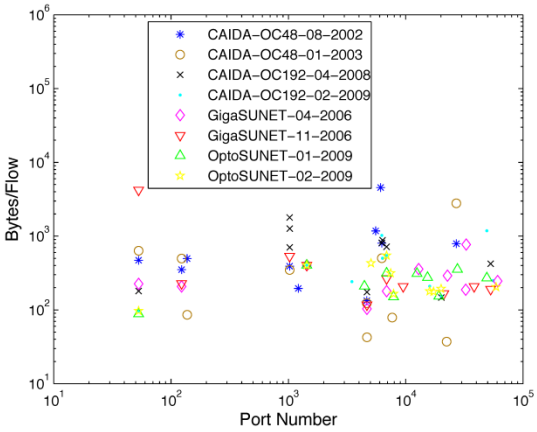


Abbildung 27: Bytes/Flow der UDP-Portnummer [29]

Die Entwicklung des steigenden Datenvolumens über UDP lässt sich dadurch erklären, dass in letzter Zeit die Verbreitung von Audio- und Videostreaming Diensten gestiegen ist. Die Kommunikationssoftware Skype [54] zählt inzwischen über 700 Millionen registrierte Nutzer und wird mit der Übernahme von Microsoft eine noch größere Verbreitung finden. Der Datenaustausch erfolgt dabei über UDP. Aber auch andere neue Streaming-Applikationen wie etwa IP-TV nutzen das UDP als Transportprotokoll. Besonders einige P2P Video-Streaming Programme wie etwa Sopcast [55] nutzen auch UDP zur Übermittlung der Daten. Im Gegensatz zu Skype werden hierbei jedoch sehr große Pakete nahe der MTU versendet. Skype hingegen nutzt zur Datenübertragung recht kleine Pakete mit nur etwa 80KByte Gesamtgröße. Der Grund dafür könnte sein, dass kleinere Paketgrößen bei UDP eine bessere Performance bieten, da im Zusammenspiel mit TCP Daten weniger häufig verworfen werden

[56]. Außerdem kommuniziert Skype auch über sehr hohe Portnummern, üblicherweise etwa <12000.

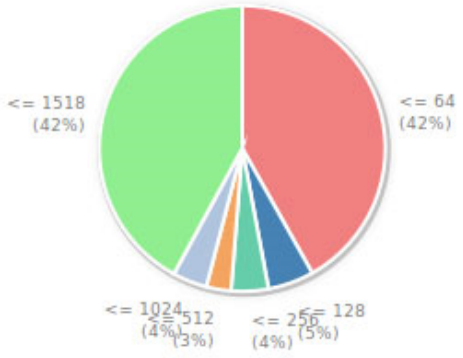
Zusammenfassend aus [45], [49], [50], [51] und den Statistiken von 2011 ist festzuhalten, dass TCP nach wie vor das dominierende Transportprotokoll ist. Die steigende Anzahl an UDP-Paketen und auch der zuletzt größere Anstieg des Datenvolumens über UDP sind auf die gestiegene Nutzung von Audio- und Video-Streaming Diensten zurückzuführen. Bildet man aus den Ergebnissen einen Mittelwert so liegt die durchschnittliche Verteilung bei etwa 16-19% UDP- und 80-83% TCP-Paketen. Beim Datenvolumen ist TCP mit etwa 84-89% gegenüber 10-15% Datenvolumen UDP weiter vorne.

Eine prozentual höhere Anzahl und mehr Datenvolumen an UDP Verkehr wären für die Infrastruktur des Internet auch nicht sehr positiv. Da UDP keinerlei Staukontrolle bietet, ergibt sich im Zusammenspiel von UDP und TCP ein unschöner Nebeneffekt. Erhöht sich die Belastung eines Routers und steigt die Warteschlange in dessen Puffer an, steigt auch die Wahrscheinlichkeit, dass mit dem Random Early Detection (RED) Algorithmus Datenpakete verworfen werden. Die TCP-Verbindung wird in diesem Fall ihre Übertragungsrate verringern, während die UDP-Verbindung unvermindert den Router belastet [57]. Aufgrund dieser unfairen Nutzung der Ressourcen im Netz müsste RED dahingehend erweitert werden, dass UDP Pakete mit einer höheren Wahrscheinlichkeit als TCP-Pakete verworfen werden.

2.8 DURCHSCHNITTLICHE VERTEILUNG DER PAKETGRÖßEN[2]

In früheren Untersuchungen wurde die Verteilung der Paketgrößen im Internet als trimodal beschrieben. Dabei wurde von einem großen Anteil an Paketen mit einer geringeren Größe als 100 Byte berichtet, welche sich vor allem durch TCP-Acknowledgements erklären lassen. Weiterhin wurden dort viele Pakete mit einer Größe von exakt 576 Byte registriert, was auf die standardmäßige Festlegung der Maximum Segment Size (MSS) in RFC 879 [58] zurückführt. Der dritte große Anteil bestand aus Paketen, welche die volle MTU ausschöpfen, also eine Größe zwischen 1400 – 1500 Byte besitzen.

Shortest		46 bytes
Average Size		587 bytes
Longest		1,504 bytes
Size <= 64 bytes	41.9%	1,351,346
64 < Size <= 128 bytes	5.2%	168,721
128 < Size <= 256 bytes	4.2%	134,471
256 < Size <= 512 bytes	2.8%	88,963
512 < Size <= 1024 bytes	3.9%	126,639
1024 < Size <= 1518 bytes	42.0%	1,353,833
Size > 1518 bytes	0.0%	0



Packets too long [> 65355]	0.0%	0

Abbildung 28: Auswertung Trace-Datei nach Paketgrößen

Spätere Untersuchungen zeigen eine Verschiebung hin zu einer bimodalen Verteilung. Darin findet sich ein großer Anteil an sehr kleinen Paketen mit weniger als 100 Byte Größe sowie ein zweites großes Vorkommen an Paketen mit einer Größe zwischen 1400 – 1500 Byte [49]. Der große Anteil an Paketen mit einer Größe von 576 Byte ist dagegen nahezu verschwunden, was auch darauf zurückzuführen ist das in 93,7% aller TCP Pakete der Wert der MSS auf eine Größe von 1400-1460Byte gesetzt wurde, was wiederum der MTU von Ethernet abzüglich der Headergröße von IP und TCP entspricht.

Bestätigt werden diese Untersuchungen auch von aktuellen Studien [45] und der Untersuchung einer Trace Datei von CAIDA in Abbildung 28. Dabei wurde in [45] eine durchschnittliche Größe für TCP Pakete von 736 Byte ermittelt und insgesamt ist die von uns ermittelte Durchschnittsgröße etwa 587 Bytes. Auch wenn solche Durchschnittswerte existieren, sollten Forschungen diese nicht als Maßstab in Ihren Untersuchungen nehmen. Dieser Durchschnitt ist nur das Produkt der beiden extremen Verteilungen von sehr kleinen und sehr großen Paketen nahe der MTU von Ethernet.

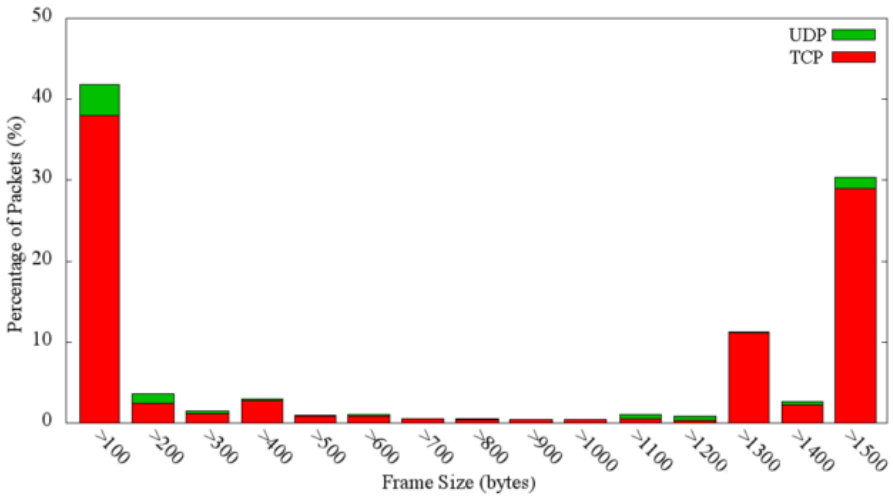


Abbildung 29: Verteilung der Paketgrößen [45]

In Abbildung 29 ist auch zu erkennen, dass diese Verteilung sowohl für TCP als auch für UDP Pakete zutrifft. Der erkennbar prozentual höhere Anteil an UDP-Paketen für sehr kleine Datenpakete ist vor allem durch DNS Anfragen zu erklären. Wie wir aber auch schon in Abschnitt 2.8. erwähnt haben, nutzen viele auf UDP basierte Dienste sehr kleine Pakete um einen effektiveren Datendurchsatz im Zusammenspiel mit TCP zu erreichen [56].

2.9 DURCHSCHNITTLICHE TTL[2]

Wie wir bereits in Abschnitt 2.7.1 beim Betrachten des Internetprotokolls festgestellt haben, ist die „Lebensdauer“ eines IP Pakets ein entscheidender Faktor für das Verhalten von Internetverkehr innerhalb einer Forwarding-Schleife. Mit der Time-To-Live wird dem Paket eine maximale Lebenszeit eingeräumt, welche angibt, wie viele Stationen es auf seinem Weg zum Zielpunkt passieren darf, bevor ein Router es verwirft. Ein IP-Paket zirkuliert also solange in einer Forwarding-Schleife, bis die TTL den Wert Null erreicht, es sei denn, der Router verwirft das Paket bereits vorher wegen Überlastung.

Mit welcher TTL ein Paket abgesendet wird ist abhängig von der Implementierung des IP-Stack auf dem jeweiligen Betriebssystem. In der untenstehenden Tabelle sind die standardmäßigen Werte der TTL der einzelnen Betriebssysteme zusammengefasst. Die Angabe „sicher“ bezieht sich dabei auf die Wahrscheinlichkeit, dass ein Paket einen regulären Weg durch das Internet nimmt und trotzdem verworfen wird. Bei einer TTL <=32 kann es durchaus vorkommen, dass ein Paket verworfen wird, obwohl es sich nicht in einer Forwarding-Schleife befindet. Wie man jedoch sieht, ist vor allem für das UDP oft eine geringere TTL angegeben, was darauf zurückzuführen ist, dass UDP-Pakete für einen schnellen Datenaustausch verwendet werden. Wird dabei ein zu langer Pfad durch das Internet gewählt ist die Wahrscheinlichkeit, dass dieses Paket zu spät beim Empfänger ankommt sehr groß und kann bereits vorher verworfen werden.

Betriebssystem	"sicher"	tcp_ttl	udp_ttl
AIX	n	60	30
DEC Pathworks V5	n	30	30
FreeBSD 2.1R	j	64	64
HP/UX 10.01	j	64	64
Irix 6.x	j	60	60
Linux	j	64	64
MacOS/MacTCP 2.0.x	j	60	60
OS/2 TCP/IP 3.0	j	64	64
OSF/1 V3.2A	n	60	30
Solaris 2.x	j	255	255
SunOS 4.1.3/4.1.4	j	60	60
Ultrix V4.1/V4.2A	n	60	30
VMS/TCPware	j	60	64
VMS/Wollongong 1.1.1.1	n	128	30
VMS/UCX (latest rel.)	j	128	128
MS WfW	n	32	32
MS Windows NT 3.51	n	32	32
MS Windows NT 4.0	j	128	128
CISCO IOS 12.4+	j	255	255

Abbildung 30: Standart-TTL unterschiedlicher Betriebssysteme

Je größer die TTL ist, mit welcher ein IP-Paket in eine Forwarding-Schleife gerät, umso größer ist dementsprechende die Belastung der Router durch dieses Paket.

3 SCHLEIFEN (FORWARDING LOOPS)[1]

Schleifen (Forwarding Loops) führen dazu, dass Pakete einen Link mehrere Male durchlaufen. Dies kann bei Schleifen, die länger andauern, bis hin zum Paketverlust führen. Eine mathematische Definition liefert [59]: Im Allgemeinen durchläuft ein Paket *s* eine Sequenz von Routern, um zum Ziel *d* zu gelangen. Ein Paket durchläuft eine Schleife (Forwarding Loop), wenn es eine Menge an Routern mehrmals durchläuft. Eine Sequenz von Router Interfaces $(r_1, r_2,...,r_n)$, die ein Paket von *s* nach *d* durchläuft, wird als Forwarding Path bezeichnet. Wenn $r_i=r_j$ und $r_i<>r_j$ enthält der Pfad eine Schleife. Die Länge der Schleife wird bestimmt durch *j-i*.

Im Allgemeinen werden Schleifen anhand ihrer Lebensdauer in zwei Arten gegliedert [59][60][61][62][63][64].

Schleifen, die temporär in der Konvergenzzeit der Routing Protokolle durch z.B. einen Linkausfall entstehen, werden als temporäre Schleifen (Transient Loops) bezeichnet. Aufgrund der Dynamik der Routing-Protokolle werden temporäre Schleifen als unausweichlich angesehen. Erweiterungen für IGP-Protokolle [65][60][66] oder das Feature Equal-Cost Multi-Path in modernen Link-State-Protokollen lösen bzw. minimieren das temporäre Schleifenproblem. Bei ECMP (Equal-Cost Multi-Path) werden alternative Next Hops für ein identisches Ziel propagiert. Neben einer Lastverteilung (Load Balancing) bietet der alternative Next Hop im Fehlerfall ein Backup. Für BGP und RIP gibt es keine Erweiterung, um temporäre Schleifen zu verhindern oder die Ausfallzeit zu verringern. [67] belegt, dass die Konvergenzzeit von BGP und damit auch die Dauer von temporären Schleifen unterschätzt wird. Die Zahl der Router Zustände und der auszutauschenden Kontrollnachrichten im Fehlerfall ist faktoriell verbunden mit der Zahl der AS [67]. Eine längere Konvergenzzeit hat maßgeblichen Einfluss auf die Dauer von temporären Schleifen. Equal-Cost Multi-Path in BGP ist ein nicht standardisiertes Feature. Manche Hersteller bieten Multipath in ihren Produkten an. In der Forschung existieren Ansätze, um eine Schleifenfreiheit in BGP unter Verwendung von ECMP zu garantieren [68].

Durch z.B. Konfigurationsfehler entstandene Schleifen, werden als persistente Schleifen (Persistent Loops) eingegliedert und benötigen einen menschlichen Eingriff. Wurden

persistente Schleifen in der Vergangenheit oft unterschätzt und als wenig verbreitet angesehen, belegen neuere Analysen das Gegenteil und gehen von einer nicht zu unterschätzenden und wachsenden Verbreitung aus [59].

Unabhängig von der Lebensdauer existiert eine weitere Klassifizierung in Routing-Schleifen (Routing Loops) und Forwarding-Schleifen (Forwarding Loops). Als Routing wird der Informationsaustausch der Router über die Netztopologie durch ein Protokoll bezeichnet. Die eigentliche Weiterleitung der Pakete wird als Forwarding bezeichnet. Dementsprechend beeinträchtigt eine Forwarding-Schleife ein Datenpaket und eine Routing-Schleife ein Routing-Paket oder den Zustand des Routers. In diesem Kapitel werden nur Forwarding Loops betrachtet.

3.1 GRUNDLAGEN – ROUTING-PROTOKOLLE[1]

Routing-Protokolle dienen dem Austausch von Informationen der Netztopologie und dem Aufbau der Routing-Tabellen. Durch diesen Austausch an Informationen kann jeder Router entscheiden, über welchen Link er ein an ein bestimmtes Ziel adressiertes eingehendes Paket senden muss. Dynamisches Routing ist sehr flexibel und ermöglicht das exponentielle Wachstum des Internet [69].

Das Internet ist in Autonome Systeme gegliedert, die im Allgemeinen von verschiedenen administrativen Einheiten konfiguriert werden. Oftmals findet eine Klassifizierung der Netzwerke in Tier-1 bis Tier-5 Netzwerke statt. Ein Netzwerk wird als Tier-1 bezeichnet, wenn es jedes Netzwerk im Internet ohne Bezahlung einer Transit-Gebühr erreichen kann [70]. Als Tier-2 Netzwerke werden Provider bezeichnet, die hauptsächlich mit anderen Tier-2 Netzwerken Peering betreiben, aber letztlich nicht jedes Netzwerk ohne eine Entrichtung von Transitgebühren erreichen können [71]. Tier-3 Netzwerke sind letztlich auf die Bezahlung von Transitgebühren angewiesen, um das Internet zu erreichen [71]. Netzwerke in Unternehmen werden als Tier-4 und Netzwerke in kleinen Büros als Tier-5 bezeichnet.

Routing-Protokolle werden grob in zwei Klassen gegliedert. Interior-Gateway-Protokolle (IGP) dienen dem Informationsaustausch innerhalb eines Autonomen Systems (AS). Exterior-Gateway-Protokolle (EGP) sind für den Informationsaustausch zwischen Autonomen Systemen verantwortlich.

3.1.1 IGP-PROTOKOLLE[1]

Zu den bekanntesten Vertretern der IGPs gehören RIP, OSPF und IS-IS. In der Forschung findet IS-IS wenig Beachtung. In der Praxis hat IS-IS durch die einfache Konfiguration wieder eine kleine Renaissance erfahren [65]. Bei IGPs werden zwei Arten von Protokollklassen unterschieden. Zu den Vertretern der Distance-Vector Algorithmen gehört RIP. Die bekanntesten Vertreter der Link-State-Routing Algorithmen sind OSPF und IS-IS. Im Folgenden wird nur kurz auf die grundlegende Funktionsweise und Unterschiede der einzelnen Vertreter eingegangen, um ein kompaktes Gesamtbild zu erzeugen.

3.1.1.1 RIP (DISTANCE-VECTOR PROTOKOLL)[1]

RIP ist ein verteiltes Routing-Protokoll, d.h. jeder Knoten im Netzwerk bildet seine eigene Routing-Tabelle. Als Distanzmaß verwendet RIP die Anzahl der Gateways (Hop Counts) bis zum Ziel [72]. Routing-Informationen werden über UDP versendet. Ein Paketverlust muss durch Timer im Protokoll abgefangen werden. Durch das CTI-Problem (Count To Infinity) ist der maximale Wert der Hop Counts auf 16 festgelegt, d.h. der maximale „Durchmesser“ des Netzwerks darf höchstens 15 Zwischenstationen beinhalten. Danach wird ein Paket verworfen, um ein Zirkulieren der Pakete bis zum Ablauf der TTL (Time To Live) zu verhindern. Verschiedene Protokollerweiterungen versuchen das CTI-Problem (Split Horizon With Poisoned Reverse) zu minimieren. Das Protokoll RIP-MTI (auch unter RMTI bekannt) verhindert das CTI-Problem vollständig [73][74].

Am Anfang kennt jeder Knoten seine direkt angeschlossenen Netzwerke. Die Distanz zu direkt angeschlossenen Netzwerken beträgt 0. Zusätzlich zur Netz-ID und der Distanz wird der nächste Nachbar (Next Hop) gespeichert, über den das Ziel zu erreichen ist. Ein eingehendes Paket für ein bestimmtes Ziel wird zum jeweiligen Next Hop weitergeleitet (Forwarding). Damit besteht ein Eintrag in der Routing-Tabelle aus Netz-ID, Distanz und dem zugehörigen Next Hop. Im nächsten Schritt sendet jeder Knoten seine Routing-Tabelle an alle direkten Nachbarn. Jeder Nachbar vergleicht die empfangenen Einträge mit seinen vorhandenen Einträgen, ob ein kürzerer Weg zu einem Ziel existiert. Bei einem kürzeren Pfad wird der eigene Eintrag aktualisiert und die Distanz um +1 erhöht [73].

Kommt es zu keiner Änderung im Netzwerk sendet jeder Knoten alle 30 Sek. seine komplette Routing-Tabelle an alle Nachbarn (Periodic Update). Das Ausbleiben der regelmäßigen Updates wird genutzt, um Ausfälle im Netzwerk zu erkennen. Um gelegentliche Paketverluste auszuschließen, wird der Expiration Timer auf 180 Sek. gesetzt. Jedes empfangene Update für einen Eintrag in der Routing-Tabelle führt zu einem Reset des Expiration Timers. Bleibt ein Update in dieser Zeitspanne aus, wird die Distanz auf den Maximalwert (16) gesetzt und das Zielnetz ist somit als unerreichbar markiert. Diese neue Information wird dann an alle Nachbarn weitergeleitet (Triggered Update). Eine Route bleibt solange als unerreichbar markiert bis der Flush Timer (240 Sek.) abläuft. Daraufhin wird der Eintrag aus der Routing-Tabelle gelöscht [75].

Die größten Nachteile von RIP sind:

- Einfache Metrik (Anzahl der Hops)
- Die fehlende Unterstützung für konkurrierende Routen (Equal-Cost Multi-Paths, Load Balancing)
- CTI-Problem
- Relativ langsam (insbes. Fehlererkennung, kann durch die zusätzliche Verwendung von BFD ausgeglichen werden)
- Durch die Verwendung von UDP kann nicht sichergestellt werden, dass Routing-Informationen überhaupt ankommen.
- Maximaler „Durchmesser“ des Netzes darf 15 Hops nicht überschreiten.
- RIP produziert viel Routing-Verkehr. Die kompletten Routing-Tabellen werden alle 30 Sekunden ausgetauscht.

3.1.1.2 OSPF (LINK-STATE-ROUTING PROTOKOLL)[1]

Link-State Routing-Protokolle sind komplexer als Distance-Vector Routing-Protokolle. Im Gegensatz zu RIP kennt jeder einzelne Knoten im Netz die gesamte Topologie des Netzwerkes und speichert diese in seiner Link-State-Database (LSD). Ist diese Datenbank einmal aufgebaut, besteht keine Notwendigkeit, weitere Routing-Informationen auszutauschen (nur Hello-Nachrichten alle 10 Sek.). Jeder Router versendet seine LSAs (Link State Advertisements) dann nur noch alle 30 Minuten (Refreshment). Im Falle einer Topologieänderung werden inkrementelle Updates versendet. Ein Router erkennt einen

fehlerhaften Link, wenn HELLO-Nachrichten für 40 Sek. Ausbleiben (schnellere Fehlererkennung als RIP). Die Kosten eines Links werden anhand der Bandbreite durch OSPF berechnet. Link-State-Routing-Protokolle arbeiten nach dem Shortest-Path-First-Algorithmus, der im Wesentlichen eine praktische Umsetzung des bekannten Dijkstra-Algorithmus darstellt. Dieser Algorithmus erzeugt aus der LSD die notwendigen Routing-Einträge [73].

Als Kontrollnachrichten setzt OSPF Link-State-Advertisements (LSAs) ein. In diesen sind die sogenannten Link-State-Updates verpackt. Ein Link-State-Update besteht im Wesentlichen aus einem Teil der eigenen LSD. Ein LSA besteht aus der ID des Urhebers, einem oder mehreren Link-State-Updates, einer Sequenznummer und einem Alter (LSAge). Ein „neuer" Knoten beginnt mit der Sequenznummer 0. Die Sequenznummer wird inkrementiert, wenn sich das LSA geändert hat. Anhand der Sequenznummer können alte Informationen erkannt und aus dem Netzwerk entfernt werden. Beim Fluten und während ein LSA in der LSD verweilt, inkrementiert jeder Router das Alter. Nach 60 Minuten wird ein LSA aus der LSD entfernt, wenn kein Refreshment (alle 30 Minuten, LSRefreshTime) vollzogen wird, sonst wird die LSAge auf 0 zurückgesetzt. Wird ein LSA entfernt, flutet der Knoten unmittelbar das Netzwerk mit der neuen Information. Treffen zwei LSAs mit der gleichen Sequenznummer ein, wird das mit der jüngeren LSAge bevorzugt, falls die Differenz größer als MaxAgeDiff ist. Ist die Differenz kleiner, werden beide als identisch angesehen [73].

Die LSAs werden direkt in IP-Datagrammen gekapselt, d.h. OSPF nutzt im Gegensatz zu RIP kein UDP, sondern besitzt ein eigenes Protokoll (Nummer 89). Fehlererkennung (LSAck) und Session Management (HELLO-Protokoll) wird von OSPF selbst gesichert und gesteuert. Da das Fluten des Netzwerks mit LSAs gegen einen Fehlerfall abgesichert ist, spricht man von Reliable Flooding. Die Multicast Adresse 224.0.0.5 spricht alle OSPF-Router in einem Netz an. Ein Router nimmt ein LSA an, aktualisiert die eigene LSD und flutet die Nachricht an alle Nachbarn. Im Gegensatz dazu muss RIP eine Routing-Information empfangen, untersuchen, ändern (Distanz inkrementieren) und senden [73].

OSPF enthält drei Protokolle: Das HELLO -, EXCHANGE - und FLOODING -Protokoll. Das HELLO-Protokoll hat im Grunde zwei wesentliche Aufgaben. Zum einen wird die Funktionsfähigkeit durch regelmäßige HELLO-Nachrichten überprüft und neue Nachbarn gefunden und initialisiert. Die Initialisierung geschieht durch das EXCHANGE-Protokoll.

Hierbei wird die komplette Link-State-Database beider Knoten synchronisiert. Danach werden Updates anhand des FLOODING-Protokolls durch das komplette Netz verbreitet. Ein Knoten sendet ein erhaltenes LSP an alle seine Nachbarn weiter (Reliable Flooding) [73].

Da ein Knoten das komplette Wissen über das Netzwerk in seiner LSD vorhält, kann OSPF in großen Netzwerken sehr speicher- und rechenintensiv sein. Ein Netzwerk kann in Areas und Stub Areas unterteilt werden, um das Fluten des Netzwerks mit LSAs zu begrenzen (Areas) und die LSD und die Routing-Tabellen zu minimieren (Stub Areas). Ein LSA wird nur in der eigenen Area geflutet. Den Austausch zwischen den Areas übernehmen Area Border Router, die LSAs der eigenen Area herausfiltern und sog. Network Summary LSAs (Distance-Vector!) an die Area 0 übermitteln. Jede Area muss mit der Area 0 (Backbone Area) verbunden sein. OSPF kennt vier Arten von Router-Typen [73]:

- **Internal Routers** (IR): Alle Schnittstellen sind innerhalb einer Area
- **Backbone Routers** (BR): Mind. eine Schnittstelle befindet sich in der Backbone Area
- **Area Border Routers** (ABR): Hat Schnittstellen in mind. zwei Areas
- **Autonomous System Boundary Routers** (ASBR): Hat mind. eine Schnittstelle in einer nicht-OSPF-Domain und verteilt externe Routen in die OSPF-Domain. Auf einem ASBR läuft immer noch ein zweites Routing-Protokoll. Der Router propagiert die Routen zwischen den beiden Protokollen.

Ein ASBR kann viele externe Routen in OSPF fluten. Die LSD wächst dadurch dramatisch an. Um dies zu minimieren, kann eine Area als Stub Area deklariert werden. D.h. externe Routen werden durch die ABRs nicht in die Stub Area geleitet, sondern stattdessen wird in der Stub Area eine Default Route (0.0.0.0) eingerichtet. Eine weitere Möglichkeit die LSD zu minimieren ist die Summarization (in BGP Aggregation). Anstatt jedes einzelne Subnetz weiterzuleiten, wird das aggregierte Netz propagiert [73].

OSPF hat einige Vorteile gegenüber RIP:

- Weniger Verkehr bei häufigen Änderungen der Netzstruktur
- Kürzere Fehlererkennungszeiten als RIP (ohne BFD)
- Kompliziertere Metrik
- Subnetting
- Summarization

- Areas, Stub Areas
- Konkurrierende Pfade für das gleiche Ziel (Equal Cost Multipaths, Load Balancing)
- Multicast

Für OSPF stehen Erweiterungen zur Verfügung, um die Konvergenzzeit zu minimieren bzw. temporäre Schleifen auszuschließen. Bei zeitintensiven Anwendungen wie Audio- oder Video-Streams können so Paketverluste evtl. ausgeschlossen werden (IPFRR) [65][60][66]. Ein Ansatz verhindert temporäre Schleifen durch eine eingeführte Ordnung, wie die LSPs durch das Netz verteilt werden [65][60]. Ein weiterer Ansatz vermeidet die Entstehung temporärer Schleifen, indem das hereinkommende und herausführende Interface die Pfadauswahl beeinflusst [66].

3.1.2 EGP-PROTOKOLLE [1]

IGPs eignen sich nicht für Inter-ISP Routing. Routing zwischen ISPs ist eine Frage von Verträgen, angebotenen Diensten, Richtlinien und somit Geld. Für ISPs ist es daher von besonderer Bedeutung, den Verkehrsfluss manipulieren zu können. Seit ca. 20 Jahren wird BGP als Inter-Domain Routing-Protokoll im Internet genutzt. Autonome Systeme werden anhand ihrer Nummer unterschieden. Ein Austausch der Routing-Informationen zwischen Autonomen Systemen geschieht über BGP. Allein das Anwendungsgebiet lässt eine weitaus höhere Komplexität als bei IGPs vermuten.

3.1.2.1 BGP (BORDER GATEWAY PROTOCOL)[1]

BGP wird als Path-Vector-Protokoll klassifiziert. BGP-Updates sind ähnlich wie bei OSPF zuverlässig (TCP), d.h. ein Sender bekommt immer eine Bestätigung, ob die Information beim Empfänger eingetroffen ist. Aus diesem Grund müssen nur bei einer Änderung Routing-Informationen versendet werden. Um einen Ausfall erkennen zu können, versendet BGP alle 60 Sek. (Default) ein Kontrollpaket (Keepalive Packet). Benachbarte Router werden in BGP manuell konfiguriert. Eine BGP-Session wird immer von zwei Border-Routern aufgebaut, die sich in unterschiedlichen Autonomen Systemen befinden. BGP verwendet vier Arten von Nachrichten [73]:

- **Open**: Wird für den Verbindungsaufbau verwendet. Enthält unter anderem die BGP-Version, die AS Nummer und eine Hold Time (wie lange wird auf ein Keepalive gewartet).
- **Update**: Enthält neue bzw. geänderte Routing Informationen (inkl. der Attribute) und/oder verworfene Routen (Withdrawn Routes)
- **Notification**: Wird benutzt, um Fehler mitzuteilen (Status Code). Nach einer Notification kehrt der Router in den Idle-Zustand zurück.
- **Keepalive**: Wird benutzt, um eine Open-Anfrage zu beantworten und alle 60 Sek. (Default) den Status der Verbindung zu prüfen. Es wird empfohlen, ein Drittel der Hold Time als Intervall für die Keepalive-Nachrichten zu verwenden.

Ein Autonomes System ist eine Gruppe von IP-Netzwerken, die von einem oder mehreren Administratoren eingerichtet werden und eine klar definierte Routing-Richtlinie verfolgen (Routing Policy). RFC 1930 unterscheidet drei Arten von Autonomen Systemen [76]:

- **Stub bzw. Single-Homed**: Ein AS ist durch einen einzigen ISP mit dem Internet verbunden. Normalerweise wird in dieser Situation kein BGP verwendet, sondern oftmals statische oder Default Routen.
- **Multi-Homed Non-Transit**: Ein AS ist mit mehreren ISPs verbunden, die Zugang zum Internet gewähren. Es wird empfohlen, eine öffentliche AS Nummer zu vergeben und einen vom Provider unabhängigen Adressbereich zu verwenden.
- **Multi-Homed Transit**: Ein Transit AS tauscht Routing-Informationen mit anderen AS und leitet von einem AS empfangene Routing Informationen an andere weiter.

In Vereinbarungen (Peering Agreements) zwischen IPSs wird festgehalten, wie hoch das Trafficvolumen in jede Richtung sein darf. Profitieren beide Seiten vom Peering, ist dies im Allgemeinen kostenlos, ein Ungleichgewicht wird finanziell ausgeglichen [73].

Ein AS teilt einem benachbarten AS mit, welche Präfixe es erreichen kann. Der Empfänger weiß nun, dass er diese Präfixe erreichen kann, wenn er den Verkehr an den Sender weiterleitet. Jedes BGP-Update enthält eine oder mehrere IP-Subnetze (NLRI, Network Layer Reachability Information) mit einer Menge an Attributen. Zu löschende Routen werden nur per NRLI ohne Attribute propagiert. Autonome Systeme (bzw. Confederations) hängen die eigene Nummer an eine durchlaufende Routing-Information an (AS_PATH Attribut). Dadurch werden Schleifen erkannt und ggf. wird die Information verworfen. Die

Schleifenerkennung ermöglicht es, dass das AS_PATH Attribut ausgenutzt werden kann, um bestimmte Policies durchzusetzen, indem durch Manipulation Schleifen im AS_PATH Attribut erzeugt werden. Richtlinien (Policies) in BGP definieren, welche Routen akzeptiert, propagiert und verwendet werden. Jede Route besitzt mehrere Attribute, die Metrik ist nur eine von vielen. BGP kann anhand der Attribute eine bestimmte Policy umsetzen. Die Routing-Informationen werden in der RIB (Routing Information Base) gespeichert. Nur die besten Pfade werden selektiert und propagiert [73].

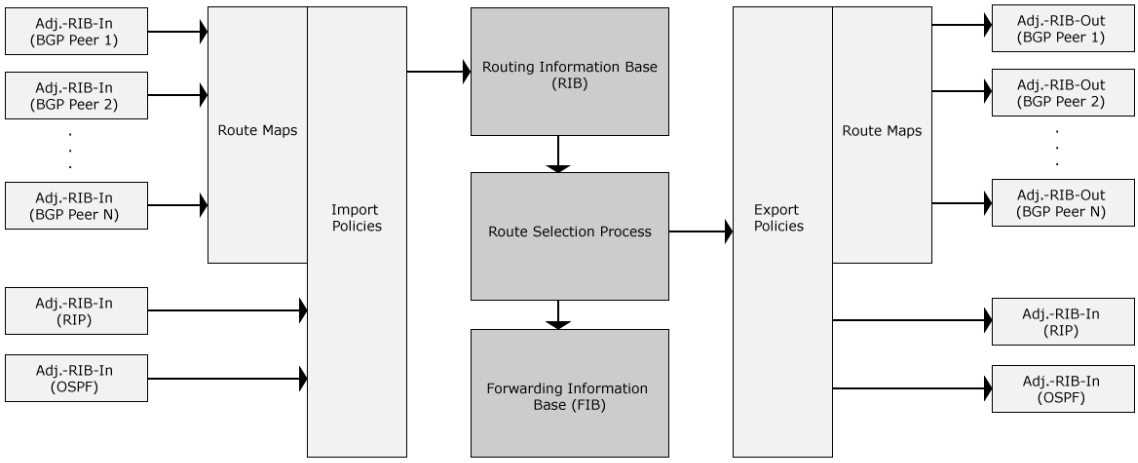


Abbildung 31: BGP Routing Information Base [73]

Attribute werden in BGP in Gruppen (Well Known, Optional) und Untergruppen (Mandatory, Discretionary, Non-Transitive, Transitive) unterteilt und bestehen aus dem Typ (Type), der Länge (Length) und dem eigentlichen Wert (Value) [73]. Folgende Attribute sind im Hinblick auf Schleifen interessant:

- **Well-Known**: Diese Attribute müssen von allen BGP-Implementationen erkannt werden.
    - **Mandatory**: Diese Attribute müssen in einem Update vorkommen:
        - **ORIGIN** gibt an, wie die Route gelernt wurde. Ein Wert von 0 (IGP) zeigt an, dass die Route per *network* Kommando oder per Aggregation gelernt wurde. Insbesondere die Aggregation ist eine Schlüsselkomponente bei der Entstehung von persistenten Schleifen. Wurde die Route via EBGP gelernt, wird Origin auf 1 (EGP) gesetzt.

Der Wert 2 (INCOMPLETE) zeigt an, dass die Route von einem IGP stammt.

- **AS_PATH** beinhaltet alle traversierten AS und liefert die Grundlage für die Schleifenerkennung in BGP. Wird ein AS mehr als einmal durchlaufen, deutet dieser Umstand auf eine Schleife hin und das Update wird verworfen. Dadurch wird ein Zirkulieren von BGP-Updates vermieden.
- **NEXT_HOP** zeigt die IP des letzten EBGP-Router an, der zum Ziel führt. Das muss nicht notwendigerweise der nächste physische Hop sein.

- **Discretionary**: Diese Attribute dürfen in einem Update vorkommen:
  - **ATOMIC_AGGREGATE** zeigt an, dass ein Router eine Aggregation vorgenommen hat, also Routing-Information zusammengefasst hat. Der AS_PATH der aggregierten Route enthält normalerweise ein AS_SET, d.h. die Menge von AS aus denen das Aggregat geformt wurde. Ein Netzwerkadministrator kann, wenn Schleifen ausgeschlossen werden können, das AS_SET entfernen. In diesem Fall wird das Attribut ATOMIC_AGGREGATE gesetzt, also sobald eine Information aus dem AS_PATH entfernt wurde.

- **Optional**: Diese Attribute müssen nicht von allen BGP-Implementationen erkannt werden.
  - **Non-Transitive**: Diese Attribute werden NICHT an andere AS propagiert:
    - **ORIGINATOR_ID**: Jeder Routing-Information wird die ID des Urhebers beigefügt, um Schleifen erkennen zu können. Ist die Originator-ID identisch mit der ID des Routers, wird das Update verworfen.
    - **CLUSTER_ID**: Wenn mehrere Route-Reflektoren redundant eine gleiche Menge an Clients versorgen, wird die Cluster-ID zur Erkennung der Redundanz benutzt.
  - Diese Attribute werden an andere AS propagiert (**Transitive**):
    - **AGGREGATOR** beinhaltet die IP-Adresse des BGP Routers (und Nummer des AS), der die Aggregation vorgenommen hat.

Der Selektionsprozess einer Route (Route Selection) wird aufgrund einer Liste von verschiedenen Kriterien (Tie Breakers) von oben nach unten abgearbeitet, um im Falle einer Gleichheit von Attributen eine Wahl treffen zu können. Die Kriterien des Selektionsprozesses variieren leicht je nach Hersteller. Mögliche Kriterien des Selektionsprozesses sind [73]:

- Gewicht (Weight, nur Cisco)
- LOCAL_PREFERENCE (höchster Wert wird bevorzugt)
- Länge des AS_PATH (kürzester Wert wird bevorzugt)
- Origin: IGP>EGP>INCOMPLETE
- IGP Kosten zum Next Hop (niedrigster Wert wird bevorzugt)
- MED (niedrigster Wert wird bevorzugt)
- EBGP>IBGP
- ID des BGP Router (niedrigster Wert wird bevorzugt)

Ein BGP Router besitzt drei Möglichkeiten, den Routing-Prozess zu beeinflussen [73]:

- **Prefix List**: Bestimmte Prefixe oder Bereiche von Prefixen können explizit erlaubt oder verboten werden.
- **Filter List**: Reguläre Ausdrücke werden benutzt, um Routen mit bestimmten Attributen zu filtern, z.B. selektiert *_4_* alle Routen die AS4 passiert haben.
- **Route Maps** sind das mächtigste Mittel und bestehen aus einer Bedingung und einer Anweisung. Trifft die Bedingung zu, wird die Anweisung ausgeführt.

Da ein AS oftmals mehrere Border-Router unterhält, werden diese durch IBGP synchronisiert. IBGP hat jedoch keine Schleifenerkennung, da diese sich im gleichen Autonomen System befinden und das AS_PATH Attribut für die Schleifenerkennung im gleichen AS ausscheidet. Über IBGP empfangene Routing-Informationen dürfen nicht über IBGP weitergegeben werden. Als Folgerung müssen alle IBGP-Router direkt vernetzt sein (Full Mesh), d.h. jeder IBGP-Router muss eine Session mit jedem anderen IBGP-Router unterhalten. Die Zahl der Verbindungen steigt damit rapide mit der Zahl der IBGP-Router. Aus diesem Grund wurde in BGP das Konzept der Route Reflectors und Confederations eingefügt. Confederations ermöglichen die Unterteilung eines AS in mehrere kleinere Systeme (sub-AS). Nur innerhalb dieser Systeme ist ein Full Mesh aller IBGP-Router notwendig. Die Confederations wiederum sind mit EBGP untereinander verbunden.

Allerdings ist dieses Konzept schwieriger zu konfigurieren, weshalb oftmals Route Reflectors eingesetzt werden. Ein Route-Reflektor ist ein IBGP-Router der die besten Routing-Informationen an alle verbundenen IBGP-Router weitersendet. Somit muss jeder IBGP-Router eine Verbindung zum Route-Reflektor unterhalten. Dieser sendet eine über eine Schnittstelle empfangene Routing-Information über alle anderen Schnittstellen weiter. Damit wird die Anzahl der Verbindungen von einem Full-Mesh auf eine Baumstruktur reduziert. Um Schleifen erkennen zu können, wird jedem Update die ID des Urhebers (ORIGINATOR_ID) und die ID des Clusters (CLUSTER_ID) beigefügt. Ist die Originator-ID identisch mit der ID des Routers, wird das Update verworfen. Die Cluster-ID wurde von Cisco für den Fall eingeführt, falls mehrere Route Reflektoren redundant die gleiche Menge an Clients versorgen. Wenn mehrere IBGP-Sessions in einem Netz ausfallen, kann die Verwendung der Cluster ID zu einer eingeschränkten Verbindung führen, weshalb vom Einsatz abgeraten wird [73].

Um das Fluten der Netze mit immer neuen Routing-Informationen zu verhindern, die durch „flatternde“ Verbindungen oder Attribute ausgelöst werden können, bietet BGP das sogenannte Route Flap Dampening. Eine „flatternde“ Verbindung schwankt ständig zwischen funktionstüchtig und funktionsuntüchtig hin und her. Damit nicht jedes Mal alle Netze mit neuen BGP-Updates geflutet werden müssen, wird eine Verbindung ab einem Suppress Limit von 2000 nicht mehr propagiert. Ein Flap löst per Default eine Strafe von 1000 aus. Die Strafe wird alle 15 Minuten ohne Flap halbiert und ab einem Reuse Limit von 750 wird die Verbindung wieder propagiert. Die maximale Suppress Time liegt bei 60 Minuten. Leidet die Erreichbarkeit von Teilnetzen unter einem Flattern, wird dies oftmals durch die Aggregation an späterer Stelle im Netz unterdrückt [73].

3.2 NACHWEIS VON SCHLEIFEN[1]

Der Nachweis von Schleifen ist mehr als trivial und der Hauptgrund für die lang unterschätzte Beachtung in der Forschung. Zum einen existieren aufgrund der vielen AS und damit einer Vielzahl von involvierten Unternehmen/Organisationen keine ganzheitlichen Aufzeichnungen. Richtlinien, interne Ziele und Wettbewerb der administrativen Domains erschweren den Zugang zu vorhandenen Aufzeichnungen. Aufgrund des enorm gestiegenen Datenverkehrs ist eine temporäre Protokollierung an einem oder mehreren Knoten eine

erhebliche Investition und eine komplette Protokollierung des Datenverkehrs unter finanziellen Gesichtspunkten nahezu unmöglich für gewinnorientierte Unternehmen. Weiterhin existiert kein automatisierter Erkennungs- und Warnmechanismus für auftretende temporäre und persistente Schleifen. Eine für diesen Mechanismus nähere Untersuchung der Pakete (Deep Packet Inspection) wäre bei heutiger Geschwindigkeit und Volumen des Datenverkehrs zu aufwendig.

Oftmals werden in der Forschung beide nachfolgend vorgestellten Verfahren kombiniert. Entweder weil beide Schleifenarten untersucht werden sollen oder man die gefundenen Ergebnisse zusätzlich absichern möchte.

3.2.1 ANALYSE VON PAKETAUFZEICHNUNGEN[1]

Eine Möglichkeit Schleifen zu finden, besteht in der Analyse von Paketaufzeichnungen eines Knoten im Netz. Eine Paketaufzeichnung stellt ein lückenloses Abbild des aufgezeichneten Intervalls dar. Es ist offensichtlich, dass der Knoten der Aufzeichnung ein Mitglied der gefundenen Schleife sein muss.

Ein Algorithmus zum Auffinden von Schleifen in Paketaufzeichnungen wird in [61] beschrieben:

1. **Auffinden von replizierten Paketen**: Ein repliziertes Paket enthält die identischen Nutzdaten (Payload) und bis auf TTL und Header-Checksum einen identischen IP-Header. Die TTL muss dabei eine Abweichung von mindestens 2 enthalten. Oftmals werden nur die ersten z.B. 40 Bytes eines Pakets aufgezeichnet, um das Datenvolumen zu minimieren und/oder eine aus juristischen Gründen notwendige Anonymisierung zu erreichen (siehe Kapitel 2.4.2). Um die fehlende Payload trotzdem zuverlässig vergleichen zu können, wird die TCP- oder UDP-Checksumme verglichen, die im Gegensatz zur IP-Checksumme aus TCP-Header und der TCP-Payload berechnet wird.
2. **Replizierte Datenströme validieren**: Als Beweis für eine Routing-Schleife in einer Paketaufzeichnung müssen zwei Bedingungen erfüllt sein. Zunächst müssen mindestens mehr als zwei replizierte Pakete vorhanden sein. Es kann durch Fehler vorkommen, dass ein Paket durch die Link-Layer-Schicht dupliziert wird. Z.B. wenn

ein fehlerhaft konfigurierter SONET-Protection-Layer das Paket auf der eigentlichen und der redundanten Verbindung weiterleitet. Weiterhin muss bestimmt werden, ob das replizierte Paket zum Zeitfenster der vermuteten Schleife gehört. Das längste Präfix bei Tier-1 ISPs besitzt eine Länge von 24-Bits, d.h. replizierte Pakete mit einer Übereinstimmung der ersten 24-Bits in der Zieladresse, werden zu replizierten Datenströmen zusammengefasst (Replica Streams).

3. **Zusammenfassen der replizierten Datenströme**: Die identifizierten Datenströme aus Schritt 2 sind ein Indiz für eine aufgetretene Schleife. Replizierte Datenströme mit der gleichen Zieladresse und einer zeitlichen Überlappung sind mit höchster Wahrscheinlichkeit einer Schleife zuzuordnen und ein Kriterium für die Zusammenfassung. Weiterhin werden Datenströme mit der gleichen Zieladresse und einem zeitlichen Abstand von einer Minute einer Schleife zugeordnet, um den Fall abzufangen, dass es eine kurze Pause im Datenverkehr gab.

3.2.1.1 SCHLEIFEN DEN VERURSACHENDEN BGP-EVENTS ZUORDNEN[1]

In [77] wird eine Methodik vorgestellt, um BGP-Events den gefundenen Schleifen zuzuordnen. BGP-Statusänderungen können temporäre Schleifen erzeugen z.B. eine Änderung im AS_PATH oder NEXT_HOP, die aufgrund zeitlicher Verzögerungen von Timern (z.B. Route Flap Dampening) oder Übertragungszeiten noch nicht durch das gesamte Netz propagiert wurden. Eine Änderung im AS_PATH oder NEXT_HOP ist eine notwendige aber nicht hinreichende Bedingung. Die Formulierung einer hinreichenden Bedingung benötigt die kompletten BGP-Tabellen aller Nachbarn zur Überprüfung. Aus diesem Grund wird in [77] nur eine Änderung im AS_PATH oder NEXT_HOP inkl. einer zeitlichen Nähe zur resultierenden Schleife überwacht.

Die benötigten BGP-Updates werden von einem passiven Empfänger mitgeschnitten, der sich im Sprint Backbone befindet und als Client eines Route Reflectors alle BGP-Updates innerhalb des Sprint-Netzwerks empfangen kann. Zudem werden Daten des Routeview-Projekts [78] zur Unterstützung herangezogen. Das Routeviews-Projekt sammelt externe BGP-Updates und liefert somit eine breitere Datengrundlage.

Jedes BGP-Update durchläuft folgende Schritte, um Assoziationen mit gefundenen Schleifen zu erkennen [77]:

1. Es wird geprüft, ob die Zieladresse der betroffenen Pakete (d.h. Pakete in der Schleife) zu propagierten (Advertised) oder zurückgezogenen (Withdrawn) Präfixen des BGP-Updates passt (Longest Prefix Match).
2. Das BGP-Update muss in einer zeitlichen Nähe von 2 Minuten zur gefundenen Schleife liegen.
3. Ein Zebra-Router (Projektseite: www.zebra.org) wird mit dem BGP-Update versorgt, um zu überprüfen, ob der BGP-Entscheidungsprozess die Attribute NEXT_HOP oder AS_PATH für das Zielpräfix ändert.

Das Verfahren ist sowohl für temporäre als auch persistente Schleifen geeignet. Bei persistenten Schleifen kann das verursachende BGP-Update schon länger zurückliegen und dementsprechend nicht im Datenstrom aufgezeichnet worden sein.

3.2.1.2 DER ZUORDNUNGS-ALGORITHMUS IN DER PRAXIS[1]

In [77] werden im Jahr 2003 mehrere Paketaufzeichnungen von OC-48 Links zwischen Backbone-Routern aus dem SprintLink Netzwerk analysiert. Drei Aufzeichnungen sind 1 Stunde und drei 12 Stunden lang (Abbildung 32). Als Bedingung für persistente Schleifen, müssen diese eine Woche lang bestehen. Nach dem Finden der Schleifen in den Aufzeichnungen werden Traceroutes zu verschiedenen Zeitpunkten über eine Woche ausgeführt, um die Dauer zu ermitteln.

Trace Name	No. of Packet Loops	Duration (hrs)
NYC-20	2476	1
NYC-21	3838	1
NYC-23	1895	1
NYC-22	8672	12
NYC-24	719	12
NYC-25	1691	12

Abbildung 32: Anzahl der in Schleifen geratenen Pakete (2003) [77]

Auffällig ist, dass die Anzahl in extremen Größenordnungen variiert (Abbildung 33). Es wird versucht, gesammelte BGP- und IS-IS-Updates als Ursache den gefundenen Schleifen zuzuordnen. BGP-Updates werden zum einen selbst gesammelt und zugeordnet, zum anderen werden Informationen des Routeviews-Projekts herangezogen, und diese versucht den Schleifen zuzuordnen. Es kann kein einziges IS-IS Routing-Update den gefundenen Schleifen zugeordnet werden. Die Autoren gehen davon aus, dass die Konvergenzzeit bei IS-IS zum einen nicht kritisch ist und das Feature Equal-Cost Multi-Path das schnelle Auffinden von Alternativen ermöglicht.

Trace	% Transient & BGP Updates	% Persistent BGP Updates	%Persistent No Updates	Total
NYC-20	40.1	0	50.8	90.8
NYC-21	80.2	0	7.5	87.9
NYC-23	3.3	0	0	3.3
NYC-22	18.8	0	80.6	99.4
NYC-24	70.0	0	0	70.0
NYC-25	43.7	15.5	0	59.2

Abbildung 33: Erfolgsrate der Zuordnung der BGP-Updates zu Schleifen (2003) [77]

Die starke Varianz der erfolgreichen Zuordnung lässt sich aus mehreren Gründen ableiten. Zum einen können Schleifen schon vor der Aufzeichnung existieren und das verursachende BGP-Update konnte nicht aufgezeichnet werden. Weiterhin kann eine Schleife durch externe BGP-Updates verursacht werden. 50,8% bzw. 80,6% der gefundenen Schleifen in NYC-20 und NYC-22 sind persistent, jedoch kann keine Zuordnung zu BGP-Updates gefunden werden (Abbildung 33). Ein Sonderfall besitzt NYC-25. Hier können alle persistenten Schleifen (15,5% aller gefundenen Schleifen) einem BGP-Update zugeordnet werden (Abbildung 33). In NYC-24 und NYC-23 können keine persistenten Schleifen gefunden werden, was nicht die Abstinenz beweist, sondern auf andere Faktoren schließen lässt, die die Erkennungstechnik beeinflusst haben (Abbildung 33).

Trace	% Sprint Matches	% RouteViews Matches
NYC-20	40.1	43.1
NYC-21	80.2	82
NYC-23	3.3	10.6

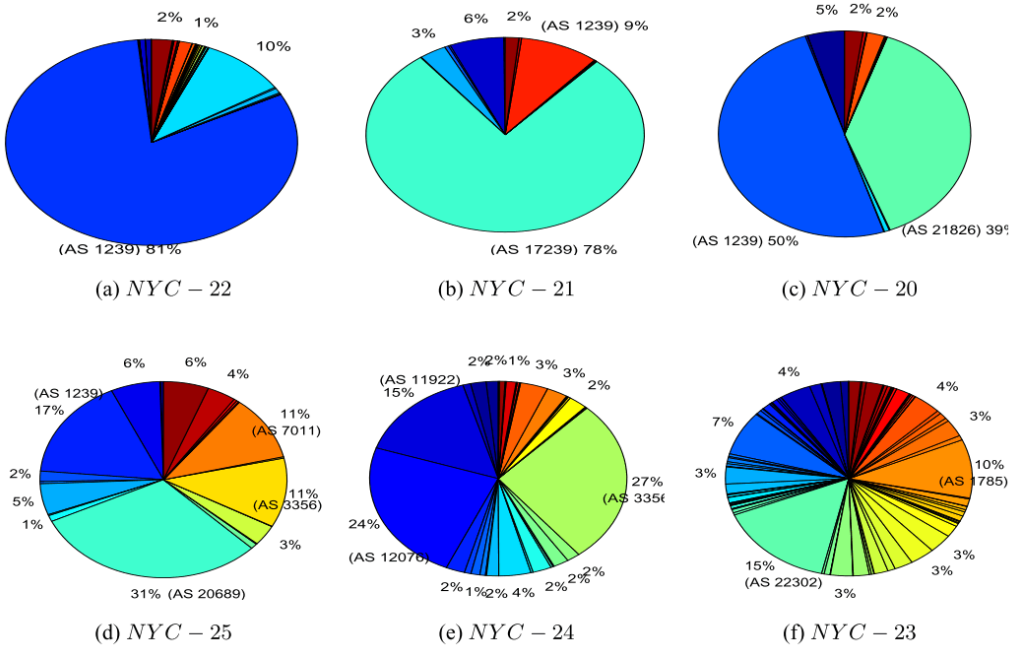
Abbildung 34: Erfolgsrate bei der Zuordnung nach Quelle der BGP-Updates (2003) [77]

Um externe BGP-Updates als Ursache miteinzubeziehen, werden die aufgezeichneten BGP-Updates des Routeviews-Projekts zugeordnet und die Erfolgsrate verglichen (Abbildung 34).

Eine Verbesserung von 7% bei NYC-23 lässt auf vermehrt externe Ursachen schließen (Abbildung 34), was wiederum die niedrige Erfolgsrate bei der Zuordnung aufgezeichneter BGP-Updates aus dem SprintLink-Netzwerk erklärt. Unterstützt wird diese Annahme durch die durchschnittliche Anzahl durchlaufener AS der gefundenen Schleifen, die bei NYC-23 am höchsten ist (Abbildung 35).

Trace	Avg. No. Of ASes traversed
NYC-20	1.34
NYC-21	1.04
NYC-23	1.74
NYC-22	0.513
NYC-24	1.61
NYC-25	1.63

Abbildung 35: Durchschnittlich Anzahl durchlaufener AS in Schleifen (2003) [77]



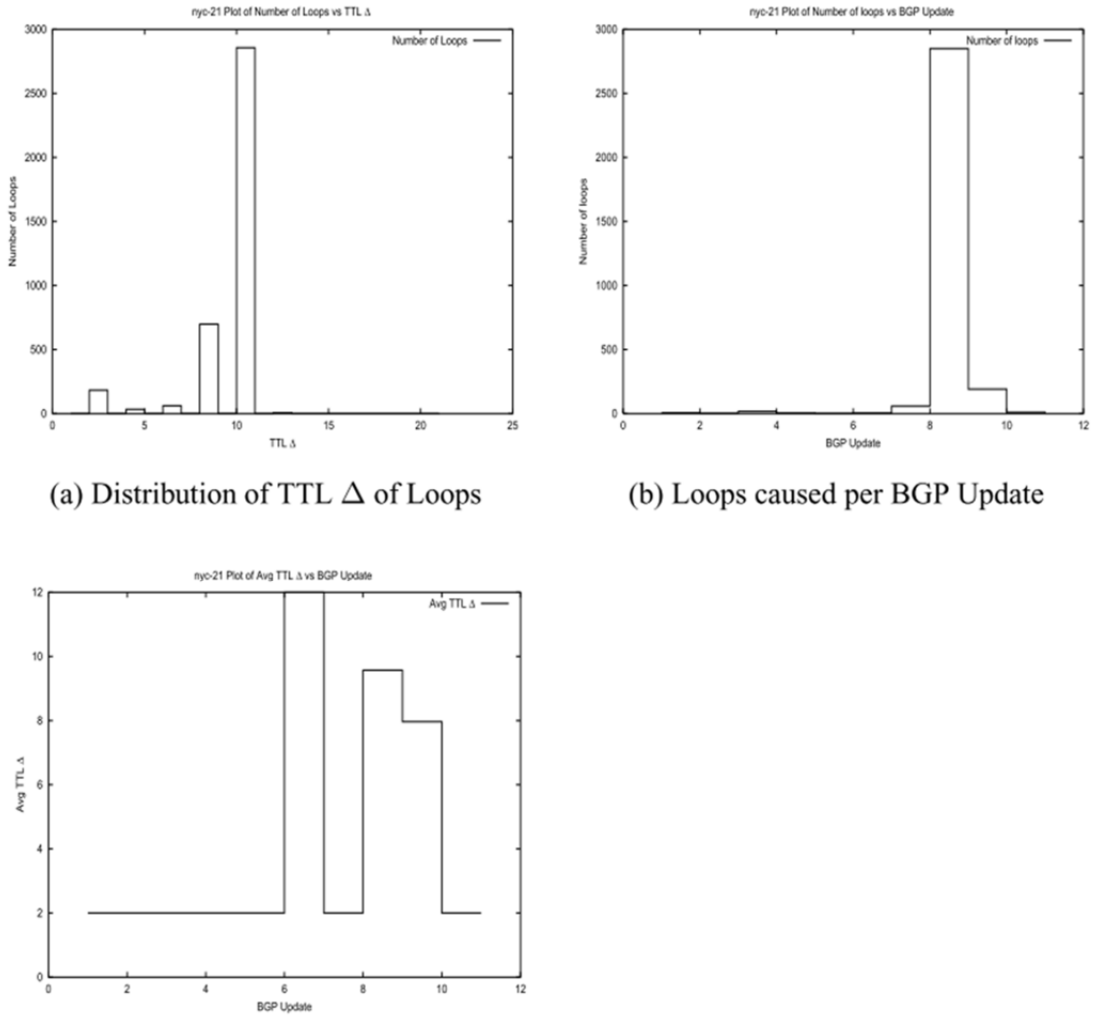
(a) $NYC-22$ (b) $NYC-21$ (c) $NYC-20$

(d) $NYC-25$ (e) $NYC-24$ (f) $NYC-23$

Abbildung 36: Verteilung der Zieladressen der Schleifen auf AS# (2003) [77]

Abbildung 36 beschreibt die Verteilung der Zieladressen auf die Anzahl der Autonomen Systeme. Die Mehrheit der Zieladressen der Schleifen in NYC-21 (78%) und NYC-22 (81%)

gehören zu einem AS. Das AS in NYC-21 gehört zu einem Kunden und im Fall von NYC-22 ist das AS Teil des SprintLink-Netzwerks, was die Trefferquote von 99,4% bei der Zuordnung erklärt, da alle BGP-Updates aufgezeichnet werden konnten (Abbildung 33). Zieladressen der Schleifen in NYC-20 sind über mehrere AS verteilt, wobei zwei dominante AS (verantwortlich für 50% bzw. 39% der Schleifen) hervorstechen. In NYC-24 und NYC-25 verteilt sich die Dominanz auf weitere AS. Die meiste Variation bietet NYC-23 und liefert eine weitere Erklärung für die niedrige Erfolgsrate bei der Zuordnung aufgezeichneter BGP-Updates.



(a) Distribution of TTL Δ of Loops

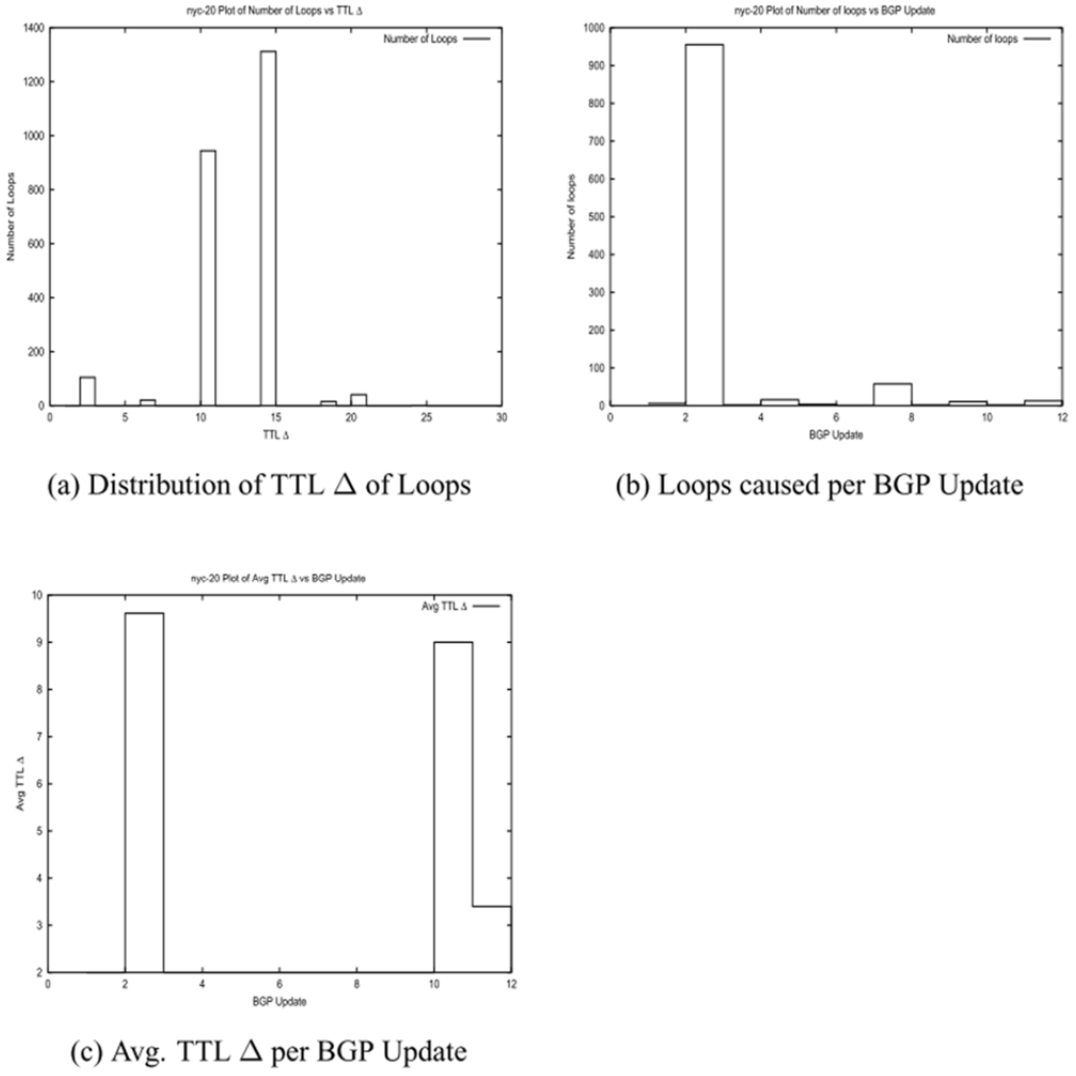
(b) Loops caused per BGP Update

(c) Avg. TTL Δ per BGP Update

Figure 5: NYC-21: Routing Loop characteristics

Abbildung 37: Charakteristika der Schleifen in NYC-21 (2003) [77]

Sind nur wenige BGP-Updates für die Schleifenentstehung verantwortlich, verteilt sich das TTL-Delta auf ein kleineres Spektrum. Ist die Anzahl der für Schleifen verantwortlichen BGP-Updates relativ klein, ist die Verteilung der Zieladressen ebenfalls klein. Ein großer Teil der Schleifen folgt somit dem gleichen Pfad, was sich in einer schmaleren Verteilung der Schleifengrößen widerspiegelt.



(a) Distribution of TTL Δ of Loops

(b) Loops caused per BGP Update

(c) Avg. TTL Δ per BGP Update

Figure 6: NYC-20: Routing Loop characteristics

Abbildung 38: Charakteristika der Schleifen in NYC-20 (2003) [72]

Abbildung 37 (5a) zeigt, dass 74% der Schleifen in NYC-21 eine Größe von 10 und 18% eine Größe von 8 haben. Diese schmale Verteilung sollte implizieren, dass nur wenige BGP-Updates für die Schleifen verantwortlich sind. Ungefähr 74% der Schleifen sind mit einem

einzigen BGP-Update verbunden (Index 8 in Abbildung 37 (5b)). Das BGP-Update mit dem Index 8 erzeugt Schleifen mit einem TTL-Delta von 10 und 8 (Abbildung 37 (5c)).

50,8% der Schleifen in NYC-20 haben ein TTL-Delta von 14 (Abbildung 38). Interessanterweise existieren diese vor der Aufzeichnung, sodass keine verursachenden BGP-Updates aufgezeichnet werden konnten. Abbildung 38 (6b) zeigt wiederum, dass das BGP-Update mit dem Index 2 an der Bildung der meisten Schleifen beteiligt ist und durchschnittlich Schleifen der Größe 9,5 formt.

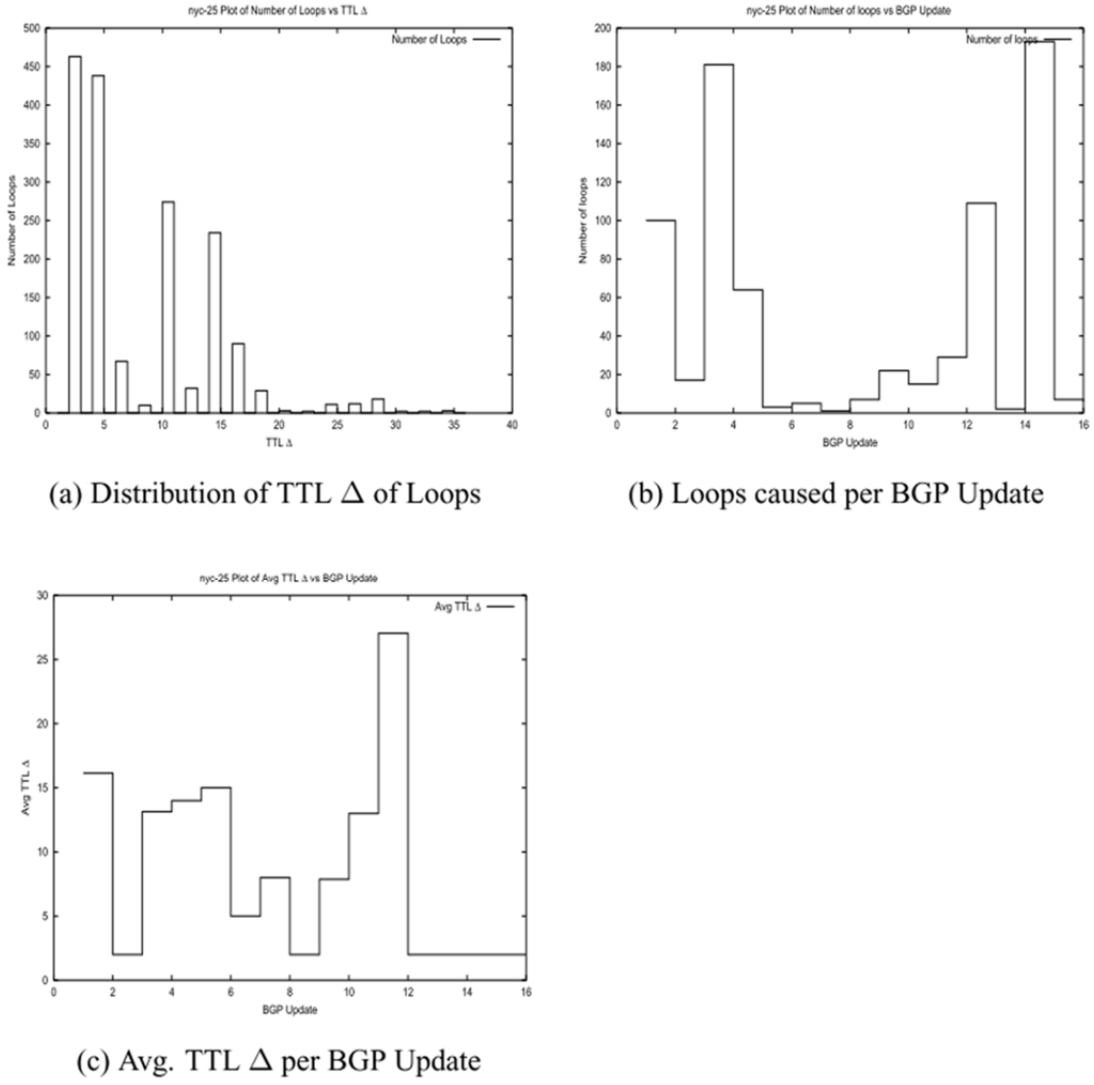


Abbildung 39: Charakteristika der Schleifen in NYC-25 (2003) [77]

Die Gegenprobe bilden die Charakteristika der Schleifen in NYC-25. Hier sind eine große Anzahl an BGP-Updates für die Schleifenbildung verantwortlich Abbildung 39 (9b), was sich in einem größeren Spektrum an TTL-Deltas ausdrückt Abbildung 39 (9a).

3.2.2 TRACEROUTE[1]

Traceroute unter Windows (tracert.exe) ermittelt mit ICMP-Echo-Requests den Weg durch das Netz zu einem bestimmten Ziel. Aufgrund der Dynamik des Internet kann sich dieser Weg über die Zeit ändern. Traceroute bietet demnach nur eine Momentaufnahme. Das erste Paket wird mit einer TTL von 1 gesendet, woraufhin der erste Knoten das Paket verwirft und eine ICMP-Nachricht (Typ 11: Time Exceeded) mit seiner Adresse als Absender zurücksendet. Das nächste Paket wird mit einer TTL von 2 versendet, womit der nächste Knoten identifiziert werden kann. Diese Schritte werden wiederholt bis das eigentliche Ziel erreicht ist [79].

Traceroute unter Unix arbeitet nach dem gleichen Prinzip, verwendet aber UDP-Pakete als Standardeinstellung. Dabei wird eine große Portnummer (33434 oder höher) benutzt, die sehr wahrscheinlich vom Zielrechner nicht benutzt wird. Der Zielrechner antwortet daraufhin mit einer ICMP-Nachricht, dass der verwendete Port nicht erreichbar ist (Typ 3: Destination Unreachable) [79].

Da alle Zwischenstationen entweder ICMP-Nachrichten oder UDP-Nachrichten aufgrund von Firewalls oder Richtlinien (Policies) filtern können, kann die Ausgabe variieren bzw. eine erfolgreiche Ausgabe des Pfads unterbunden werden. Unter Unix kann Traceroute mit einem Kommandozeilenparameter angewiesen werden, ICMP-Nachrichten zu benutzen (*traceroute –I www.google.de).*

Eine gleiche Sequenz von Routern muss in vielen Studien innerhalb des Traceroutes mindestens drei Mal erscheinen, um als Schleife deklariert zu werden [80][64]. Hier gibt es keine generelle Vorgabe für eine Klassifizierung als Schleife. Eine temporäre Schleife sehr kurzer Dauer kann sich durchaus obigem Merkmal entziehen.

Nachstehende Ausgabe zeigt den Einsatz des Befehls *tracert.exe* unter Windows ohne Schleife. Das Ziel 10.0.10.100 wird über 10.0.0.2, 10.0.1.2 und 10.0.20.2 erreicht.

```
C:\Users\root>tracert 10.0.10.100

Routenverfolgung zu 10.0.10.100 über maximal 30 Abschnitte

  1    <1 ms    <1 ms    <1 ms  192.168.137.2
  2    <1 ms    <1 ms    <1 ms  10.0.0.2
  3     1 ms     1 ms     1 ms  10.0.1.2
  4     1 ms     1 ms    <1 ms  10.0.20.2
  5     1 ms     1 ms     1 ms  10.0.10.100

Ablaufverfolgung beendet.
```

Befindet sich eine Schleife auf dem Weg zu einem Ziel, werden alle Knoten der Schleife nacheinander traversiert. Im folgenden Beispiel sind das die Knoten 10.0.0.2, 10.0.1.2 und 10.0.2.1. Solange die Schleife existiert, wird das eigentliche Ziel 10.0.10.100 nicht erreicht. Die Datenpakete würden bis zum Ablauf der TTL die Schleife durchlaufen.

```
C:\Users\root>tracert 10.0.10.100

Routenverfolgung zu 10.0.10.100 über maximal 30 Abschnitte

  1    <1 ms    <1 ms    <1 ms  192.168.137.2
  2    <1 ms    <1 ms    <1 ms  10.0.0.2
  3     1 ms    <1 ms    <1 ms  10.0.1.2
  4    <1 ms    <1 ms    <1 ms  10.0.2.1
  5     1 ms    <1 ms    <1 ms  10.0.0.2
  6     1 ms     1 ms     1 ms  10.0.1.2
  7     1 ms     1 ms    <1 ms  10.0.2.1
  8     1 ms     1 ms     1 ms  10.0.0.2
  9     1 ms     1 ms     1 ms  10.0.1.2
 10     1 ms     1 ms     1 ms  10.0.2.1
 11     2 ms     1 ms     1 ms  10.0.0.2
 12     2 ms     1 ms     1 ms  10.0.1.2
 13     1 ms     1 ms     1 ms  10.0.2.1
 14     2 ms     1 ms     1 ms  10.0.0.2

Ablaufverfolgung beendet.
```

Die Ausgabe von traceroute unter Linux liefert ein semantisch ähnliches Bild. Pro TTL werden jedoch nacheinander 3 IP-Datagramme versendet und die Zeit bis zum Eintreffen der ICMP-Nachricht gemessen. Trifft die ICMP-Nachricht nicht innerhalb von 5 Sekunden ein, wird das nächste IP-Paket mit identischer TTL versendet und ein „*“ ausgegeben. Die Verzögerung lässt sich damit erklären, dass der Traceroute in ein virtuelles Netz gesendet wird, welches auf einem physischen Host simuliert wird.

```
lubuntu@lubuntu-VirtualBox:~$ traceroute 10.0.10.100
traceroute to 10.0.10.100 (10.0.10.100), 30 hops max, 60 byte packets
 1  192.168.137.2 (192.168.137.2)  0.599 ms  0.488 ms  0.406 ms
 2  10.0.0.2 (10.0.0.2)  0.785 ms  1.926 ms  1.850 ms
 3  10.0.1.2 (10.0.1.2)  2.149 ms  2.185 ms  2.217 ms
 4  10.0.2.1 (10.0.2.1)  1.614 ms  1.539 ms  1.606 ms
 5  10.0.0.2 (10.0.0.2)  1.996 ms  2.050 ms  2.188 ms
 6  10.0.1.2 (10.0.1.2)  2.664 ms  1.652 ms  1.672 ms
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * 10.0.1.2 (10.0.1.2)  2.252 ms  2.368 ms
13  10.0.2.1 (10.0.2.1)  2.542 ms  6.587 ms  6.606 ms
14  10.0.0.2 (10.0.0.2)  6.602 ms  6.495 ms  6.396 ms
15  10.0.1.2 (10.0.1.2)  6.420 ms  6.321 ms  6.274 ms
16  10.0.2.1 (10.0.2.1)  6.029 ms  6.002 ms *
17  10.0.0.2 (10.0.0.2)  5.888 ms  5.868 ms *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * 10.0.2.1 (10.0.2.1)  5.584 ms
23  10.0.0.2 (10.0.0.2)  6.357 ms  6.278 ms  9.258 ms
24  10.0.1.2 (10.0.1.2)  9.370 ms  9.360 ms  9.279 ms
25  10.0.2.1 (10.0.2.1)  8.990 ms  8.907 ms  9.102 ms
26  10.0.0.2 (10.0.0.2)  9.023 ms  8.943 ms *
27  10.0.1.2 (10.0.1.2)  10.196 ms  10.595 ms *
28  10.0.2.1 (10.0.2.1)  4.215 ms * *
29  * * *
30  * * *
lubuntu@lubuntu-VirtualBox:~$
```

Der Vorteil gegenüber der Analyse von Datenaufzeichnungen ist die zeitliche Unmittelbarkeit der vorliegenden Ergebnisse. Weiterhin ist man relativ frei in der Auswahl der zu prüfenden Knoten. Es können auch entfernte Schleifen gefunden werden, im Gegensatz zur Analyse von Paketaufzeichnungen. Jedoch spielt der Startpunkt der Traceroutes eine Rolle, da der Pfad zum Ziel vom Eintrittspunkt beeinflusst wird. Der zu prüfende Pfad kann nur indirekt beeinflusst werden, indem Eintrittspunkt und/oder Zieladresse geändert werden. Demgegenüber kann jeweils nur ein kleiner zeitlicher Ausschnitt betrachtet werden, während eine Datenaufzeichnung ein Abbild eines lückenlosen Zeitraums bietet.

3.3 TEMPORÄRE SCHLEIFEN (TRANSIENT LOOPS)[1]

Temporäre Schleifen können durch eine Änderung in der Netztopologie entstehen. Kommt es zu einer Änderung, benötigen Routing-Protokolle eine Konvergenzzeit, um die neuen

Informationen durch das komplette Netzwerk zu propagieren. Die Dauer ist abhängig von der Topologie des Netzwerks und des verwendeten Routing-Protokolls.

Es gibt Ansätze für OSPF [65] [60], um temporäre Schleifen in der Konvergenzzeit zu verhindern. Dadurch können zwar temporäre Schleifen verhindert werden, jedoch entstehen Paketverluste bis der Link-Ausfall (im Fehlerfall und nicht bei geplanter Wartung) erkannt wird. Der Expiration Timer bei RIP ist normalerweise 180 Sek. und bei OSPF wird nach 40 Sek. das Ausbleiben einer HELLO-Nachricht als Linkausfall erkannt. Um diese Zeiten möglichst gering zu halten und die Paketverluste zu minimieren, kann BFD von allen Routing-Protokollen genutzt werden. Das Ziel von BFD ist die Erbringung eines schnellen Fehlererkennungsmechanismus zwischen benachbarten Weiterleitungsknoten für deren Interfaces, den Datenlinks und den eigentlichen „Forwarding Engines“. Eine BFD-Session wird vom zu nutzenden Protokoll initiiert [65]. [81] beschreibt wie die Interaktion mit RIP, OSPF, IS-IS und BGP abzulaufen hat. Nach der Erkennung des Link-Ausfalls beeinflussen weitere Faktoren die Konvergenzzeit. In BGP können Timer des Route Flap Dampenings die Propagierung von Routing-Informationen verzögern, um unnötige Updates zu vermeiden. Die Größe des Netzwerks bzw. der Durchmesser beeinflusst die Zeit, die benötigt wird, um das komplette Netz zu fluten. Schließlich beeinflusst die Implementierung des Algorithmus zur Berechnung des besten Pfads ebenso die Konvergenzzeit, wie die Leistungsfähigkeit der zur Verfügung stehenden Hardware [61].

Die Konvergenzzeit eines Protokolls summiert sich aus folgenden Komponenten [65] [60]:

a. **Detection**: Die Zeit, die ein Router benötigt, um den Fehler zu erkennen. Während dieser Zeit gehen alle Pakete verloren.
b. **Propagation And Trigger**: Die Zeitspanne die vergeht, um eine neue Kontrollnachricht zu erstellen und abzusenden inkl. der Übertragunszeit auf dem Medium.
c. **Computation**: Die Zeitspanne die benötigt wird, um die neue Information in den eigenen Datenbestand einzufügen und ggf. neue Pfade zu berechnen (RIB- & FIB-Update)

Abhängig von der Anzahl der Zwischenstationen, um ein gesamtes Netz fluten zu können, summieren sich (b) und (c). OSPF und IS-IS sind ein Sonderfall, denn sie leiten die LSPs direkt weiter und berechnen danach erst die kürzesten Pfade. In RIP und BGP werden erst die

Berechnungen durchgeführt und dann Routing-Updates versendet. BGP besitzt den Sonderfall, dass Filter und Richtlinien (Policies) für ein- und ausgehende Updates zusätzlich angewendet werden müssen.

Eine Messung und Untersuchung der Konvergenzzeiten bei BGP ergibt, dass diese oftmals unterschätzt werden und unter bestimmten Konstellationen von 3 bis hin zu 15 Minuten dauern können. Die theoretische Obergrenze von *O(n!)* (*n*: Anzahl der AS) wird jedoch selten erreicht [67]. Generell sind temporäre Schleifen durch ihre begrenzte Dauer schwieriger zu erfassen.

[80] ordnet Schleifen, die während des Traceroute (Maximum von 32 Hops) aufgelöst werden können, als temporär ein. Es gibt keine festgelegten zeitlichen Bedingungen, die eine Eingliederung in temporär oder persistent erlauben. Eine temporäre Schleife wird durch Protokolle automatisch aufgelöst im Gegensatz zu persistenten Schleifen. Zum Auffinden von temporären Schleifen eignet sich am besten die Analyse von Paketaufzeichnungen. In [61] und [77] werden daher konsequent Paketaufzeichnungen als Datengrundlage verwendet.

Es existiert ein wichtiger Unterschied zwischen der Schleifenentstehung in EGPs und IGPs. Während temporäre Schleifen in EGPs aufgrund von Ereignissen außerhalb des eigenen AS entstehen, entstehen Schleifen in IGPs durch Ereignisse im eigenen AS. Aus diesem Grund ist es schwieriger, den Einfluss von durch EGP verursachten Schleifen zu kontrollieren, da es keine direkten Einflussmöglichkeiten eines Administrators für andere AS gibt [61].

3.3.1 BEISPIELE[1]

Ein grundsätzliches Beispiel in Abbildung 40 soll die Entstehung einer temporären Schleife verdeutlichen [77]: Router *R2* und *R3* senden Daten über *R1* im Anfangszustand an andere Netze (Abbildung 40 (a)). Der Upload-Link an *R1* fällt nun aus und *R1* bemerkt als erster den Ausfall (Abbildung 40 (b)). Verkehr zu anderen Netzen sollte nun über den Upload-Link von *R2* fließen (Abbildung 40 (d)). *R3* wird über den Link-Ausfall an *R1* informiert und sendet Verkehr für andere Netze an *R2* (Abbildung 40 (c)). Da *R2* noch keine Kenntnis über den Ausfall besitzt, leitet *R2* den Verkehr an *R1* (Abbildung 40 (c)). *R1* sendet den Verkehr zurück an *R2*. Solange *R2* noch keine Kenntnis über den Ausfall hat, bleibt diese temporäre Schleife bestehen [77].

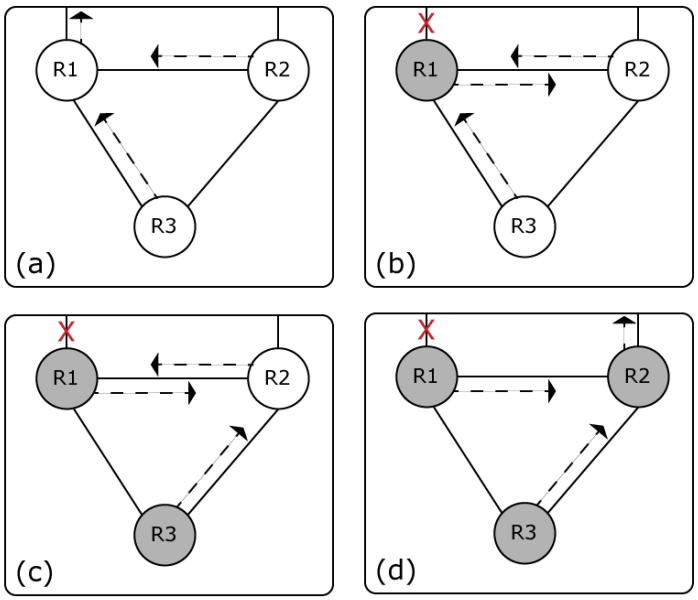


Abbildung 40: Entstehung einer temporären Schleife [77]

Im Fall von BGP können temporäre Schleifen aus folgenden Gründen entstehen:

- Eine oder mehrere Route(n), die an andere Peers propagiert wurde, wird zurückgezogen (Withdrawn) und alle Nachbarn müssen alternative Routen berechnen und installieren.
- Ein Link einer BGP-Session fällt aus, was zur Folge hat, dass alle über diese Session propagierten Präfixe zurückgezogen (Withdrawn) werden müssen.
- Eine bessere Route für ein Präfix wird propagiert, d.h. die alte Route wird zurückgezogen und die neue installiert.

Eine Möglichkeit der Schleifenentstehung in BGP soll an einem weiteren Beispiel [77] betrachtet werden (Abbildung 41): Ein Kunde ist mit zwei Providern AS *A* und AS *B* verbunden (Multihomed). Die Elemente *Y, X, U, V* sind als eine Sammlung von Routern mit einer geographischen Nähe (Point Of Presence) anzusehen und stellen keinen einzelnen Router dar. Eine bevorzugte Route ist über AS *A*, d.h. über *X* hereinkommender Traffic wird über *Y* an den Kunden geleitet. Wird nun die BGP-Policy vom Kunden geändert, dass die bevorzugte Verbindung über AS *B* läuft, erfahren die Border-Router in *Y* als erstes von dieser Änderung. Bis die Änderung an die Border-Router in *X* eintreffen, wird Verkehr für den

Kunden von *X* zu *Y* geleitet und von *Y* zurück zu *X*. Eine temporäre Schleife entsteht, bis BGP (in diesem Fall IBGP) konvergiert.

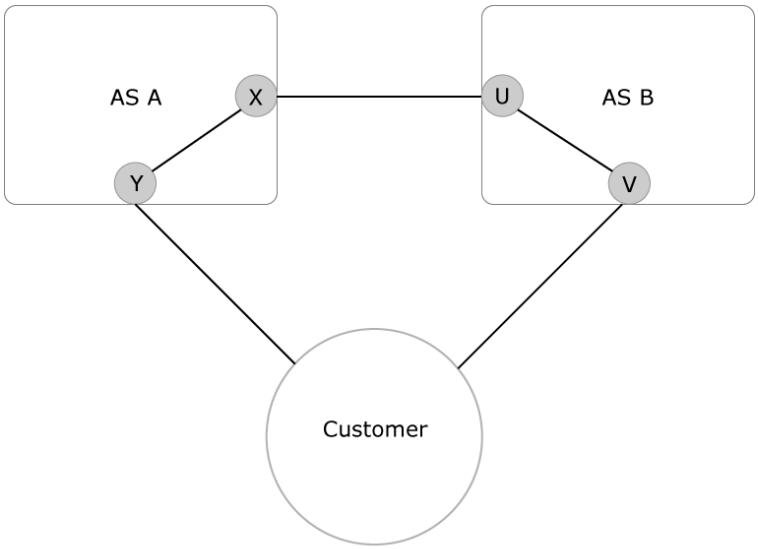


Abbildung 41: Beispiel für die Entstehung einer temporären Schleife in BGP [77]

3.3.2 TEMPORÄRE SCHLEIFEN IM INTERNET[1]

In diesem Kapitel untersuchen wir alle uns vorliegenden Studien, die temporäre Schleifen im Internet messen. Das Ziel dabei ist, Gemeinsamkeiten zu bestimmen und Trends herauszuarbeiten.

3.3.2.1 VERTEILTE TRACEROUTES ZUR MESSUNG VON TEMPORÄREN SCHLEIFEN (1996)[1]

Als einer der ersten analysiert Paxson im Jahre 1996 Netzwerkanomalien und damit auch Schleifen in [64]. Paxson untersucht ca. 40.000 Ende-zu-Ende-Verbindungen, die wiederholende Traceroutes zwischen 37 Internetsites generieren, die weltweit verteilt waren. Sein Hauptaugenmerk liegt auf Netzwerkfehler (Pathologies) im Allgemeinen, was den Einsatz von Traceroute erklären mag. Seine Ergebnisse zu persistenten Schleifen und die genaue Untersuchungsmethodik werden in Kapitel 3.4.3.1 näher ausgeführt. Auch weil Traceroutes kein geeignetes Mittel darstellen, um quantitative und qualitative Angaben über

temporäre Schleifen zu erstellen. Paxson deklarierte Schleifen, die sich innerhalb eines Traceroutes auflösen, als temporär. Auch wenn man seine Messmethodik im Zusammenhang mit temporären Schleifen anzweifeln darf, verweist er auf einen interessanten Effekt. In seinen beiden Messungen *D1* und *D2* entdeckt er 2 bzw. 23 temporäre Schleifen. Paxson verweist auf ein interessantes Detail, dass mit temporären Schleifen einhergeht: Paxson nennt dies den „Ripple Effect“: Ein Ausfall wird wie eine auslaufende Welle durch das gesamte Netz propagiert und zieht weitere temporäre Fehler mit sich. In einem Traceroute *rain->inria* wird ein Ausfall von 40 Sekunden beobachtet; gefolgt von einer Schleife von fünf MCINET Routern bei Washington D.C.; gefolgt von einem Verbindungsabbruch zurück zu *inria* und einer nachfolgenden Verbindungswiederaufnahmen. Interessant ist hierbei zu sehen, dass eine Schleife in Washington einen Verbindungsabbruch zwischen Portland und Seattle zur Folge hat. Alle beobachteten temporären Schleifen (bis auf eine Ausnahme) treten innerhalb eines AS auf. Die Studie in [77] unterstützt diese klare Tendenz, nimmt aber keine Unterscheidung zwischen temporär und persistent vor (siehe Kapitel 3.2.1.2).

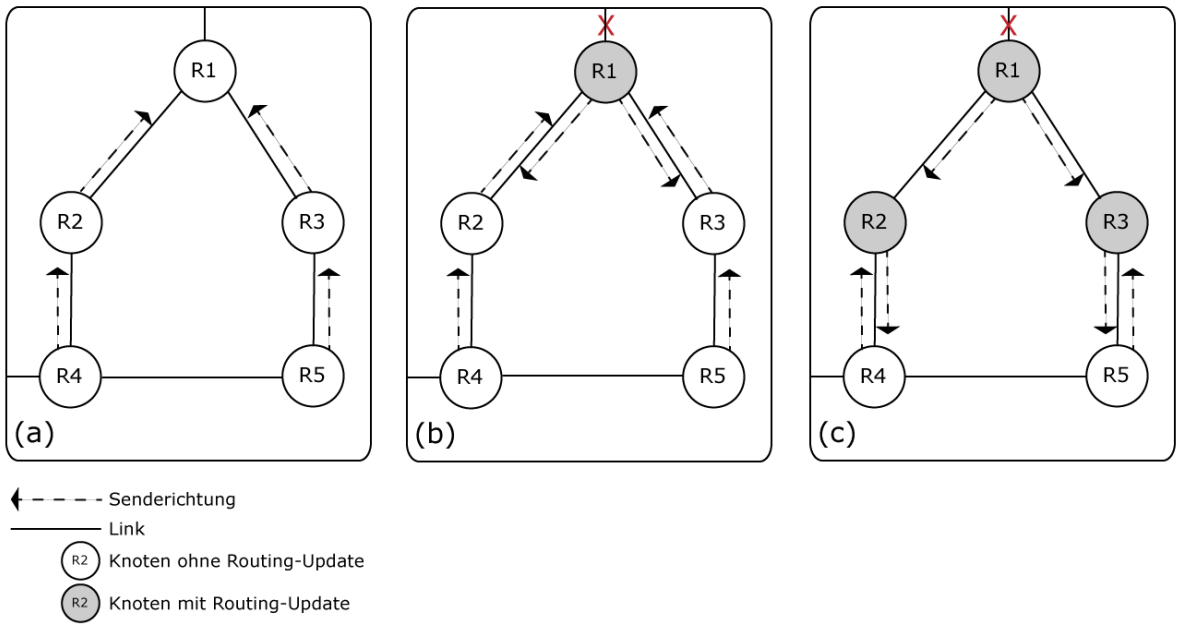


Abbildung 42: Beispiel Ripple Effect

In Abbildung 42 senden die Knoten *R2*, *R3*, *R4* und *R5* Verkehr nach außen über Knoten *R1* (Abbildung 42 (a)). Alternativ kann *R4* genutzt werden, um Verkehr nach außen zu leiten. Fällt die Verbindung an *R1* aus, entstehen unmittelbar zwei temporäre Schleifen zwischen *R2*<->*R1* und *R3*<->*R1* (Abbildung 42 (b)). In Abbildung 42 (c) erfahren *R2* und *R3* vom

Ausfall des Links an *R1*. Da *R4* und *R5* noch keine Kenntnis vom Ausfall besitzen, entstehen zwei weitere temporäre Schleifen zwischen *R2*<->*R4* und *R3*<->*R5*. Dies ist ein Beispiel für den von Paxson beobachteten Ripple Effect. Temporäre Schleifen können sich in der Konvergenzzeit wie eine auslaufende Welle durch das gesamte Netz ziehen.

3.3.2.2 Eine Messung von temporären Schleifen mittels Paketaufzeichnungen (2002) [1]

Eine genauere Untersuchung von temporären Schleifen wird in [61] im Jahr 2002 unternommen. Hier werden Paketaufzeichnungen als Grundlage der Analyse genommen. Diese erlauben eine lückenlose Analyse im Zeitraum der Aufzeichnung, können allerdings nur Schleifen erkennen, an denen der aufzeichnende Knoten direkt teilnimmt. Als Datengrundlage dienen mehrere Aufzeichnungen aus dem Tier-1 Internet Backbone von Sprint. Es wird parallel der Datenverkehr von mehreren uni-direktionalen OC-12 Links an der Ostküste der USA erfasst. Jeder Link verbindet zwei AS, die unter verschiedener Kontrolle liegen. Die ersten zwei Aufzeichnungen starten am 08.11.2001 um 13:00 Uhr, zwei weitere am 03.02.2002 um 20 Uhr.

Trace	Length (hours)	Avg BW (Mbps)	Packets Total (10^6)	Looped Packets
Backbone 1	24	1	50	2 419 792
Backbone 2	7.5	243	1 677	1 987 309
Backbone 3	11	2.2	20	337 570
Backbone 4	11	107	1 350	364 230

Abbildung 43: Details der Paketaufzeichnungen (2002) [61]

Trace	Replica Streams	Routing Loops
Backbone 1	79257	852
Backbone 2	70144	413
Backbone 3	7377	1485
Backbone 4	11997	1568

Abbildung 44: Anzahl der gefundenen Schleifen (2002) [61]

In Abbildung 43 sind die Länge der Aufzeichnung, die durchschnittliche Bandbreite des Links, die Anzahl der aufgezeichneten Pakete und die Anzahl der Pakete, die durch Schleifen laufen, zu erkennen. Der Anteil an Paketen die durch Schleifen laufen beträgt 4,84% (Backbone 1), 0,11% (Backbone 2), 1,68 % (Backbone 3) und 0,03% (Backbone 4). Das TTL-Delta zweier aufeinanderfolgender identischer Pakete gibt die Größe der Schleife an. Abbildung 44 ist die Gesamtanzahl der gefundenen Schleifen zu entnehmen.

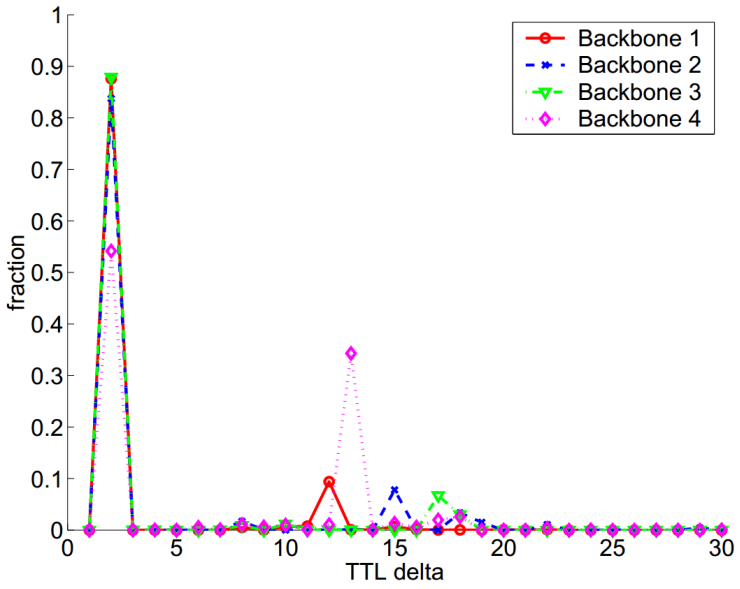


Abbildung 45: Verteilung der TTL-Deltas (2002) [61]

Es ist deutlich zu erkennen, dass der überwiegende Teil der entdeckten Schleifen in Backbone 1,2 und 3 lediglich zwei Knoten involvieren und 5-10% der Schleifen werden aus 12-18 Knoten geformt (Abbildung 45). In Backbone 4 bestehen 55% der Schleifen aus 2 Hops und 35% aus 13 Hops (Abbildung 45). Ein TTL-Delta von 2 ist am verbreitesten, da nur zwei benachbarte Router einen inkonsistenten Zustand aufweisen müssen. Im Fall von IBGP können zwei Router zwar benachbart, aber physisch über mehrere Hops verbunden sein, was Einfluss auf die Geschwindigkeit der Verteilung der Routing-Updates hat. Routing-Updates kommen an verschiedenen Punkten des Netzes zeitlich verteilt und nicht gleichzeitig an, was die Bildung von Schleifen mit mehr als zwei involvierten Hops erklärt.

Die Anzahl der replizierten Pakete pro Schleife in Abbildung 46 erfährt zwei Sprünge bei 30 und 60 Paketen. Dieser Sachverhalt ist durch den TTL-Standardwert zu erklären, der in Linux bei 64 und in Windows bei 128 liegt. Da Schleifen mit zwei Hops vorherrschend sind, erzeugt

ein Anfangswert von 64 ungefähr 30 replizierte Pakete. In Backbone 1 und 2 liegt die Zwischenankunftszeit zweier identischer Pakete zu 90% bei weniger als 80ms und zu 100% unter 150ms, in Backbone 3 und 4 zu 55% unter 10ms und zu 35% unter 150ms (Abbildung 46). Die Zwischenankunftszeit erhöht sich mit der Größe der Schleife. Da in Backbone 3 und 4 vermehrt Schleifen mit einem TTL-Delta>2 auftreten, erklärt dies die erhöhte Zwischenankunftszeit.

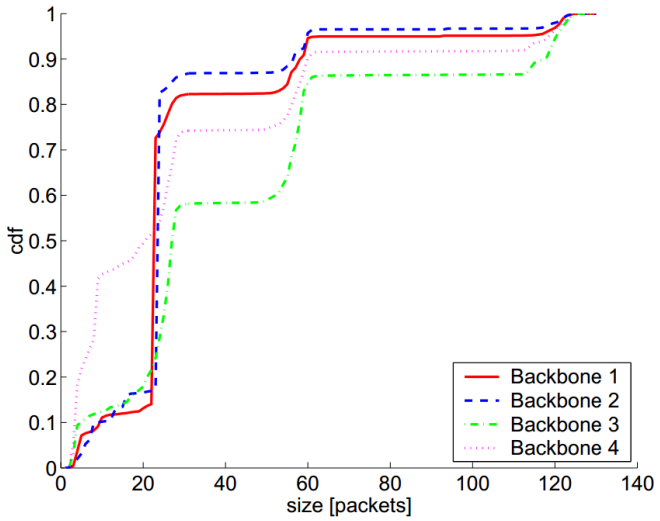


Abbildung 46: Kumulative Verteilungsfunktion der replizierten Pakete pro Schleife (2002) [61]

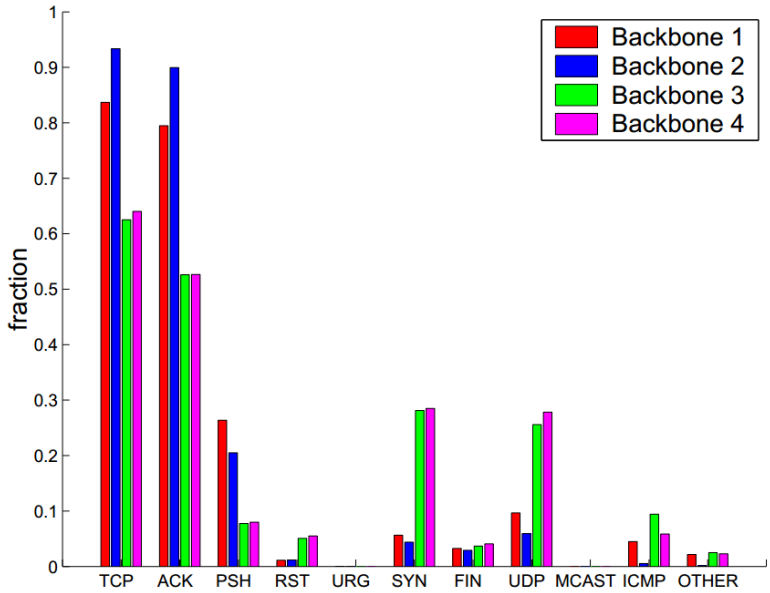


Abbildung 47: Verteilung des gesamten Datenverkehrs nach Typ (2002) [61]

Der aufgezeichnete Datenverkehr besteht zu 80% aus TCP-Paketen und zu 5-15% aus UDP-Paketen (Abbildung 47). Hierbei ist zu beachten, dass z.B. ein TCP SYN ACK mehreren Kategorien (TCP, ACK, SYN) zugeordnet wird.

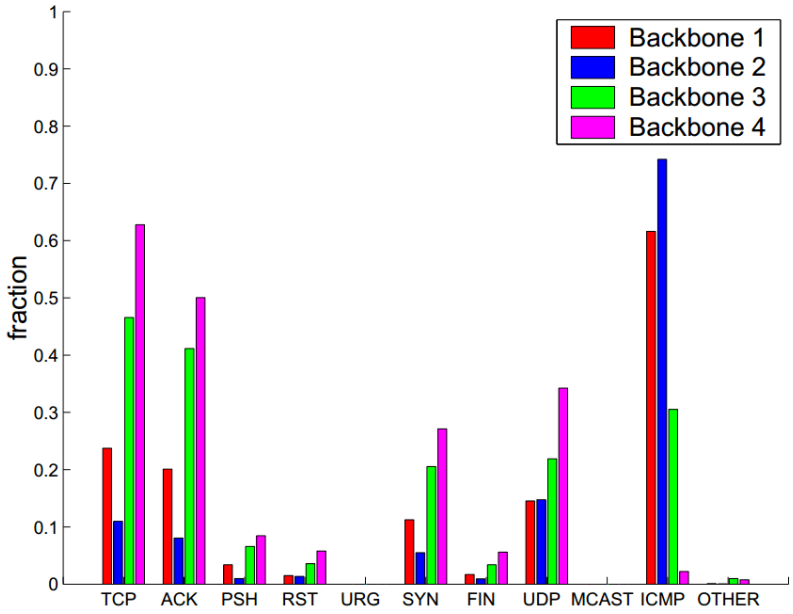


Abbildung 48: Verteilung des gesamten Datenverkehrs nach Typ in den Schleifen (2002) [61]

Auffällig ist der prozentual höhere Anteil an SYN-Paketen und der niedrigere Anteil an TCP-Paketen in Schleifen (Abbildung 48). Da durch die Schleife keine TCP-Verbindung erfolgreich aufgebaut werden kann, können keine weiteren TCP-Pakete in der Verbindung folgen bzw. TCP reagiert sofort auf verlorengegangene Pakete. Bei UDP existiert kein Verbindungsaufbau, sodass der Anteil an UDP-Traffic konstant bleibt (Abbildung 48). Auffällig ist weiterhin der hohe Anteil an ICMP-Paketen in Backbone 1, 2 und 3. Router, die ein Paket aufgrund der abgelaufenen TTL verwerfen, erzeugen und versenden ein ICMP-Paket (Typ 11). Weiterhin vermuten die Autoren, dass die Quellen die Verbindung anhand von Pings und Traceroutes prüfen, wenn sie Paketverluste feststellen.

Die Adressen von 192.0.0.0 bis 223.255.255.255 sind am Häufigsten durch Einflüsse von Schleifen betroffen, was nach Ansicht der Autoren an der höheren Auslastung dieses Adressbereichs zurückzuführen ist und/oder auf eine höhere Ereignisdynamik der Links (Abbildung 49).

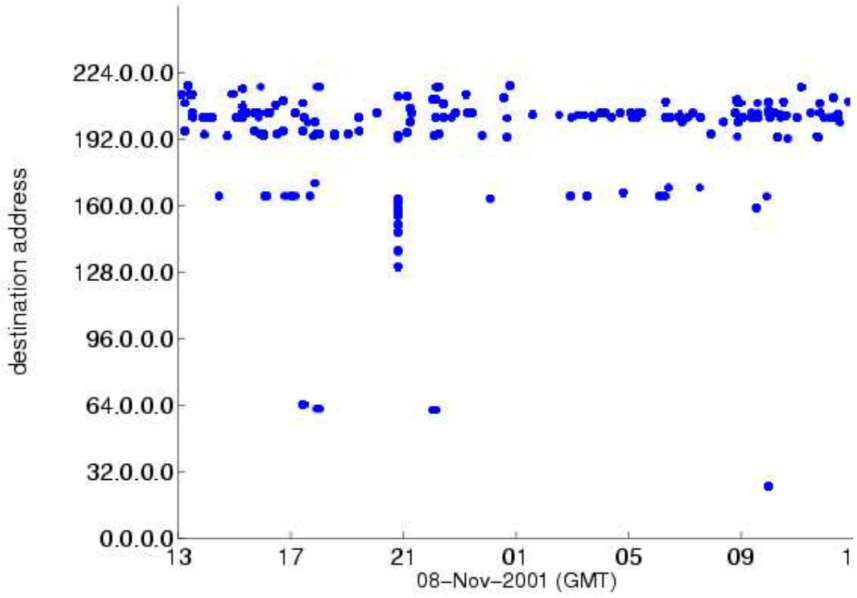


Abbildung 49: Zieladressen der replizierten Pakete bzw. Streams (2002) [61]

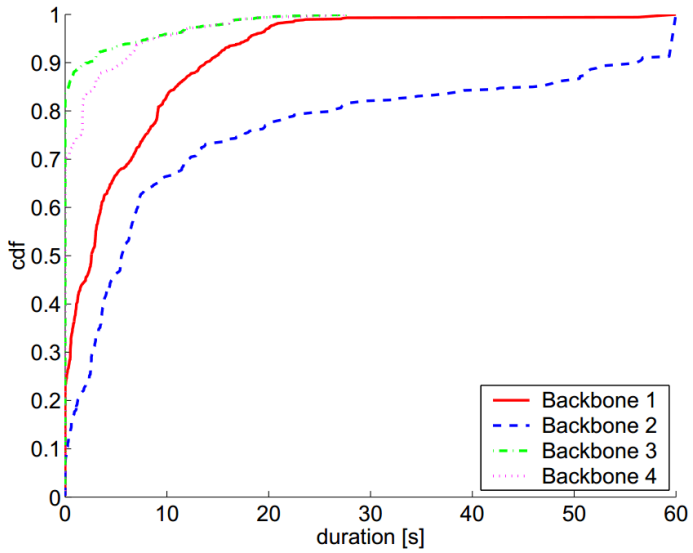


Abbildung 50: Kumulative Verteilungsfunktion der Dauer der Schleifen (2002) [61]

Die meisten temporären Schleifen existieren weniger als 10 Sekunden. Ungefähr 30% der temporären Schleifen existieren länger als 10 Sekunden (Abbildung 50).

Temporäre Schleifen haben einen erheblichen Performance-Einfluss und können zu einem Paketverlust von bis zu 90% führen, abhängig von der Auslastung des Links bzw. den Puffern in den Routern. Aufgrund der relativ kurzen Dauer der gemessenen temporären Schleifen

bleibt der Paketverlust im Allgemeinen niedriger. In jedem Fall führen temporäre Schleifen zu einer erheblichen Verzögerung bei Paketen, die vor Ablauf der TTL die temporäre Schleife verlassen können. Zwischen 0,6% und 11% der Pakete können die Schleife vor Ablauf der TTL verlassen und erfahren eine zusätzliche Verzögerung zwischen 25ms und 1300ms.

3.3.2.3 EINE MESSUNG VON TEMPORÄREN SCHLEIFEN MITTELS PLANETLAB (2004)[1]

Eine weitere Studie aus dem Jahr 2004 nutzt PlanetLab als physische Grundlage der Messung [80]. PlanetLab (Projektseite: http://www.planet-lab.org) besteht aus einer Vielzahl von weltweit verteilten Rechnern, die als verteilte Systeme der Forschung zur Verfügung stehen. PlanetLab wird 2002 von Prof. Larry L. Peterson gegründet und unterhält im Juni 2010 1090 Knoten an 570 weltweit verteilten Stellen. Zugänge sind auf Personen begrenzt, deren Unternehmen oder universitären Einrichtungen ebenfalls PlanetLab-Knoten unterhalten. Trotzdem sind die letzten Jahre ebenfalls freie Services wie CoDeeN (Coral Content Distribution Network, Projektseite: http://codeen.cs.princeton.edu) auf PlanetLab-Knoten entstanden [82].

CoDeeN wiederum hilft der 2004 durchgeführten Statistik durch enorme Traffic-Generierung. In [80] wird ein weltweit verteiltes Monitoring-System zur Auffindung und Auswertung von Anomalien (PlanetSeer) beschrieben. Das CoDeeN-Netzwerk wird eingesetzt, weil es täglich 7.000-12.000 Clients bedient und dabei 100-200 GB Traffic pro Tag generiert. PlanetSeer besteht im Kern aus zwei Komponenten: Einem passiven (MonD) und einem aktiven Monitoring Daemon (ProbeD). MonD läuft auf allen CoDeeN-Knoten und untersucht eingehenden wie ausgehenden Datenverkehr per TCPDump (Projektseite: http://www.tcpdump.org). TCPDump ist ein Kommandozeilentool für die Paketanalyse. Zwei Indikatoren helfen bei der Identifizierung von Anomalien:

1. Ändert sich die TTL in einem Stream von Paketen, deutet dies auf eine Pfadänderung.
2. Mehrere aufeinanderfolgende Timeouts signalisieren eine mögliche Anomalie. Eine TCP-Verbindung kann mehrere Male einen Timeout erleiden, wenn ein Datenpaket nicht mit einem ACK bestätigt wird.

Stellt MonD eine Anomalie fest, wird der lokale ProbeD angewiesen, den Umstand per Traceroute genauer zu untersuchen. ProbeD-Daemons laufen auf allen PlanetLab-Knoten (inkl. den CoDeeN-Knoten). Der lokale ProbeD-Daemon kontaktiert weitere ProbeD-Daemons (einen von jeder Gruppe), um eine verteilte Untersuchung zu organisieren. Die ProbeDs werden manuell ausgewählt und in Gruppen organisiert, sodass nicht alle in einer Untersuchung involviert sind und um den generierten Datenverkehr gering zu halten. Die geographische Verteilung ist Abbildung 51 zu entnehmen.

Category	Grps	Sites	Descriptions
US (edu)	11	70	US Universities
US (non-edu)	5	13	Intel, HP, NEC, etc.
Canada	2	11	Eastern & Western Canada
Europe	7	31	UK, France, Germany, etc.
Asia & MidE	4	14	China, Korea, Israel, etc.
Others	1	6	Australia, Brazil, etc.
Total	30	145	

Abbildung 51: Verteilung der teilnehmenden Knoten (ProbeDs) (2004) [80]

Um eine Anomalie als Routing-Schleife zu werten, muss die gleiche Sequenz in einem Traceroute sich mindestens drei Mal wiederholen. Mit dieser Bedingung können 21.565 Schleifen identifiziert werden. Eine Lockerung der Bedingung auf eine zweifache Sequenz führt zu 119.936 gefundenen Schleifen.

Wird eine Schleife innerhalb eines Traceroutes (32 Hops) nicht aufgelöst, wurde sie als persistent klassifiziert. 17% der Schleifen werden als temporär klassifiziert (Abbildung 52). Die kurze Dauer von temporären Schleifen und die daraus resultierende schwierigere Erkennbarkeit, relativiert diese Zahl etwas. Zumal nach Erkennung einer Anomalie Traceroutes für eine genauere Untersuchung eingesetzt werden. Aus diesem Grund können die Autoren auch keine genaueren Angaben zu der Dauer von temporären Schleifen machen. Paxson stellt in [64] fest, dass Schleifen eine örtliche Ballung aufweisen. Es ist sehr wahrscheinlich, dass ein Router mit inkonsistenten Daten, diese mit Nachbarn teilt und diese somit wiederum zur Schleifenbildung tendieren. In [80] wird ein ähnliches Phänomen entdeckt. Hierzu werden Schleifen in Traceroutes von anderen ProbeDs in der gleichen Periode gemessen. In Abbildung 52 werden 16% der temporären Schleifen von weiteren Schleifen begleitet.

	Temporary
Total	3565 (17%)
$\leq$ 30min	N/A
$\leq$ 1.5 hrs	N/A
$\leq$ 3.5 hrs	N/A
$\leq$ 7.5 hrs	N/A
$>$ 7.5 hrs	N/A
Single Loop	3008 (84%)
Multiple Loops	557 (16%)
1 AS	3021, (85%)
2 ASes	416, (12%)
3 ASes	106, (3%)
$\geq$4 ASes	22, (0%)
Tier-1 AS	510 (15%)
Tier-2 AS	859 (25%)
Tier-3 AS	1378(40%)
Tier-4 AS	197 (5%)
Tier-5 AS	538 (15%)
Total	3482

Abbildung 52: Zusammenfassung der PlanetSeer-Studie für temporäre Schleifen (2004) [80]

Über 85% der temporären Schleifen sind Intra-AS (Abbildung 52), d.h. erstrecken sich nicht über AS-Grenzen hinweg. Dieses Hauptmerkmal wird von allen in dieser Arbeit erwähnten Studien über Schleifen gestützt [64][61][77].

Ein Großteil aller temporären Schleifen entsteht in Tier-3 Netzwerken (Outer Core, Abbildung 52). 40% der temporären Schleifen werden hier gebildet, was auf eine häufige Änderung der Infrastruktur schließen lässt.

	2	3	4	5	6+
Temporary/All	51%	29%	11%	7%	2%
Temporary/Core	45%	31%	13%	8%	3%
Temporary/Edge	53%	27%	12%	6%	2%

Abbildung 53: Größe der gefundenen temporären Schleifen (2004) [80]

Ein deutliches Ergebnis ist, dass 51% der temporären Schleifen aus 2 Hops bestehen (Abbildung 53). [77] unterstützt diese These. Daraus folgt, dass eine Schleife mit steigender Größe an Stabilität verliert. Intuitiv ist dies dadurch zu erklären, dass nur ein Router der Schleife ein Routing-Update erfahren muss, um die Schleife aufzulösen.

Abbildung 53 bezeichnet Tier-1 bis Tier-3 Netzwerke als Core und Tier-4 bis Tier-5 Netzwerke als Edge. Hier wird ebenfalls ersichtlich, dass entstehende temporäre Schleifen im Kernnetzwerk dazu neigen, mehr Knoten zu involvieren.

Temporäre Schleifen haben einen Einfluss auf die Stabilität von Verbindungen auf mehrere Arten. Sie können die Puffer in Routern zum Überlaufen bringen, was Paketverluste initiiert, sie können den Verbindungsaufbau komplett verhindern, wenn sie vorher und ausreichend lange existieren und sie können eine bestehende Verbindung im Fall von verbindungsorientierten Protokollen (z.B. TCP) beenden.

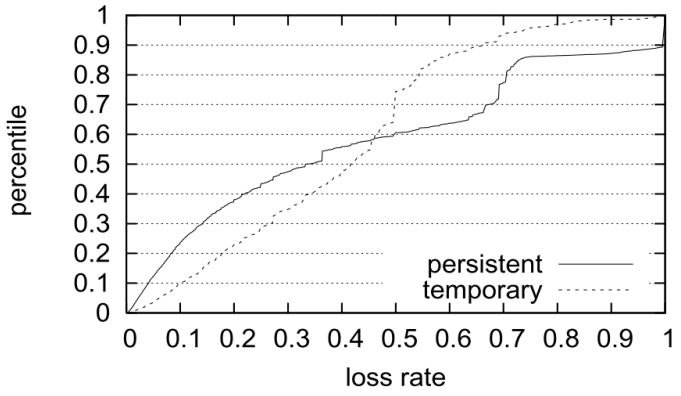


Abbildung 54: Kumulative Verteilungsfunktion der Rate der Paketverluste unmittelbar vor der Schleifenbildung (2004) [80]

Abbildung 54 zeigt, dass 65% der temporären Schleifen kurz vor der Bildung eine Fehlerrate von mehr als 30% implizieren. Die typische Fehlerrate im Internet beträgt weniger als 5%. Zu erklären ist dies durch die Änderungen in den Routing-Tabelle eines Knoten. Für ein Update muss ein Router einen alten Eintrag löschen und den neuen Eintrag installieren. Auch wenn dies zeitlich unmittelbar hintereinander passiert, können Pakete in dieser Zeitspanne verworfen werden, wenn die neue Route noch nicht installiert ist.

Schleifen beeinträchtigen in erster Linie die Performance, bevor es zu Verbindungsabbrüchen kommt. In Verbindung mit einem erhöhten Paketverlust sind Schleifen für einen rapiden Anstieg der RTTs einer Verbindung verantwortlich. In Abbildung 55 wird die Verteilung der RTTs unter normalen Bedingungen und unmittelbar vor Bildung einer Schleife verglichen. Es ist offensichtlich, dass Schleifen einen deutlichen Einfluss auf die RTT ausüben.

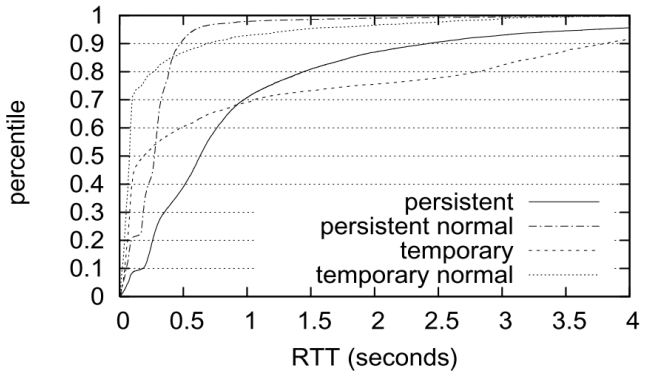


Abbildung 55: Kumulative Verteilungsfunktion der RTTs unmittelbar vor der Schleifenbildung und unter normalen Bedingungen (2004) [80]

3.4 PERSISTENTE SCHLEIFEN (PERSISTENT LOOPS)[1]

Ein einheitliches Kriterium für persistente Schleifen ist, dass diese einen menschlichen Eingriff erfordern und nicht durch Protokolle selbst aufgelöst werden können.

Es gibt für die Kategorisierung als persistente Schleife kein einheitliches zeitliches Kriterium. [64] und [80] deklarieren eine Schleife als persistent, sobald sie innerhalb eines Traceroutes nicht aufgelöst wurde. In [77] und [59] werden gefundene Schleifen, die länger als sechs Tage existieren, als persistent angesehen. Zur Überprüfung wurde in periodischen Abständen die Existenz der Schleife durch ein Traceroute überprüft. Die Zeitspanne von 1 Woche ist eher als willkürlich anzusehen, um den Unterschied zwischen temporären und persistenten Schleifen zu verdeutlichen und eine fehlerhafte Klassifizierung zu vermeiden.

Traceroutes stellen ein gutes Mittel für die Untersuchung persistenter Schleifen dar, da diese über einen längeren Zeitraum existieren. Es können komplette Pfade geprüft werden und man ist nicht auf den Knoten der Aufzeichnung beschränkt. Allerdings kann ein Pfad nur Schleifen für bestimmte Ziele beinhalten und der gleiche Pfad bei anderen Zielen schleifenfrei sein [59].

3.4.1 URSACHEN (AGGREGATION)[1]

Die integrierte Schleifenerkennung (AS_PATH) in BGP funktioniert zuverlässig. Route Aggregation gilt als eine der Hauptursachen für die Entstehung von persistenten Schleifen [59][62]. Als Route Aggregation bezeichnet man die Zusammenfassung einzelner Subnetze

z.B. 10.1.0.0/24 bis 10.1.15.0/24 zu einem generelleren Netz z.B. 10.1.0.0/20. Dieses generellere Netz wird an alle Nachbarn propagiert.

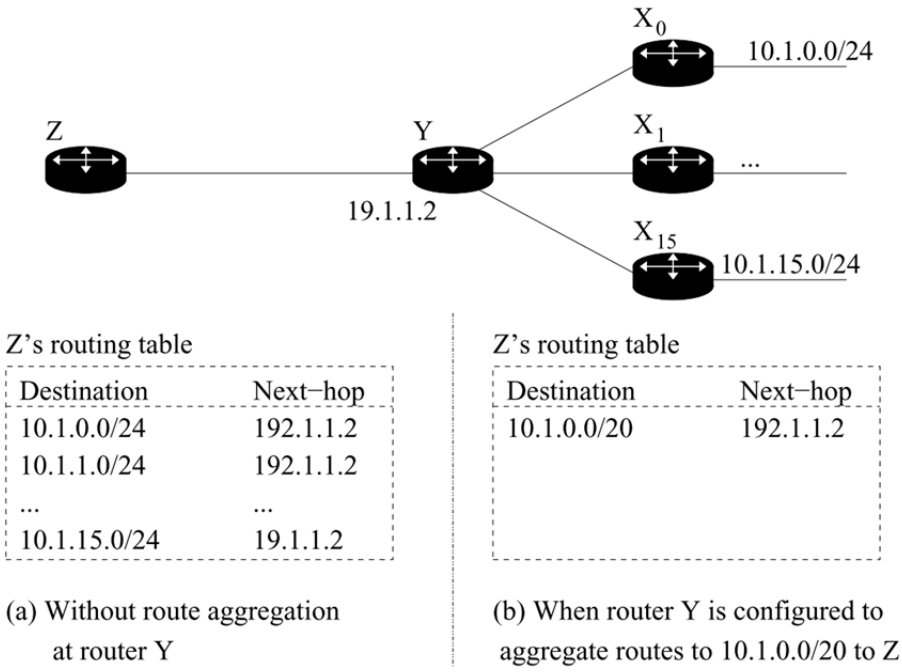


Abbildung 56: Route Aggregation [62]

Im Beispiel in Abbildung 56 würde die Routing Tabelle von Router *Z* ohne Route Aggregation jedes einzelne Präfix von 10.1.0.0/24 bis 10.1.15.0/24 beinhalten. Mit Route Aggregation würde nur das generellere Präfix 10.1.0.0/20 eingetragen. Wird ein Aggregat in einem Router gebildet (Router *Y*), werden alle spezifischeren Präfixe als Child Route vorgehalten. Das Aggregat wird nur propagiert, wenn mindestens eine Child Route vorhanden ist. Route Aggregation kann durch statische Routen oder dynamische Features des Routing-Protokolls ausgelöst werden. Alle Routing-Protokolle (BGP, RIP, OSPF usw.) unterstützen automatische Mechanismen, um Routen zu aggregieren. Manche Implementierungen aktivieren diese in der Standardeinstellung [62].

Es gibt mehrere Gründe, warum Routen aggregiert werden [62]:

- Route Aggregation wird im Zusammenhang mit CIDR benutzt, um die Routing-Tabellen wie im obigen Beispiel zu minimieren.
- Route Aggregation maskiert Route Flaps in den spezifischeren Netzen. Wenn im obigen Beispiel Router X_0 aufgrund eines Hardware-Fehlers beginnt sein Netz

kontinuierlich im Wechsel von funktionstüchtig zu funktionsuntüchtig zu propagieren, bleiben diese Flaps für Router *Z* aufgrund der Aggregation in *Y* verborgen.

- Letztlich kann Route Aggregation noch dazu benutzt werden, um netzwerkweite Designziele umzusetzen. Im Beispiel in Abbildung 57 hat das Enterprise Netzwerk zwei ISPs (ISP A und ISP B). Traffic aus dem Internet für 20.1.0.0/16 soll durch den Router *Seattle* fließen. Verkehr für 20.0.0.0/16 soll den Weg über Router *Pittsburgh* nehmen. Zusätzlich soll Verkehr aus dem Internet den jeweils anderen Pfad als Backup nehmen. Um dies zu erreichen, propagieren beide Router das generellere 20.0.0.0/15 Präfix und das jeweilige spezifischere Präfix (20.1.0.0/16 für Router *Seattle* oder 20.0.0.0/16 für Router *Pittsburgh*). Sind Netzwerke über mehrere Border Router mit den/dem Provider(n) verbunden, kann Route Aggregation genutzt werden, um Load Balancing zu erreichen. Auch hier wird wieder das generelle und spezifischere Präfix gleichzeitig propagiert. Weiterhin kann Route Aggregation die interne Netzstruktur maskieren.

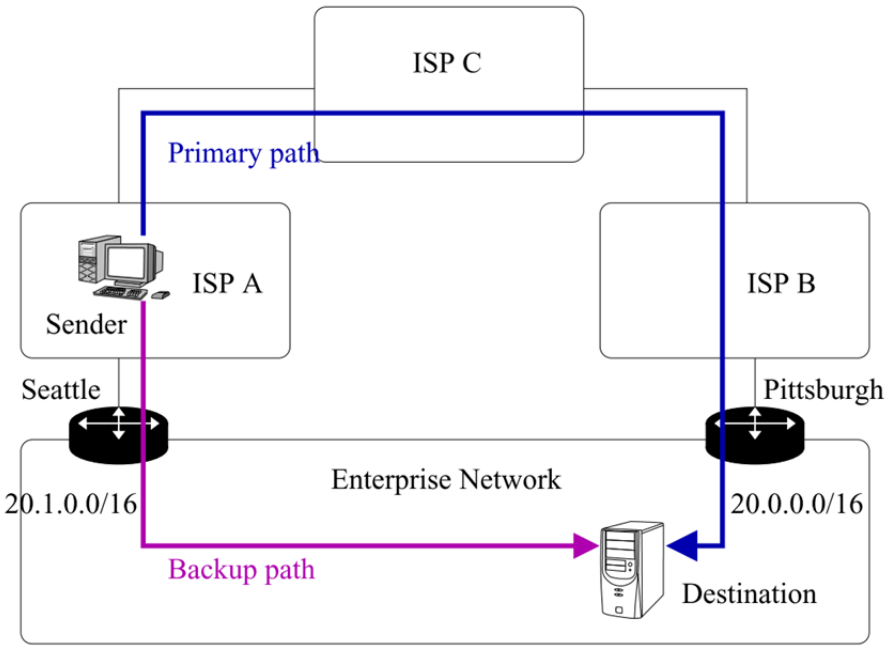


Abbildung 57: Backup-Pfad mittels Route Aggregation [62]

Das nächste Beispiel verdeutlicht die Entstehung einer persistenten Schleife.

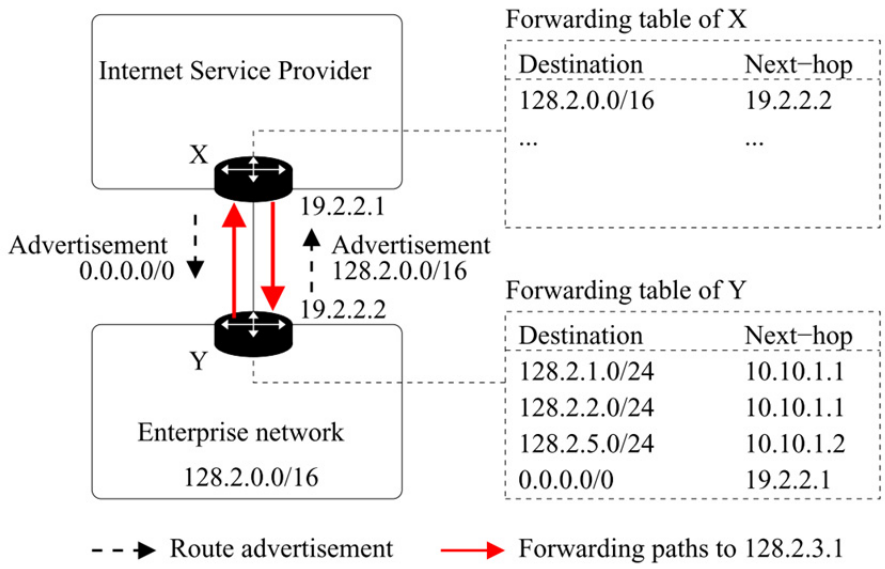


Abbildung 58: Persistente Schleife durch Aggregation [62]

Im Beispiel in Abbildung 58 aggregiert Router *Y* seine Teilnetze 128.2.1.0/24, 128.2.2.0/24 und 128.2.5.0/24. Das Aggregat 128.2.0.0/16 wird an Router *X* propagiert. Die Ursachen für die Entstehung einer Schleife liegen zum einen in der Default Route (kann statisch oder per Routing-Update installiert werden) von *Y* zu *X* und in der Tatsache, dass nicht alle Subnetze des propagierten Aggregats erreichbar sind. Sendet *X* Pakete für ein nicht vergebenes Subnetz z.B. 128.2.3.0/24 an Router *Y* , hat Router Y keinen Eintrag in seiner Routing-Tabelle für 128.2.3.0/24 und sendet den Verkehr über seine Default Route an *Y* zurück. Die Pakete traversieren die persistente Schleife bis zum Ablauf der TTL. Zwar beeinflusst die persistente Schleife Verkehr für vergebene und damit erreichbare Subnetze nicht direkt, das Traversieren der Pakete in der Schleife jedoch erhöht die Auslastung der Router *X* und *Y* und kann damit zu einem Stau führen [62].

Um obiges Szenario zu verhindern wird empfohlen, eine sogenannte Sink Route zu installieren. Eine Sink Route verwirft alle Pakete, die an ein nicht vergebenes Subnetz gerichtet sind. Im obigen Beispiel würde ein Eintrag 128.2.0.0/16 der auf ein Null Interface verweist, alle Pakete für nicht vergebene Subnetze verwerfen und damit die persistente Schleife verhindern [62].

Im vorigen Beispiel entsteht die Schleife nahe der Zieldomain und weist eine Länge von zwei Hops auf. Analog lässt sich dieses Szenario auf persistente Schleifen mit mehr als zwei Hops erweitern.

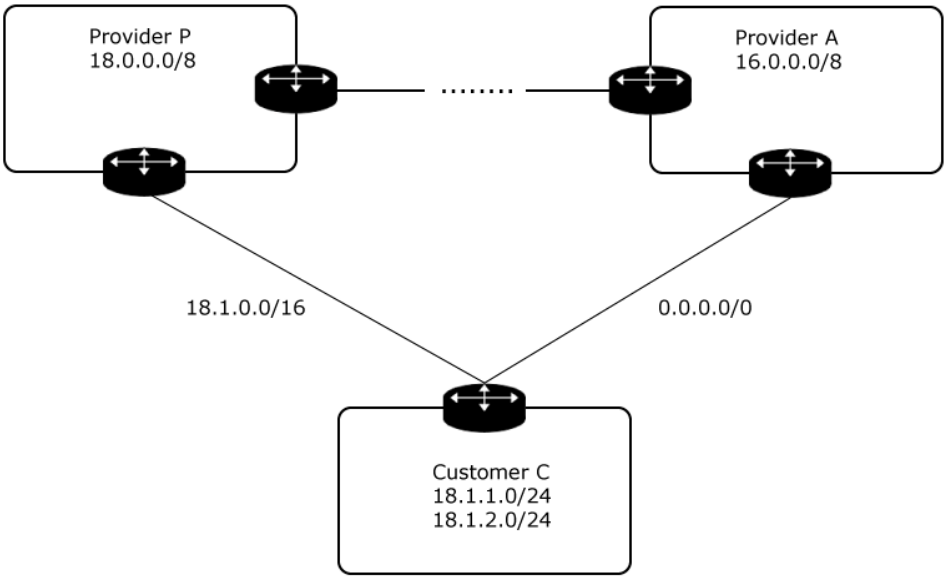


Abbildung 59: Persistente Schleife mit mehr als zwei Hops [59]

Im Beispiel in Abbildung 59 ist Customer *C* mit mehreren Providern verbunden. Eingehender Verkehr soll über Provider *P* gelangen, ausgehender über Provider *A* versendet werden. Customer *C* propagiert das aggregierte Präfix 18.1.0.0/16 an Provider *P*. Um ausgehenden Verkehr über Provider *A* zu leiten, installiert Customer *C* eine Default Route 0.0.0.0/0 über Provider *A*. Pakete für z.B. 18.1.3.0/24 werden von Provider *P* an Customer *C* gesendet. Da Customer *C* das Subnetz nicht unterhält und keine Sink Route installiert hat, werden die Pakete über die Default Route an Provider *A* gesendet. Provider *A* sendet die Pakete weiter, sodass sie evtl. über mehrere Hops wieder Provider *P* erreichen. Eine persistente Schleife mit mehr als zwei Hops ist entstanden [59].

In [59] werden Kriterien aufgestellt, um zu prüfen inwieweit Route Aggregation in Verbindung mit Default Routes und fehlenden Sink Routes für die Entstehung von persistenten Schleifen verantwortlich ist. Im Folgenden wird dieser Umstand allgemein als fehlerhafte Konfiguration beschrieben:

1. Ein Netzwerk propagiert ein aggregiertes Präfix ins Internet. Pfade zu manchen Subnetzen weisen persistente Schleifen auf, Pfade zu anderen Subnetzen weisen wiederum keine persistenten Schleifen auf.
2. Pfade zu Subnetzen aus einer bestimmten Quelle weisen den gleichen Pfad im Internet auf, divergieren aber innerhalb des Netzwerks.
3. Es befindet sich ein fehlerhaft konfigurierter Router auf dem Pfad. Manche Pfade zu Subnetzen weisen persistente Schleifen auf, manche nicht.

In [59] wird folgender Algorithmus auf alle gefundenen persistenten Schleifen angewendet (siehe Kapitel 3.4.3.3):

1. Finde das globale Präfix *p* in BGP Routing-Tabellen, welches ein Supernet zu einem gefundenen Präfix mit persistenter Schleife darstellt.
2. Finde alle Traces in D_A die zu Zieladressen von Präfix *p* führen und füge sie der Menge *T(p)* zu.
3. Zerlege T(p) in zwei Klassen $T_1(p)$ und $T_2(p)$
    a. $T_1(p)$ enthält eine persistente Schleife
    b. $T_2(p)$ enthält keine persistente Schleife
4. Existiert ein Trace t_1 aus $T_1(p)$ und ein Trace t_2 aus *T2(p)*, für die gilt
    a. Wenn t_1 und t_2 eine Interface Adresse *r* enthält und
    b. *r* zu der persistenten Schleife gehört und
    c. beide Pfade von der Quelle zum Ziel *r* in t_1 und t_2 identisch sind, dann KANN die persistente Schleife aufgrund fehlerhafter Konfiguration entstanden sein.

Es werden alle Präfixe, die in den Routing-Tabellen des Routeview-Projekts erscheinen mit diesem Algorithmus untersucht. Für 21% der persistenten Schleifen treffen alle Bedingungen zu [59].

Bedingung (4c) prüft, ob die Pfade identisch sind. Aufgrund von Paketverlusten, ICMP Rate Limiting oder Filtern können die Traces leichte Unterschiede aufweisen, obwohl die Pfade identisch sind.

Hop	Trace t_1 to 208.53.185.80 (contains a persistent forwarding loop)	Trace t_2 to 208.53.191.239 (does not contain any forwarding loop)
1	10.221.248.1	10.221.248.1
2	68.87.159.17	68.87.159.17
3	68.87.37.66	68.87.37.66
4	**68.87.144.221**	*
5	68.87.144.217	68.87.144.217
6	68.87.144.77	68.87.144.77
7	68.87.144.197	68.87.144.197
8	*	**68.87.145.9**
9	12.118.88.9	12.118.88.9
10	12.122.81.14	12.122.81.14
11	12.122.10.106	12.122.10.106
12	12.123.6.69	12.123.6.69
13	154.54.11.205	154.54.11.205
14	154.54.2.237	154.54.2.237
15	66.28.6.142	66.28.6.142
16	**38.112.4.98**	**38.112.0.234**
17	**66.90.127.254**	**66.90.127.254**
18	**66.90.127.253**	208.53.191.239
19	**66.90.127.254**	
20	**66.90.127.253**	
21	**66.90.127.254**	
...	...	...

Abbildung 60: Zwei Traces mit dem gleichen Weiterleitungspfad [43]

Im Traceroute zum Ziel 208.53.185.80 in Abbildung 60 wird von einem identischen Weiterleitungspfad ausgegangen. Aufgrund von Timeouts bei Hop 4 und Hop 8 weisen die Traces Unterschiede auf. In anderen Fällen unterscheiden sich zwei Traces nur in ein oder zwei Router Interfaces. Zum Beispiel kann aufgrund von Load Balancing Hop 16 im obigen Beispiel differieren. Aus diesen Gründen können eventuell manche persistente Schleifen aufgrund von fehlerhafter Konfiguration sich dem Algorithmus entziehen. Bedingung (4) aus dem Algorithmus wird deshalb in zwei Varianten aufgeweicht [59]:

- **Variante 1**: Die Idee hierbei ist, isolierte Unstimmigkeiten zu erlauben. Weisen zwei Traces einen Unterschied im Interface r_2 auf, so müssen die Interfaces unmittelbar vorher (r_1) und unmittelbar nachher (r_3) identisch sein. Mit dieser Bedingung konnten 39% der persistenten Schleifen einer fehlerhaften Konfiguration zugeordnet werden.
- **Variante 2**: Die Bedingung (4c) wird vollständig entfernt. Traces müssen nun nicht mehr bis zum Router Interface *r* identisch sein. Mit dieser Bedingung konnten 50% der persistenten Schleifen einer fehlerhaften Konfiguration zugeordnet werden.

Obige Aufweichungen könnten zu einer Überschatzung von persistenten Schleifen aufgrund fehlerhafter Konfiguration führen. Wichtig ist anzumerken, dass der Algorithmus notwendige aber nicht hinreichende Bedingungen formuliert. Wenn ein Netzwerk keine zusammenhängenden Subnetze verwendet und diese Präfixe einzeln propagiert werden, werden diese vom Algorithmus nicht gefunden, obwohl persistente Schleifen aufgrund von fehlerhafter Konfiguration trotzdem existieren können. Weiterhin kann ein Pfad nur Router Interfaces enthalten, die in keiner persistenten Schleife vorkommen. Aufgrund der Vergabe von Aliasen für Interfaces ist es dennoch möglich, dass ein Interface im Pfad in einer persistenten Schleife involviert ist [59].

3.4.2 URSACHEN (BGP MISCONFIGURATION)[1]

Eine falsche Konfiguration von BGP und Route Reflektoren kann zu einer persistenten Schleife führen. In [83] wird eine wichtige Design-Regel beim Einsatz von Route-Reflektoren beschrieben. Route-Reflektoren sollen entlang eines Weiterleitungspfades in Clustern platziert werden. Dies entspricht dem Design-Prinzip, dass IBGP-Sessions der physikalischen Topologie folgen sollen. Eine Verletzung dieser Regel in Verbindung mit der Tatsache, dass Route-Reflektoren nur die besten Routen an die Clients propagieren, resultiert in persistenten Schleifen.

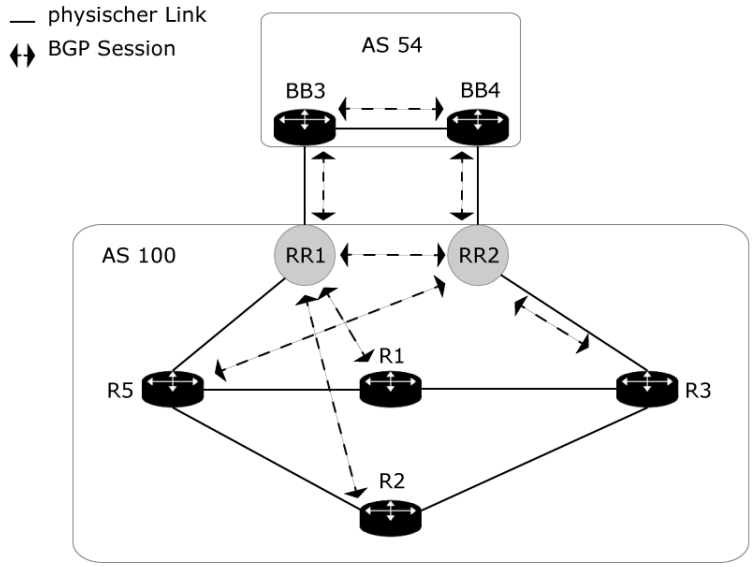


Abbildung 61: Persistente Schleife durch Fehlkonfiguration [83]

Router *BB3* im Beispiel in Abbildung 61 propagiert Präfixe aus AS54 an *RR1* (Route Reflector). Analog propagiert *BB4* identische Präfixe an *RR2*. Beide Route-Reflektoren tauschen die empfangenen Informationen aus und jeder bevorzugt seine direkte Verbindung zu AS54. *RR1* sendet die besten Pfade an *R1* und *R2*, analog sendet *RR2* die eigenen besten Pfade an *R5* und *R3*. Das bedeutet *R5* bevorzugt Pfade über *RR2*. Ein Pfad zu *RR2* läuft entweder über *R2* oder *R1*. *R1* und *R2* bevorzugen jedoch einen Pfad über *RR1*. Eine persistente Schleife ist entstanden (entweder *R5* <-> *R1* oder *R5* <->*R2*) [83].

3.4.3 PERSISTENTE SCHLEIFEN IM INTERNET[1]

Wie zuvor bei temporären Schleifen in Kapitel 3.3.2 untersuchen wir in diesem Kapitel alle uns vorliegenden Studien zu persistenten Schleifen und versuchen, Gemeinsamkeiten und Trends herauszuarbeiten.

3.4.3.1 VERTEILTE TRACEROUTES ZUR MESSUNG VON PERSISTENTEN SCHLEIFEN (1996)[1]

Paxson untersucht in [64] 40.000 generierte Traceroutes neben Netzwerkanomalien auch auf persistente Schleifen. Die Traceroutes werden zwischen 37 Internetknoten generiert. Hierzu wird bei allen Teilnehmern ein „Network Probe Daemon“ (NPD) installiert, die von einer entfernten Kontrollinstanz angewiesen werden, verschiedene Routen per Traceroute zu analysieren. Die Besonderheit lag damals in der weltweiten Verteilung der NPDs.
In einer ersten Messung D_1 wird jeder Pfad in einem Intervall von 1-2 Tagen geprüft. Eine weitere Messung D_2 simuliert zu 60 % „Bursts“ in einem Intervall von 2 Stunden und zu 40% Messungen in einer größeren zeitlichen Periode, um Langzeitbeobachtungen über die Stabilität zu erhalten. Die zeitliche Verteilung zweier aufeinanderfolgenden Messungen eines Pfads ist unabhängig und exponentiell verteilt.

Schleifen, die sich innerhalb eines Traceroutes nicht auflösen, werden als persistent deklariert. In D_1 werden 10 Traceroutes mit persistenten Schleifen entdeckt. In D_2 zeigen 50 Traceroutes persistente Schleifen.

Durch den Vergleich aufeinanderfolgender Messungen können Aussagen über eine untere bzw. obere zeitliche Grenze der Schleifen getroffen werden, je nachdem ob sie in einer späteren Messung weiterhin auftritt oder verschwindet.

Die geographische Verteilung der NPDs und die getesteten Verbidnungen werden aus Abbildung 62 ersichtlich.

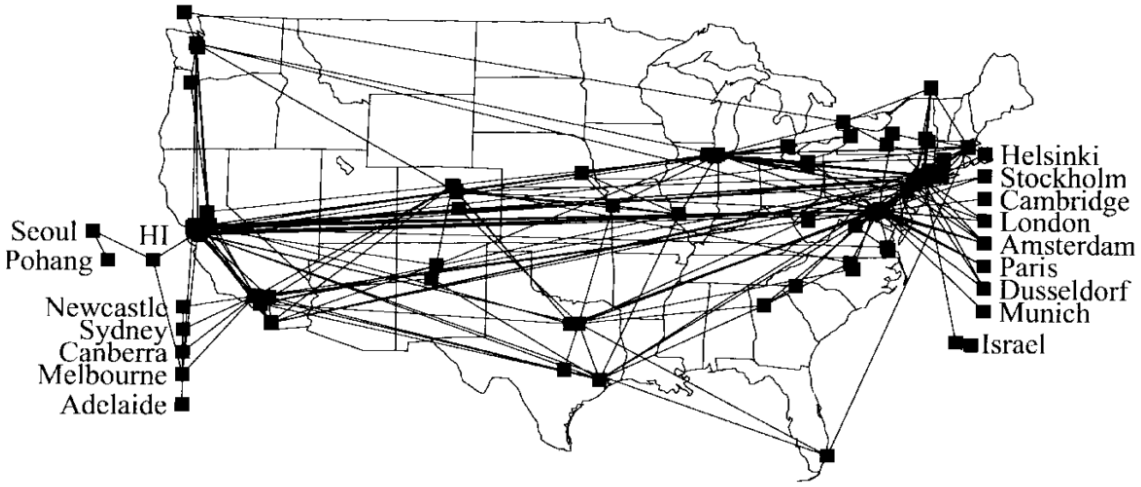


Abbildung 62: Verteilung der Verbindungen von Paxson (1996) [64]

Neben Quelle, Ziel, Datum, Ort und Dauer ist die Anzahl der aufeinanderfolgenden Traceroutes in Abbildung 63 angegeben, in denen die persistente Schleife auftritt. Die Dauer der persistenten Schleifen ist grob in zwei Kategorien teilbar: Unter 3 Stunden und größer als ein halber Tag. Lediglich die letzte Schleife *bsdi->sintef1* spannt sich über mehrere Städte und Kontinente.

Paxson beobachtet eine Tendenz zur regionalen und zeitlichen Ballung. Persistente Schleifen treten häufiger in und um Washington D.C. auf, was allerdings auf den hohen Grad der Vernetzung der ISPs in dieser Region zurückzuführen sein könnte. Weiterhin werden neben einer Schleife (*pubnix->sintef1*) zwei weitere Schleifen in Routern der näheren Umgebung gemessen. Paxson erklärt dies, indem zwei Router der näheren Umgebung eine inkonsistente Sicht der Topologie schneller austauschen können. In der Implikation würden gemessene persistente Schleifen weitere Schleifen in unmittelbarer Nähe erzeugen. Spätere Messungen zeigen, dass dies zu einem größeren Anteil auf temporäre Schleifen zutrifft.

Source	Dest.	Date	#	Location	Duration
inria	adv	Nov. 6	1	Washington	?
inria	near	Nov. 11	1	Washington	$\leq$ 3 hr
wustl	inria	Nov. 24	1	Washington	?
inria	pubnix	Nov. 12	1	Washington	?
inria	austr2	Nov. 15	1	Washington	?
sintef1	adv	Nov. 12	1	Washington	?
pubnix	sintef1	Nov. 8	1	Anaheim	?
ustutt	ucl	Nov. 11	16	Stuttgart	16–32 hr
connix	bsdi	Nov. 14	1	MAE-East	$\geq$ 10 hr
ustutt	austr	Nov. 14	1	same loop	
pubnix	sintef1	Nov. 14	1	Washington	$\leq$ 5.5 hr
austr	nrao	Nov. 15	1	College Park	?
many	oce	Nov. 23	12	Amsterdam	14–17 hr
ucol	ustutt	Nov. 24	1	San Francisco	?
ucol	inria	Nov. 27	1	Paris	$\leq$ 14 hr
mid	bsdi	Nov. 28	1	Washington	$\leq$ 3 hr
mid	austr	Dec. 6	1	Chicago	$\leq$ 3 hr
mit	wustl	Dec. 10	1	St. Louis	?
umann	nrao	Dec. 13	1	Heidelberg	?
ucl	mit	Dec. 14	1	Cambridge	$\leq$ 3 hr
near	ucla	Dec. 16	1	Los Angeles	?
sri	near	Dec. 17	1*	Palo Alto	?
near	sri	same	1*	San Francisco	?
bsdi	sintef1	Dec. 21	1	NJ, London	$\leq$ 10 hr

Abbildung 63: Verteilung der persistenten Schleifen in D2 von Paxson (1996) [64]

Alle beobachteten persistenten Schleifen (bis auf eine Ausnahme) treten innerhalb eines AS auf, was auf eine korrekt arbeitende Schleifenerkennung von BGP (AS_Path) schließen lässt.

3.4.3.2 Eine Messung von persistenten Schleifen mittels PlanetLab (2004)[1]

Eine PlanetLab-Studie aus dem Jahr 2004 misst ebenfalls persistente Schleifen und wertet die Ergebnisse aus [80]. Der genaue Aufbau der Studie ist in Kapitel 3.3.2.3 beschrieben. Weltweit verteilte Daemons werten den Datenverkehr per TCPDump aus. Entdeckte Anomalien werden mit Traceroutes nähergehend untersucht.

	Persistent
Total	18000 (83%)
$\leq$ 30min	54%
$\leq$ 1.5 hrs	11%
$\leq$ 3.5 hrs	6%
$\leq$ 7.5 hrs	6%
$>$ 7.5 hrs	23%
Single Loop	17007(94%)
Multiple Loops	993 (6%)
1 AS	17895, (99%)
2 ASes	101, (1%)
3 ASes	4, (0%)
$\geq$4 ASes	0, (0%)
Tier-1 AS	244 (2%)
Tier-2 AS	789 (6%)
Tier-3 AS	6263 (46%)
Tier-4 AS	3899 (29%)
Tier-5 AS	2401 (17%)
Total	13596

Abbildung 64: Zusammenfassung der PlanetSeer-Studie (2004) [80]

Wird eine Schleife innerhalb eines Traceroutes nicht aufgelöst, wurde sie als persistent klassifiziert. 83% der persistenten Schleifen werden innerhalb eines Traceroutes nicht aufgelöst und als persistent klassifiziert (Abbildung 64). Diese Zahl ist aufgrund der verwendeten Traceroutes jedoch mit Vorsicht zu genießen. Bei persistenten Schleifen konnte oftmals der genaue Zeitpunkt wegen eines Ausfalls oder Reboots des untersuchenden Knoten nicht genau festgestellt werden. Aus diesem Grund wird nur eine exakt ausgewertete Dauer miteinbezogen. Die meisten Schleifen (54%) werden entweder schnell wieder aufgelöst (kürzer als 30 Minuten) oder existieren sehr lange (23%, größer 7,5 Stunden) (Abbildung 64). Diese Tendenz deckt sich mit den vorher ausgeführten Ergebnissen von Paxson [64]. Die von Paxson [64] beobachtete Tendenz zur örtlichen und zeitlichen Ballung von persistenten Schleifen wird in dieser Messung [80] nicht bestätigt. In obiger Tabelle werden nur 6% der persistenten Schleifen von weiteren Schleifen begleitet, da persistente Schleifen hauptsächlich auf eine fehlerhafte Konfiguration zurückzuführen sind [59]. Über 99% der persistenten Schleifen sind Intra-AS, d.h. erstrecken sich nicht über AS-Grenzen hinweg (Abbildung 64). Diese These decken die bisher vorgestellten Messungen [64] und [77]. Um eine Art Qualitätsmerkmal von Schichten (Tiers) zu berechnen, werden die 10% der „schlechtesten“ AS eines Tiers untersucht. In Tier-1 sind zwei AS (10% von 22) für 35% der persistenten Schleifen verantwortlich, während in Tier-2 (10% von 215) 21 AS 97%

der Schleifen verursachen. Diese Zahlen lassen evtl. Rückschlüsse auf die Wichtigkeit der Kunden und des Datenverkehrs zu, die bedient werden. Allerdings sollte der Anteil am Gesamtverkehr noch berücksichtigt werden. In Tier-1 sind das 20% und in Tier-2 97% des Gesamtverkehrs, was darauf schließen lässt, dass diese AS extrem problematisch sein können.

	2	3	4	5	6+
Persistent/All	97%	2%	1%	0%	0%
Persistent/Core	94%	4%	1%	1%	0%
Persistent/Edge	97%	2%	1%	0%	0%

Abbildung 65: Größe der gefundenen Schleifen (2004) [80]

Ein deutliches Ergebnis ist, dass 97% der persistenten Schleifen aus 2 Hops bestehen (Abbildung 65). Tier-1 und Tier-2 Netzwerke erzeugen weniger persistente Schleifen, evtl. weil sie wichtiger und dementsprechend eine bessere „Versorgung“ erfahren. Ein Großteil (46%) der persistenten Schleifen entsteht in Tier-3 Netzwerken (Outer Core) (Abbildung 64). Die Abbildung 65 bezeichnet Tier-1 bis Tier-3 Netzwerke als Core und Tier-4 bis Tier-5 Netzwerke als Edge. Hieraus wird ebenfalls ersichtlich, dass persistente Schleifen im Kernnetzwerk (Core, Tier-1 bis Tier-3) dazu tendieren, mehr Knoten zu involvieren. Durch eine höhere Dauer der persistenten Schleifen ist es wahrscheinlicher, dass Puffer überlaufen, eine Verbindung nicht aufgebaut werden kann oder eine Verbindung beendet wird.

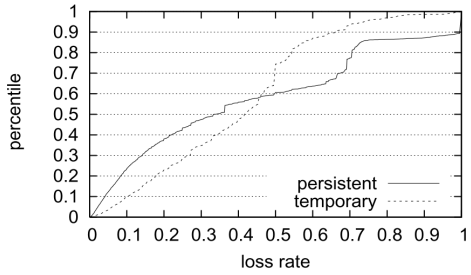


Abbildung 66: Kumulative Verteilungsfunktion der Rate der Paketverluste unmittelbar vor der Schleifenbildung (2004) [80]

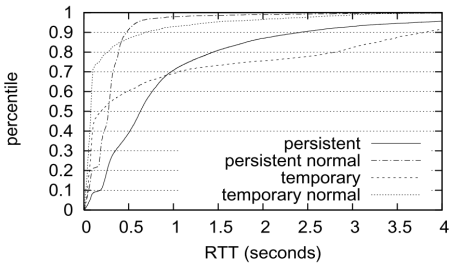


Abbildung 67: Kumulative Verteilungsfunktion der RTTs unmittelbar vor der Schleifenbildung und unter normalen Bedingungen (2004) [80]

55% der persistenten Schleifen implizieren kurz vor der Bildung eine Fehlerrate von mehr als 30% (Abbildung 66) und erhöhen die RTT einer Verbindung (Abbildung 67).

3.4.3.3 EINE MESSUNG VON PERSISTENTEN SCHLEIFEN IN ALLEN GEROUTETEN NETZEN DER KLASSE C (2007) [1]

In [59] wird versucht im Jahr 2007 persistente Schleifen mittels Traceroute aufzuspüren. Die Besonderheit der Studie liegt darin, dass versucht wird, alle IP-Adressen herauszufiltern, die eine persistente Schleife im Pfad beherbergen. Hierbei wird versucht, eine Menge an Zielpräfixen zu generieren, die möglichst klein ist. Im Gegensatz hierzu soll die Menge an Präfixen eine möglichst große Zahl an Pfaden überdecken, um ein repräsentatives Gesamtbild zu generieren. Viele IP-Adressen eines Netzes nehmen den gleichen Pfad zum Ziel. Deswegen wäre eine Prüfung aller IP-Adressen zu aufwendig und nicht zielführend. Der Adressraum eines großen Netzwerks kann dagegen aus mehreren Subnetzen aggregiert sein, die z.B. zu verschiedenen Kunden gehören und einen unterschiedliche Pfad implizieren. Die kleinste Einheit, um ein Netz in Subnetze zu unterteilen, besitzt eine Netzmaske mit 24 Bits (/24er-Netze). Aus diesem Grund werden IP-Adressen pro Subnetz geprüft. Um alle im Internet gerouteten Präfixe zu erhalten, werden die BGP-Tabellen des Routeviews-Projekts analysiert. Eine Analyse aller von der IANA vergebenen IP-Adressen würde nicht zum Ziel führen, da nicht alle vergebenen Adressen auch im Internet genutzt werden. Alle extrahierten Präfixe werden in alle möglichen Subnetze der Länge 24 zerlegt und als Fine-Grained Prefixes bezeichnet. Netze in militärischem Besitz oder Netze von Regierungen wurden vorher durch eine Analyse mittels WHOIS herausgefiltert. Aus diesen möglichen 254 Adressen werden die erste und eine zufällig ausgewählte und mittels Traceroute geprüft. Manche Traceroutes können ein „*“ oder „!“ innerhalb einer wiederkehrenden IP-Adresse enthalten, wenn z.B. ein Router nicht mit einem ICMP-Paket antwortet oder diese verloren gehen oder herausgefiltert werden. Andere Traceroutes enthalten eine IP-Adresse, die kontinuierlich wiederholt wird. Eine Firewall könnte eine mögliche Ursache sein. Diese Traceroutes werden herausgefiltert.

Zwei Messungen (D_A und D_B) werden im September 2005 von einem Kunden des Comcast High Speed Internet Service im westlichen Massachusetts aus gesammelt und weitere von verschiedenen globalen Standorten, um gefundene Thesen zu bestätigen.

Table 2
Summary on measurement design and trace data

Data set	Fine-grained prefix selection	# of selected IP addresses per prefix
D_A	All fine-grained prefixes	2
D_B	All candidate prefixes	2

Data set	# of traces for each selected address	# of prefixes traced	# of addresses traced
D_A	Once	5,499,618	10,999,236
D_B	10 times	207,891	415,782

Abbildung 68: Zusammenfassung der Messungen (2007) [59]

Im ersten Datensatz D_A (Abbildung 68) werden aus 0,18 Millionen im Internet gerouteten Präfixen ca. 5,5 Millionen Adressen zur Überprüfung generiert. Diese führen zu ca. 11 Millionen Traceroutes innerhalb von 16 Tagen. Präfixe mit einer Schleife im Pfad werden als Kandidaten gehandelt und weiter untersucht (Candidate Prefixes). Von 5,5 Millionen untersuchten Präfixen erwiesen sich 207.891 als Candidate Prefixes (3,77% aller im Internet gerouteten IP-Adressen). Ein Candidate Prefix wird in einer Zeitspanne von 6 Tagen 10 Mal per Traceroute untersucht. Pro Präfix werden wieder die erste IP-Adresse und eine zufällig zur Überprüfung ausgewählt. Insgesamt werden 4,2 Millionen Traceroutes zu 415.782 IP-Adressen in der zweiten Messung D_B (Abbildung 68) generiert. Persistente Schleifen werden in 135.973 Präfixen gefunden, was 2,47% aller im Internet gerouteten Adressen (ca. 35 Millionen Adressen) entspricht. Diese Präfixe sind auf 5341 AS verteilt, was ein weltweit verteiltes Problem suggeriert.

Da in den ersten beiden Messungen nur zwei IP-Adressen aus einem Subnetz geprüft werden, liefert eine dritte Messung D_C (Abbildung 69) eine Aussage über 50 zufällig im Subnetz verteilte IP-Adressen und damit eine deutlichere Tendenz. Hierzu werden 50 IP-Adressen aus 3705 Präfixen aus D_B vom Netzwerk der Universität von Massachusetts aus geprüft. Wie bisher werden wieder Traceroutes mit „*“ oder „!“ gefiltert, was zu insgesamt 3533 überprüften Präfixen führt.

Table 3
Persistent forwarding loops to additional sampled addresses

Category	# of shadowed prefixes	Percentage (%)
All sampled addresses have forwarding loops	**2401**	**68.96**
Partial sampled addresses have forwarding loops	**1083**	**30.65**
(a) Infrastructure addresses are sampled	791	–
(b) Unknown reason	292	–
None of sampled addresses has forwarding loops	**49**	**1.39**
Total	**3533**	**100**

Abbildung 69: Persistente Schleifen in D_C (2007) [59]

68,96% der untersuchten Präfixe bestätigen, dass alle 50 weiterhin untersuchten IP-Adressen eine persistente Schleife im Pfad haben. Die restlichen 30,65% werden auf Gründe untersucht, warum persistente Schleifen nicht in allen untersuchten IP-Adressen vorkommen. Ein Grund kann darin bestehen, dass Adressen geprüft werden, die zur Infrastruktur gehören also z.B. einem Router-Interface. Im Beispiel aus Kapitel 3.4.4 besteht zwar eine Schleife zum Host Y, aber keine zur Adresse des Router-Interfaces R_C. Auf 73,41% der IP-Adressen trifft diese Erklärung zu.

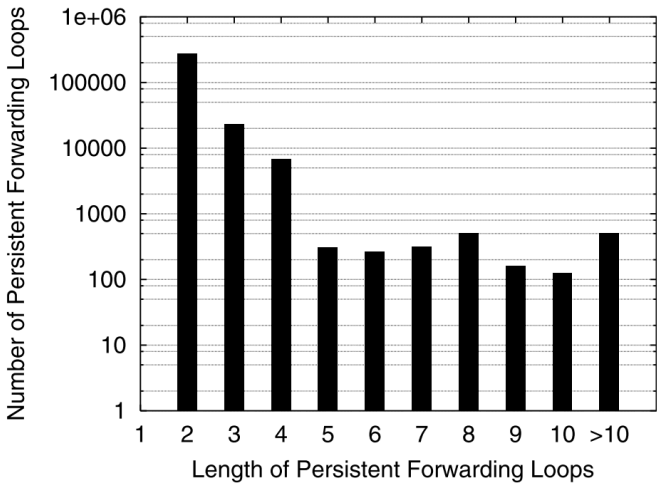


Abbildung 70: Verteilung der Schleifen nach Größe (2007) [59]

Je nach Startpunkt der Messung führen natürlich unterschiedliche Pfade zum Ziel. Aus diesem Grund werden auf vier Knoten von PlanetLab-Rechnern in Asien, Europa und der Ost- und Westküste der USA Traceroutes initiiert (D_{D1}, D_{D2}, D_{D3}, D_{D4}). Ca. 90% aller Präfixe weisen persistente Schleifen in allen vier Traceroutes auf.

Je kleiner eine Schleife ist, desto öfter werden Pakete die einzelnen Hops traversieren. 89,4% aller 302.989 gefundenen persistenten Schleifen bestehen aus zwei Knoten (Abbildung 70), dies deckt sich mit [80]. 10,4% besitzen eine Größe zwischen 3 und 9. Der Rest besteht aus mehr als 10 Knoten, darunter einige mit mehr als 16 Knoten.

Die Lage der Schleifen wird aufgrund zweier Krtiterien überprüft. Involviert die persistente Schleife Router der Zieldomain und involviert die persistente Schleife Router über mehrere AS hinweg. Die Hauptschwierigkeit besteht in der Zuordnung von IP-Adressen zu AS. An den Grenzen können Ungenauigkeiten auftreten.

Ein Pfad $(r_1, r_2, \ldots, r_n)$ enthält eine persistente Schleife $(r_i, \ldots, r_j)$. Wenn sich mindestens ein Router-Interface $(r_i, \ldots, r_j)$ in der gleichen Domain wie das Ziel *d* befindet, dann involviert die persistente Schleife Router der Zieldomain. Auf 87,44% der persistenten Schleifen trifft dieses Kriterium zu (Abbildung 71). 12,56% der persistenten Schleifen involvieren dagegen keine Router der Zieldomain (Abbildung 71). Von den 87,44% involvieren 83,66% zwei oder mehr Router der Zieldomain und 3,78% nur einen Router (Abbildung 71). Das Zuordnen von IPs zu AS kann wie bereits erwähnt an den Grenzen Ungenauigkeiten aufweisen. Die Wahrscheinlichkeit, dass dies bei zwei Routern passiert, ist dagegen relativ klein. Aus diesem Grund sollte die Ungenauigkeit weniger als 3,78% betragen.

Um die Besonderheit zu prüfen, dass sich alle Router der persistenten Schleife in der Zieldomain befinden, wird eine weitere Anpassung vorgenommen. Das vorangegangene Interface vor der Schleife $(r_i, \ldots, r_j)$ also $r_i\text{-}1$ wird ebenfalls untersucht, um Ungenauigkeiten in der Zuordnung IP zu AS möglichst gering zu halten. Für 58,21% der persistenten Schleifen befinden sich die Router-Interfaces $(r_{i-1}, r_i, \ldots, r_n)$ alle in der Zieldomain (Intra-AS) (Abbildung 71). Bei den 12,56% der persistenten Schleifen, die keine Router in der Zieldomain involvieren, wird ebenfalls das Interface des unmittelbaren Vorgängers geprüft. Nur bei 1,47% befindet sich das Vorgänger-Interface in der Zieldomain und deutet evtl. auf eine fehlerhafte Zuordnung hin (Abbildung 71).

Table 5
Classifying persistent forwarding loops based on whether destination domain is involved

Category of persistent forwarding loops	Percentage (%)
1. Destination domain is involved	**87.44**
i. loops contains only one address in the destination domain	3.78
ii. loops contain two or more addresses in the destination domain	83.66
— *All addresses in the loop are in the destination domain*	*58.21*
2. Destination domain is not involved	**12.56**
The preceding router interface of the loop in the traces	
i. is in the destination domain	1.47
ii. is not in the destination domain	11.09
Total	**100**

Abbildung 71: Zuordnung der Zieldomain zu persistenten Schleifen (2007) [59]

Table 6
Classifying persistent forwarding loops based on the number of involved domains

Category of persistent forwarding loops	Percentage (%)
1. Only a single domain is involved	**94.27**
The preceding router interface of the loop in the traces	
i. is in the same domain as the loop	67.06
ii. is not in the same domain as the loop	27.21
2. Multiple domains are involved	**5.73**
i. Two domains are involved	5.35
ii. Three or more domains are involved	0.38
Total	**100%**

Abbildung 72: Anzahl der involvierten AS (Domains) (2007) [59]

Weiterhin interessant ist die Verteilung der Schleifen auf die AS, bzw. wie viele AS involviert eine persistente Schleife oder befindet sie sich vollständig in einem einzigen AS. Abbildung 72 fasst die Ergebnisse zusammen. 94,27% der persistenten Schleifen werden als Intra-AS deklariert. Andere Studien kommen zu einem vergleichbaren Ergebnis [64][80]. Um wiederum einen Fehler abzuschätzen, wird das Vorgänger-Interface in das Kriterium

einbezogen. Allerdings ist bei 27,21% das Vorgänger-Interface nicht im gleichen AS, was auf einen höheren Fehler schließen lässt. 67,06% Prozent lassen sich dagegen eindeutig einem AS zuordnen. Von den 5,73% involviert die Mehrheit (5,35%) zwei AS und 0,38% mehr als zwei AS.

Obwohl die Zahl der persistenten Schleifen gering ist, die mehr als ein AS involvieren, kann ihr Einfluss auf die Stabilität des Internet je nach Lage enorm sein. 19 Tier-1 AS besitzen eine oder mehrere Router, die in persistenten Schleifen involviert sind. Mittels DNS werden die IP-Adressen der involvierten Router-Interfaces Domainnamen zugeordnet. 52,4% (82.626) werden erfolgreich zugeordnet und bestätigen die Erkenntnis. Aufgrund des Domainnamens wird festgestellt, dass diese Router quer über den Erdball verteilt sind und somit persistente Schleifen eine weltweite Gefahr für die Netzstabilität darstellen.

3.4.4 Auswirkungen und mögliche Angriffsarten (Flooding Attacks)[1]

[4] beschäftigt sich mit möglichen Angriffsszenarien, die Eigenheiten von Schleifen ausnutzen. Eine der weit verbreitesten Art Rechner im Netz anzugreifen, ist das sogenannte Distributed Denial Of Service (DDOS). Hierbei werden Anfragen von vielen verschiedenen Rechnern gleichzeitig an ein Ziel gesendet, sodass dieses keine Ressourcen mehr für „echte“ Anfragen zur Verfügung hat. Persistente Schleifen intensivieren eine DDOS-Attacke.

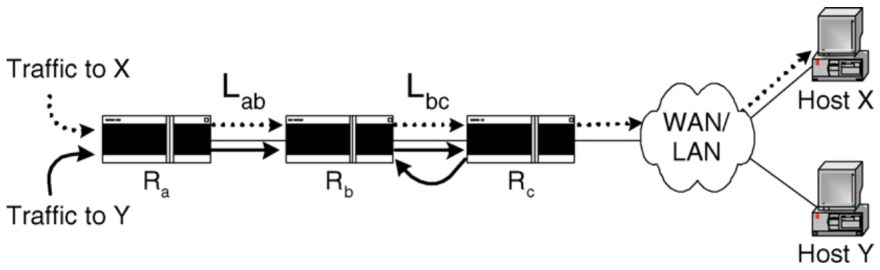


Abbildung 73: Beispiel für ein mögliches Angriffsszenario (2005) [4]

Das Beispiel Abbildung 73 erklärt ein mögliches Szenario, um eine persistente Schleife für einen DDOS-Angriff auszunutzen. Der Verkehr für Router *X* und *Y* nimmt den gleichen Pfad über L_{ab} und L_{bc}. Zwischen Router R_b und R_c existiert eine persistente Schleife, die allerdings

durch eine fehlerhafte Konfiguration nur Verkehr für Host *Y* betrifft. Ein Angreifer kann nun versuchen, die Router durch Verkehr an *Y* zu „verstopfen“, was ebenfalls eine Unerreichbarkeit von Host *X* impliziert. Pakete in Schleifen traversieren die involvierten Knoten mehrmals bis zum Ablauf der TTL, sodass ein Angreifer weitaus weniger Verkehr generieren muss, um ein Überlaufen zu erzwingen. Hierbei ist zu beachten, dass die Größe der Schleife eine Kernrolle einnimmt. Je kleiner die Schleife, desto öfter wird ein Knoten durchlaufen und desto weniger Verkehr ist notwendig. Weniger Verkehr macht das Entdecken der Attacke wiederum schwieriger, zumal sie auch nicht auf Host X sondern auf Host Y ausgerichtet ist.

Existiert eine persistente Schleife im Pfad zu einem Ziel *d*, wird die Adresse von *d* als abgeschattet (Shadowed Adress) deklariert. Im obigen Beispiel wäre das die IP-Adresse von Host *Y*. Die Besonderheit ist, dass der Pfad trotzdem Verkehr an andere Ziele transportieren kann. Ein Ziel *d‘* ist erreichbar und teilt einen oder mehrere Links eines Pfads mit einer persistenten Schleife, die bei einem anderen Ziel auftritt. Im Beispiel wäre das Host *X*, dessen IP-Adresse als gefährdete Adresse (Imperiled Address) bezeichnet wird. Gefährdete Adressen können unter der Angriffsmöglichkeit einer persistenten Schleife „leiden“, auch wenn sie nicht direkt angegriffen werden. Bei einem Angriff erfahren die Verbindungen zu gefährdeten Adressen (Imperiled Adresses) ebenfalls höhere Latenzzeiten bis hin zu einem Verbindungsabbruch.

An example of shadowed address and imperiled address

Hop	Traceroute to shadowed address 81.181.31.127	Traceroute to imperiled address 80.96.192.10
1	128.119.91.254	128.119.91.254
2	128.119.2.238	128.119.2.238
3	128.119.2.194	128.119.2.194
4	65.77.95.161	65.77.95.161
...	...	...
18	166.49.147.134	166.49.147.134
19	195.39.208.82	195.39.208.66
20	193.226.179.18	193.226.179.18
21	**193.226.130.226**	**193.226.130.226**
22	**194.176.189.42**	**194.176.189.42**
23	**193.226.130.226**	194.105.11.178
24	**194.176.189.42**	80.96.192.10
25	**193.226.130.226**	
26	**194.176.189.42**	

Abbildung 74: Traceroutes zu einer abgeschatteten (Shadowed) und einer gefährdeten (Imperiled Adress) Adresse (2005) [4]

Die Schleife im Beispiel in Abbildung 74 involviert zwei Interfaces 193.226.130.226 und 194.176.189.42, sie tritt auf, wenn Verkehr an 81.181.31.127 (Shadowed Adress) gesendet wird. Beide Interfaces tauchen auch im Pfad zum Ziel 80.96.192.10 (Imperiled Adress) auf jedoch ohne das Auftreten einer persistenten Schleife.

Die Messungen in [59] identifizierten 135.973 Präfixe als abgeschattete Präfixe (Shadowed Prefixes), was ungefähr 35 Millionen IP-Adressen (2,47% aller gerouteten IP-Adressen) entspricht. Diese sind auf 5341 AS verteilt.

Jede Adresse in einem Präfix, die sich den Pfad mit einer gemessenen Schleife teilt, wird als gefährdete Adresse angesehen. Die Schwierigkeit besteht darin, alle Adressen eines Subnetzes zu prüfen. Deshalb wird ein Präfix mit gefährdeten Adressen als gefährdetes Präfix (Imperiled Prefix) angesehen, auch wenn evtl. nicht alle Adressen des Subnetzes als gefährdet gelten. Letztlich werden 42.887 Präfixe als gefährdet gemessen, was ungefähr 0,78% (11 Millionen Adressen) aller gerouteten IP-Adressen entspricht. Diese Präfixe sind auf 2117 AS verteilt. Jedoch teilen nicht alle gefundenen persistenten Schleifen ihren Pfad mit weiteren gefährdeten Adressen. Von 302.989 gefundenen persistenten Schleifen teilen 24,1% ihren Pfad mit weiteren Adressen (Imperiled Adresses). Diese werden als dunkle Adressen (Dark Adresses/Prefixes) deklariert und können für einen Angriff auf andere Ziele ausgenutzt werden. Von 135.973 identifizierten Shadowed Prefixes können 18,4% für dieses Ziel ausgenutzt werden (Dark Prefixes). Manche persistente Schleifen beinhalten Links zu bis zu tausend angreifbaren Präfixen.

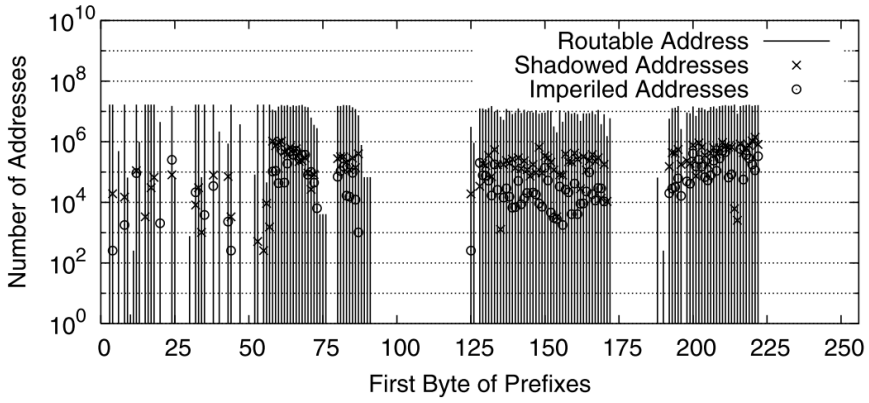


Abbildung 75: Verteilung der IP-Adressen (Routable, Shadowed, Imperiled) (2005) [4]

Abbildung 75 zeigt eine Verteilung von gerouteten, abgeschatteten und gefährdeten Adressen nach dem ersten Byte des zugehörigen Präfixes. Nicht alle Adressen werden gleichmäßig über den IPV4-Adressraum verteilt. Manche Blöcke sind z.B. für Multicast reserviert oder noch nicht von der IANA vergeben. Das Gleiche gilt für Shadowed und Imperiled Adresses. Diese stehen jedoch in einem proportionalen Verhältnis zu den gerouteten Adressen eines Präfixes. In fast jedem Präfix aus 8 Bits finden sich abgeschattete und gefährdete Adressen, was für eine globale Verteilung spricht. Eine Besonderheit bietet das Präfix 20.0.0.0/8. Hier werden keine persistenten Schleifen gefunden (keine Shadowed Adresses) jedoch einige gefährdete Adressen (Imperiled Adresses). D.h. manche Adressen sind angreifbar, obwohl keine persistenten Schleifen in Subnetzen existieren. Der umgekehrte Fall ergibt sich beim Präfix 56.0.0.0/8. Hier werden abgeschattete Adressen jedoch keine gefährdeten Adressen gefunden.

Die in [59] gefundene Tatsache, dass persistente Schleifen von weltweit verteilten Knoten beobachtet werden können, vergrößert die Angriffsfläche. Wenn Angreifer von verschiedenen Standorten verschiedene IP-Adressen (Shadowed Adresses) angreifen, erschwert dies die Erkennung erheblich.

Um einen Einfluss von Schleifen auf DDOS-Angriffe zu ermitteln, errechnen die Autoren einen Verstärkungsfaktor (Traffic Amplification Factor). Dieser ermittelt, wie oft im Durchschnitt ein Paket einen Knoten der Schleife durchläuft. Ein Angreifer würde die TTL eines Pakets wahrscheinlich auf 255 setzen und damit komplett ausreizen. Im Durchschnitt sind persistente Schleifen 14-15 Hops von der Quelle entfernt. Da persistente Schleifen größtenteils eine Länge von zwei Hops aufweisen, wäre der Verstärkungsfaktor (255-14)/2=120. D.h. ein Paket durchläuft jeden Knoten der Schleife 120 Mal. Auch für eine persistente Schleife der Länge 16, verstärkt diese einen Angriff um den Faktor 15. Angenommen der Link L_{bc} Abbildung 73 besitzt eine Bandbreite von 50Mbps und der Verstärkungsfaktor beträgt 25. Ein Angreifer müsste dementsprechend 2Mbps an Verkehr generieren, um den Link zum Überlaufen zu bringen. Hätte dieser 100 Rechner im Netz kompromittiert, müsste jeder Rechner 20Kbps generieren. Solch ein Angriff wäre kaum zu entdecken.

3.5 FAZIT[1]

Die Unterschiede in der Methodik der Studien und der untersuchte Rahmen, machen es schwierig, einen kompletten und einheitlichen Überblick zu generieren. Allerdings lassen sich einige Kernpunkte herausarbeiten.

Es gibt keine zeitlichen Kriterien, die eine Klassifizierung in persistent oder temporär erlauben. Das einzige Unterscheidungskriterium ist, dass temporäre Schleifen sich durch Protokolle automatisch auflösen, während persistente Schleifen durch einen menschlichen Eingriff aufgelöst werden müssen.

Die Analyse von Paketaufzeichnungen eignet sich in erster Linie für die Entdeckung von temporären Schleifen, da alle Pakete eines zeitlichen Intervalls ohne Unterbrechung zur Verfügung stehen. Dem zeitlichen Vorteil steht die geographische Limitierung gegenüber. Es können nur Schleifen entdeckt werden, an die der aufzeichnende Knoten teilnimmt. Traceroutes heben diese geographische Beschränkung auf und können einzelne Pfade weltweit auf Schleifen untersuchen, aber jeweils nur für einen kleinen Moment. Die weitaus größere Dauer von persistenten Schleifen hebt diesen Nachteil auf, weshalb Traceroutes ideal für die Analyse von persistenten Schleifen auch in entfernten Netzwerken sind.

Schleifen sind ein weltweites Problem und nicht auf geographische Bereiche beschränkt [64][61][80][59]. Ebenso existiert keine Beschränkung auf einige Schichten der Internetstruktur (d.h. z.B. Tier-1) [80][59], jedoch treten ungefähr 40% aller Schleifen in Tier-3 AS auf [80]. „Kernkomponenten“ des Internets leiden weniger an Schleifen, was an einem höheren finanziellen und administrativen Aufwand liegen kann, da Fehler hier ganze Bereiche des Internet beeinflussen.

Ein klarer Trend ist die Zunahme von Schleifen, bedingt durch das Wachstum und die bessere Vernetzung des Internets. Konnte Paxson [64] in 40.000 Traceroutes im Jahr 1996 ca. 50 persistente Schleifen ausfindig machen, belegen spätere Messungen eine signifikant höhere Anzahl [77][61][80][59]. Da jegliche Messungen nur einen Teil des Internets abdecken und sich in der Durchführung unterscheiden, sind absolute Zahlen in diesem Zusammenhang irreführend, ein Trend aber klar zu erkennen.

Eine Schleife verliert mit steigender Größe an Stabilität [61][80][59]. Hieraus folgt, dass ein großer Teil der Schleifen aus zwei Hops bestehen. Dies deckt sich mit allen Messungen. Temporäre Schleifen bestehen hauptsächlich (55%-90%) aus zwei Knoten [61][80]. Sobald ein Knoten ein Routing-Update erfährt, kann sich die Schleife auflösen. Die Wahrscheinlichkeit steigt mit der Anzahl der Knoten. Bei temporären Schleifen ist die Anzahl der für die Schleifenentstehung verantwortlichen BGP-Updates proportional zur Größe des Spektrums der resultierenden Schleifengrößen [77].

Persistente Schleifen bestehen zu mehr als 90% aus zwei Knoten [80][59]. Sobald ein beteiligter Knoten eine Änderung der Konfiguration erfährt, kann dies die Schleife auflösen. Persistente Schleifen in Kernnetzwerken (Tier-1 und Tier-2) neigen im Durchschnitt dazu, mehr Knoten als zwei zu involvieren [80].

Ein großer Teil aller in Studien gemessenen Schleifen sind Intra-AS, d.h. sie involvieren nur ein AS. Über 80% der temporären Schleifen ([80][77]) und über 94% der persistenten Schleifen ([59][80][77]) sind Intra-AS. Persistente Schleifen treten zudem zu über 85% in der Zieldomain auf [59].

Schleifen beeinflussen eine Verbindung auf mehrere Arten. Unabhängig von der Art der Schleife, wird die RTT einer Verbindung immer erhöht. Mit steigender Größe der Schleifen erhöht sich die Zwischenankunftszeit replizierter Pakete, da mehr Hops durchlaufen werden [80]. Hierbei ist zu beachten, dass eine Schleife auf einem Pfad entstehen kann, über den weitere Ziele erreicht werden können. Demnach wird jeglicher Verkehr beeinflusst, der über den Pfad der Schleife läuft [59]. Da TCP schnell auf Paketverluste oder eine Erhöhung der RTT reagiert, verringert sich der prozentuale Anteil an TCP-Verkehr in Schleifen [61]. Zudem kann eine Schleife einen TCP-Verbindungsaufbau komplett unterbinden oder eine Verbindung beenden, wenn sie vorher bzw. lange genug existiert. Im Allgemeinen reagiert TCP durch Stau- und Flusskontrolle auf eine Veränderung der RTT und Paketverluste. UDP verhält sich je nach Anwendung ganz unterschiedlich, da zusätzliche „Steuerungslogik“ nicht vom Protokoll sondern der Anwendung übernommen werden muss. Im schlimmsten Fall sendet eine Anwendung fortlaufend UDP-Pakete. Jedes in der Schleife verworfene UDP- oder TCP-Paket erzeugt eine ICMP-Nachricht (Typ11, Time-To-Live Exceeded). Dieser erhöhte Anteil an ICMP-Paketen in Schleifen wurde in [61] gemessen. Schleifen rufen bereits vor der

Bildung Paketverluste hervor [80]. 65% der temporären Schleifen und 55% der persistenten Schleifen implizieren kurz vor der Bildung eine Fehlerrrate von mehr als 30% [80].

Temporäre Schleifen ziehen weitere temporäre Ausfälle mit sich und/oder unterstützen die Bildung weiterer temporärer Schleifen in der unmittelbaren Umgebung [64][80]. [61] belegt eine überwiegende Dauer (70%) von weniger als 10 Sekunden für temporäre Schleifen, 30% existieren dagegen länger als 10 Sekunden. Der Adressbereich von 192.0.0.0 bis 223.255.255.255 ist am Häufigsten durch die Bildung von temporären Schleifen betroffen, da sich hier häufige strukturelle Änderungen ergeben [61]. Da temporäre Schleifen sich vollautomatisch durch Protokolle auflösen und die Dauer sehr kurz ist, sind sie letztlich nur für zeitkritische Anwendungen wie z.B. Audio- oder Videoübertragungen ein Problem. Ein gezieltes Ausnutzen für DDOS-Attacken ist unwahrscheinlich. Um temporäre Schleifen zu vemeiden, müssen Protokolle und die Infrastruktur erweitert werden. Jeder Link benötigt einen (berechneten) alternativen Next Hop, der bei einem Ausfall umgehend genutzt werden kann. Auch wenn in der Forschung mehrere Ansätze seit Jahren bestehen, temporäre Schleifen zu verhindern, ist man von einer realen Verbreitung weit entfernt. Hier scheint IPFRR am weitesten zu sein, zumindest wird in Gremien wie der IETF (Internet Engineering Task Force, Webseite: www.ietf.org) über Standardisierungsmöglichkeiten diskutiert, ohne das IP-Konzept aufgeben zu müssen [84].

Die Hälfte aller persistenten Schleifen ist auf Konfigurationsfehler zurückzuführen. 50% der persistenten Schleifen entstehen durch Aggregation von Subnetzen und das Fehlen einer Sink Route [59]. Ein großer Teil persistenter Schleifen existiert weniger als 30 Minuten (ca. 50%) oder länger als 7,5 Stunden (ca 20%) [80]. Die wesentlich längere Dauer von persistenten Schleifen ist für Angreifer ideal und bietet ein enormes Angriffpotential. Eine persistente Schleife bietet eine Angriffsmöglichkeit für alle Ziele, die den gleichen Pfad nutzen und intensiviert/verschleiert DDOS-Attacken [59]. Ca. 2% aller gerouteten IP-Adressen leiden unter persistenten Schleifen (Shadowed Adresses). Diese persistenten Schleifen sind von weltweit verteilten Knoten aus beobachtbar [59]. Ca. 18% aller abgeschatteten Adressen können für einen Angriff auf andere Ziele ausgenutzt werden (Imperiled Adresses) und damit einen Angriff verschleiern [59]. In fast jedem Präfix aus 8 Bits finden sich abgeschattete (Shadowed Adresses) und gefährdete Adressen (Imperiled Adresses) [59]. Damit sind persistente Schleifen ein ernsthaftes Problem für die Sicherheit und Stabilität des Internet. Da

sie letztlich hauptsächlich „nur“ durch Konfigurationsfehler entstehen, sind die Gegenmaßnahmen offensichtlich und leichter durchzuführen als bei temporären Schleifen.

4 TESTUMGEBUNG / MESSERGEBNISSE[2]

Nachdem wir uns in Kapitel 2 mit der Verteilung verschiedener Transportprotokolle beschäftigt haben und in Kapitel 3 das Vorkommen und die Ursachen von Forwarding-Schleifen beschrieben wurden, wollen wir in diesem Kapitel das Verhalten von Datenpaketen und dessen Auswirkungen in einem Testszenario untersuchen.

Mit den aus Kapitel 2 gewonnen Kenntnissen über die Protokolleigenschaften werden wir dabei analysieren, wie sehr die verschiedenen Pakete in einer Forwarding-Schleife das Datenaufkommen beeinflussen und welche Transportprotokolle gegenüber den negativen Auswirkungen resistent sind.

4.1 GNS3[2]

Um ein geeignetes Testszenario zu erstellen, benutzen wir für unsere Untersuchungen das Netzwerk-Simulationstool GNS3 [85]. Die Software basiert auf den Emulatoren Dynamips bzw. Dynagen [86], welche dafür entwickelt wurde Cisco-Hardware auf PC-Systemen zu emulieren. Die Unterstützung von Cisco Geräten beschränkt sich dabei jedoch auf die Router der Serien 7200, 3600, 3700, 2600 und 1700, schließt aber die Simulation von Geräten der Cisco Catalyst Switch Reihe aus. Der Grund dafür ist, dass eine Emulation von ASIC-Prozessoren, welche diese Geräte benutzen, nicht möglich ist. Darüber hinaus bietet dieses Werkzeug jedoch eine exzellente Möglichkeit, um sich mit den unterstützten Geräten auf die Cisco Zertifizierungen CCNA, CCNP, CCIP und CCIE vorzubereiten. GNS3 ist für alle gängigen Betriebssysteme wie Linux, Windows und MacOS X verfügbar und stellt eine intuitiv zu bedienende grafische Benutzeroberfläche zur Verfügung.

Neben der Emulation von Cisco-Hardware ist mit GNS3 auch die virtuelle Vernetzung von QEMU-, PEMU- und VirtualBox-Instanzen möglich. Ein Vorteil bei der Verwendung von GNS3 gegenüber anderen Netzwerksimulatoren, wie etwa VNUML[87], [88], sind dabei die realistischen Laufzeiten auf den Links, welche von Dynampis an die tatsächliche Geschwindigkeit des jeweils gewählten Interface angepasst wird.

Die Performanz von GNS3 ist dabei sehr hoch und ermöglicht die Simulation kleiner Netzwerke auf handelsüblichen Computersystemen. Bei der Verwendung von Cisco Routern ist jedoch unbedingt die Justierung der Idle-Time notwendig um die CPU des Systems nicht dauerhaft auszulasten. Bei der Simulation von virtuellen Maschinen hat sich unter Windows die Verwendung von VirtualBox-Instanzen als sehr performant erwiesen, da hier die volle Hardwarebeschleunigung für emulierte Systeme gegeben ist. Eine Verwendung von QEMU hingegen ist nur unter Linux zu empfehlen, da nur hier die Unterstützung für KVM gegeben ist und die emulierten Systeme ohne diese sehr langsam sind.

Die Konfiguration der einzelnen Schnittstellen erfolgt dabei in den emulierten Geräten selbst, d.h. GNS3 bietet keine Möglichkeit die Interfaces direkt aus seiner Oberfläche heraus zu konfigurieren.

Ein weiterer Vorteil von GNS3 gegenüber VNUML ist die Möglichkeit, die virtuellen Links direkt aus der Oberfläche hinaus zu überwachen. Dazu wird der Netzwerkverkehr an dem gewählten Link in einer Capture-Datei fortlaufend abgelegt und kann in Echtzeit über Wireshark analysiert werden. Bei der Verwendung von Cisco Systemen kann diese Überwachung im laufenden Betrieb zu- und abgeschaltet werden, bei QEMU- oder VirtualBox-Systemen muss diese Funktion vor dem Starten der virtuellen Maschinen aktiviert sein.

Zusätzlich kann ein in GNS3 simuliertes Netzwerk über ein virtuelles Interface mit dem Host-System verbunden werden. Dadurch ist es möglich die in GNS3 emulierten Systeme mit dem Internet zu verbinden oder auch mehrere Szenarien über eine VPN Verbindung miteinander zu verknüpfen.

4.2 VORÜBERLEGUNGEN ZUR TESTUMGEBUNG[1]

Bevor wir unser Testszenario entwerfen, überlegen wir uns, wie wir am Besten das grundlegende Verhalten der Protokolle in Schleifen nachweisen können. Aus Kapitel 3 wissen wir, dass die Mehrheit der Schleifen im Internet 2 Hops involvieren. Nach reiflicher Überlegung entschließen wir uns jedoch dazu, eine Schleife mit 3 Hops aufzubauen, da dies ein unserer Meinung nach anschaulicheres Beispiel darstellt. Zudem vermuten wir bei unserem simulierten Test eine sehr niedrige RTT und wir hoffen durch einen zusätzlichen

Knoten eine geringe Erhöhung der RTT. Die Größe der Schleife hat letztlich nur einen Einfluss darauf, wie oft ein Knoten bis zum Ablauf der TTL durchlaufen wird und demnach keinen entscheidenden Einfluss auf den grundlegenden Umgang der Protokolle mit Schleifen. Da wir unsere Ergebnisse durch mehrere Durchläufe absichern und auch die verschiedenen Tests der unterschiedlichen Protokolle untereinander vergleichbar halten möchten, entwerfen wir ein statisches Testszenario, welches bei allen Messungen Anwendung findet und entscheiden uns gegen einen z.B. geskripteten Netzentwurf mit evtl. variierenden Variablen.

4.3 TESTSZENARIO[2]

Für unser Testszenario benutzen wir mehrere VirtualBox-Instanzen, in welchen die auf Debian basierende virtuelle Router-Software Vyatta [89] installiert ist. Um eine Forwarding-Schleife zu simulieren, benötigen wir dafür 4 virtuelle Router welche in einer Y-Topologie angeordnet sind. Zum Erzeugen von Netzwerkverkehr mit unterschiedlichen Transportprotokollen sind zusätzlich an den entsprechenden Router zwei weitere virtuelle Maschinen angeschlossen, welche es ermöglichen die für unsere Tests benötigten Client-Server Applikationen auszuführen.

In Abbildung 76 ist der schematische Aufbau des Testnetzwerks zu sehen. Die mit R1 – R4 gekennzeichneten Router werden dabei mit Vyatta 6.4 Distributionen simuliert. Die beiden Hostsysteme, Host A und Host B, werden ebenfalls mit VirtualBox-Instanzen simuliert und bilden ein minimales Linux-System ab.

Die Forwarding-Schleife wird dabei in den späteren Test über die Router R1-R3 aufgebaut, so dass alle Pakete, die für das Zielnetz 10.0.10.0/24 bestimmt werden, durch diese Router zirkulieren.

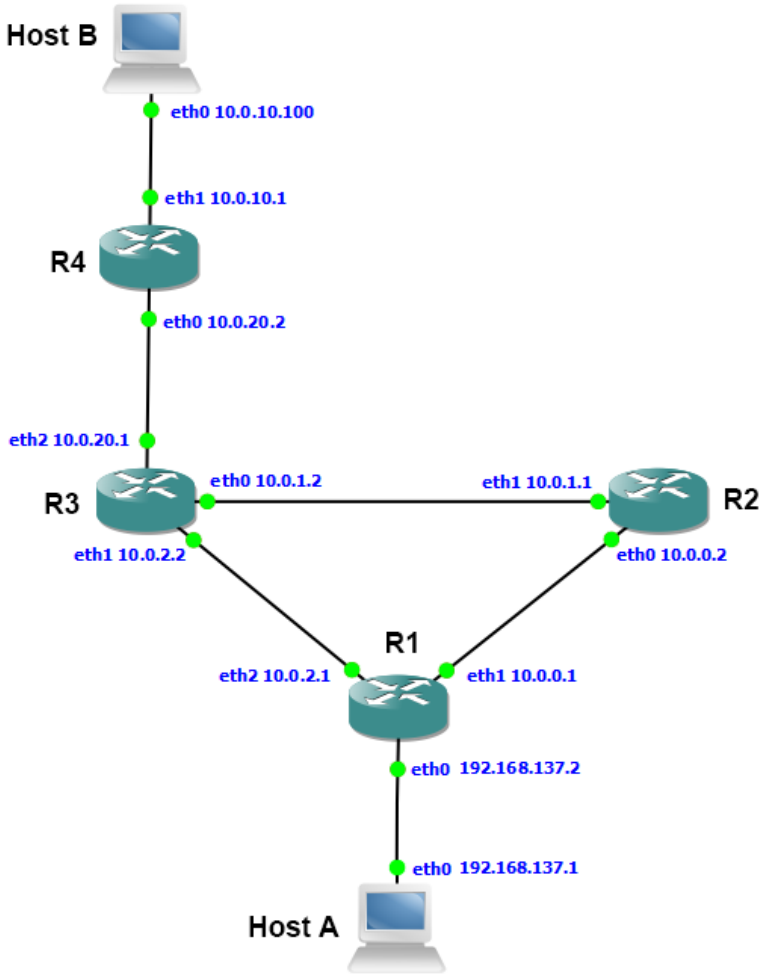


Abbildung 76: Testszenario in GNS3 in Y-Topologie

Für die durchgeführten Tests wurde GNS3 in der Version 0.8.4 RC1 (64Bit) verwendet. VirtualBox wurde in der Version 4.2.10 eingesetzt.

Soweit es in den folgenden Fallstudien nicht anders beschrieben ist, wurde jede Testreihe mindestens 20-mal durchlaufen, um die Testergebnisse zu verifizieren. Für eine detaillierte Auswertung der Ergebnisse wurde im Einzelfall das Testergebnis ausgewertet, welches am repräsentativsten für die entsprechende Versuchsreihe war.

4.3.1 INTERFACE-KONFIGURATION[2]

Um das in Abbildung 76 gezeigte Szenario zu erstellen müssen die Interfaces der Vyatta-Router entsprechend der Abbildung konfiguriert werden. Um in den Konfigurationsmodus

wechseln zu können muss die Anmeldung als Superuser erfolgen. Anschließend kann mit dem Befehl

```
#  configure
```

der Konfigurationsmodus gestartet werden.

Um einem Interface nun eine IP-Adresse mit der zugehörigen Netzmaske zuzuweisen wird folgender Befehl

```
#  set interface ethernet INTERFACE address IP-ADRESSE/NETZMASKE
```

verwendet.
Zum Übernehmen der Änderungen muss anschließend der Befehl

```
#  commit
```

eingeben werden und mit dem Befehl

```
#  save
```

werden die Einstellungen dauerhaft gespeichert.

Führen wir die Konfioguration anlog der schematischen Darstellung in aus, erhalten wir folgende Interfacekonfiguration:

Router R1

```
root@vyatta:/home/vyatta# show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                     S/L  Description
---------        ----------                     ---  -----------
eth0             192.168.137.2/24               u/u
eth1             10.0.0.1/24                    u/u
eth2             10.0.2.1/24                    u/u
lo               127.0.0.1/8                    u/u
```

Router R2

```
root@vyatta:/home/vyatta# show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                     S/L  Description
---------        ----------                     ---  -----------
eth0             10.0.0.2/24                    u/u
eth1             10.0.1.1/24                    u/u
lo               127.0.0.1/8                    u/u
```

Router R3

```
root@vyatta:/home/vyatta# show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                     S/L  Description
---------        ----------                     ---  -----------
eth0             10.0.1.2/24                    u/u
eth1             10.0.2.2/24                    u/u
eth2             10.0.20.1/24                   u/u
lo               127.0.0.1/8                    u/u
```

Router R4

```
root@vyatta:/home/vyatta# show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                     S/L  Description
---------        ----------                     ---  -----------
eth0             10.0.20.2/24                   u/u
eth1             10.0.10.1/24                   u/u
lo               127.0.0.1/8                    u/u
```

4.3.2 ROUTING TABELLEN[2]

Zum Konfigurieren der Routen für die dargestellte Topologie wechseln wir ebenfalls zuerst wieder in den Konfigurationsmodus (vgl. Abschnitt 4.2.1.) und legen anschließend mit dem Befehl

```
#  set protocols static route NETZ/NETZMASKE next-hop IP-ADRESSE
```

die entsprechenden Routen an.

Für unser Testszenario legen wir die Routen so an, dass Host A die Pakete über die Router R1, R2, R3 und R4 zu Host B sendet. Entsprechend kommuniziert Host B über die Router R4, R3, R2 und R1 mit Host A.

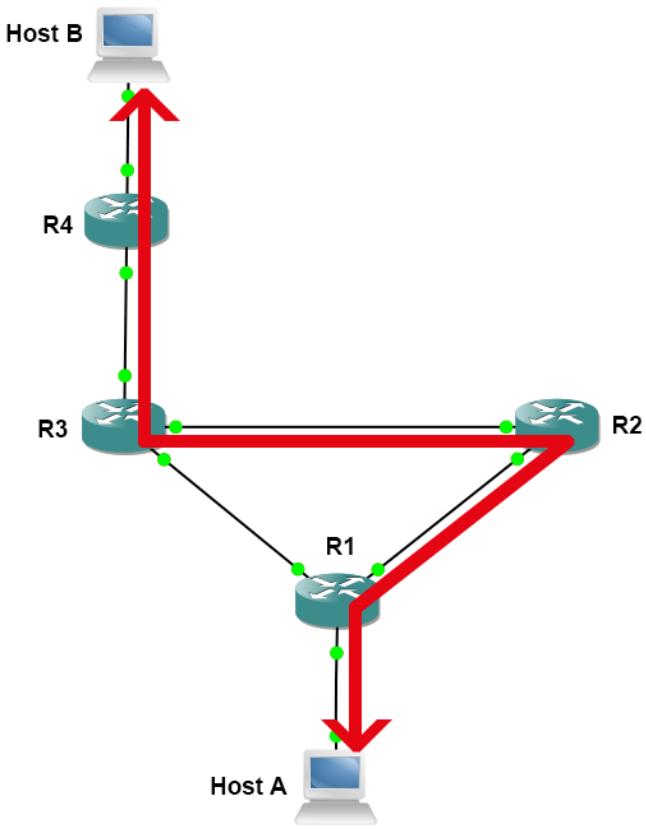


Abbildung 77: Routing-Konfiguration des Testszenarios

Daraus resultieren für unsere Topologie folgende Routing Tabellen:

Router R1

```
root@vyatta:/home/vyatta# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        *               255.255.255.0   U     0      0        0 eth1
10.0.1.0        10.0.0.2        255.255.255.0   UG    0      0        0 eth1
10.0.2.0        *               255.255.255.0   U     0      0        0 eth2
10.0.10.0       10.0.0.2        255.255.255.0   UG    0      0        0 eth1
10.0.20.0       10.0.0.2        255.255.255.0   UG    0      0        0 eth1
loopback        *               255.0.0.0       U     0      0        0 lo
192.168.137.0   *               255.255.255.0   U     0      0        0 eth0
```

Router R2

```
root@vyatta:/home/vyatta# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        *               255.255.255.0   U     0      0        0 eth0
10.0.1.0        *               255.255.255.0   U     0      0        0 eth1
10.0.10.0       10.0.1.2        255.255.255.0   UG    0      0        0 eth1
10.0.20.0       10.0.1.2        255.255.255.0   UG    0      0        0 eth1
loopback        *               255.0.0.0       U     0      0        0 lo
192.168.137.0   10.0.0.1        255.255.255.0   UG    0      0        0 eth0
```

Router R3

```
root@vyatta:/home/vyatta# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.1.1        255.255.255.0   UG    0      0        0 eth0
10.0.1.0        *               255.255.255.0   U     0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     0      0        0 eth1
10.0.10.0       10.0.20.2       255.255.255.0   UG    0      0        0 eth2
10.0.20.0       *               255.255.255.0   U     0      0        0 eth2
loopback        *               255.0.0.0       U     0      0        0 lo
192.168.137.0   10.0.1.1        255.255.255.0   UG    0      0        0 eth0
```

Router R4

```
root@vyatta:/home/vyatta# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.20.1       0.0.0.0         UG    0      0        0 eth0
10.0.0.0        10.0.20.1       255.255.255.0   UG    0      0        0 eth0
10.0.1.0        10.0.20.1       255.255.255.0   UG    0      0        0 eth0
10.0.2.0        10.0.20.1       255.255.255.0   UG    0      0        0 eth0
10.0.10.0       *               255.255.255.0   U     0      0        0 eth1
10.0.20.0       *               255.255.255.0   U     0      0        0 eth0
loopback        *               255.0.0.0       U     0      0        0 lo
192.168.137.0   10.0.20.1       255.255.255.0   UG    0      0        0 eth0
```

4.3.3 TESTSZENARIO MIT FORWARDING -SCHLEIFE[2]

Um eine Forwarding -Schleife in unserem Testszenario zu erzwingen, werden wir in unseren Tests die Routing-Tabelle von R3 entsprechend verändern. Um dies zu erreichen wechseln wir wieder in den Konfigurationsmodus und löschen den Routeneintrag für das Zielnetz 10.0.10.0/24.

```
#  delete protocols static route 10.0.10.0/24 next-hop 10.0.20.2
```

Anschließend setzen wir einen neuen Routing-Eintrag mit dem Befehl

```
#  set protocols static route 10.0.10.0/24 next-hop 10.0.2.1
```

Nachdem wir die Änderungen der beiden Befehle mit dem Befehl *commit* bestätigen, werden diese in R3 übernommen und die Forwarding-Schleife ist erstellt. Die Routing-Tabelle und der schematische Aufbau der Topologie sehen dabei anschließend wie folgt aus:

Router R3

```
root@vyatta:/home/vyatta# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.1.1        255.255.255.0   UG    0      0        0 eth0
10.0.1.0        *               255.255.255.0   U     0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     0      0        0 eth1
10.0.10.0       10.0.2.1        255.255.255.0   UG    0      0        0 eth1
10.0.20.0       *               255.255.255.0   U     0      0        0 eth2
loopback        *               255.0.0.0       U     0      0        0 lo
192.168.137.0   10.0.1.1        255.255.255.0   UG    0      0        0 eth0
```

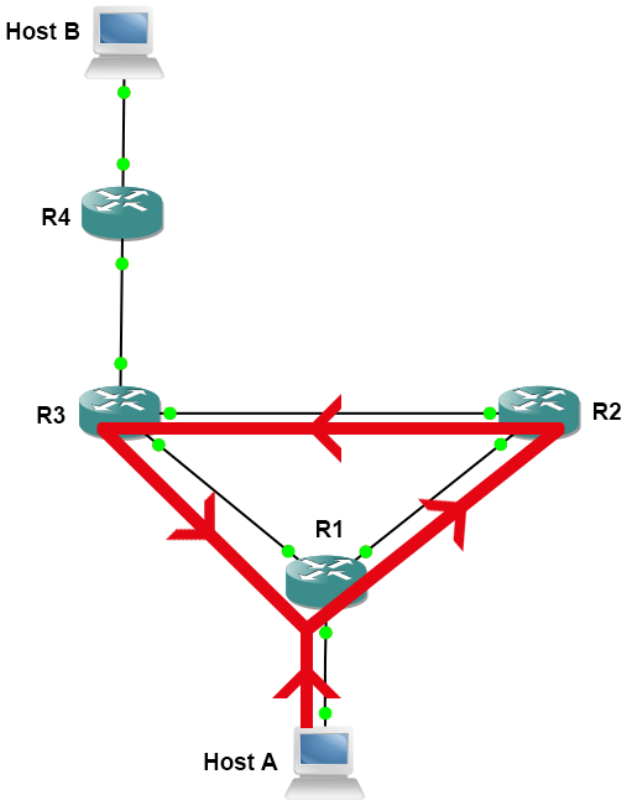


Abbildung 78: Darstellung der Forwarding-Schleife im Testszenario

4.3.4 PING-TEST[1]

Für einen ersten Praxistest lassen wir ein ICMP Echo-Request-Paket (Ping) in eine Schleife laufen und beobachten was passiert. Dabei messen wir in mehreren Durchgängen und variieren die TTL-Anfangswerte. Windows-Rechner benutzen meist eine TTL von 128, während Linux die TTL standardmäßig auf 64 einstellt.

Ping ist ein kleines Komandozeilentool mit dem die Erreichbarkeit von Rechnern im Netzwerk geprüft werden kann [90]. Der Zielrechner antwortet auf ein ICMP Echo-Request-Paket (ICMP-Pakettyp 8) mit einem ICMP Echo-Reply (ICMP-Pakettyp 0). Die Zeit vom Versand der Anfrage bis zum Eintreffen der Antwort wird gemessen und als Paketumlaufzeit bezeichnet (RTT, Round-Trip-Time). Ist der Rechner im Netzwerk nicht erreichbar, antwortet der zuständige Router entweder mit „Network Unreachable“ oder „Host Unreachable“. Ein Ausbleiben der Echo-Reply-Nachricht ist kein Beweis für die Abstinenz des geprüften Netzwerkknotens, da ICMP-Nachrichten von Hosts ignoriert bzw. von Firewalls gefiltert

werden können. Die Messung führen wir ca. 20 Mal aus, um vergleichbare Ergebnisse zu erhalten und einmalige Anomalien herauszufiltern.

Ein Aufruf von Ping mit einer TTL von 64 in der Kommandozeile unter Windows sieht wie folgt aus: *ping www.google.de –i 64*

4.3.4.1 TESTERGEBNISSE[1]

In unserem Testnetz sendet *Host A* kontinuierlich ein Echo-Request-Paket an *Host B* ab und wartet auf die Antwort. Nach 10 Sekunden aktivieren wir die Schleife. Wir messen mit den TTL-Amfangswerten von 64, 128 und 255 also dem Maximalwert. Die Pakete werden an Router *R2 eth1* also dem Link zwischen *R2* und *R3* aufgezeichnet und ausgewertet.

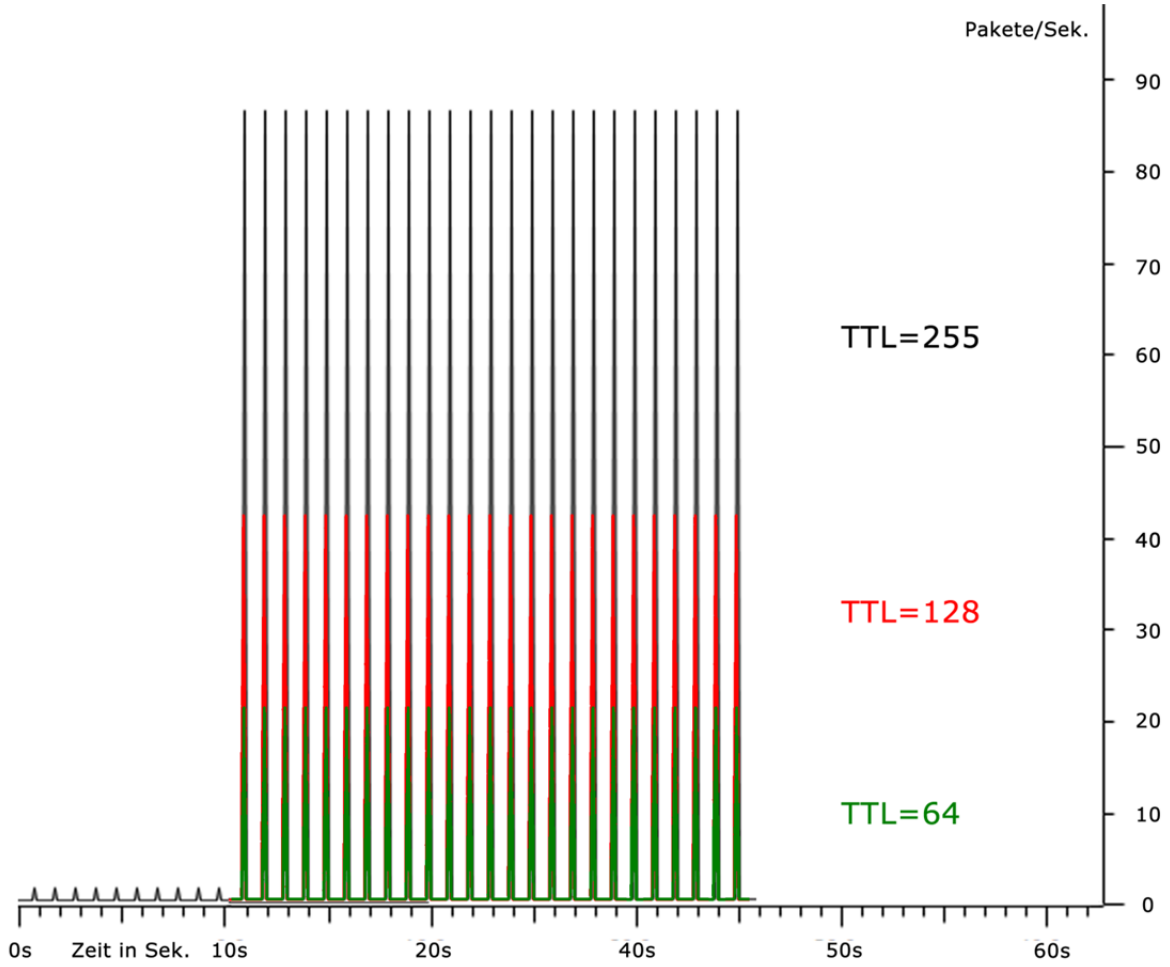


Abbildung 79: Echo-Request-Pakete mit einer TTL von 64, 128 und 255 (Pakete pro Sekunde)

In den ersten 10 Sekunden existiert keine Schleife. Es ist erkennbar, dass jeder Echo-Request mit einem Echo-Reply beantwortet wird. Es ist immer nur ein Paket zu einem Zeitpunkt im Transit. Pro Sekunde wird durchschnittlich ein Echo-Request mit einem Echo-Reply beantwortet, d.h. der Graph in Abbildung 79 zeigt zwei Pakete pro Sekunde an (bis zur Aktivierung der Schleife ab Sekunde 10).

Nach der Aktivierung der Schleife durchläuft jeder Echo-Request die Schleife bis zum Ablauf seiner TTL. Ein Echo-Request erreicht die Schleife (*R1*) mit einer TTL von 64 und durchläuft jeden Knoten der Schleife mindestens 21 Mal (TTL/Anzahl der Knoten der Schleife). Router *R1* erhält das Paket mit einer TTL von 1 (22. Durchlauf), dekrementiert diese und verwirft daraufhin das Paket. Eine ICMP-Nachricht (Typ11, Time-To-Live Exceeded) informiert den Absender (Host A) über den Paketverlust. Das Zirkulieren der Pakete in der Schleife bis zum Ablauf der TTL geschieht innerhalb einer Sekunde. Aus diesem Grund zeigt der Graph nach Aktivierung der Schleife 21 Pakete pro Sekunde an (Sekunde 10 bis 45, grüner Graph in Abbildung 79).

Analog ist das Verhalten mit einer TTL von 128 und 255 zu beobachten. Jedoch erkennt man die enorme Auswirkung auf die Schleife. Ein Paket traversiert jeden Knoten mind. 42 (TTL=128) (Sekunde 10 bis 45, roter Graph in Abbildung 79) bzw. 85 (TTL=255) (Sekunde 10 bis 45, schwarzer Graph in Abbildung 79) Mal bis zum Ablauf der TTL.

4.3.4.2 PING TEST TTL MIT VIER HOPS IN EINER SCHLEIFE[1]

Um verlässliche Aussagen über das Verhältnis von Schleifengröße zur Anzahl der Traversierungen pro Hop zu erhalten, erweitern wir unser Testnetz zu einer Schleife mit vier Hops (Abbildung 80). Um eine Schleife zu erhalten, müssen die Routen in Router *R3* und *R5* angepasst werden. Die ursprüngliche Route für das Zielnetz 10.0.10.0/24 wird dabei entfernt und durch einen Eintrag ersetzt, der auf *R5* bzw. *R1* als Next Hop verweist.

Die Paketaufzeichnung wird wieder auf *R2 eth1* also dem Link zwischen *R2* und *R3* gestartet. Um Anomalien der virtuellen Umgebung auszuschließen, lassen wir den Test 20 Mal durchlaufen und werten die Ergebnisse aus.

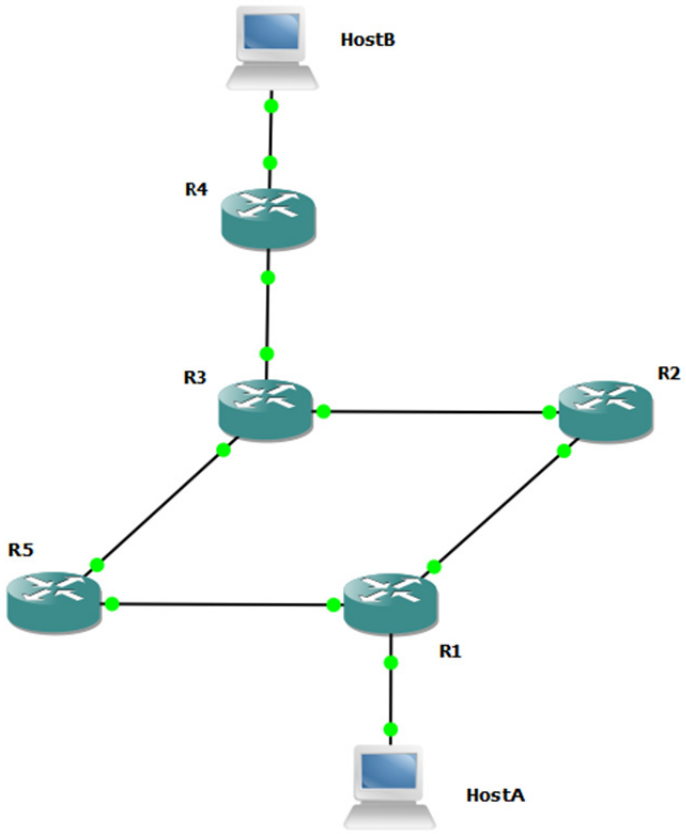


Abbildung 80: Testnetz mit einer Schleifengröße von vier Hops

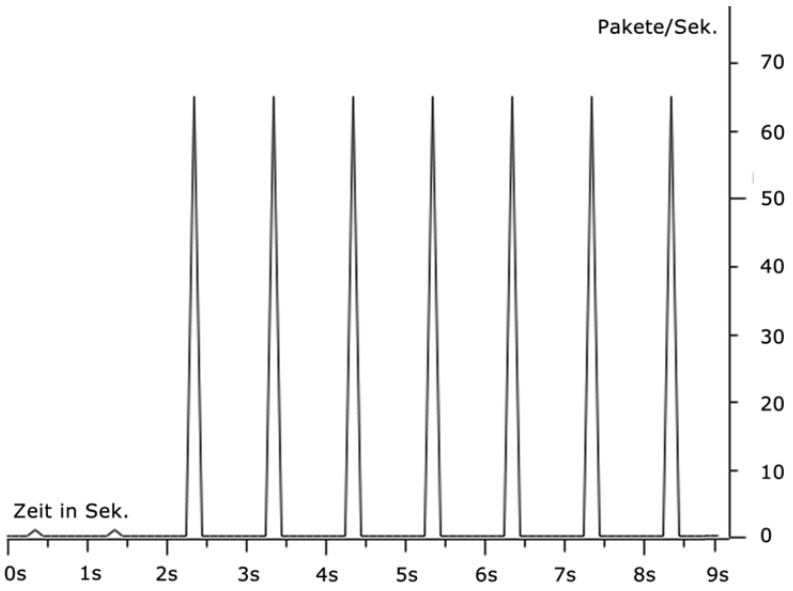


Abbildung 81: Echo-Request-Pakete mit einer TTL von 255 (Schleife mit 4 Hops)

Jeder Knoten wird 65 Mal durchlaufen (Abbildung 81). Die TTL wird in Knoten *R5* (letzter Knoten der Schleife) auf 0 dekrementiert, verworfen und eine entsprechende ICMP-Nachricht (Typ11, Time-To-Live Exceeded) an *Host A* gesendet.

4.3.4.3 FAZIT[1]

Ein Angreifer hat mit der TTL ein mächtiges Instrument zur Verfügung, um Angriffe ohne großen Aufwand erheblich zu intensivieren. Aus Kapitel 3 wissen wir, dass die Stabilität von Schleifen mit der Größe abnimmt. In der Implikation bedeutet dies, dass der überwiegende Teil temporärer und persistenter Schleifen aus zwei Knoten bestehen. Für temporäre Schleifen ist dies zu ca. 55-90% der Fall (siehe Kapitel 3.5). Persistente Schleifen bestehen zu über 90% aus lediglich zwei Knoten (siehe Kapitel 3.5). Eine Verdoppelung der TTL bedeutet eine Verdoppelung der Anzahl der Durchläufe pro Paket bei einer Schleifengröße von zwei Hops.

Allgemein lässt sich folgende Formel für die Anzahl der minimalen Durchläufe pro Knoten (x) in Abhängigkeit von der Schleifengröße (size) und der TTL (ttl) beim Eintritt in die Schleife herleiten (floor bedeutet ein Abrunden des Ergebnisses):

x = floor (ttl/size)

Analog dazu kann die maximale Anzahl der Durchläufe pro Knoten (x) in Abhängigkeit von der Schleifengröße (size) und der TTL (ttl) beim Eintritt in die Schleife hergeleitet werden (ceil bedeutet ein Aufrunden des Ergebnisses):

x = ceil (ttl/size)

Der Knoten bei dem das Paket verworfen wird, lässt sich durch folgende Formel herleiten (mod bestimmt den Rest aus einer ganzzahligen Division):

node = (ttl mod size) //node = 0 bedeutet, der letzte Knoten der Schleife verwirft das Paket

Alle Knoten davor erfahren die maximale Anzahl an Durchläufen, Knoten dahinter nur die minimale Anzahl an Durchläufen. Der Knoten der die TTL auf 0 dekrementiert erfährt ebenfalls die maximale Anzahl an Durchläufen, da er das Paket empfängt und verarbeiten muss.

4.4 EIN SERVER-CLIENT-TEST[2]

Ein realistischeres Szenario wird durch einen Server-Client-Test nachgestellt. Hierbei sendet ein Client eine bestimmte Anzahl an Nutzdaten an einen Server. Je nachdem ob TCP oder UDP benutzt wird, muss vorher ein Verbindungsaufbau erfolgen.

Wir benutzen den in der Lehre bekannten Echo-Server. Ein Client schickt hierbei einen beliebigen String an den Server. Dieser wiederum sendet den empfangenen String zurück. Um den Ablauf zu automatisieren, wandeln wir die Skripte ab, so dass kontinuierlich ein bestimmter String („This is the message.“) in einer Schleife an den Server gesendet wird und dieser nicht mit dem String antwortet. Dadurch ist es einfacher den Paketstrom zu analysieren und die zurückgesendeten ACKs im Fall von TCP zu identifizieren.

TCP ist ein verbindungsorientiertes Protokoll, d.h. erst nach einem Verbindungsaufbau (Three-Way-Handshake) erfolgt eine Übertragung der Nutzdaten. Um weder den Empfänger noch das Netzwerk zu überfluten besitzt TCP eine Fluss- und Staukontrolle (siehe Kapitel 2.6.2). Wir erwarten, dass TCP nach Aktivierung der Schleife den nicht zu erreichenden Host bzw. einen Paketverlust bemerkt und das Netz nicht unnötig mit Datenpaketen geflutet wird.

Im Gegensatz dazu besitzt das verbindungslose UDP-Protokoll keine Fluss- und Staukontrolle. Das Abfangen eines Fehlerfalls und damit ein unnötiges Fluten des Netzwerks muss von der Applikation sichergestellt werden. Aus diesem Grund setzt z.B. Skype eine zusätzliche TCP-Verbindung ein, um einen Verbindungsabbruch zu erkennen. Die eigentlichen Nutzdaten der Telefonie werden oftmals per UDP übertragen [91]. Wir erwarten, dass in unserem Client-Server-Szenario das Netz bzw. die Schleife unkontrolliert mit UDP-Nachrichten geflutet wird.

4.4.1 TCP[1]

Da TCP einen Verbindungsaufbau benötigt, bevor Nutzdaten übertragen werden können, werden wir die Schleife erst nach einem erfolgreichen Verbindungsaufbau aktivieren. Um alle ICMP-Nachrichten zu erfassen, wird der Datenverkehr dieses Mal an Router *R1* (genauer am Link zwischen *R1* und *R2*) protokolliert, da Router *R2* aufgrund des Ablaufs der TTL die Pakete verwerfen wird und eine ICMP-Nachricht an Host *A* sendet. Zur Messung verwenden

wir das ursprüngliche Testnetz mit einer Schleifengröße von drei Hops (siehe Kapitel 4.3.3). Die Pakete haben eine TTL von 126 nach einem erstmaligen Durchlauf von *R2*. Demenstprechend verlassen die Pakete am Schluss der Traversierung der Schleife *R1* mit einer TTL von 1. Aus dem TTL-Test wissen wir damit, dass *R1* und *R2* eine maximale Anzahl an Durchläufen der Pakete von *Host A* erfährt. Um temporäre Anomalien der virtuellen Umgebung auszuschließen, haben wir den Test ca. 20 Mal unter gleichen Bedingungen durchgeführt und die Ergebnisse ausgewertet.

4.4.1.1 PYTHON SCRIPTE TCP[1]

Das Python-Skript des Clients erzeugt einen Socket (Zeile 5) und versucht eine Verbindung zum Server mit der IP 192.168.1.134 an Port 51600 aufzubauen (Zeile 11). Eine While-Schleife erzeugt 3000 Pakete mit dem String („This is the message.“) und versendet diese an den Server (Zeile 13-18).

Python TCP-Client

```
01    import socket
02    import sys
03
04    # Create a TCP/IP socket
05    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
06    i= 0
07
08    # Connect the socket to the port where the server is listening
09    server_address = ('192.168.1.134', 51600)
10    print ('connecting to %s port %s' % server_address)
11    sock.connect(server_address)
12    try:
13       while i < 3000:
14        # Send data
15        message = 'This is the message.'
16        print ('sending "%s"' % message)
17        sock.sendall(message.encode())
18       i = i + 1
19
20    finally:
21        print ('closing socket')
22        sock.close()
```

Das Server-Skript erzeugt einen Socket (Zeile 8) und bindet diesen an die IP 192.168.1.134 mit dem Port 51600 (Zeile 11). Zeile 14 beauftragt den Socket am gegebenen Port auf Datenpakete zu lauschen. Eine erste Endlosschleife (Zeile 16-33) wartet auf Verbindungsanfragen und akzeptiert diese. Eine verschachtelte Schleife in Zeile 24-30

verarbeitet den empfangenen Datenstrom und bricht ab, sobald jegliche Daten empfangen und verarbeitet wurden. Um Fehlerfälle abzufangen wird dieser Block von einer Try-Anweisung umschlossen (Zeile 20-30). Nach dem Verlassen der Try-Anweisung wird der Finally-Block (Zeile 31-33) ausgeführt und die Verbindung geschlossen. Da der Client fortlaufend Daten an den Server sendet, bricht dieser erst nach der Aktivierung der Schleife und einem einhergehenden Abbruch des Datenstroms bzw. einem Ablauf des Connection-Timers die Verbindung ab.

Python TCP-Server

```
01    import socket
02    import sys
03
04    # Create a TCP/IP socket
05    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
06
07    # Bind the socket to the port
08    server_address = ('192.168.1.134', 51600)
09
10    print ('starting up on %s port %s' % server_address)
11    sock.bind(server_address)
12
13    # Listen for incoming connections
14    sock.listen(1)
15
16    while True:
17        # Wait for a connection
18        print ('waiting for a connection')
19        connection, client_address = sock.accept()
20        try:
21            print ('connection from', client_address)
22
23            # Receive the data
24            while True:
25                data = connection.recv(4096)
26                print ('received "%s"' % data)
27                if data:
28                    print ('still busy...')
29                else:
30                    break
31        finally:
32            # Clean up the connection
33            connection.close()
```

4.4.1.2 TESTERGEBNISSE FÜR KLEINE DATENPAKETE[1]

Wie in Kapitel 2.6.2 beschrieben, wird erst eine TCP-Verbindung (Three-Way-Handshake) aufgebaut.

No.	Source	Destination	Protocol	Length	Info
1			Ethernet	60	[Packet size limited during capture]
2	192.168.137.1	10.0.10.100	TCP	62	49315 > 51600 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
3	10.0.10.100	192.168.137.1	TCP	62	51600 > 49315 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
4	192.168.137.1	10.0.10.100	TCP	54	49315 > 51600 [ACK] Seq=1 Ack=1 Win=17520 Len=0
5	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=20
6	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=21 Win=14600 Len=0
7	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=21 Ack=1 Win=17520 Len=20
8	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=41 Win=14600 Len=0
9	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=41 Ack=1 Win=17520 Len=20
10	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=61 Win=14600 Len=0
11	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=61 Ack=1 Win=17520 Len=20
12	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=81 Win=14600 Len=0
13	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=81 Ack=1 Win=17520 Len=20
14	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=101 Win=14600 Len=0
15	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=101 Ack=1 Win=17520 Len=20
16	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=121 Win=14600 Len=0
17	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=121 Ack=1 Win=17520 Len=20
18	10.0.10.100	192.168.137.1	TCP	54	51600 > 49315 [ACK] Seq=1 Ack=141 Win=14600 Len=0
19	192.168.137.1	10.0.10.100	TCP	74	49315 > 51600 [PSH, ACK] Seq=141 Ack=1 Win=17520 Len=20

Abbildung 82: Paketaufzeichnung des Verbindungsaufbaus (Wireshark)

Der Client sendet ein SYN-Paket an den Server mit einer Sequenznummer x (in unserem Fall 0). Der Server antwortet mit einem SYN/ACK-Paket. Dieses enthält die zuvor gesendete Sequenznummer (Seq=0) und eine Ack-Nummer. Der Client antwortet daraufhin mit einem ACK-Paket, wobei die Sequenznummer der zurückgesendeten ACK-Nummer entspricht (Abbildung 82). Damit ist der Verbindungsaufbau abgeschlossen. Alle Pakete des Verbindungsaufbaus haben eine TCP-Payload von 0, enthalten also keine Nutzdaten.

Das erste Paket nach dem Verbindungsaufbau enthält die zuvor ausgehandelte Sequenznummer von 1 und besitzt eine TCP-Payload von 20 Bytes. Der Server empfängt dieses, addiert die Payloadlänge zur Sequenznummer und teilt dem Client diese Nummer per ACK=21 mit. Da der Client die Paketlänge der Payload des ersten gesendeten Pakets kennt, kann er das ACK dem zuvor gesendeten Paket zuordnen. Gleichzeitig benutzt er diese Nummer als Sequenznummer für das nächste Paket. In der Lehre wird der Prozess der Nummerierung der Sequenznummern oft vereinfacht als fortlaufend dargestellt, in der Praxis wird jedoch die Paketgröße berücksichtigt (vgl. Kapitel 2.6.2).

Unser Testnetz besitzt durch die geringe Größe und Auslastung eine sehr niedrige RTT. Zudem ist die Datenmenge (zu versendender String) sehr klein und das PSH-Flag sorgt für eine umgehende Abgabe an den TCP-Stack des Betriebssystems. So ist zu erklären, dass die Bestätigung für einen erfolgreichen Datentransfer (ACK) vor der Übertragung des nächsten Pakets erfolgt. Somit befindet sich immer nur ein Paket im Transit (entweder Nutzdaten oder die Bestätigung). TCP erhöht für jedes erfolgreiche ACK das Congestion-Window, was aus den oben genannten Gründen jedoch keine sichtbare Auswirkung hat. Die Pakete haben eine

Gesamtgröße von 81 Bytes. Damit liegt die Nutzlast der TCP-Daten im Paket weit unter der MSS (Maximum Segment Size). Die MSS definiert in einem Netz die maximale Anzahl von Bytes, die als Nutzdaten in einem TCP-Segment versendet werden können [92].

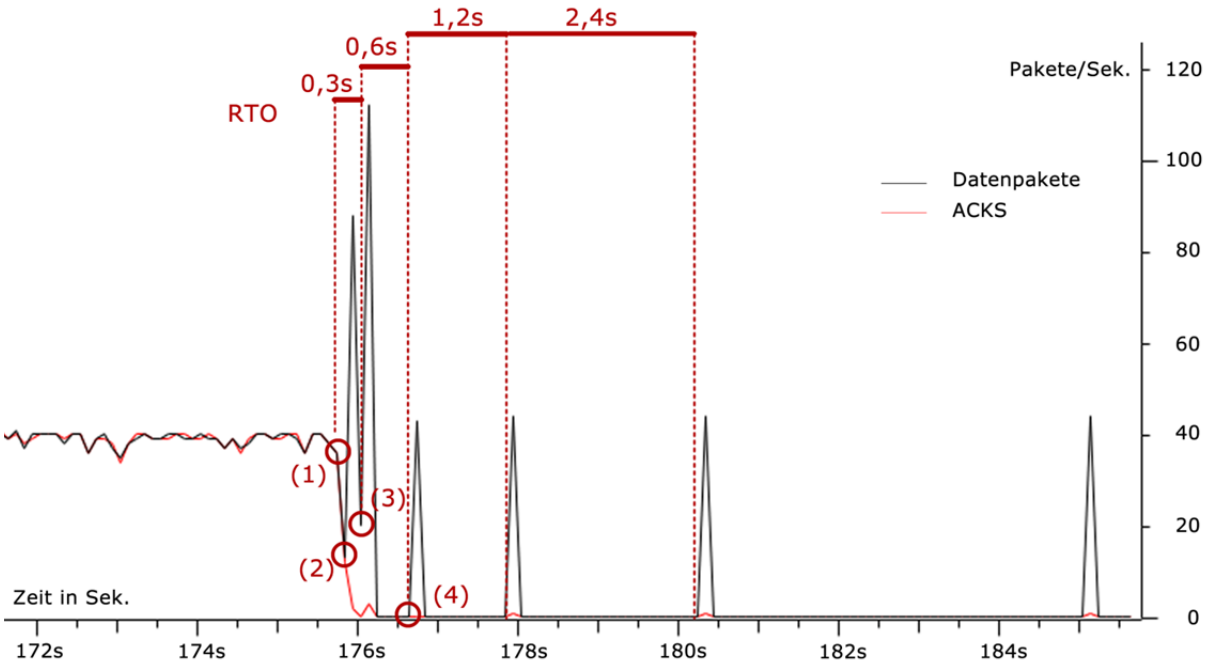
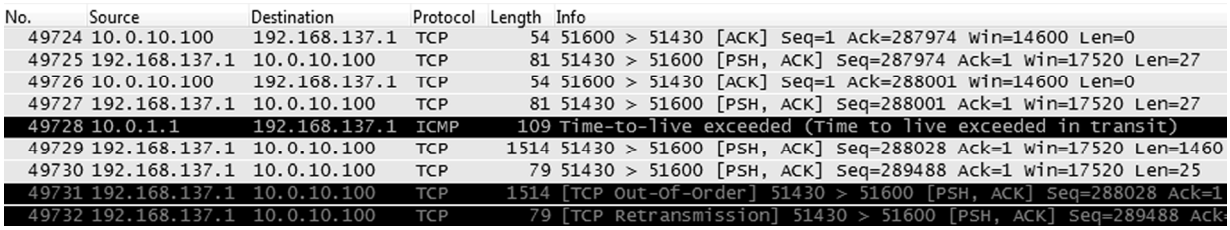


Abbildung 83: TCP-Verbindung mit zugeschalteter Schleife (Pakete pro Sekunde, TCP Reno bzw. NewReno)

Ab ca. Sekunde 175,5 (Zeitpunkt (1) Abbildung 83) aktivieren wir die Schleife. Vorher bleibt der Paketdurchsatz des messenden Routers bis auf eine leichte Varianz fast konstant. TCP konnte vorher ca. 35 Sekunden ohne Paketverlust senden. Unmittelbar nach Aktivierung der Schleife führt ein Paketverlust dazu, dass TCP eine sog. Repacketization vornimmt, d.h. mehrere Segmente werden zu einem größeren Paket zusammengefasst, sofern die MSS nicht überschritten wird. Zuerst glaubten wir beim Paketverlust an eine Anomalie durch die virtuelle Umgebung. Nachdem wir den Code des Clients änderten, sodass eine fortlaufende Nummer am zu versendenden String angehängt wird, konnten wir die ICMP-Nachricht genau zuordnen. Das Paket mit der Sequenznummer 288001 (siehe Abbildung 84) geht verloren und produziert die ICMP-Nachricht über den Paketverlust. Dieser Paketverlust ist reproduzierbar in allen Messungen aufgetreten und ist demnach keine temporäre Anomalie aufgrund der Virtualisierung der Testumgebung. Es ist in folgender Abbildung zu erkennen, dass die Paketgröße danach schlagartig von 81 auf 1514 Bytes anwächst (Seq=288028). Wir können den Paketverlust nur im Zusammenhang mit der Umkonfiguration der Router während des

Betriebs erklären, da vorher noch kein Paket in der Schleife zirkulierte. Ein durchaus vergleichbares Phänomen wird in [80] beobachtet. Hier wird beschrieben, dass Schleifen unmittelbar vor der Bildung schon Paketverluste bis zu 30% erzeugen können (siehe Kapitel 3.3.2.3 und Kapitel 3.4.3.2).



No.	Source	Destination	Protocol	Length	Info
49724	10.0.10.100	192.168.137.1	TCP	54	51600 > 51430 [ACK] Seq=1 Ack=287974 Win=14600 Len=0
49725	192.168.137.1	10.0.10.100	TCP	81	51430 > 51600 [PSH, ACK] Seq=287974 Ack=1 Win=17520 Len=27
49726	10.0.10.100	192.168.137.1	TCP	54	51600 > 51430 [ACK] Seq=1 Ack=288001 Win=14600 Len=0
49727	192.168.137.1	10.0.10.100	TCP	81	51430 > 51600 [PSH, ACK] Seq=288001 Ack=1 Win=17520 Len=27
49728	10.0.1.1	192.168.137.1	ICMP	109	Time-to-live exceeded (Time to live exceeded in transit)
49729	192.168.137.1	10.0.10.100	TCP	1514	51430 > 51600 [PSH, ACK] Seq=288028 Ack=1 Win=17520 Len=1460
49730	192.168.137.1	10.0.10.100	TCP	79	51430 > 51600 [PSH, ACK] Seq=289488 Ack=1 Win=17520 Len=25
49731	192.168.137.1	10.0.10.100	TCP	1514	[TCP Out-Of-Order] 51430 > 51600 [PSH, ACK] Seq=288028 Ack=1
49732	192.168.137.1	10.0.10.100	TCP	79	[TCP Retransmission] 51430 > 51600 [PSH, ACK] Seq=289488 Ack

Abbildung 84: Paketverlust vor Zuschaltung der Schleife (Wireshark)

Die Kenntnis über den Paketverlust löst jedoch keine direkte Retransmission des verlorenen Pakets (Seq=288001) aus. Es werden zunächst zwei weitere Pakete (Seq=288028 und Seq=289488) gesendet, d.h. das Effective Window von TCP ist zu diesem Zeitpunkt größer als 1*MSS (Zeitpunkt 1. in Abbildung 83). Das Effective Window berechnet sich aus dem Minimum zwischen Congestion Window und Advertised Window abzüglich der, sich schon im Transit befindenden, unbestätigten Bytes. Folgende Formeln dienen der Berechnung:

MaxWindow = MIN(Congestion Window, Advertised Window)
Effective Window = MaxWindow - (LastByteSent - LastByteAcked)

Extrem deutlich wird die Repacketization, wenn man sich folgenden Graphen betrachtet. Der Durchsatz gemessen in Bytes steigt durch die Schleife und der größeren Payload extrem, obwohl sich nur jeweils zwei Pakete in der Schleife befinden (ab Zeitpunkt (2) in Abbildung 83 und Abbildung 85).

Zwischen Sekunde 175,9 und 176,1 (Zeitpunkt (2) bis (3) in Abbildung 83 und Abbildung 85) zirkulieren die beiden Pakete in der Schleife bis zum Ablauf der TTL. Nachdem die TTL der in der Schleife zirkulierenden Pakete abgelaufen ist, wird der Sender wieder per ICMP von *R2* über den Ablauf der TTL und den damit einhergehenden Paketverlust informiert. Daraufhin werden zwei weitere Pakete in die Schleife gesendet. Zu beachten ist hierbei, dass es sich um keine Retransmission verlorengegangener Pakete handelt, sondern wieder um neue Daten. In der Implikation bedeutet dies, dass trotz der bisherigen 3 Paketverluste das Effective Window von TCP immer noch größer 1*MSS ist und der Retransmission Timer des

ersten verlorengegangenen Pakets noch nicht abgelaufen ist. Die Schleife besteht aus 3 Hops und wird mit einer TTL von 128 betreten, da die Messung auf R1 stattfindet, also unmittelbar hinter dem Sender. Daraus ergibt sich folgende Anzahl der Durchläufe (siehe Kapitel 4.3.4.3):

Minimale Anzahl der Durchläufe: $x = floor\ (128/3)=42$

Maximale Anzahl der Durchläufe: $x = ceil\ (128/3)=43$

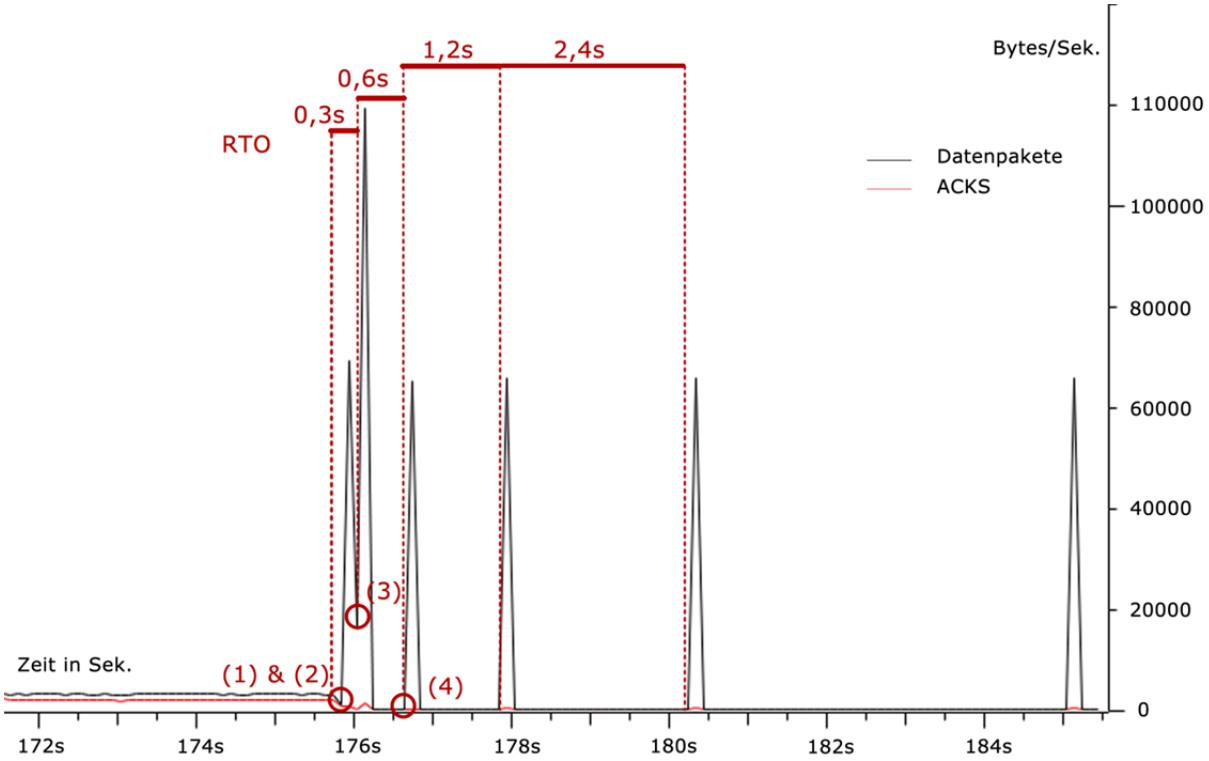


Abbildung 85: Ausschnitt der TCP-Verbindung mit zugeschalteter Schleife (Bytes pro Sekunde, TCP Reno bzw. NewReno)

Da Router *R2* die Pakete verwirft, erfahren *R1* und *R2* die maximale Anzahl der Durchläufe. Nachdem der Sender über beide Paketverluste informiert wird, sendet er zwei weitere Pakete mit neuen Daten, die wiederum in der Schleife zirkulieren und verworfen werden (ab Zeitpunkt (2) in Abbildung 83 und Abbildung 85).

Zunächst vermuten wir, dass daraufhin das Effective Window ausgeschöpft ist und ein RTO abgewartet wird. Bei genauerer Betrachtung erkennt man, dass jeweils ca. 0,13 Sek zwischen dem ersten verloren gegangenen Paket und den zwei Mal zwei weiteren gesendeten Paketen mit neuen Daten liegen (Abbildung 86). Durch die Repacketization dauert es länger bis ein Paket nahe der MSS „gefüllt“ ist und versendet werden kann, da unser Client pro Durchlauf

nur einen sehr kurzen String („This is the message“) generiert. Bevor zwei weitere Pakete versendet werden können, gibt es ein RTO für das erste Paket. Der Minimale Wert bei Windows 7 liegt entgegen der Spezifikation [41] bei 0,3 Sek. In der Spezifikation wird ein Minimalwert von 1 Sekunde angegeben (siehe Kapitel 2.6.2). Für Verbindungen mit sehr kleiner RTT ist dieser Minimalwert zu groß gewählt. Nachdem ein RTO (Retransmission Timeout) erfolgt, werden die ersten verlorengegangenen Daten erneut versendet (ungefähr bei Zeitpunkt (3) in Abbildung 83 und Abbildung 85) und das Congestion Window auf 1 gesetzt. Auch hier bewegt sich TCP an die Grenze der MSS mit einer gemessenen Gesamtpaketgröße von 1514 Bytes. Zu beachten ist, dass die Zeitpunkte (2) und (3) in Abbildung 83 und Abbildung 85 nicht ganz genau bestimmt werden können, da unser Netz eine sehr kleine RTT besitzt. Dadurch ist es der bei Zeitpunkt (3) größte Ausschlag noch oben zu erklären, obwohl die Paketgrößen der in die Schleife gesendeten Pakete identisch ist. Bei Zeitpunkt (4) in Abbildung 83 und Abbildung 85 läuft der verdoppelte Retransmission Timer (2*0,3 Sek.) erneut ab.

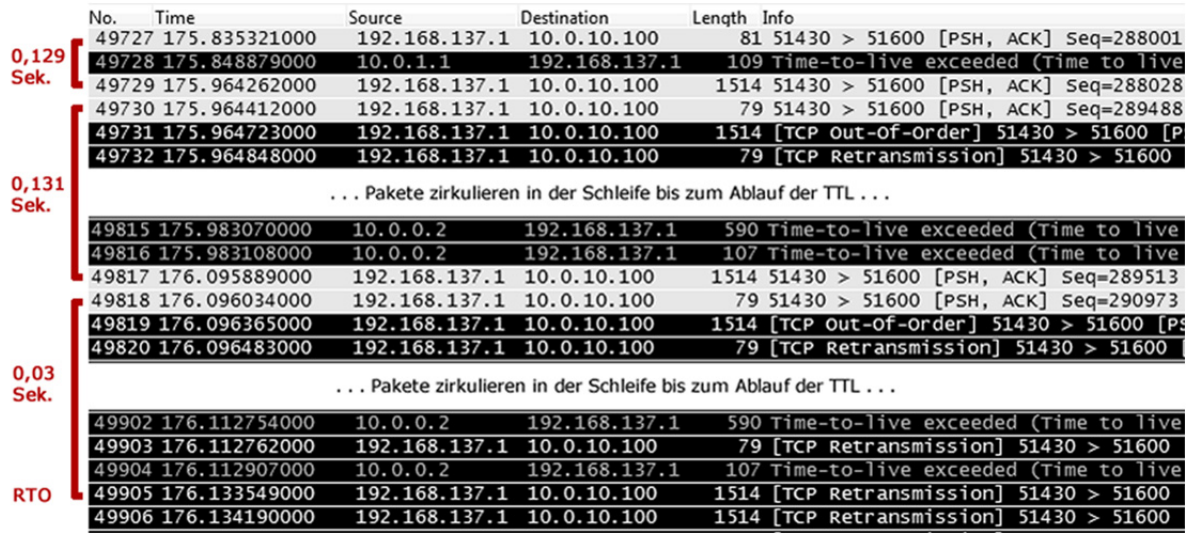


Abbildung 86: Zeitlicher Verlauf der Paketverluste bis zum Ablauf des Retransmission Timeouts (Wireshark)

Der sog. Exponential Backoff des Retransmission Timers lässt sich am leichtesten zwischen den Sekunden 176 und 185 in Abbildung 83 und Abbildung 85 erkennen. Die Wartezeit wird nach jedem Paketverlust verdoppelt und vergrößert sich von 0,3 Sek. auf 4,8 Sekunden. Danach brachen wir die Messung ab.

Aus obiger Testauswertung lässt sich schließen, dass sich TCP bei kleinen Datenpaketen sehr schnell zurückzieht und das Netz nach Paketverlusten nicht unnötig belastet. Der Schlüssel zum Grad der Belastung liegt im Staukontrollverfahren und dem zugehörigen Congestion Window. Unsere Datenpakete werden unter Windows 7 erzeugt. Windows 7 verwendet standardmäßig TCP Reno bzw. NewReno. Dabei ist entscheidend wie das Congestion Window auf Paketverluste reagiert. In unserem Fall wird Additive Increase Multiplicative Decrease (AIMD) verwendet. Das Congestion Window wird bei einem Paketverlust halbiert, so dass noch weitere Pakete vor Ablauf des RTO versendet werden können [73]. Im schlimmsten Fall sendet TCP Daten in der Größe des Effective Window in die Schleife. Der Retransmission Timer kann einen entscheidenden Einfluss haben, wenn das Congestion Window sehr groß ist bzw. wenige Daten anfallen und die Zeitspanne zwischen dem Versenden der Pakete relativ groß ist. Hier kann ein niedrig gewähltes RTO verhindern, dass das komplette Effective Window in die Schleife gesendet wird. Zu beachten ist hierbei auch, dass ein Paketverlust das Congestion Window halbiert, während ein Retransmission Timeout das Congestion Window auf 1 setzt.

In einem weiteren Versuch stellen wir Compound TCP [93] als Staukontrollverfahren beim Sender ein und werten die Ergebnisse aus. Compound TCP nutzt die Summe aus zwei Fenstern zur Berechnung des Congestion Window. Eins arbeitet streng nach dem AIMD-Prinzip und ein weiteres wird aus der aktuellen Verzögerung berechnet. Erhöht sich die RTT der Verbindung wird das zweite Fenster verkleinert, um den weiteren linearen Anstieg des AIMD-Fensters zu kompensieren. Im Falle eines Paketverlusts wird jedoch auch hier das Congestion Window halbiert. Aus diesem Grund erwarten wir ähnliche Ergebnisse. Eventuell hat Compound TCP ein etwas größeres Effective Window, da wir keine Erhöhung der RTT haben, sondern direkt Paketverluste beim Aktivieren der Schleife eintreten. Dies sollte aber keine Auswirkungen haben, da wir vorher ein Retransmission Timeout erwarten wie im Fall von TCP Reno bzw. NewReno.

Das Verhalten von Compound TCP bei einer Schleife ist komplett analog zu Reno bzw. NewReno. Vor dem Zuschalten der Schleife geht ein erstes Paket verloren (Zeitpunkt (1) in Abbildung 87), worauf TCP eine Repacketization vornimmt und zwei Mal zwei weitere Datenpakete in die Schleife sendet, eins davon jeweils nahe der MSS von TCP mit einer Gesamtpaketgröße von 1514 Bytes (ungefähr bei Zeitpunkt (2)). Daraufhin läuft der Retransmission Timer des ersten verlorengegangenen Pakets ab (ungefähr bei Zeitpunkt (3)).

Das Congestion Window wird auf 1 gesetzt und der Retransmission Timer bei jedem weiteren Paketverlust verdoppelt (Zeitpunkt (4)).

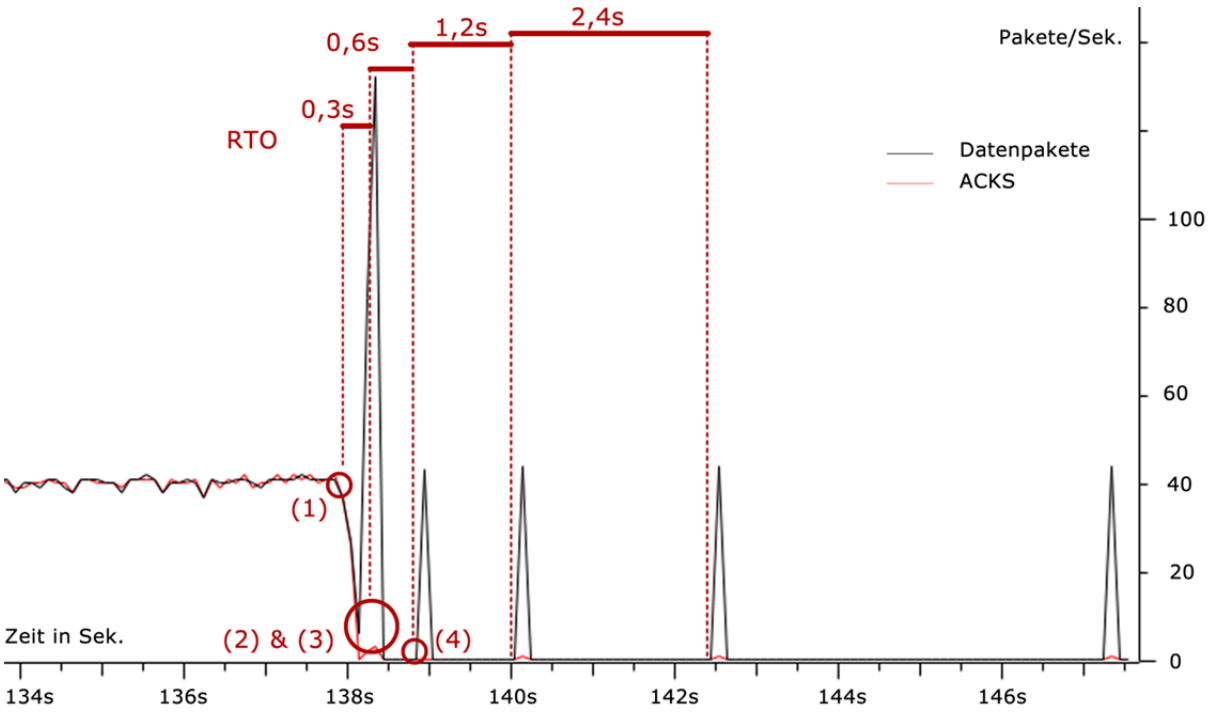


Abbildung 87: TCP-Verbindung mit zugeschalteter Schleife (Pakete pro Sekunde, Compound TCP)

4.4.1.3 TESTERGEBNISSE FÜR GROSSE DATENPAKETE[1]

Um das Verhalten bei größeren Datenpaketen zu evaluieren, änderten wir unseren Client derart ab, dass er größere Strings generiert und versendet. Dadurch erreichen wir schon zu Beginn eine TCP-Nutzlast näher der MSS und es werden mehrere Pakete versendet, bevor eine Bestätigung eintrifft. Zudem hoffen wir darauf, dass trotz Repacketization genug Daten in kurzer Zeit anfallen, um klar erkennen zu können, ob das Effective Window von TCP komplett ausgenutzt werden kann. Auch diesen Test führen wir ca. 20 Mal aus und werten das Ergebnis aus. Zuerst verwenden wir das von Windows 7 standardmäßig verwendete TCP Reno bzw. NewReno.

Da größere Pakete einen Knoten länger belasten bzw. die zeitliche Lücke zwischen einzelnen Paketen ebenfalls größer ist, können wir den im vorigen Kapitel aufgetretenen Paketverlust vor der Bildung der Schleife nur sporadisch reproduzieren. Dies bestätigt aber unsere Vermutung, dass der Paketverlust aufgrund der Zustandsänderung des Routers (Löschen der

alten Route, Installieren der neuen Route) zustande kommt. Auch hier löst ein Paketverlust eine Repacketization von TCP aus, der aber die Paketgrößen nicht signifikant verändert, da wir uns schon vorher nahe am Maximum bewegen. Der Paketverlust bei Schleifenbildung hat in unserer Umgebung keinen Einfluss darauf, wieviel Pakete zu Anfang hintereinander in die Schleife gesendet werden. Je nach Größe des Advertised Window und des Congestion Window kann dies aber durchaus in anderen Szenarien Einfluss auf das Effective Window haben und damit einhergehend wie viele Pakete in die Schleife gesendet werden.

Wir aktivieren die Schleife nach ca. 128 Sekunden (Zeitpunkt (1) in Abbildung 89). Vorher konnte TCP ohne Beeinträchtigung 20 Sekunden senden und empfangen, d.h. das Effective Window ist definitiv größer als 1*MSS. Kurz vor Aktivierung der Schleife versendet TCP drei Pakete nahe der MSS ohne auf eine Bestätigung zu warten (Seq=1753774, Seq=1755145, Seq=1756605) (Abbildung 88).

No.	Time	Source	Destination	Length	Info
5861	128.552	10.0.10.100	192.168.137.1	54	51600 > 49578 [ACK] Seq=1 Ack=1751121 Win=42240 Len=0
5862	128.552	10.0.10.100	192.168.137.1	54	51600 > 49578 [ACK] Seq=1 Ack=1752403 Win=42240 Len=0
5863	128.568	192.168.137.1	10.0.10.100	1425	49578 > 51600 [PSH, ACK] Seq=1752403 Ack=1 Win=17408 Len=1371
5864	128.570	10.0.10.100	192.168.137.1	54	51600 > 49578 [ACK] Seq=1 Ack=1753774 Win=42240 Len=0
5865	128.586	192.168.137.1	10.0.10.100	1425	49578 > 51600 [PSH, ACK] Seq=1753774 Ack=1 Win=17408 Len=1371
5866	128.619	192.168.137.1	10.0.10.100	1514	49578 > 51600 [PSH, ACK] Seq=1755145 Ack=1 Win=17408 Len=1460
5867	128.620	192.168.137.1	10.0.10.100	1336	49578 > 51600 [PSH, ACK] Seq=1756605 Ack=1 Win=17408 Len=1282
5868	128.621	192.168.137.1	10.0.10.100	1514	[TCP Out-Of-Order] 49578 > 51600 [PSH, ACK] Seq=1755145 Ack=1
5869	128.621	192.168.137.1	10.0.10.100	1336	[TCP Retransmission] 49578 > 51600 [PSH, ACK] Seq=1756605 Ack=
5870	128.622	192.168.137.1	10.0.10.100	1514	[TCP Out-Of-Order] 49578 > 51600 [PSH, ACK] Seq=1755145 Ack=1
5871	128.622	192.168.137.1	10.0.10.100	1336	[TCP Retransmission] 49578 > 51600 [PSH, ACK] Seq=1756605 Ack=
5872	128.622	192.168.137.1	10.0.10.100	1514	[TCP Out-Of-Order] 49578 > 51600 [PSH, ACK] Seq=1755145 Ack=1
5873	128.622	192.168.137.1	10.0.10.100	1336	[TCP Retransmission] 49578 > 51600 [PSH, ACK] Seq=1756605 Ack=
5874	128.622	192.168.137.1	10.0.10.100	1514	[TCP Out-Of-Order] 49578 > 51600 [PSH, ACK] Seq=1755145 Ack=1

Abbildung 88: Paketaufzeichnung unmittelbar vor Aktivierung der Schleife (Wireshark, TCP Reno bzw. NewReno, große Datenpakete))

Das Paket mit der Sequenznummer 1753774 kommt noch beim Empfänger an, jedoch wird die Bestätigung durch die zwei in der Schleife zirkulierenden Pakete verzögert.

TCP versendet daraufhin zwei Mal zwei weitere Pakete in die Schleife (Zeitpunkt (2) in Abbildung 89), die dort bis zum Ablauf der TTL zirkulieren. Nach Ablauf der TTL wird der Empfänger wieder per ICMP-Nachricht über den Paketverlust informiert und wartet den Ablauf des Retransmission Timers ab, d.h. das Effective Window wurde ausgeschöpft. Nach dessen Ablauf wird das erste verworfene Paket erneut übermittelt. Zwischen dem Paketverlust und der Retransmission (Zeitpunkt (3) in Abbildung 89) liegen wieder 0,3 Sek., d.h. der minimal zulässige Wert des RTO von Windows 7. Die Wartezeit zwischen den Versuchen wird nach jedem Paketverlust verdoppelt (Exponential Backoff). Nach 0,6 Sekunden

(Zeitpunkt (4) in Abbildung 89) veranlasst TCP eine erneute Retransmission. Diese Schritte wiederholen sich und TCP zieht sich sehr schnell zurück. Dass das Effective Window vor dem RTO ausgeschöpft wird, wird deutlich, wenn wir uns den zeitlichen Verlauf genauer betrachten.

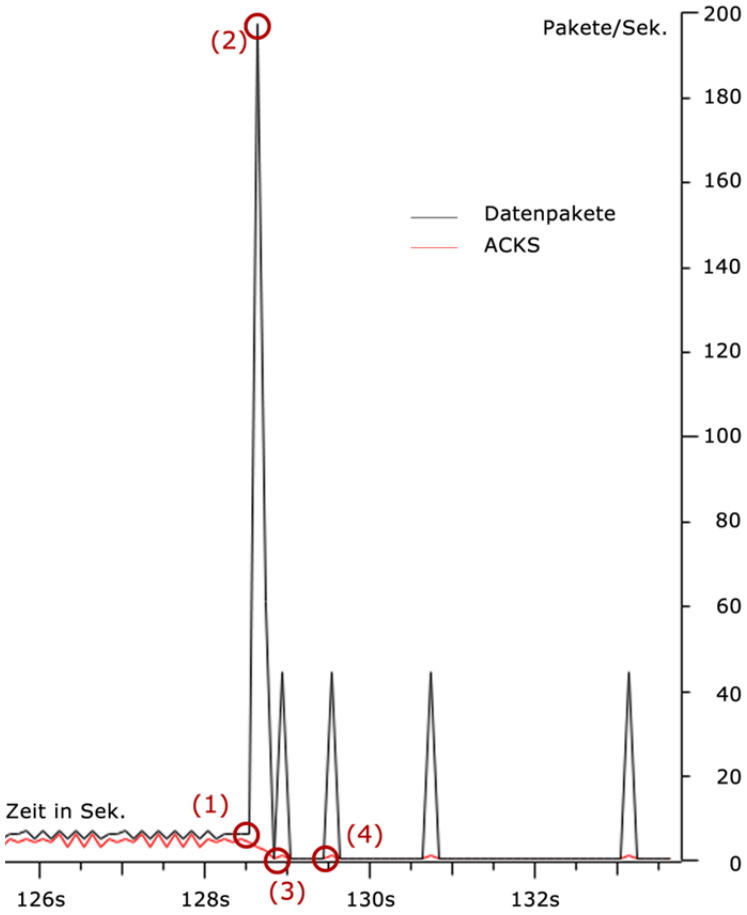


Abbildung 89: Ausschnitt der TCP-Verbindung mit zugeschalteter Schleife (Bytes pro Sekunde, TCP Reno bzw. NewReno, große Datenpakete)

Es ist deutlich in Abbildung 90 zu erkennen, dass ca. 0,035 Sekunden vergehen, bis neue Daten vorliegen und neue Pakete versendet werden können. Die Lücke der letzten versendeten Pakete bis zum RTO ist bedeutend größer (0,2 Sekunden). Das bedeutet, dass TCP das Effective Window komplett ausreizen kann und dann das RTO abwartet.

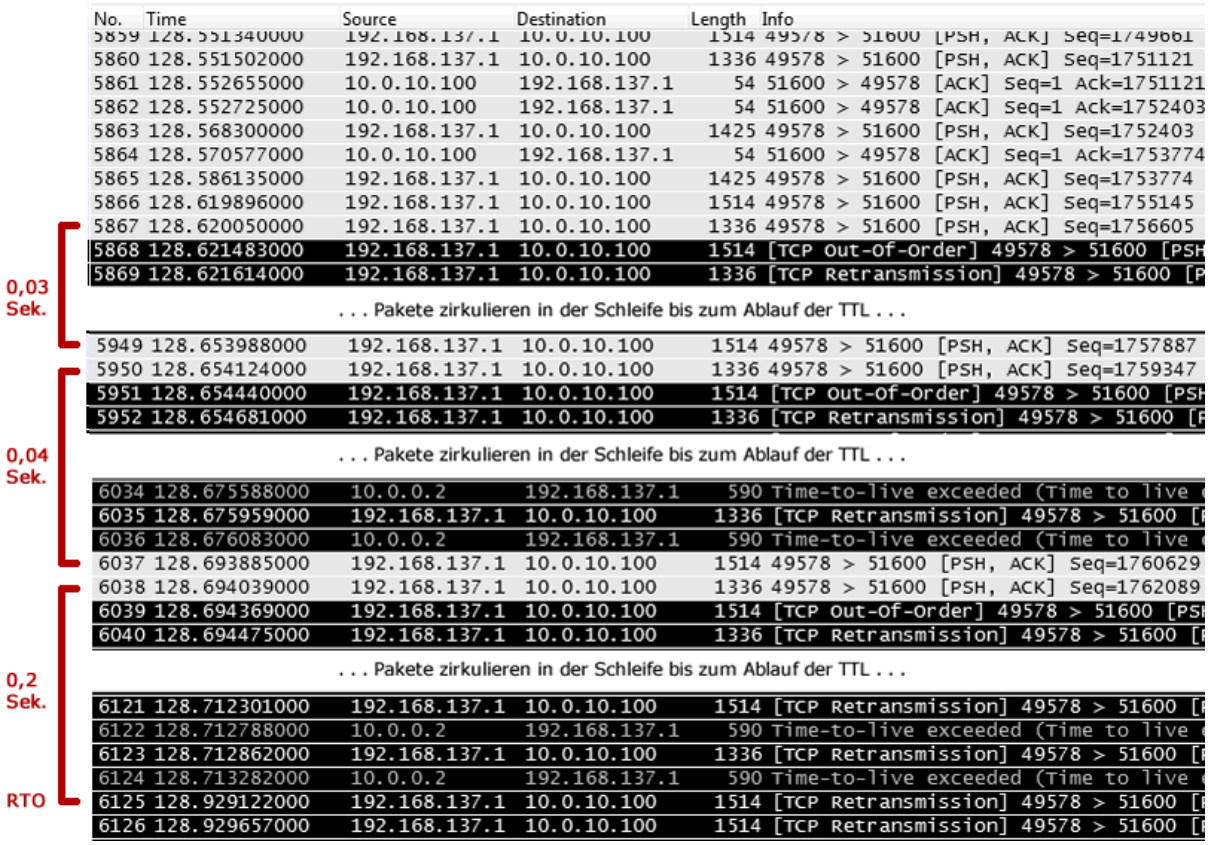


Abbildung 90: Zeitlicher Verlauf der Paketverluste bis zum Ablauf des Retransmission Timeouts (Wireshark, TCP Reno bzw. NewReno, große Datenpakete)

Diesen Test wiederholen wir mit Compound TCP, um eventuelle Unterschiede zu entdecken bzw. auszuschließen. Auch hier lassen wir das Szenario ca. 20 Mal durchlaufen, um temporäre Anomalien auszuschließen. Wie zuvor tritt ein Paketverlust unmittelbar vor Bildung der Schleife nur sporadisch auf. Nach genauerer Analyse entdecken wir, dass unter Compound TCP vier Mal zwei Pakete in die Schleife gesendet werden statt wie zuvor drei Mal zwei Pakete unter TCP Reno bzw. NewReno (siehe Abbildung 90). Zuerst vermuten wir, dass die abweichende Berechnung des Congestion Windows sich dafür verantwortlich zeichnet und ein größeres Congestion Window beim Eintritt in die Schleife existiert. Nach genauerer Betrachtung fällt uns auf, dass in unseren Tests mit TCP Reno bzw. NewReno die TCP-Verbindung ca. 20 Sekunden Zeit hatte, das Congestion Window und Advertised Window aufzubauen. Bei unserem Test mit Compound TCP hatte die Verbindung ca. 30 Sekunden Zeit, die Fenster aufzubauen. Wir entschließen uns dazu, die Tests mit TCP Reno bzw. NewReno zu wiederholen. Hat TCP Reno bzw. NewReno 30 Sekunden Zeit, die Fenster aufzubauen, sendet es analog vier Mal zwei Pakete in die Schleife. Wir können demnach

abschließend feststellen, dass Compound TCP sich in unserem Szenario komplett analog zu Reno bzw. NewReno verhält. In weiteren Tests verdoppeln wir die Zeit der störungsfreien TCP-Verbindung auf 60 Sekunden, um auszuschließen, dass TCP evtl. noch mehr Pakete in die Schleife sendet. Das Gegenteil ist der Fall, d.h. bei vier Mal zwei Paketen ist die maximale Belastung der Schleife durch eine TCP-Verbindung in unserer Umgebung erreicht.

4.4.1.4 FAZIT[1]

Durch eine komplexe Fluss- und Staukontrolle schafft es TCP, kurzfristig auf Paketverluste durch die auftretende Schleife zu reagieren. Ein sinnloses Fluten des Netzes mit Paketen wird vermieden. Eine Schlüsselrolle spielt dabei das Effective Window (Minimum des Advertised Window und/oder Congestion Window) und das Retransmission Timeout. Das Effective Window bestimmt bei einem hinreichend großen RTO, wie viele Pakete ohne Bestätigung zu Anfang in die Schleife gesendet werden. Ist das Effective Window ausgeschöpft, wird der Ablauf des RTO abgewartet und eine Retransmission des ersten verlustigen Pakets veranlasst. Wie bei den Tests mit den kleinen Datenpaketen gesehen, kann das RTO bei einer Verbindung mit einer kurzen RTT eintreten bevor das Effective Window von TCP ausgeschöpft ist und das Verhalten von TCP bei Schleifen zusätzlich verbessern. Das Minimieren des Congestion Window auf den Wert 1 nach einem RTO und die Verdoppelung der Wartezeit zwischen den folgenden Versuchen lässt ein über die Zeit immer geringeres Paketaufkommen in der Schleife zu. TCP gelingt ein guter Trade-Off zwischen Paketverlusten und einer fortwährenden Prüfung der Verbindung.

4.4.2 UDP[2]

Als nächstes wollen wir das Verhalten von UDP-Paketen in einer Forwarding-Schleife testen. Dazu werden wir wie bei unserem Test des Transport Control Protocol eine störungsfreie Datenübertragung starten und inmitten derer die Forwarding-Schleife erzeugen.

Da wir in Kapitel 2 auch für UDP Pakete eine Verteilung auf sehr kleine und sehr große Pakete nahe der MTU festgestellt haben, werden wir auch diesen Test wieder mit der Übertragung von kleinen und großen Paketen durchführen.

4.4.2.1 PYTHON SCRIPT UDP²

Zum Testen benutzen wir wie bei TCP wieder eine Client-Server-Variante, die auf Python basiert. Auf Host B, mit der IP-Adresse 10.0.10.100, wird dazu wieder die Server-Variante gestartet, welche auf Port 54600 auf eingehende Verbindungen wartet.

UDP - Server

```
import socket
import sys

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the port
server_address = ('192.168.1.134', 54600)

print ('starting up on %s port %s' % server_address)
sock.bind(server_address)

while True:
    print ('waiting to receive message')
    data, address = sock.recvfrom(4096)

    print ('received %s bytes from %s' % (len(data), address))
    print (data)
```

Der dazugehörige Client baut entsprechend eine Verbindung zu dem geöffneten Port des Servers auf und sendet anschließend in einer Schleife fortlaufend 30.000 UDP Pakete, sofern die Übertragung nicht abgebrochen wird.

Um den Test mit kleinen und großen Paketen durchzuführen wird analog wie in Abschnitt 4.4.1. der String *message* entsprechend in der Länge verändert um größere Pakete zu generieren.

UDP - Client

```
import socket
import sys

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
i=0
server_address = ('10.0.10.100', 54600)
message = 'This is the message. It will NOT be repeated!'

try:
 while i <= 100:
    # Send data
    print ('sending "%s"' % message)
    sent = sock.sendto(message.encode(), server_address)
    i = i + 1

finally:
    print ('closing socket')
    sock.close()
```

4.4.2.2 TESTERGEBNISSE[2]

Anders als bei TCP ist direkt erkennbar, dass die Übertragung der UDP-Pakete nach dem Erzeugen der Forwarding-Schleife nicht abbricht und fortlaufend Pakete in die existierende Forwarding-Schleife gesendet werden.

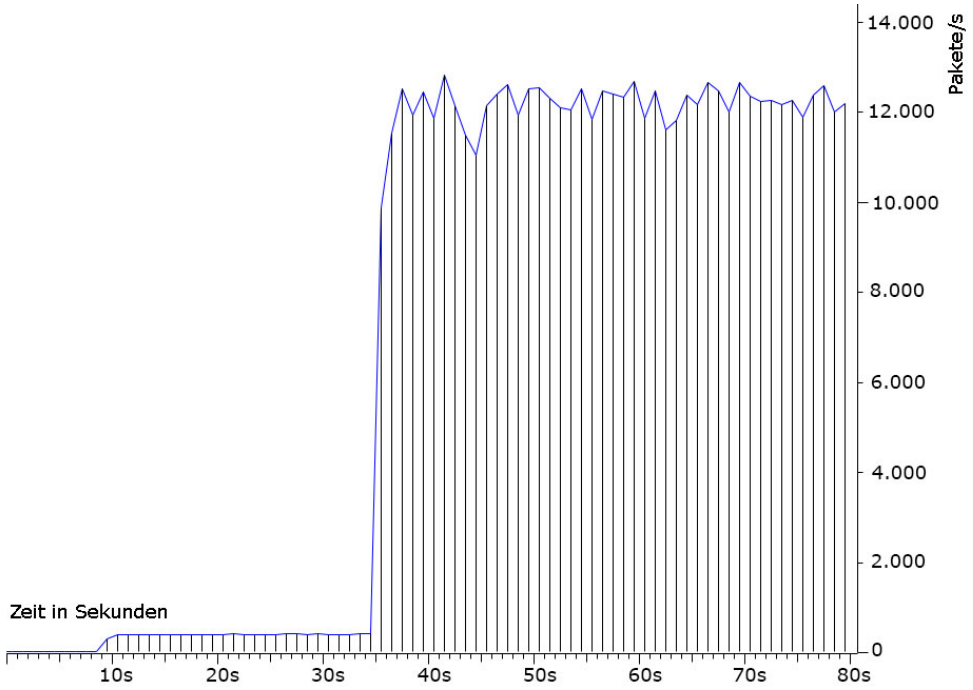


Abbildung 91: 46 Byte UDP Pakte in Forwarding-Schleife, TTL 128 – Pakete/Sekunde

Im ersten Test initiieren wir eine Übertragung von UDP-Paketen mit einer Gesamtgröße von 46 Byte pro IP-Paket. Host A in unserem Testszenario ist dabei ein System mit dem Betriebssystem Windows7 und sendet dementsprechend standadtmäßig IP-Pakete mit einer TTL von 128 ab.

Bevor wir die Forwarding-Schleife in unserem Testszenario erzeugen, messen wir eine sehr konstante Übertragung von etwa 400 Paketen pro Sekunde (Abbildung 91) mit einem Datendurchsatz von 25.000 Bytes pro Sekunde (Abbildung 92).

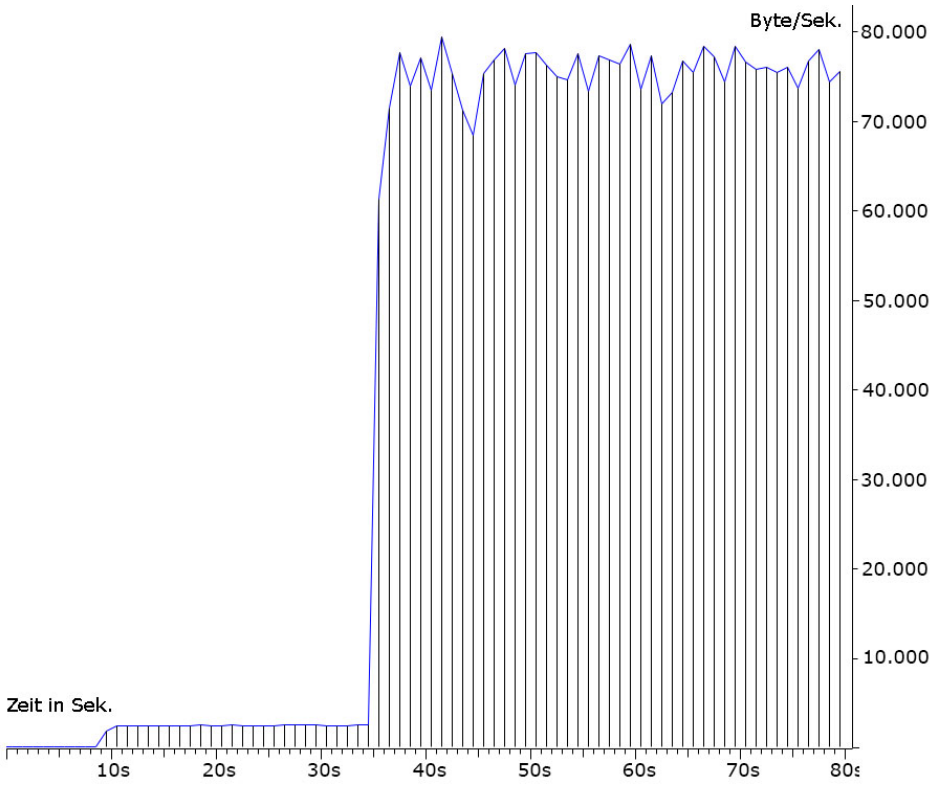


Abbildung 92: 46 Byte UDP Pakte in Forwarding-Schleife, TTL 128 – Byte/Sekunde

Nachdem die Forwarding-Schleife in unserem Test nach etwa 35 Sekunden erzeugt wurde, steigt der Verkehr in unserer Testumgebung stark an. Pro Sekunde messen wir nun 12.000 Pakete (Abbildung 91) welche die Verbindung von R1 und R2 pro Sekunde passieren und der Datendurchsatz steigt auf etwa 780.000 Byte pro Sekunde (Abbildung 92). Dies bedeutet also, dass sich die Anzahl der Pakete und des Datenvolumen innerhalb der Forwarding-Schleife um etwa den Faktor 30 erhöht hat.

Als nächstes testen wir das Verhalten von großen UDP-Paketen in der Forwarding-Schleife. Dazu starten wir die Übertragung mit einer Paketgröße von 1442 Byte. Analog zu unseren Ergebnissen mit den kleinen UDP-Paketen erhalten wir anfänglich einen sehr konstanten Datendurchsatz, welcher für die signifikant größeren Datenpakete bei etwa 60 Paketen pro Sekunde liegt (Abbildung 93). Das Datenvolumen, welches pro Sekunde vor dem Erzeugen der Forwarding-Schleife gemessen wird, liegt bei etwa 97.000 Bytes (Abbildung 94).

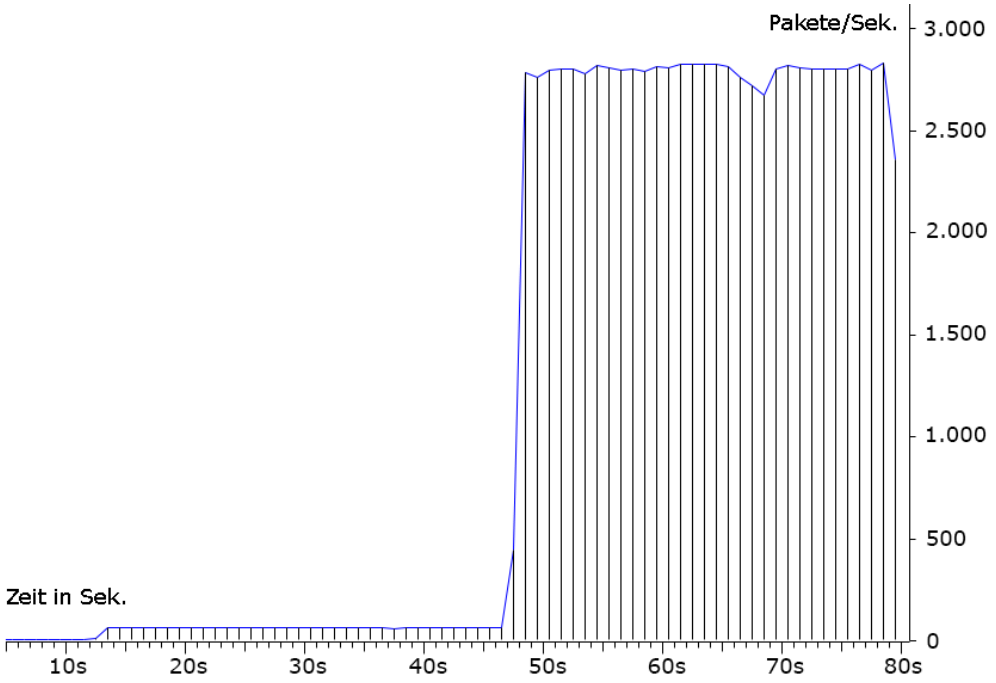


Abbildung 93: 1440 Byte UDP Pakete in Forwarding-Schleife, TTL 128 - Pakete/Sekunde

In diesem zweiten Test erzeugen wir die Forwarding-Schleife nach etwa 46 Sekunden nach dem Start unserer Messung und auch hier ist wieder ein sehr starker Anstieg des Paket- und Datenaufkommen zu verzeichnen. Die Anzahl der Pakete steigt dabei auf etwa 2580 pro Sekunde (Abbildung 93) und das Datenvolumen erhöht sich gleichermaßen auf ca. 4.000.000 Bytes pro Sekunde (Abbildung 94). Dies bedeutet also bei einer Übertragung von großen UDP-Paketen, dass sich das Datenaufkommen in der Forwarding-Schleife um mehr als das 40-fache erhöht.

Im Vergleich zu den Testergebnissen mit 46 Byte großen UDP Paketen ist festzustellen, dass dabei der anfängliche Durchsatz an Datenpaketen viel höher ist, das Datenvolumen jedoch bei

der Übertragung von 1442 Byte großen Paketen wesentlich höher liegt. Dies lässt sich dadurch erklären, dass die Puffer der Router eine größere Anzahl von Datenpaketen aufnehmen können, wenn diese eine geringere Größe besitzen und dementsprechend mehr kleine als große Datenpakete verarbeiten können. Da jedes Paket jedoch vom Router auf seine Zieladresse hin überprüft werden muss, bedeutet dies einen größeren Rechenaufwand bei der Verarbeitung vieler Pakete mit geringer Größe. Dadurch ist im Verhältnis der Durchsatz bezüglich des Datenvolumens bei großen UDP-Paketen höher, auch wenn hier weniger Pakete pro Sekunde verarbeitet werden.

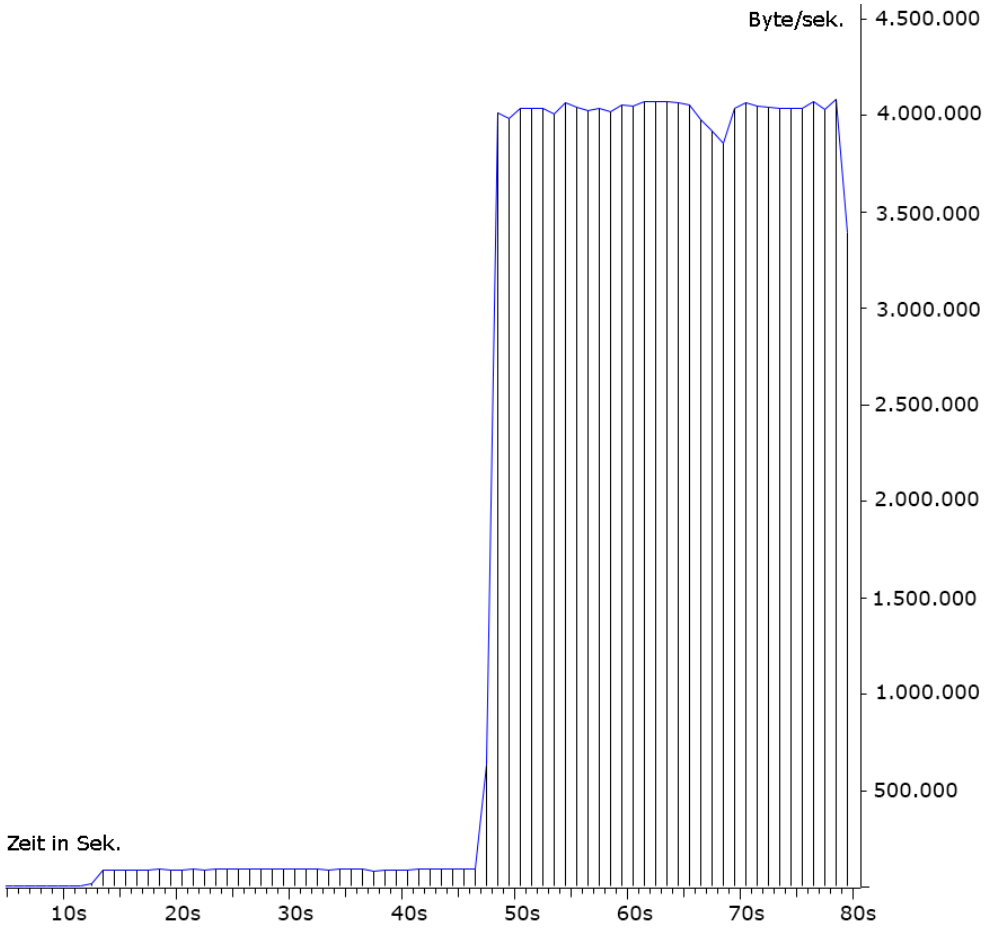


Abbildung 94: 1440 Byte UDP Pakte in Forwarding-Schleife, TTL 128 - Bytes/Sekunde

Um die aus Abschnitt 4.3. bekannten Auswirkungen der TTL auf die Belastung der Router in einer Forwarding-Schleife zu untersuchen, führen wir die gleichen Tests mit veränderten Bedingungen durch. Für die folgende Untersuchung reduzieren wir den Wert der TTL in unserem Windows7 System auf 64 und wiederholen unsere Messungen.

Der Versuch mit einer Paketgröße von 46 Byte zeigt dabei die aus dem ersten Test bekannten anfänglichen Datenraten von 400 Paketen und 25.000 Byte pro Sekunde. Nach dem Erzeugen der Forwarding-Schleife ist auch hier ein Vielfaches an Datenverkehr gegenüber der normalen Übertragung zu beobachten (Abbildung 95).

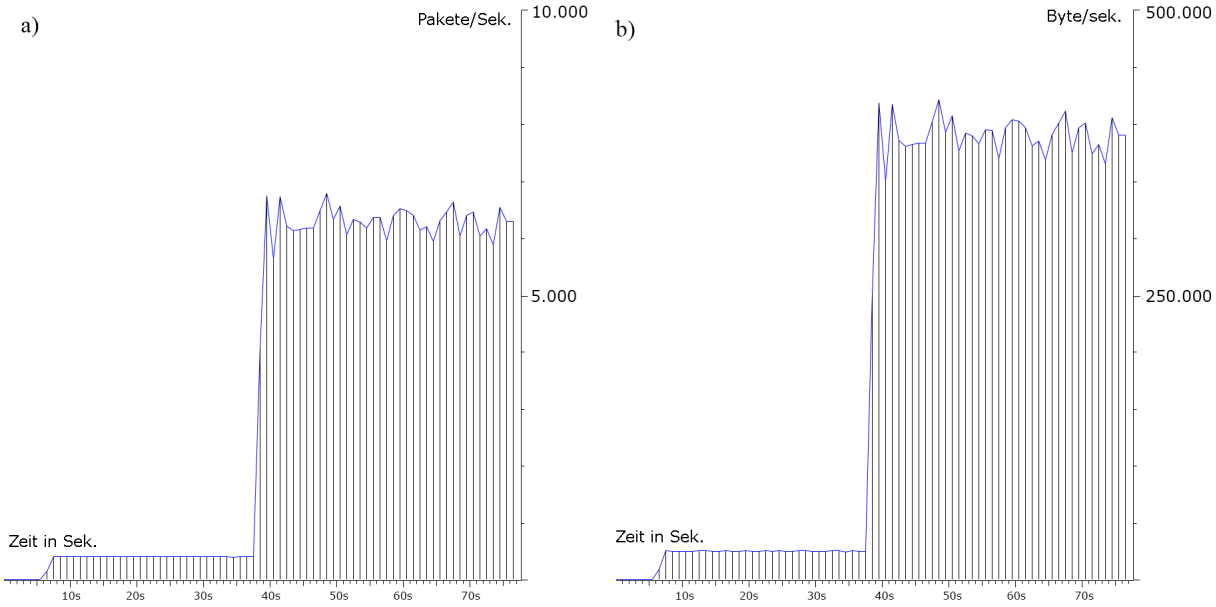


Abbildung 95: 46 Byte UDP Pakete in Forwarding-Schleife, TTL=64 a) Pakete/s b) Byte/s

Im Vergleich zur erhöhten Belastung bei IP-Paketen mit einer TTL von 128, zeigt sich in diesem Versuch ein deutlich geringerer Anstieg des Datenverkehrs innerhalb der Forwarding-Schleife. Die Anzahl der Datenpakete steigt dabei auf etwa 6.000 Pakete pro Sekunde (Abbildung 95 a)) und auf ein Datenvolumen von etwa durchschnittlich 390.000 Byte/s (Abbildung 95 b)). Verglichen mit den Testwerten aus unserem Versuch mit einer TTL von 128 zeigt sich, dass für eine TTL von 64 das Datenaufkommen von UDP-Paketen nur halb so groß ist. Dies bestätigt also unsere Ergebnisse aus 4.3., dass die Belastung innerhalb einer Forwarding-Schleife entscheidend von der TTL eines IP-Paket abhängt.

Vergleichbare Ergebnisse erhalten wir auch mit UDP-Paketen mit einer Größe von 1442 Byte. Auch hier ist bei einer TTL von 64 der Anstieg von anfänglichen 60 Paketen/s auf ca. 1.300 Pakete/s (Abbildung 96 a)) nur etwa halb so groß wie beim vergleichbaren Versuch mit einer TTL von 128. Analog steigt auch das Datenvolumen von 87.000 Byte/s auf ~1.960.000 Byte/s (Abbildung 96 b)) in der Forwarding-Schleife nur etwa um die Hälfte an.

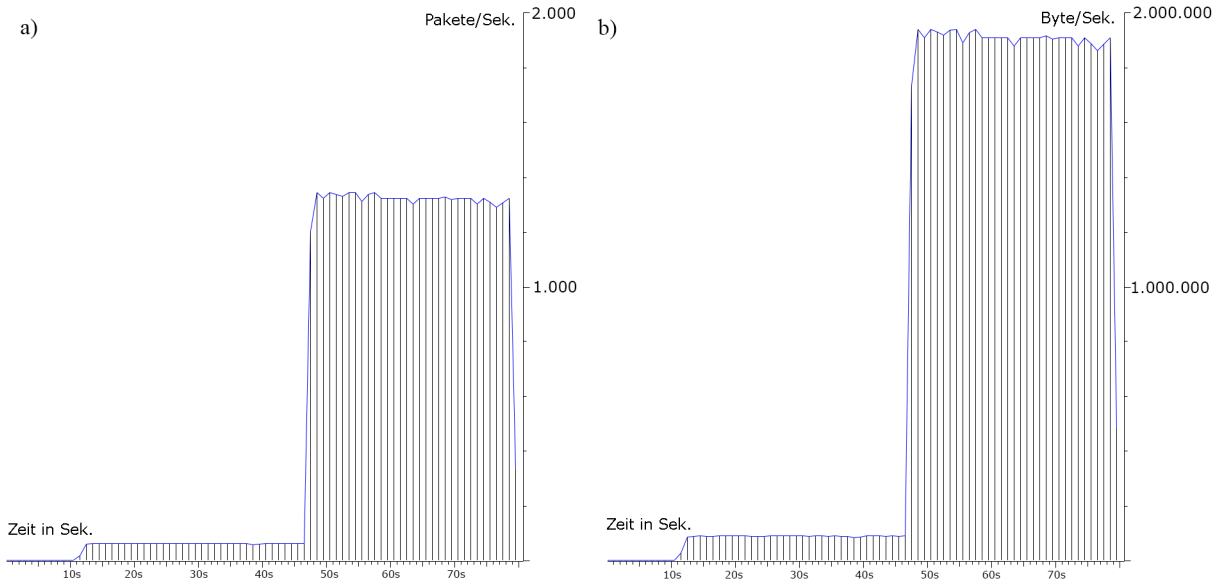


Abbildung 96: 1442 Byte UDP Pakete in Forwarding-Schleife, TTL 64 a) Pakete/s b) Byte/s

4.4.2.3 FAZIT[2]

Im Gegensatz zu einer Verbindung die auf dem TCP als Transportprotokoll basiert, bietet eine Datenübertragung über UDP keinen Schutz vor einer Forwarding-Schleife. Wie in Abschnitt 4.4.1. beobachtet unterbricht die Staukontrolle des Transport Control Protokoll umgehend die Datenübertragung und beendet die Verbindung. Beim User Datagram Protocol hingegen wird die Übertragung der Daten vom Sender unverändert fortgesetzt und so die Forwarding-Schleife mit Datenpaketen belastet. Der Grund hierfür ist, wie in Kapitel 2 beschrieben, dass UDP ein verbindungsloses Transportprotokoll ist und so keinerlei Rückmeldung vom Empfänger über eine erfolgreiche Datenübertragung erhält. Dadurch kann mit dem UDP auch keine Stau- oder Flusskontrolle ausgeführt werden, welche zu einer Regulierung des Datenverkehrs im Falle einer Forwardingschleife eingreifen könnte. Die Datenpakete zirkulieren bei UDP daher entsprechend ihrer TTL in dieser Schleife bis die Übertragung durch den Sender abgeschlossen ist. Dabei ist, wie in Abschnitt 4.3. festgestellt, die Größe der Schleife sowie die TTL maßgebend dafür, wie groß die Belastung innerhalb der Forwardingschleife zunimmt. Wie bereits festgestellt steht dabei die TTL und die Anzahl der Router, welche die Forwarding-Schleife aufbauen, in einem festen Verhältnis. Bei einer größeren TTL zirkulieren die Datenpakete entsprechend länger in der Schleife, bis sie mit dem TTL-Wert 0 vom Router verworfen werden. Erstreckt sich die Schleife über eine größere Anzahl an Routern, verteilt sich die Belastung auf diesen und auch die TTL wird schneller

dekrementiert, also verringert sich die Anzahl der Schleifendurchläufe eines Pakets. Wie wir aber in Kapitel 3 beschrieben haben, erstreckt sich eine Forwarding-Schleife für gewöhnlich über eine kleine Anzahl an Routern, so dass im Falle einer Schleife immer von einer großen Belastung der beteiligten Router mit UDP Datentransfer auszugehen ist.

Betrachtet man die beschriebene Entwicklung des Internetverkehrs in Kapitel 2, so ergibt sich aus dem steigenden Datenverkehr an UDP basierten Paketen ein zunehmendes Risiko. Auch abseits von Forwarding-Schleifen sorgt der zunehmende UDP Verkehr dafür, dass TCP-basierten Verbindungen im Verhältnis weniger Bandbreite zur Verfügung steht. Im Fall, dass dieser Internetverkehr in eine Forwarding-Schleife gerät, sorgt dies unweigerlich dafür, dass die beteiligten Router überlastet werden und auch Pakete, welche nicht in der Schleife zirkulieren, verworfen werden.

Diese Ergebnisse bestätigen auch die Gefahr von DoS Attacken in Verbindung mit Forwarding-Schleifen, wie wir sie bereits in Kapitel 3 beschrieben haben. Existiert eine Forwarding-Schleife und ist dem Angreifer diese bekannt, ist es einfach für ihn die beteiligten Router durch das fortlaufende Versenden von UDP Pakten in diese Schleife zu überlasten.

4.4.3 DCCP[2]

In Kapitel 2 haben wir neben den dominierenden Transportprotokollen UDP und TCP auch alternative Entwicklungen beschrieben, welche versuchen die Vorteile dieser beiden Protokolle miteinander zu verbinden. An dieser Stelle wollen wir uns daher das Forwarding-Verhalten von Paketen betrachten, die über das DCCP-Protokoll übermittelt werden. Da sich in den beiden vorangegangenen Abschnitten zum Verhalten von TCP und UDP das erwartete Verhalten in Forwarding-Schleifen bestätigt hat, wird es nun interessant sein wie die unterschiedlichen Staukontrollverfahren des DCCP-Protokolls auf eine Schleife reagieren.

Da wir bisher noch kein Vorkommen von DCCP-Paketen im Internet messen konnten, können wir auch keine Aussage darüber machen wie groß die üblicherweise verwendete Paketgröße ist. Aus diesem Grund werden wir auch für DCCP die Messungen für Pakete mit einer kleinen und großen Paketgröße vornehmen.

4.4.3.1 PYTHON SKRIPTE DCCP[2]

Wie auch bei unseren Tests mit UDP und TCP benutzen wir Python-Skripte um uns eine Client-Server Umgebung aufzubauen. Der Server erzeugt dazu einen DCCP Socket (Zeile 13) und bindet diesen an die IP-Adresse von Host B und den Port 12345 (Zeile 17 bzw. 10) in unserem Testszenario. Mit der Protokollnummer 33 (Zeile 8) wird dabei der Socket definiert, welche nach IANA [40] für das DCCP-Protokoll registriert ist. Eine Endlosschleife (Zeile 21-27) wartet auf eingehende Verbindungen und in einer verschachtelten Schleife (Zeile 25-27) werden die empfangenen Daten ausgegeben.

DCCP-Server:

```
01    #!/usr/bin/python
02
03    import socket
04
05
06    socket.DCCP_SOCKOPT_SERVICE     = 2
07    socket.SOCK_DCCP                = 6
08    socket.IPPROTO_DCCP             = 33
09    socket.SOL_DCCP                 = 269
10    address                         = ('10.0.10.100', 12345)
11
12    # Create sockets
13    server       =       socket.socket(socket.AF_INET,       socket.SOCK_DCCP,
      socket.IPPROTO_DCCP)
14    server.setsockopt(socket.SOL_DCCP, socket.DCCP_SOCKOPT_SERVICE, True)
15
16    # Connect sockets
17    server.bind(address)
18    server.listen(1)
19
20    # Echo
21    while True:
22        client,address = server.accept()
23        print address[0]
24        data = client.recv(1024)
25        while data:
26            print data
27            data = client.recv(1024)
```

Das DCCP-Client-Skript erzeugt ebenfalls einen DCCP-Socket (Zeile 13) und versucht diesen mit dem zuvor gestarteten Server unter der Adresse 10.0.10.100 an Port 12345 (Zeile 10 bzw. 17) zu verbinden. Nach erfolgreicher Verbindung sendet der Client anschließend in einer Endlosschleife (Zeile 20-26) eine Stringvariable (Zeile 23) *message* an den Server. Um jede Nachricht eindeutig identifizieren zu können, wird diesem String eine fortlaufende Nummerierung angehängt (Zeile 22 und 23). Zur Erzeugung von großen Paketen wird der String entsprechend mit einer längeren Zeichenkette gefüllt.

DCCP-Client

```
01    #!/usr/bin/python
02
03    import socket
04    import time
05
06    socket.DCCP_SOCKOPT_SERVICE     = 2
07    socket.SOCK_DCCP                = 6
08    socket.IPPROTO_DCCP             = 33
09    socket.SOL_DCCP                 = 269
10    address                         = ('10.0.10.100', 12345)
11
12    # Create sockets
13    client                  =                   socket.socket(socket.AF_INET,
      socket.SOCK_DCCP,socket.IPPROTO_DCCP)
14    client.setsockopt(socket.SOL_DCCP, socket.DCCP_SOCKOPT_SERVICE, True)
15
16    # Connect sockets
17    client.connect(address)
18
19    # Echo
20    count=0
21    while True:
22        count=count+1
23        message="This is the message: %s " % (count)
24        print "send:", message
25        client.send(message)
26        #time.sleep(0.01)
```

Im DCCP-Client-Skript findet sich in Zeile 26 eine auskommentierte Zeile, welche wir zu einem späteren Zeitpunkt nutzen werden, um die Senderate etwas zu verzögern.

In Zeile 6 beider Skripte findet sich die Angabe zum DCCP_SOCKOPT_SERVICE. Dieser Wert ist entscheident für die Auswahl des Staukontrollverfahrens. Wie wir in Kapitel 2 beschrieben haben, stellt DCCP zwei unterschiedliche Verfahren zur Staukontrolle zur Verfügung, CCID2 und CCID3. Da das Staukontrollverfahren von der Anwendung festgelegt wird, wird dies an dieser Stelle in unserem Skript mit der entsprechenden Zahlenvariable definiert. Der Wert 2 steht dabei selbsterklärend für das Verfahren CCID2 und ein Wert von 3 für CCID3.

4.4.3.2 BESONDERHEITEN DER DCCP-TESTUMGEBUNG[2]

Da in den aktuellen Linux-Kernel die Implementierung des DCCP-Protokolls noch unvollständig ist, müssen an dem in Abschnitt 4.2 beschriebenen Testszenario einige Veränderungen vorgenommen werden.

Als erstes wird dabei Host A und Host B gegen eine virtuelle Maschinen auf Ubuntu-Basis ausgetauscht und ein experimenteller Kernel kompiliert, welcher über eine Unterstützung der genannten Staukontrollverfahren verfügt. Der aktuelle Quellcode dieses Kernel ist in einem GIT-Repository zu finden. Eine Anleitung zum Herunterladen und Kompilieren des Kernels findet sich auf den Internetseiten des linux-magazin.com [94].

Die beiden virtuellen Maschinen bekommen dabei von VirtualBox einen Hauptspeicher mit einer Größe von 1024MB, sowie einen Prozessorkern unseres Testsystems zugewiesen. Weiterhin sind VT-x und Nested-Paging aktiviert.

Nachdem wir auf beiden Maschinen eine vollständige DCCP-Unterstützung geschaffen haben, gilt es eine weitere Besonderheit zu beachten. Da DCCP als Modul im Kernel implementiert ist, muss es entweder durch eine Anwendung oder manuell mit dem Befehl

```
#  modprobe dccp
```

gestartet werden. Anschließend kann man sich die standardmäßigen Einstellungen mit dem Befehl

```
#  sysctl –a | grep dccp
```

anzeigen lassen.

```
root@ubuntu:/home/mario# sysctl -a | grep dccp
net.dccp.default.ecn_local = 0
net.dccp.default.request_retries = 6
net.dccp.default.retries1 = 3
net.dccp.default.retries2 = 15
net.dccp.default.rx_ccid = 2
net.dccp.default.seq_window = 100
net.dccp.default.sync_ratelimit = 124
net.dccp.default.tx_ccid = 2
net.dccp.default.tx_qlen = 10
```

Die Auswahl des Staukontrollverfahrens sollte normalerweise aus der Anwendung heraus, also unserem Python-Skript, erfolgen. Dies gelingt aber nur für CCID2, welches als standardmäßiges Verfahren implementiert ist. Um CCID3 als Staukontrollverfahren einzustellen, muss zusätzlich mit dem Befehl

```
#  sudo sysctl –w net.dccp.default.rx_ccid=3
#  sudo sysctl –w net.dccp.default.tx_ccid=3
```

das Verfahren explizit eingestellt werden. Erst danach ist es möglich über die Anwendung CCID3 als Staukontrolle zu definieren. Da dieser Umstand dafür sorgte, dass die anfänglichen Tests alle mit CCID2 durchgeführt wurden, konnten auch keine Unterschiede zwischen den beiden Verfahren feststellt werden. Erst bei einer genauen Betrachtung der beim Verbindungsaufbau ausgetauschten Pakete von Server und Client bemerkten wir diese fehlerhafte Konfiguration und konnten dies mit dem genannten Befehl korrigieren.

Neben dieser Korrektur der Standardeinstellungen im DCCP-Protokoll muss auch zwingend die Größe der Transmission-Queue-Length geändert werden. Standardmäßig ist hier ein Wert von 5 eingestellt, was aber auf unserem System dafür sorgte, dass diese Ausgangs-Warteschlange schneller mit Daten gefüllt wurde als versendet werden konnte. Dies führte dazu, dass der Sender die Verarbeitung nach wenigen Nachrichten abbrach und der Socket geschlossen wurde. Mit dem Befehl

```
#  sudo sysctl –w net.dccp.default.tx_qlen=100
```

wird die Länge entsprechend korrigiert und so konnte diesem Verhalten entgegengewirkt werden.

4.4.3.2.1 TESTERGEBNISSE FÜR DCCP MIT CCID2[2]

Da für das DCCP-Protokoll zwei völlig unterschiedliche Staukontrollverfahren implementiert sind, werden wir unsere Testergebnisse entsprechend in zwei Abschnitte unterteilen. Als erstes untersuchen wir dabei das Verhalten von DCCP-Paketen mit dem Verfahren nach CCID2 in einer Forwarding Schleife.

Besteht bereits vor dem Verbindungsaufbau eine Forwarding-Schleife zwischen Host A und Host B werden keine Pakete mit Nutzdaten in das Netz gesendet. Es wird nur ein einziges Request-Paketen zum Initiieren der Verbindung von Host A in das Testnetzwerk gesendet, welches entsprechend ihrer TTL zirkuliert und anschließend verworfen wird. Anschließend wird der Verbindungsaufbau abgebrochen und keine weiteren Versuche unternommen Host B zu erreichen. Dieses Verhalten widerspricht der eigentlichen DCCP-Konfiguration (vgl. Abschnitt 4.4.3.2), da hier ein Wert von 6 für request-retries angegeben ist. In unseren Messungen konnten wir jedoch wie beschrieben nur ein Request-Paket messen.

No.	Time	Source	Destination	Protocol	Length	Info
30	81.633856000	192.168.137.3	10.0.10.100	DCCP	78	39065 > 12345 [Request] Seq=191040702082680 (service=16777216)
31	81.635142000	10.0.10.100	192.168.137.3	DCCP	110	12345 > 39065 [Response] Seq=277302500843402 (Ack=191040702082680)
32	81.639864000	192.168.137.3	10.0.10.100	DCCP	82	39065 > 12345 [Ack] Seq=191040702082681 (Ack=277302500843402)
33	81.640209000	192.168.137.3	10.0.10.100	DCCP	97	39065 > 12345 [DataAck] Seq=191040702082682 (Ack=277302500843402)
34	81.640617000	192.168.137.3	10.0.10.100	DCCP	97	39065 > 12345 [DataAck] Seq=191040702082683 (Ack=277302500843402)

Abbildung 97: Verbindungsaufbau DCCP

Ist ein Verbindungsaufbau möglich, d.h. es besteht anfangs keine Forwarding-Schleife zwischen Host A und Host B, tauschen Sender und Empfänger grundlegende Informationen über das anschließende Transportverhalten aus. Bei DCCP bedeutet dies, dass mittels eines 3-Way-Handshakes die Verbindung initiiert wird. Dabei teilen die Kommunikationspartner auch das jeweils verwendete Staukontrollverfahren mit. Dies bedeutet jedoch nicht, dass sich Sender und Empfänger auf ein Verfahren einigen, denn beide können jeweils unterschiedliche Staukontrollverfahren verwenden. In unseren Testfällen werden aber beide Kommunikationspartner das gleiche Verfahren verwenden, um eine bessere Übersicht zu behalten.

a)
```
Options: (24 bytes)
    Padding
    Timestamp: 18067555
    Change L(CCID 2, 3, 248)
    Change R(CCID 2, 3, 248)
    Mandatory
    Change L(Allow Short Seqnums 0)
```

b)
```
Options: (48 bytes)
    Timestamp Echo: 18067555
    Elapsed Time: 1
    Timestamp: 28370804
    Confirm L(CCID 2, 2, 3, 248)
    Confirm R(CCID 2, 2, 3, 248)
    Mandatory
    Change L(Allow Short Seqnums 0)
    Confirm R(Allow Short Seqnums 0, 0)
    Mandatory
    Change L(Send Ack Vector 1)
    Mandatory
    Change R(Send Ack Vector 1)
```

Abbildung 98: CCID2-Verbindungsaufbau a) Request-Optionen b) Response-Optionen

In diesem Fall bedeutet dies, dass der Client dem Server mitteilt sein Staukontrollverfahren auf CCID2 zu ändern und er gleichzeitig den Server dazu auffordert auf seine CCID ebenfalls auf 2 zu setzen. Dies geschieht mit den Befehlen Change L(CCID 2, 3, 248), wobei L sich auf den Absender (local) bezieht und Change R(CCID 2, 3, 248), welcher einen Befehl für den Empfänger spezifiziert (R = remote) (Abbildung 98 a)). Als Antwort auf diesen Request sendet der Server eine Response-Nachricht (Abbildung 98 b)) in welcher er dem Client die angeforderte Änderung bestätigt (Confirm L(CCID 2, 2, 3, 248). Optionen, welche die Angabe *Mandatory* vorangestellt haben, sind verpflichtend. Hier bedeutet dies, dass eine Einigung über das Senden von ACK-Vektoren zwingend bestätigt werden muss. Mit einem anschließenden ACK-Paket vom Client an den Server, welches diese Bestätigung beinhaltet,

ist die Verbindung aufgebaut und der Client beginnt mit der Datenübertragung (Abbildung 97).

In unserem ersten Test untersuchen wir das Verhalten von kleinen Datenpaketen mit einer TTL von 64. Da DCCP, im Vergleich zu TCP und UDP, viele Informationen über das Options-Feld im DCCP-Header austauscht, variiert die Größe eines gesendeten Pakets zwischen 62 und 110 Byte. Um vergleichbare Messwerte zu erhalten, senden wir die DCCP-Pakete mit einer konstanten Rate von 100 Paketen pro Sekunde in unser Testszenario. Dies erreichen wir durch die Aktivierung der Zeile 26 in unserem DCCP-Client-Skript, welche dafür sorgt, dass nur 100 Pakete pro Sekunde generiert und versendet werden.

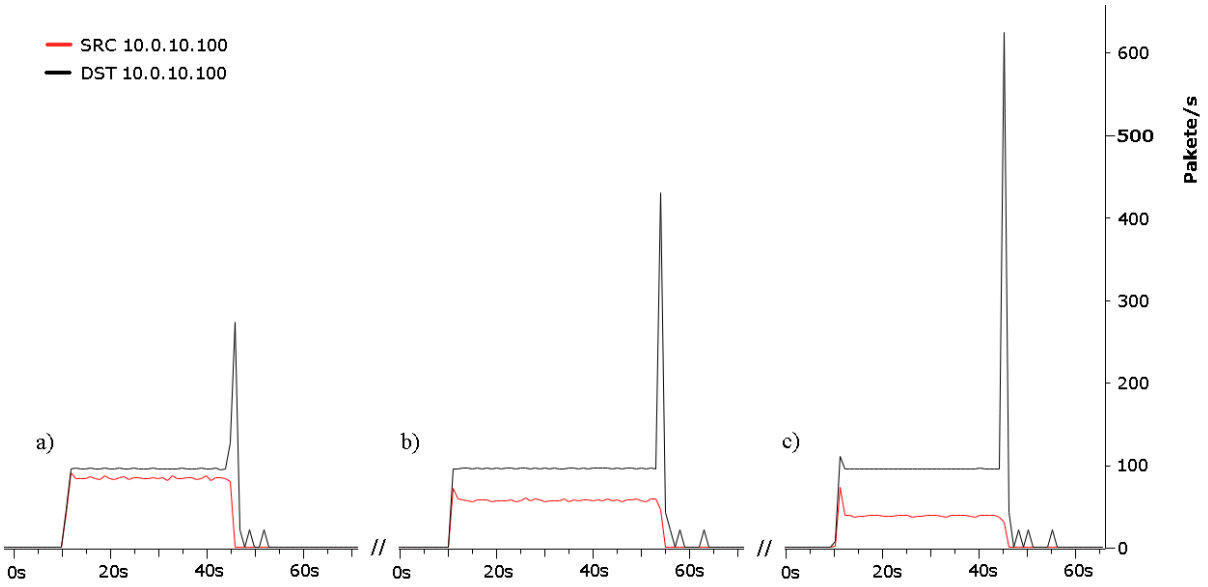


Abbildung 99: DCCP-Datenübertragung kleiner Paketgrößen mit CCID2-Staualgoritmus a) ohne Verzögerung b) 50ms ACK-Verzögerung c) 100ms ACK-Verzögerung

In unserem ersten Testlauf (Abbildung 99 a)) starten wir die Übertragung nach 10 Sekunden und beobachten einen linearen Anstieg der Übertragungsrate bis auf 100 Pakete/s. Die Pakete des Senders sind schwarz markiert, die ACK-Pakete der Empfänger-Sender Richtung sind rot. Deutlich erkennbar ist eine niedrige ACK-Ratio, welche sich während der gesamten Übertragung zwischen 1 und 2 bewegt, die meiste Zeit jedoch auf den Wert 1 gesetzt ist. Der Grund für diese geringe ACK-Ratio ist, dass CCID2 mindestens ein ACK-Paket pro RTT erwartet, was bei der geringen Senderate für fast jedes Datenpacket ein ACK-Paket bedeutet.

Nach etwa 43 Sekunden leiten wir die Datenpakete wieder in eine Forwarding-Schleife und messen im Anschluss einen sprunghaft ansteigenden Datenverkehr an unserem Messpunkt. Mit einem Spitzenwert von 273 Pakete/s bedeutet dies, dass 13 Datenpakete in die Schleife gesendet wurden. Nach verstrichenem TO versucht der Sender mit einem Congestion-Window von 1 und fortlaufend verdoppeltem TO weitere Datenpakete zu senden. Wie bereits beschrieben, handelt es sich dabei jedesmal um ein neu generiertes Paket und nicht um eine wiederholte Sendung wie bei TCP.

Um die Auswirkungen der Größe des Congestion-Windows in unserer Forwarding-Schleife zu testen, wurde in den folgenden Tests die RTT künstlich erhöht. Um dies zu erreichen wurde das Absenden von Paketen auf Serverseite mit dem Befehl

```
#  tc qdisc add dev eth0 root netem delay x
```

um die Zeitspanne *x* verzögert. Der Befehl *tc* steht dabei für Traffic-Control und ermöglicht es, eine Netzwerkschnittstelle in unterschiedlicher Hinsicht zu manipulieren. Neben dem verzögerten Absenden ist auch das zufällige Verwerfen von Paketen bestimmter Protokolle möglich [95]. Durch die damit erhöhte RTT vergrößert sich automatisch das Congestion-Window, da der Sender mehr Datenpakete absendet bevor er für diese wieder ein ACK-Paket erwartet.

Im zweiten Versuch mit kleinen DCCP-Paketen wird eine Verzögerung von 50ms bei Host B (Abbildung 99 b)) erzwungen. Deutlich erkennbar ist direkt eine höhere ACK-Ratio, was bedeutet dass weniger Datenpakete mit einem einzelnen ACK-Paket bestätigt werden. Nachdem nach etwa 54 Sekunden die Datenpakete wieder in die Forwarding-Schleife geleitet werden, ist ein kurzeitiger Spitzenwert von 420 Paketen pro Sekunde messbar. Dies entspricht bei einer TTL von 64 und einer Forwarding-Schleife über drei Hops einer Paketanzahl von 20 in der Schleife. Das bedeutet also, dass bei einem größeren Congestion-Window auch mehr Pakete in die Forwarding-Schleife gesendet werden.

Um diese Messergebnisse zu bestätigen, unternehmen wir einen dritten Versuch mit nochmals gesteigerter RTT. Dazu setzen wir die Verzögerung zum Absenden der Pakete von Host B diesmal auf 100ms (Abbildung 99 c)). Bei der Auswertung ist eine weiter gestiegene ACK-Ratio erkennbar, welche zwischen den Werten 2 und 4 wechselt. Dementsprechend ist auch der Abstand zwischen dem Versenden eines Datenpaket und der Bestätigung mit einem ACK-

Paket gegenüber dem vorherigen Versuch weiter erhöht. Nachdem wir den Datenverkehr in unserem dritten Versuch nach etwa 44 Sekunden in die Forwarding-Schleife leiten, erhalten wir als Spitzenwert 609 Pakete pro Sekunde an unserem Messpunkt. Das wiederum bedeutet, dass nun 29 Pakete entsprechend ihrer TTL im Netzwerk zirkulieren.

Daraus resultiert, dass ein direkter Zusammenhang zwischen der Größe des Congestion-Window und der Belastung in der Forwarding-Schleife besteht. Eine erhöhte Belastung kann aber nur bei einer niedrigeren Übertragungsgeschwindigkeit gemessen werden. Nämlich genau dann, wenn die Anzahl der zirkulierenden Pakete mal der Anzahl der Schleifendurchläufe die reguläre Datenrate übersteigt.

Führen wir den gleichen Versuch mit einer Übertragungsrate von über 3.800 Paketen/s durch, ist nach dem Aktivieren der Forwarding-Schleife nach etwa 37 Sekunden kein Anstieg der Datenrate zu erkennen (Abbildung 100 a)). Da die Pakete in der Forwarding-Schleife unseres Testszenarios 21-mal zirkulieren, wenn sie mit einer TTL von 64 versendet werden, bedeutet dies, dass >3.500/21 Pakete in die Schleife gesendet werden müssen um die reguläre Belastung des Netzes kurzzeitig zu erhöhen. In dieser Auswertung ist auch deutlich erkennbar, dass bei einer hohen Übertragungsrate die ACK-Ratio sehr hoch liegt. In dem

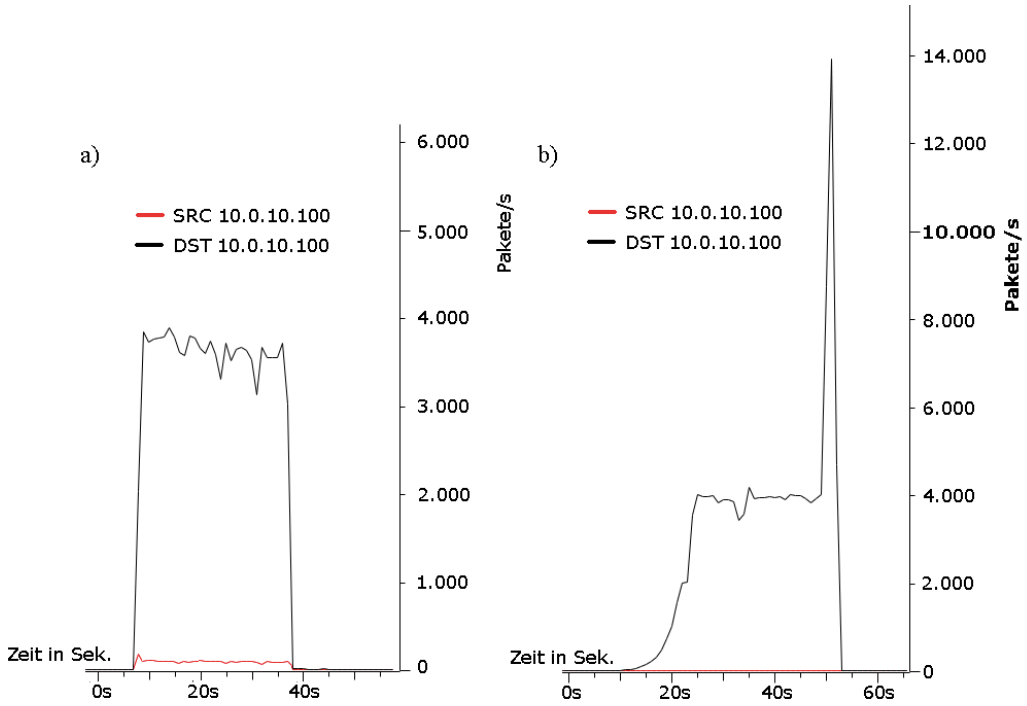


Abbildung 100: Datenübertragung kleiner DCCP-Pakete mit CCID2 bei maximaler Senderate a) minimale RTT b) RTT + 1000ms Verzögerung

Testfall mit 3.800 Pakete/s messen wir eine ACK-Ratio, welche zwischen 32 und 64 wechselt und im Mittel bei etwa 47 liegt. Nach RFC 4341 darf die ACK-Ratio einen Wert von cwnd/2 nicht überschreiten, was ein Congestion-Window von ~90 Paketen bedeuten würde. Dieser Wert reicht jedoch nicht aus um eine messbare erhöhte Belastung in der Forwarding-Schleife zu erzeugen.

In einem weiteren Versuch erzeugen wir auch bei der Übertragung mit einer hohen Paketfrequenz eine künstlich erhöhte RTT. Diesmal verzögern wir die Pakete um 1000ms (Abbildung 100 b)). Daraus resultiert eine sehr langsame Startphase und erst mit zunehmender ACK-Ratio gleicht sich die Geschwindigkeit der Übertragungsrate ohne künstliche Latenz an. Anfangs steigt bei CCID2 die Übertragungsrate der Datenpakete linear zu den ACK-Paketen an. Erst mit exponentiell steigender ACK-Ratio steigt auch die Übertragungsgeschwindigkeit der Datenpakete. Bei einer Übertragungsgeschwindigkeit von ~4.000 Paketen/s erreicht die Verbindung ihren dauerhaften Höchstwert, während die ACK-Ratio dabei auf bis zu 1024 ansteigt. Nach dem aktivieren der Forwarding-Schleife messen wir eine stark ansteigende Paketrate, da auf Grund der großen Verzögerung der RTT eine große Anzahl an Paketen in die Schleife gelangen. In den mehrfachen Versuchsläufen unterlag der gemessene Wert starken Schwankungen, was auf den Zeitpunkt der Aktivierung der Forwarding-Schleife zurückzuführen ist. In der Mehrzahl der Fälle wurde dabei ein Wert zwischen 12.000-14.000 Pakete/s gemessen, was etwa 570 bis 660 Paketen in der Schleife entspricht. Entscheidend dabei ist wie viele Pakete aus dem Congestion-Window vor dem Aktivieren der Forwarding-Schleife das Netz noch passieren und wie viele davon in die Schleife geraten.

Wie auch bei TCP und UDP untersuchen wir in der folgenden Testreihe das Verhalten von großen DCCP-Paketen nahe der MTU. Die Paketgröße der Datenpakete liegt dabei zwischen 1338 und 1355 Byte, was wiederum auf die unterschiedlichen Options-Angaben im DCCP-Header zurückzuführen ist.

Wie in der Testreihe mit kleinen DCCP-Paketen variieren wir die RTT durch eine erzwungene Verzögerung beim Absenden der Pakete von Host B (Abbildung 101). Weiterhin senden wir die Pakete mit einer konstanten Rate von 100 Paketen pro Sekunde bevor wir die Forwarding-Schleife erzwingen. Wie auch bei den kleinen Paketgrößen messen wir ohne eine künstliche Verzögerung der RTT eine sehr geringe ACK-Ratio, welche sich überwiegend bei

1 befindet. Nach etwa 31 Sekunden erzeugen wir in unserem Testszenario die Forwarding-Schleife und messen im Anschluss einen Spitzenwert von 315 Pakete/s (Abbildung 101 a)). Bei 21 Schleifendurchläufen eines Pakets entspricht dies 15 Paketen in der Forwarding Schleife. Im Gegensatz zu den kleinen DCCP-Paketen wurden also zwei Pakete mehr in die Schleife gesendet, was auf ein etwas größeres Congestion-Window hindeutet. Dies lässt sich bei den größeren Paketen darauf zurückführen, dass hier die RTT auf Grund der längeren Transportzeit gegenüber kleinen Paketen größer ist.

Anschließende Tests mit künstlich erhöhter RTT bestätigen dieses Verhalten. Bei einer erzwungen Verzögerung von 50ms messen wir einen Spitzenwert von 504 Pakete/s, was wiederum 24 Paketen in der Forwarding-Schleife entspricht und somit geringfügig mehr ist als bei kleinen Paketen (Abbildung 101 b)). Im letzten Test mit einer Senderate von 100 Paketen/s verzögern wir wieder die ACK-Pakete zusätzlich um 100ms. In der erzwungenen Forwarding-Schleife nach 43 Sekunden steigt die Anzahl der gemessenen Pakete auf 588 pro Sekunde (Abbildung 101 c)). Mit 28 Paketen in der Forwarding-Schleife entspricht dies in etwa einem gleich großen Congestion-Window wie bei dem gleichen Test mit kleinen DCCP-Paketen. Zu erklären ist dies mit der Tatsache, dass bei steigender Latenz im Netzwerk die minimal langsamere Übertragungsrate eines größeren DCCP-Pakets die RTT nicht maßgeblich beeinflusst wird.

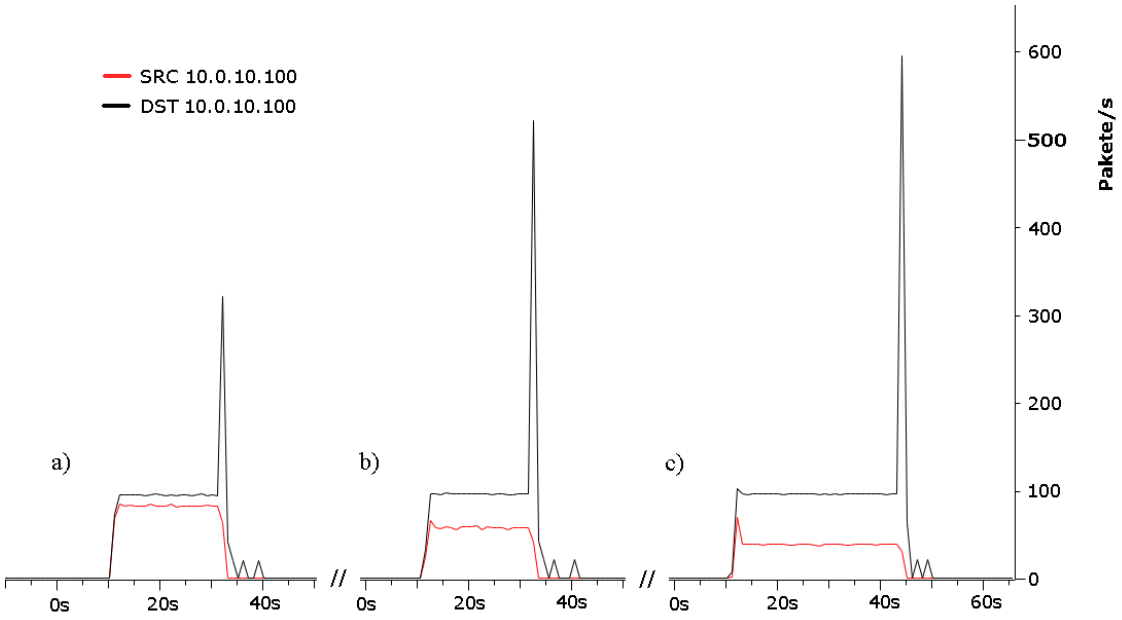


Abbildung 101: DCCP-Datenübertragung großer Paketgrößen mit CCID2-Staualgoritmus a) ohne Verzögerung b) 50ms ACK-Verzögerung c) 100ms ACK-Verzögerung

Zum Abschluss der Testreihe messen wir auch die Auswirkungen einer Forwarding-Schleife großer DCCP-Pakete bei einer höheren Senderate. Die virtuelle Ubuntu-Maschine, welche in unserem Testszenario Host A bildet, generiert jedoch nur etwa 400 DCCP-Pakete pro Sekunde, wodurch sich keine vergleichbare Testsituation wie bei kleinen Paketen herstellen lässt. Dadurch können wir also maximal ~400 DCCP-Pakete mit einer Größe von bis zu 1355 Byte pro Sekunde in unser Testnetzwerk senden.

Bei dieser Senderate lässt sich im Anschluss an das Erzwingen der Forwarding-Schleife noch eine zusätzliche Belastung im Netzwerk messen. Nachdem wir nach etwa 35 Sekunden die Schleife aktivieren ist kurzzeitig ein Paketaufkommen von 798 Paketen/s messbar, was in diesem Fall 38 Paketen in der Forwarding-Schleife entspricht. Deutlich erkennbar ist auch die größere Differenz von Daten- zu ACP-Paketen gegenüber der Senderate von 100 Paketen/s. Während der regulären Verbindung wechselt die ACK-Ratio dabei zwischen den Werten 4 und 8.

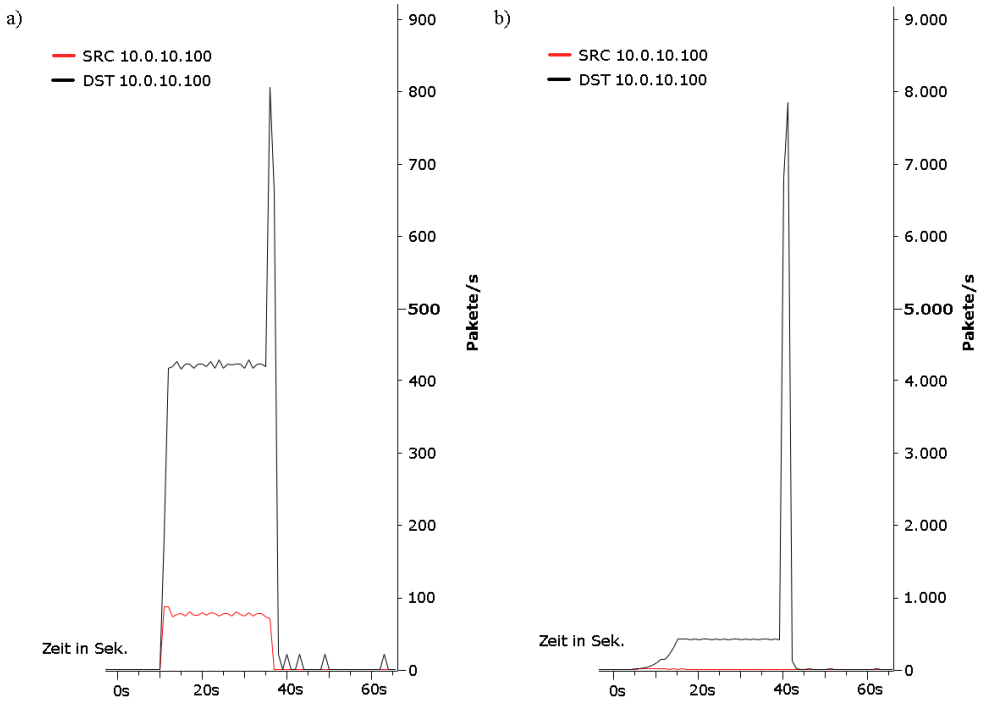


Abbildung 102: Datenübertragung großer DCCP-Pakete mit CCID2 bei maximaler Senderate a) minimale RTT b) RTT + 1000ms Verzögerung

In einer weiteren Testreihe untersuchen wir auch das Verhalten in einer Forwarding-Schleife für große DCCP-Pakete bei maximaler Senderate mit hoher Latenz im Netzwerk. Dazu erzeugen wir abermals eine Verzögerung von 1000ms auf Host B. Ebenso wie bei den Tests

mit kleinen Paketen ist zu Beginn nur eine sehr langsame Steigerung der Übertragungsrate festzustellen. Erst mit exponentiell steigender ACK-Ratio steigt auch die Paketrate entsprechend an, bis sie mit etwa 400 Paketen pro Sekunde die gleiche Übertragungsrate annimmt wie bei unserem Test mit niedriger Latenz (Abbildung 102 b)). Nachdem die Forwarding-Schleife nach etwa 39 Sekunden aktiviert wird, messen wir einen Anstieg der Paketrate auf ~8.000 Pakete pro Sekunde. Dies bedeutet, alle Pakete welche innerhalb einer RTT gesendet werden, nämlich ~400, werden auch in die Forwarding-Schleife gesendet bevor der Client den Paketverlust bemerkt.

4.4.3.2.2 FAZIT[2]

Wie auch das Staukontrollverfahren von TCP reagiert DCCP mit dem Staukontrollverfahren CCID2 sehr schnell auf eine Forwarding-Schleife. DCCP stellt ein verbindungsorientiertes Protokoll dar, welches entsprechend durch ACK-Pakete über die korrekte oder fehlerhafte Übertragung der Datenpakete informiert wird. Das Staukontrollverfahren CCID2 ist dabei derart konzipiert, dass es möglichst schnell auf veränderte Übertragungsgeschwindigkeiten reagieren kann um die verfügbaren Kapazitäten mit einer hohen Effizienz ausnutzen zu können. Dadurch werden Paketverluste, welche durch eine Forwarding-Schleife verursacht werden, schnell erkannt und nur noch vereinzelt Pakete in das Netzwerk gesendet. Lediglich kurz nach dem Auftreten der Schleife entsteht eine erhöhte Belastung der beteiligten Router durch die dort zirkulierenden Pakete. Die Anzahl der Pakete in der Schleife und das Ausmaß der Belastung ist abhängig vom Congestion-Window zu diesem Zeitpunkt. Durch die hohe ACK-Ratio bei einer schnellen Verbindung ist das Congestion-Window recht groß, wodurch auch die Belastung kurzzeitig größer ist als im Vergleich zu TCP. Insgesamt ist DCCP mit CCID2 aber sehr gut gegen eine Forwarding-Schleife abgesichert und flutet das Netzwerk nicht fortlaufend mit weiteren Datenpaketen.

4.4.3.2.3 TESTERGEBNISSE FÜR DCCP MIT CCID3[1]

Wie bereits in Kapitel 2.6.4.2.2 beschrieben, versucht die Staukontrolle CCID3 einen gleichmäßigen Datenstrom zu erzeugen. Anwendungen für Video- oder Audiostreams und insbesondere Anwendungen mit einem geringen Empfangspuffer können von dieser

Staukontrolle profitieren. Pro Szenario lassen wir die Tests ca. 20 Mal durchlaufen und werten die Ergebnisse aus.

Bei allen folgenden Tests mit CCID3 haben wir einen einheitlichen Verbindungsaufbau (Abbildung 103). Dieser besteht aus einem DCCP-Request-Paket des Clients. Nach Versenden des DCCP-Request-Pakets wechselt der Client in den Zustand REQUEST. Der Server empfängt das Paket im Zustand LISTEN, antwortet seinerseits mit einem DCCP-Response-Paket und wechselt in den Zustand RESPOND. Nachdem der Client mit einem DCCP-Ack antwortet, ist der Verbindungsaufbau abgeschlossen (3-Way-Handshake) und beide Seiten wechseln in den Zustand OPEN (siehe Kapitel 2.6.4.2).

Time	Source	Destination	Length	Info
36.616	192.168.137.3	10.0.10.100	78	38197 > 12345 [Request] Seq=203708840550816 (service=0)
36.668	10.0.10.100	192.168.137.3	134	12345 > 38197 [Response] Seq=162354623397464 (Ack=203708840550816)
36.668	192.168.137.3	10.0.10.100	106	38197 > 12345 [Ack] Seq=203708840550817 (Ack=162354623397464)

Abbildung 103: Verbindungsaufbau von DCCP mit CCID3

Alle Pakete des Verbindungsaufbaus beinhalten Feature-Requests (Change L oder Change R) und die Antwort der Gegenstelle (Commit R oder Commit L).

(a)
- Options: (24 bytes)
  - Padding
  - Timestamp: 6650328
  - Change L(CCID 3, 2, 248)
  - Change R(CCID 3, 2, 248)
  - Mandatory
  - Change L(Allow Short Seqnums 0)

(b)
- Options: (72 bytes)
  - Padding
  - Padding
  - Timestamp Echo: 6650328
  - Elapsed Time: 1
  - Timestamp: 13015784
  - Confirm L(CCID 3, 3, 2, 248)
  - Confirm R(CCID 3, 3, 2, 248)
  - Mandatory
  - Change L(Allow Short Seqnums 0)
  - Confirm R(Allow Short Seqnums 0, 0)
  - Change L(Ack Ratio 0)
  - Change L(Send Ack Vector 0)
  - Change R(Send Ack Vector 0)
  - Mandatory
  - Change R(Send NDP Count 1)
  - Change L(Send NDP Count 1)
  - Mandatory
  - Change L(Send Loss Event Rate 1)
  - Mandatory
  - Change R(Send Loss Event Rate 1)

(c)
- Options: (48 bytes)
  - Timestamp Echo: 13015784
  - Elapsed Time: 1
  - Confirm R(Allow Short Seqnums 0, 0)
  - Confirm R(Ack Ratio 0)
  - Confirm R(Send Ack Vector 0, 0)
  - Confirm L(Send Ack Vector 0, 0)
  - Confirm L(Send NDP Count 1, 1)
  - Confirm R(Send NDP Count 1, 1)
  - Confirm R(Send Loss Event Rate 1, 1)
  - Confirm L(Send Loss Event Rate 1, 1)

Abbildung 104: Feature-Requests beim Verbindungsaufbau mit DCCP (CCID3)

Der Client sendet die Anfrage an den Server, das Staukontrollverfahren für sich (Change L, Feature Location) und den Server (Change R, Feature Remote) auf CCID3 (Change L(CCID 3, 2, 248) zu ändern ((a) in Abbildung 104). Schon dieser recht zentrale Punkt unterscheidet sich von der Spezifikation [43] und unserer als experimentell ausgewiesenen und genutzten

Implementierung. Laut Spezifikation [43] müsste die Anfrage „Change L(CCID, 3 2) lauten. Weiterhin möchte der Client keine verkürzten Sequenznummern verwenden und sendet ein verpflichtendes (Mandatory vorangestellt) Change L(Allow Short Seqnums 0) an den Server. Die Antwort des Servers im DCCP-Request-Paket ((b) in Abbildung 104) auf die Feature-Requests ist ein Confirm L(CCID 3, 3, 2, 248). Laut Spezifikation [43] müsste die Antwort Confirm R(CCID, 3, 3 2) lauten. Was der Wert 248 bedeutet, ist uns unbekannt. Mit Confirm R(Allow Short Seqnums 0, 0) erhält der Client die Erlaubnis, verkürzte Sequenznummern zu verwenden. Der zweite Wert (0) ist die Präferenzliste des Servers für dieses Feature. Diese Features (CCID & Allow Short Seqnums) unterliegen der Konfliktregel Server-Priority. Im Konfliktfall bestimmt der Server damit den zu verwendenden Wert. Aus Abbildung 104 (b) ist ebenfalls ersichtlich, dass der Server für beide Seiten keine Verwendung der Optionen Ack Ratio und Ack Vector anstrebt. Die Optionen NDP und Loss Event Rate sollen dagegen von beiden Seiten genutzt werden. Der Client bestätigt diese Anfragen ((c) in Abbildung 104). Damit ist das Aushandeln der Optionen für diese Verbindung abgeschlossen. Die Optionen sind im Detail in Kapitel 2.6.4.2 erläutert.

In unserer ersten Testreihe beschränken wir die Anzahl der Pakete auf 100 Pakete/Sek. und bauen nacheinander eine künstliche Latenz von 0ms und 100ms ein.

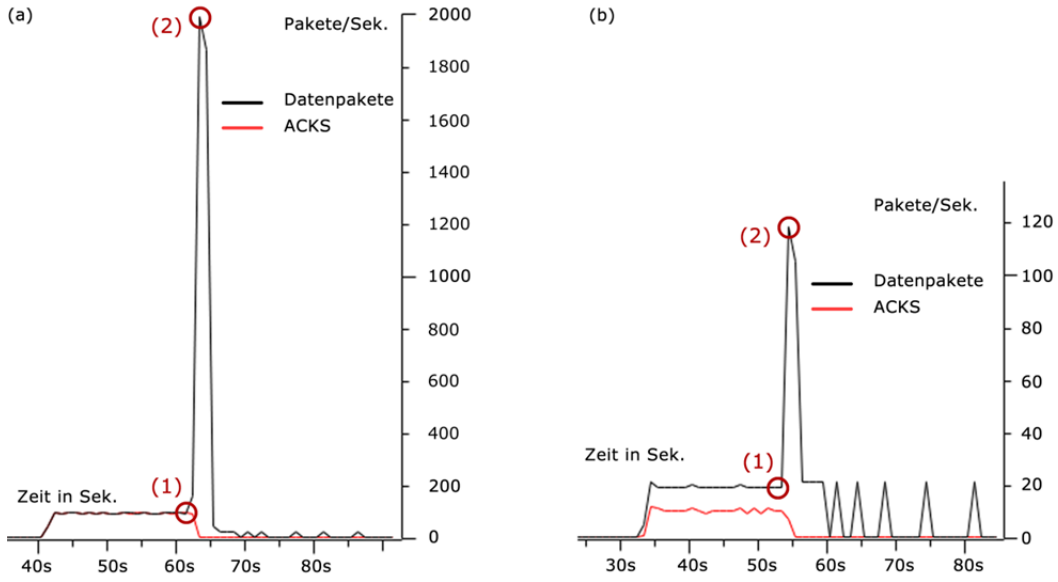


Abbildung 105: DCCP-Datenübertragung kleiner Paketgrößen (100 Pakete/Sek.) mit CCID3-Staualgoritmus a) ohne Verzögerung b) 50 ms ACK-Verzögerung

In der ersten Messreihe ohne künstliche Verzögerung ((a) in Abbildung 105) erkennt man deutlich ein langsameres Ansteigen der Datenrate als bei CCID2. Durch die extrem niedrige RTT und das Limitieren auf 100 Pakete/Sekunde wird jedes Paket direkt mit einem DCCP-Ack-Paket bestätigt. Es befindet sich demnach immer nur ein Paket im Transit. Der Sender dürfte theoretisch zwischen 2 und 4 Pakete pro RTT zu Beginn versenden. Dies wird durch unsere Drosselung verhindert. Jedes DCCP-Data-Packet enthält den DCCP-spezifischen Wert CCVal, der es dem Server ermöglicht, die Zugehörigkeit von Paketverlusten zu Intervallen (Loss Intervals) zu bestimmen ((a) in Abbildung 106). In unserem Testszenario kommt es zu Anfang nicht zu Paketverlusten und nach dem Aktivieren der Schleife, erhält der Client keine DCCP-Ack-Pakete mehr vom Server, die Loss Intervals enthalten könnten. Dieses Verhalten ist bei allen von uns durchgeführten Tests mit CCID3 zu beobachten (siehe Kapitel 2.6.4.2.2).

(a)

```
⊟ Datagram Congestion Control Protocol
    Source Port: 44664 (44664)
    Destination Port: 12345 (12345)
    Data Offset: 4
    CCVal: 4
    Checksum Coverage: 0
    Checksum: 0x39fa [correct]
    Type: Data (2)
    Extended Sequence Numbers: True
    Sequence Number: 158460044280408
```

(b)

```
⊟ Datagram Congestion Control Protocol, Src
    Source Port: 12345 (12345)
    Destination Port: 44664 (44664)
    Data Offset: 10
    CCVal: 0
    Checksum Coverage: 0
    Checksum: 0x1f05 [correct]
    Type: Ack (3)
    Extended Sequence Numbers: True
    Sequence Number: 223119697685726
    Acknowledgement Number: 158460044280408
  ⊟ Options: (16 bytes)
      Padding
      CCID3 Receive Rate: 2179 bytes/sec
      CCID3 Loss Event Rate: 0 (or max)
      NDP Count: 4
```

Abbildung 106: DCCP-spezifische (CCID3) Header-Werte in DCCP-Data-Paketen (a) und Optionswerte in DCCP-Ack-Paketen (b)

Weiterhin enthalten die Bestätigungen des Clients ((b) in Abbildung 106) die empfangene Datenrate seit dem zuletzt versendeten DCCP-Ack-Paket (CCID Receive Rate), die Loss Event Rate (in unserem Fall immer 0) und den NDP Count. Der NDP Count scheint wiederum nicht korrekt implementiert zu sein, da er in jedem DCCP-Ack-Paket stetig um 1 inkrementiert wird, obwohl keine Pakete ohne Daten vom Client versendet werden. Eine genauere Beschreibung der Optionen findet sich in Kapitel 2.6.4.2. Die vom Server übermittelte CCID Receive Rate bewegt sich konstant um den Wert 2000 Bytes/Sek.. Der Sender dürfte die Senderate maximal auf 4000 Bytes/Sek. erhöhen (2*Receive Rate). Durch die Limitierung auf 100 Pakete/Sek. und die extrem kleine RTT bewegen wir uns konstant bei 2000 Bytes/Sek..

Nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 105 (a)) sendet der Client zunächst weitere Pakete in die Schleife. Entscheidend für die Belastung der Schleife ist der NoFeedback Timer von CCID3. Dieser wird durch max(4*RTT, 2*(Zwischensendezeit)) berechnet. Unsere RTT liegt bei ca. 0,001 Sekunden und es wird ca. alle 0,1 Sekunden ein Paket gesendet. Damit liegt der NoFeedback Timer zu Anfang bei ca. 0,2 Sekunden. Nach Ablauf des NoFeedback Timers ohne zwischenzeitlich eintreffende Bestätigung wird die Senderate halbiert und der NoFeedback Timer neu berechnet. Entscheidend ist hierbei, dass das Intervall zwischen zwei Datenpaketen (Zwischensendezeit) sich mit sinkender Datenrate erhöhen muss und damit auch der Nofeedback Timer, was zu einer geforderten langsameren Anpassung der Datenrate führt. Aus Kapitel 2.6.4.2.2 wissen wir, dass CCID3 sich nur für Applikationen eignet, die eine konstante Paketgröße versenden und die Datenrate nicht durch geänderte Paketgrößen manipulieren. Dies ist für das Funktionieren von CCID3 entscheidend. Mit einem steigenden Intervall zwischen zwei zu sendenden Paketen zieht sich CCID3 langsam zurück und drosselt die Datenrate. Dies geschieht bedeutend langsamer als bei TCP. In der Spitze erfährt der messende Router R1 ca. 2000 Paketdurchläufe pro Sekunde (Zeitpunkt (2) in Abbildung 105 (a)). Nach ca. 2 Sekunden hat CCID3 188 Pakete in die Schleife gesendet und die Senderate bzw. das Intervall zwischen zwei aufeinanderfolgenden Paketen auf ca. 0,56 Sekunden erhöht (siehe Abbildung 107). Nach zwei weiteren gesendeten Paketen verdoppelt sich das Intervall dementsprechend auf 1,12 Sekunden (Halbierung der Senderate). Ab diesem Zeitpunkt zieht sich CCID3 durch die weitere Halbierung der Senderate sehr schnell zurück und belastet die involvierenden Knoten der Schleife nur noch minimal. Da wir die Pakete auf einem Linux-System generieren, besitzen diese eine TTL von 64 vor dem Eintritt in die Schleife. Dementsprechend traversiert ein Paket in der Schleife jeden Knoten 21 Mal (*floor(64/3)*, siehe Kapitel 4.3.4.3). Knoten *R1* verwirft am Anfang des 22. Umlaufs das Paket und generiert eine ICMP-Nachricht (*64 mod 3 = 1*, siehe Kapitel 4.3.4.3).

In unserer zweiten Testreihe erhöhen wir die RTT „künstlich“ um 50 ms analog zu Kapitel 4.4.3.2.1. In Abbildung 105 (b) fällt zuerst die niedrigere Paketrate durch die erhöhte RTT auf. Sie liegt bei ca. 20 Paketen/Sek. und wird durch die in Kapitel 2.6.4.2.2 beschriebene Gleichung zur Ermittlung der Senderate von CCID3 bestimmt. Da die RTT im Nenner des Quotienten vorkommt und im Zähler nur die Paketgröße verwendet wird, ist dieser Wert plausibel. Weiterhin fällt auf, dass im Durchschnitt nur noch jedes zweite Paket bestätigt

wird. Durch die niedrigere Senderate erhalten wir eine niedrigere Belastung der Knoten nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 105 (b)). Insgesamt werden 9 Pakete in die Schleife gesendet, bevor der NoFeedback Timer die Datenrate derart minimiert, dass zwischen zwei aufeinanderfolgenden Paketen eine Zeitspanne von ca. 0,7 Sekunden verstreichen muss. Damit ist die Belastung um ein Vielfaches niedriger als in unserem vorigen Test ohne Erhöhung der RTT. Die maximale Belastung der Knoten in der Schleife liegt bei ca. 120 Paketen/Sek. (Zeitpunkt (2) in Abbildung 105 (b)).

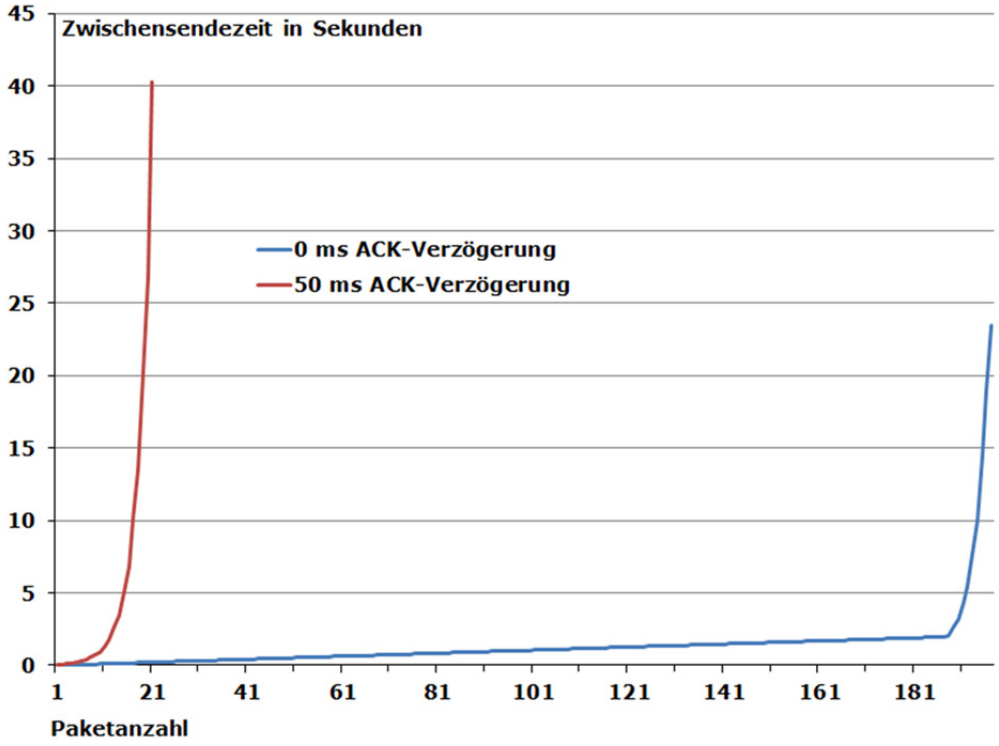


Abbildung 107: Entwicklung der Zwischensendezeit nach Aktivierung der Schleife (Drosselung auf 100 Paketen/Sekunde, kleine Pakete)

In der Entwicklung der Zwischensendezeit (Abbildung 107) erkennt man deutlich den Einfluss der Senderate auf die Anzahl der Pakete, die in die Schleife gesendet werden. Werden die ACKs nicht verzögert, liegt die Senderate vor Aktivierung der Schleife bei 100 Paketen/Sek. (20 Pakete/Sek. mit 50 ms ACK-Verzögerung). CCID3 benötigt dementsprechend länger, um die Senderate zu drosseln bzw. die Zwischensendezeit zu erhöhen. Damit hat die Senderate bei kleinen Paketen einen Einfluss auf die Belastung der Schleife. Da die Berechnung der Senderate die Paketgröße und die RTT berücksichtigt (siehe

Kapitel 2.6.4.2.2), ist bei kleinen Paketen die RTT ein entscheidender Faktor für die Belastung der Schleife. Eine höhere RTT impliziert eine geringere Belastung der Schleife.

Nun deaktivieren wir die Drosselung auf 100 Pakete/Sekunde und lassen den Client mit maximaler Paketrate senden, in einer ersten Testreihe ohne erhöhte RTT und in einer zweiten Testreihe mit einer um 1000ms erhöhten RTT.

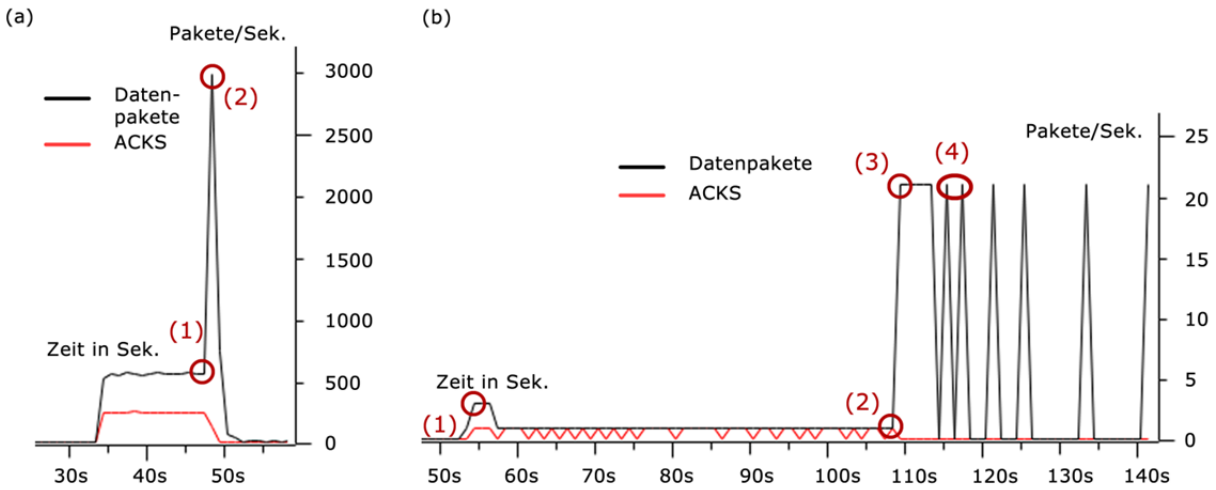


Abbildung 108: DCCP-Datenübertragung kleiner Paketgrößen (maximale Paketrate) mit CCID3-Staualgoritmus a) ohne Verzögerung b) 1000 ms ACK-Verzögerung

Ohne künstliche Verzögerung und maximaler Senderate (Paketrate) beobachten wir ein Bild ähnlich der Messung mit gedrosselter Paketrate (100 Pakete/Sek.). Allerdings wird hier im Schnitt nur jedes zweite Paket bestätigt. Die Paketrate liegt leicht oberhalb von 500 Paketen/ Sekunde. Bis sich die Zeit zwischen zwei versendeten Paketen nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 108 (a)) auf 0,6 Sekunden erhöht, werden 175 Pakete in die Schleife gesendet.

Interessanter ist der Graph, der sich aus einer zusätzlichen Verzögerung von 1000 ms ergibt. Wir erinnern uns (siehe Kapitel 2.6.4.2.2), dass der Client laut Spezifikation mit einer Paketrate zwischen 2 und 4 Paketen starten darf. In unserem Fall wird das Maximum ausgenutzt und bis zum Eintreffen der ersten Bestätigung werden 4 Pakete versendet (Zeitpunkt (1) in Abbildung 108 (b)). Daraufhin wird nur 1 Paket versendet und die nächste Bestätigung mit der Receive Rate des Servers ausgewertet. Im Mittel werden dann schließlich ca. 2 Pakete/Sekunde versendet.

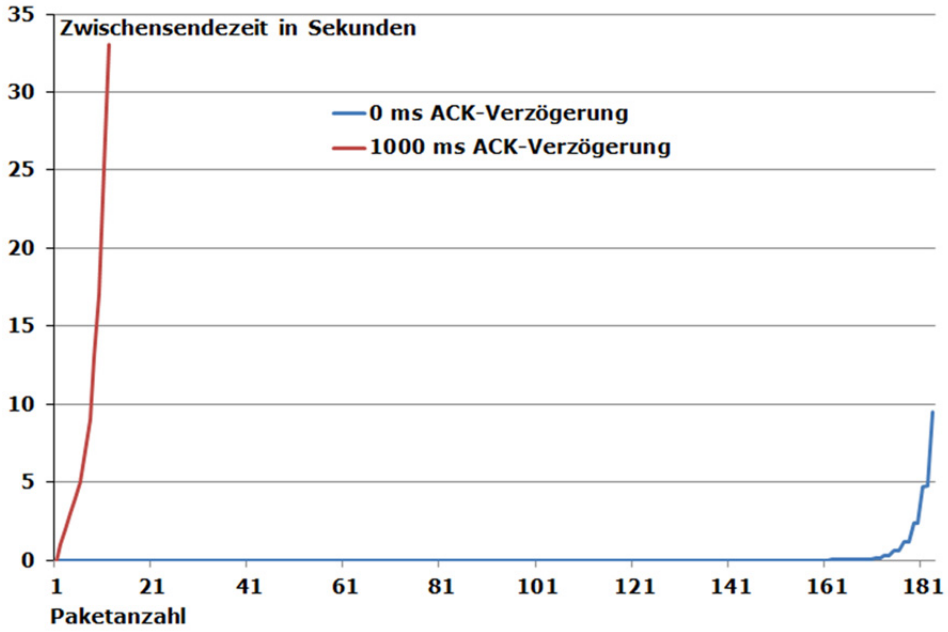


Abbildung 109: Entwicklung der Zwischensendezeit nach Aktivierung der Schleife (kleine Pakete)

Nachdem wir die Schleife aktivieren (Zeitpunkt (2) in Abbildung 108 (b)), werden 5 Pakete mit einem zeitlichen Abstand von ca. 1 Sekunde (also der RTT) in die Schleife gesendet (Zeitpunkt (3) in Abbildung 108 (b)). Die nächsten 2 Pakete haben schon einen doppelten zeitlichen Abstand von ca. 2 Sekunden zueinander (Zeitpunkt (4) in Abbildung 108 (b)). Der zeitliche Abstand verdoppelt sich weiter alle 2 Pakete. Damit haben wir die geringste bisher von uns gemessene Belastung einer Schleife mit CCID3. Der ausschlaggebende Faktor ist die niedrige Datenrate von 2 Paketen/Sekunde, die durch die extrem hohe RTT impliziert wird.

Auch in dieser Versuchsreihe ist die RTT die entscheidende Variable für die Senderate, die Erhöhung der Zwischensendezeit und die Belastung der Schleife. Eine hohe RTT impliziert eine niedrigere Senderate und damit eine geringere Belastung der Schleife (Abbildung 109).

Wir wiederholen nun unsere Testreihen für große Datenpakete, indem wir wie zuvor z.B. in Kapitel 4.4.1.3 unseren Client derart abändern, dass er längere Strings generiert und die Paketgröße daher bei durchschnittlich ca. 1500 Bytes liegt (inkl. Overhead der niedrigeren Schichten). Die ersten zwei Testreihen beschränken wir wieder auf 100 vom Client gesendete Pakete/Sekunde mit einer Verzögerung von 0 ms und 50 ms.

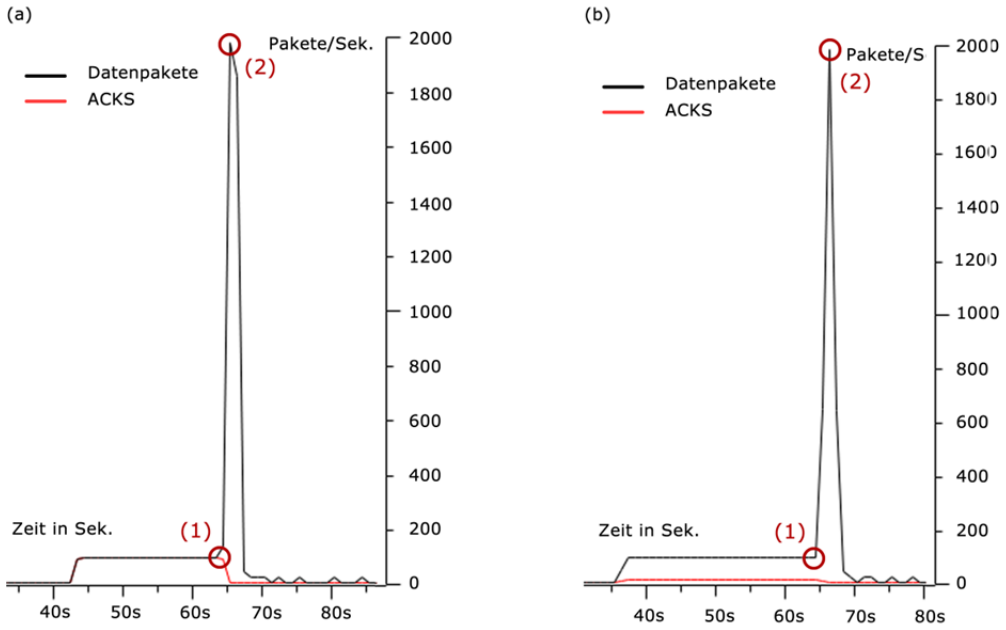


Abbildung 110: DCCP-Datenübertragung großer Paketgrößen (100 Pakete/Sek.) mit CCID3-Staualgoritmus a) ohne Verzögerung b) 50 ms ACK-Verzögerung

Für die erste Testreihe ohne Verzögerung erreichen wir eine Senderate von 100 Paketen/Sekunde. Durch die geringe RTT wird jedes Paket bestätigt. Nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 110 (a)) werden 187 Pakete in die Schleife gesendet bis die Zwischensendezeit auf ca. 0,6 Sekunden anwächst. Damit erfährt jeder Knoten eine maximale Belastung von 2000 Paketdurchläufen/Sekunde (Zeitpunkt (2) in Abbildung 110 (a)).

Die gleiche Testreihe mit einer um 50 ms erhöhten RTT liefert ein ähnliches Ergebnis, jedoch wird nur noch im Durchschnitt jedes 7. Paket bestätigt. Es fällt auf, dass die Datenrate im Gegensatz zum Test mit kleinen Paketgrößen und einer um 50 ms erhöhten RTT konstant bleibt. Da in der Gleichung zur Berechnung der Senderate die Paketgröße im Zähler steht, ist auch dieses Verhalten nachvollziehbar, d.h. größere Pakete relativieren den Effekt einer steigenden RTT. Der Client sendet 154 Pakete nach Aktivierung (Zeitpunkt (1) in Abbildung 110 (b)) in die Schleife bis die Zwischensendezeit auf ca. 0,76 Sekunden anwächst. Jeder Knoten erfährt eine maximale Belastung von 2000 Paketen/Sekunde (Zeitpunkt (2) in Abbildung 110 (b)).

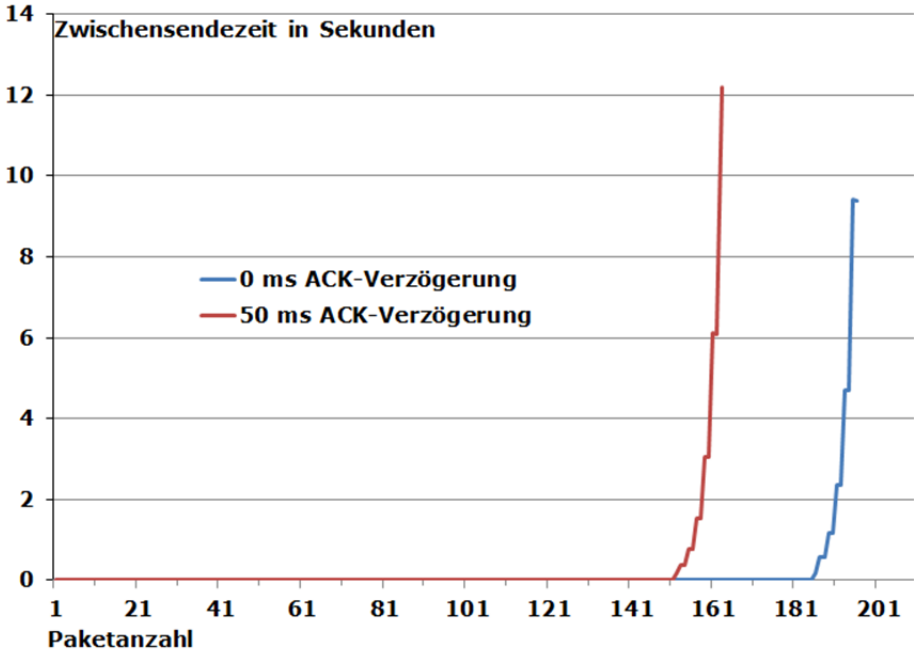


Abbildung 111: Entwicklung der Zwischensendezeit nach Aktivierung der Schleife (Drosselung auf 100 Paketen/Sekunde, große Pakete)

Zu beachten ist hierbei, dass die RTT einen geringen Effekt auf die Senderate ausübt und demnach auch die Auswirkungen auf die Schleife geringer ausfallen. In Abbildung 111 ist der Ablauf des NoFeedback Timers und die damit einhergehende Verdoppelung der Sendezwischenzeit sehr gut an den Stufen zu erkennen.

In unseren letzten beiden Testreihen deaktivieren wir die Drosselung von 100 Paketen/Sekunde und lassen den Client mit maximaler Geschwindigkeit senden.

In unserer ersten Testreihe mit maximaler Senderate und ohne Verzögerung der RTT wird durchschnittlich jedes 5. Paket bestätigt. Die Paketrate liegt bei ca. 410 Paketen/Sekunde. Nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 112 (a)) werden ca. 462 Pakete in die Schleife gesendet bis die Sendezwischenzeit sich auf 0,45 Sekunden erhöht. Jeder Knoten erfährt eine maximale Belastung von ca. 6100 Paketen/Sekunde.

Am anschaulichsten wird die Funktionsweise von CCID3 in der Testreihe mit einer ACK-Verzögerung von 1000 ms und maximaler Senderate. Nach Aktivierung der Schleife (Zeitpunkt (1) in Abbildung 112 (b)) gelangen 794 Pakete in die Schleife bis die Sendezwischenzeit sich auf 0,6 Sekunden erhöht. Die maximale Belastung der Knoten in der Schleife liegt bei ca. 2100 Paketen/Sekunde (Zeitpunkt (2) in Abbildung 112 (b)).

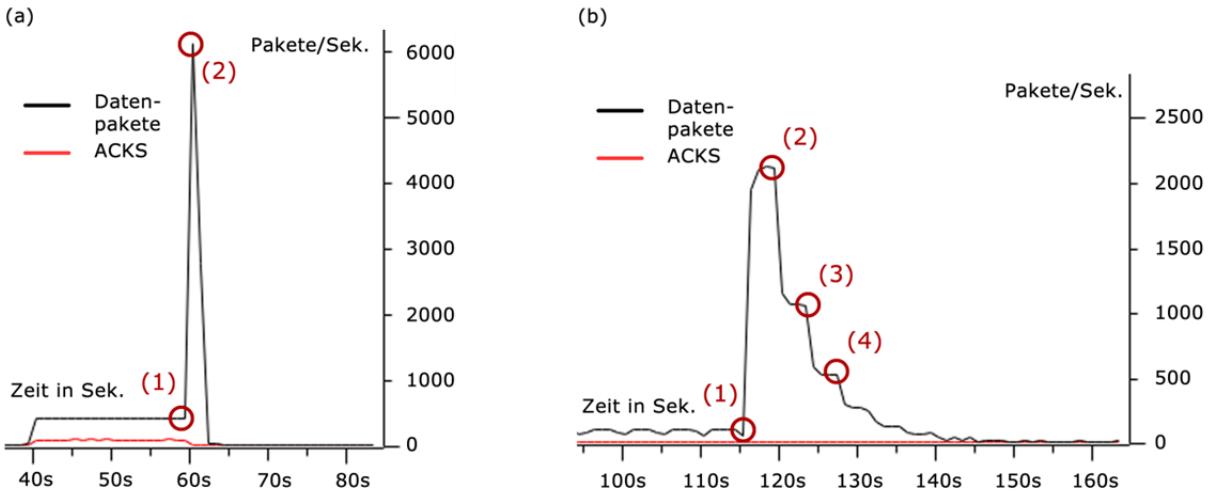


Abbildung 112: DCCP-Datenübertragung großer Paketgrößen mit CCID3-Staualgoritmus a) ohne Verzögerung b) 1000ms ACK-Verzögerung

Jede Halbierung der Senderate nach Ablauf des NoFeedback Timers von CCID3 ist deutlich an den Stufen (Zeitpunkt (3) & Zeitpunkt (4) in Abbildung 112 (b)) im Graph zu erkennen. Auch die steigenden Intervalle bis zur Halbierung der Senderate zeigen sich in der zum Ende hin abfallenden negativen Steigung des Graphen. Die immer langsamer werdende Drosselung der Datenrate wird durch einen stetig erhöhten NoFeedback Timer erreicht.

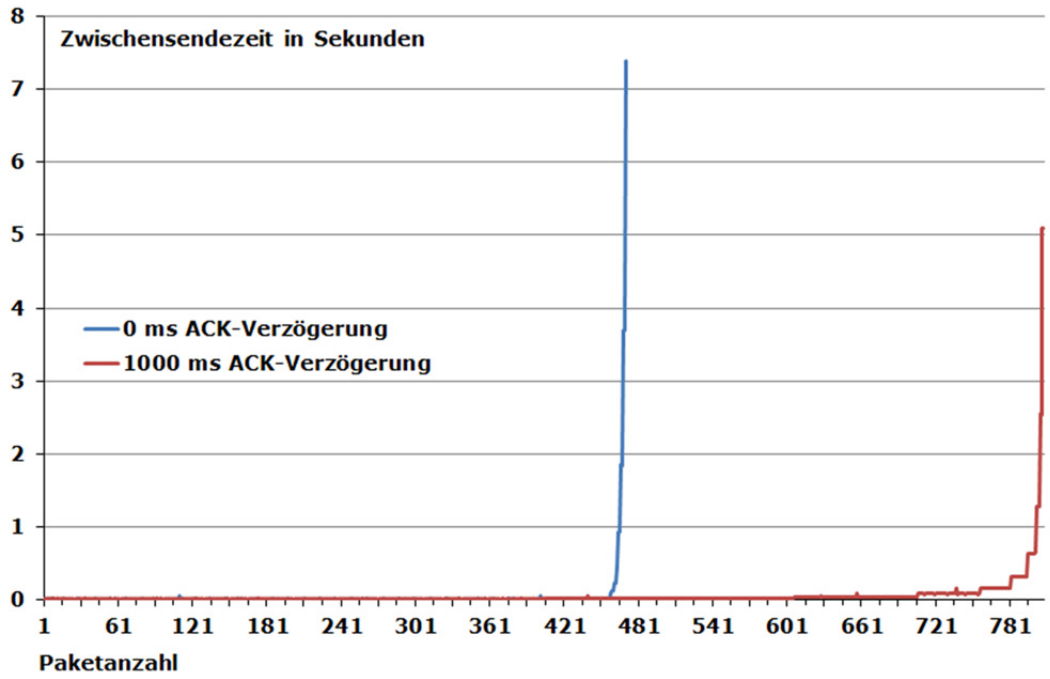


Abbildung 113: Entwicklung der Zwischensendezeit nach Aktivierung der Schleife (große Pakete)

Betrachtet man jedoch die Anzahl der in die Schleife gesendeten Pakete und die sich analog dazu entwickelnde Zwischensendezeit (Abbildung 113), fällt auf, dass die erhöhte RTT auf den ersten Blick einen umgekehrten Effekt zu den vorigen Tests ausübt. Es werden mehr Pakete in die Schleife gesendet. Jedoch wird die höhere Paketanzahl über einen längeren Zeitraum verteilt. Damit glättet eine erhöhte RTT bei großen Paketen in jedem Fall die maximale Belastung der Knoten pro Sekunde.

4.4.3.2.4 FAZIT[1]

Das Ziel von CCID3 liegt in einer gleichmäßigen Senderate. Damit wird auch eine langsamere Reaktion als CCID2 auf Schleifen impliziert. Die Kernfunktionalität von CCID3 liegt in der Berechnung der Senderate und in der Berechnung des NoFeedback Timers (siehe 2.6.4.2.2). Damit CCID3 funktionieren kann, muss die Paketgröße über die Zeit konstant bleiben. Zur Berechnung der Senderate werden die Paketgröße und die RTT miteinbezogen. Bei kleinen Paketen zeigt sich ein stärkerer Einfluss der RTT auf die Senderate als bei großen Paketen. Eine höhere RTT impliziert eine deutlich niedrigere Senderate und damit auch eine niedrigere Anzahl von Paketen, die durch eine Schleife laufen. Bei großen Paketen ist der Effekt auf die Senderate bedeutend niedriger. Unter Umständen werden sogar mehr Pakete die Schleife durchlaufen, allerdings über einen größeren Zeitraum. Damit mindert die RTT in jedem Fall die maximale Belastung einer Schleife in der Spitze. Abschließend bleibt zu sagen, dass CCID3 selbst in der experimentellen Implementierung relativ rasch auf Schleifen reagiert und dementsprechend einen Vorteil gegenüber UDP bietet.

4.5 TCP VS. UDP PAKETE[2]

In den vorherigen Versuchen haben wir bisher das Verhalten von TCP- und UDP-Paketen isoliert voneinander untersucht. In diesem Abschnitt wollen wir nun die Wechselwirkung dieser beiden dominierenden Transportprotokolle genauer betrachten. Zuerst werden wir dabei die gegenseitige Beeinflussung der Übertragungsraten ohne eine Forwarding-Schleife untersuchen und im Anschluss den UDP-Verkehr in eine Forwarding-Schleife senden, während die TCP-Verbindung unverändert bleibt.

Für diese Untersuchung müssen wir die vorhandene Netzwerktopologie um zwei Host-Systeme erweitern, sowie eine neue gesonderte Route einfügen (Abbildung 114 a)). Durch die zusätzliche Route (Zielnetzwerk 10.0.30.0/24) ist es uns später möglich diese als Forwarding-Schleife zu erzwingen während die andere Route unverändert bleibt (Abbildung 114 b)). Darin wird von Host B eine TCP-Verbindung zu Host D aufgebaut und fortlaufend Pakete mit einer Größe von 1404 Byte versendet. Parallel dazu überträgt Host A, über die gleichen Netzwerkressourcen, UDP-Pakete von jeweils 62 Byte bzw. 1442 Byte Größe an Host C.

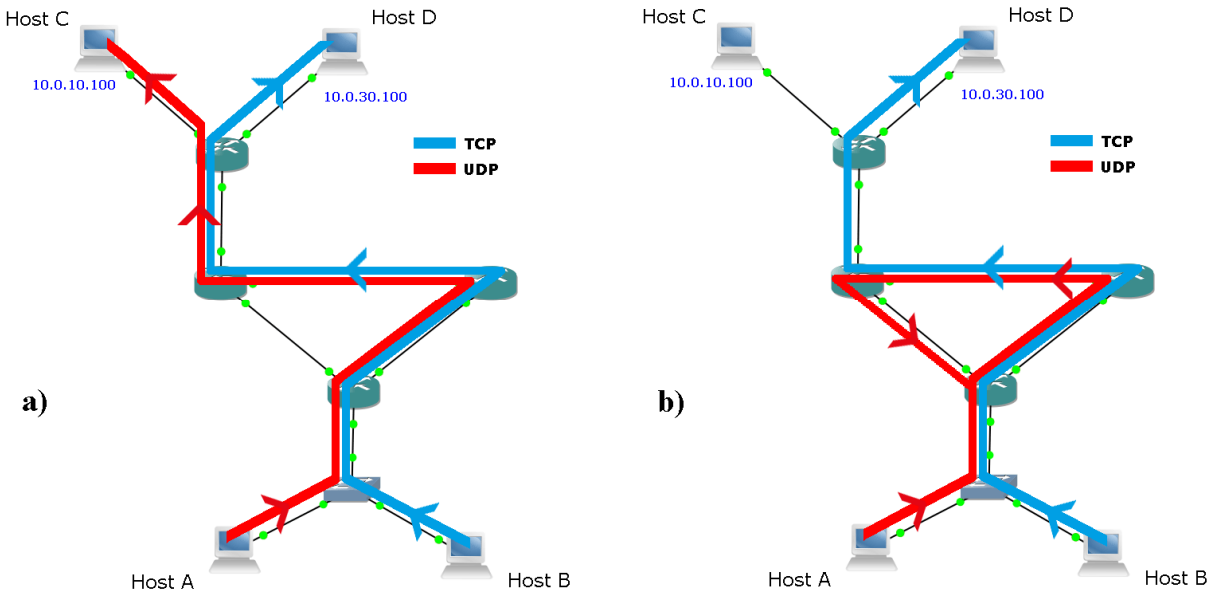


Abbildung 114: Erweiterte Netzwerktopologie zum parallelen Testen von TCP- und UDP-Verkehr a) normale Übertragung b) UDP-Verkehr in Forwarding-Schleife

Als Messpunkt wählen wir dabei das Interface eth0 von Router R3 (vgl. Abbildung 76), so dass wir auch noch in der Forwarding-Schleife beide Datenströme am gleichen Interface abgreifen können. Wichtig zu erwähnen ist dabei, dass bei den Messungen zu TCP nur der Datenverkehr von Host B in Richtung Host D gemessen wird. Dies bedeutet, dass die ACK‘s von Host D an Host B nicht gemessen und nur die eigentlichen Datenpakete erfasst werden.

4.5.1 TCP vs. UDP ohne Forwarding-Schleife[2]

Im ersten Test beschränken wir uns auf die Untersuchung einer TCP- und einer UDP-Verbindung ohne Forwarding-Schleife. Host C baut dabei eine Verbindung zu Host D auf und überträgt anschließend mit einem recht konstanten Durchsatz von 60 Paketen pro Sekunde

1404 Byte TCP-Pakete an Host C (Abbildung 115 a)). Etwa 24 Sekunden nach dem Start der Aufzeichnung startet Host A parallel eine Datenübertragung von 62 Byte großen UDP-Paketen an Host C. Wie zu erkennen ist, sinkt dadurch die Übertragungsrate der TCP-Verbindung auf etwa 53 Pakete/s und die UDP-Verbindung überträgt ca. 320 Pakete/s. Dies bedeutet also eine Reduzierung der TCP-Pakete von etwa 13%. Aus Abschnitt 4.4.2.2. wissen wir, dass die exklusive Übertragung von UDP-Paketen über unser Testszenario bei ca. 400 Paketen/s lag. In unserem jetzigen Test, bei gleichzeitiger Übertragung von TCP-Paketen über die gleiche Verbindung, sinkt die Rate auf ca. 330 Pakete/s, was einem Verlust von ca. 19% entspricht.

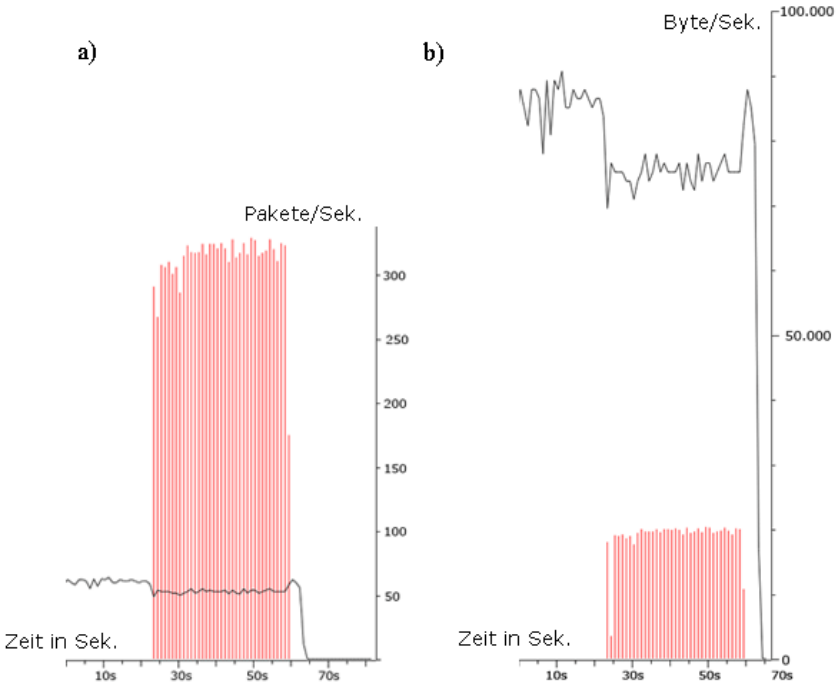


Abbildung 115: TCP vs. 46 Byte-UDP-Pakete a) Pakete/s b) Byte/s

Im gleichen Verhältnis ändert sich dabei auch das übertragene Datenvolumen. Vor dem Start der UDP-Übertragung messen wir beim TCP-Datenstrom einen Datendurchsatz von ~87.000 Byte/s welcher anschließend auf ~76.000 Byte/s sinkt. Dies entspricht, wie auch schon bei der Übertragungsrate der Pakete, einem Verlust von ~13%. Im Vergleich zum Datenvolumen der TCP-Verbindung ist die Transferrate der UDP-Übertragung recht gering, was aus der kleinen Paketgröße resultiert. Hier sinkt die übertragene Datenmenge von 25.000 Byte/s im vorherigen Versuch auf nun 20.000 Byte/s, was analog zur Paketrate einer Herabstufung um

19% entspricht. Erkennbar ist auch, dass nach dem Abbruch der UDP-Übertragung die TCP-Verbindung sowohl bei der Paket- als auch Datenrate auf ihre ursprünglichen Werte steigt.

Als nächsten betrachten wir das Verhalten bei einer parallelen Übertragung von 1442 großen UDP-Paketen. Nachdem wir 32 Sekunden nach Start der Messung die UDP-Übertragung starten erkennen wir, dass der Durchsatz von TCP-Paketen von anfangs 60 Paketen/s auf 49 Pakete/s sinkt. Gleichermaßen sinkt das transferierte Datenvolumen von ~87.000 Byte/s auf ~69.500 Byte/s. Dies bedeutet also, dass bei einer gleichzeitigen Übertragung von großen UDP-Paketen die Übertragungsrate der TCP-Verbindung um etwa 19% reduziert wird.

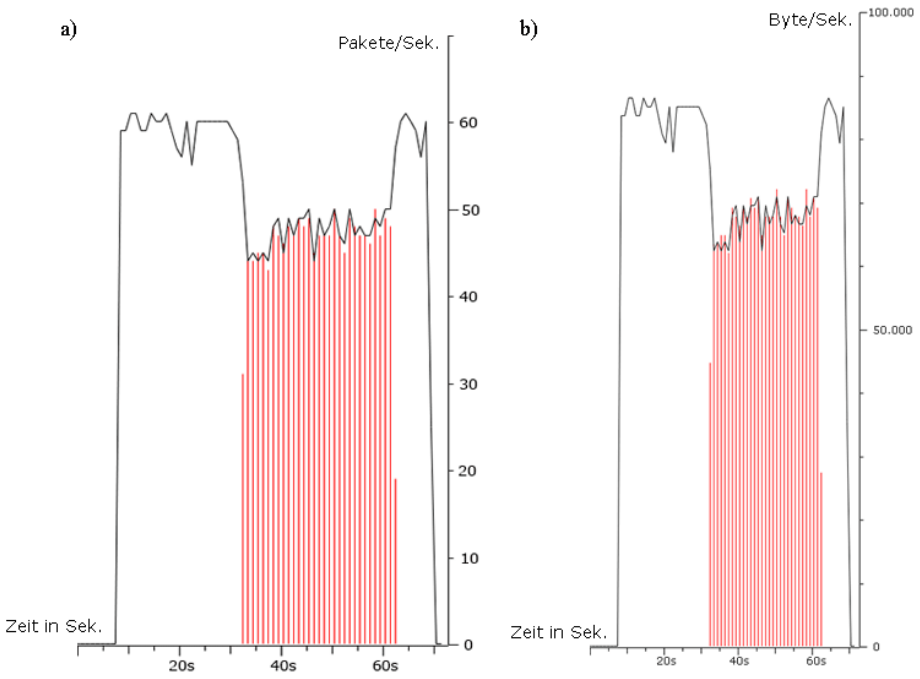


Abbildung 116: TCP vs. 1442 Byte-UDP-Pakete a) Pakete/s b) Byte/s

Bei der Übertragung der UDP-Pakete wissen wir noch aus Abschnitt 4.4.2.2, dass die Übertragungsrate ohne Beeinträchtigung bei 60 Paketen und 87.000 Byte pro Sekunde liegt. Teilt sich die UDP-Übertragung die Netzwerkressourcen mit einer TCP-Verbindung, fällt die Datentransferrate auf 48 Pakete/s und 69.000 Byte/s. Dies entspricht also auch wie bei den TCP-Paketen einem um 20% verringerten Transfervolumen. Daraus lässt sich ableiten, dass die Weiterleitung von gleichgroßen TCP- und UDP-Paketen auf einer Verbindung mit der gleichen Übertragungsrate abläuft. Ebenfalls zu erkennen ist, dass nach dem Beenden der UDP-Übertragung die TCP-Verbindung auf ihr ursprüngliches Transferniveau steigt.

4.5.2 TCP VS. UDP MIT FORWARDING-SCHLEIFE[2]

In diesem Abschnitt wollen wir die gleiche Untersuchung wie in Abschnitt 4.5.1. durchführen, diesmal werden jedoch die UDP-Pakete in eine Forwarding-Schleife geleitet (vgl. Abbildung 114 b)).

Im ersten Fall bauen wir wieder eine TCP-Verbindung auf, welche wie erwartet eine Transferrate von ~60 Paketen und ~87.000 Byte pro Sekunde aufweist. Nachdem wir nach etwa 25 Sekunden nach dem Start der Messung den UDP-Verkehr in die erzeugte Forwarding-Schleife leiten, sinkt der Durchsatz auf ~37 Pakete/s (Abbildung 117 a)) und ~51.000 Byte/s (Abbildung 117 b)). Prozentual betrachtet entspricht dies einem Verlust von etwa 40%.

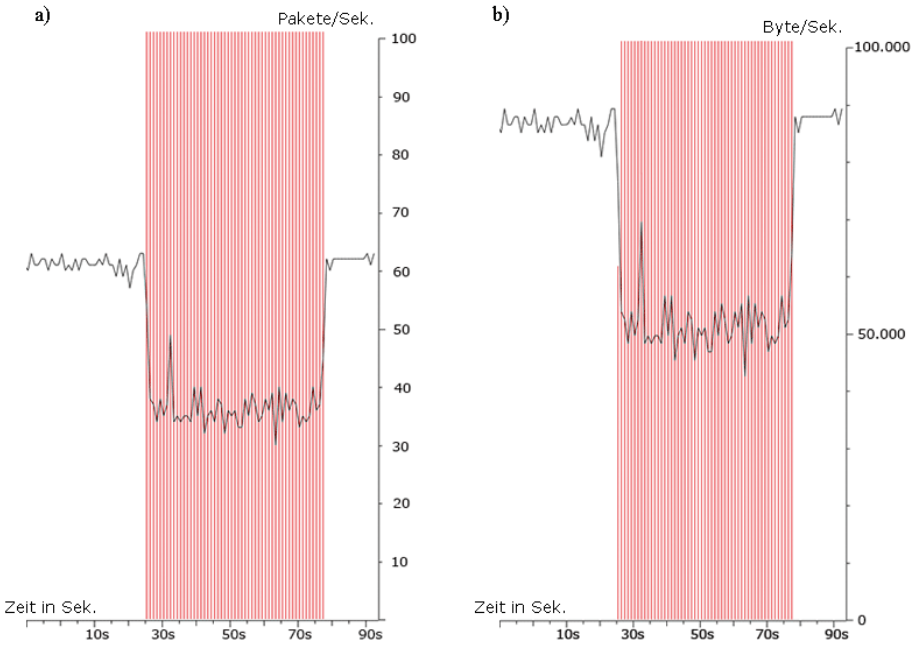


Abbildung 117: TCP vs. 46 Byte-UDP-Pakete in Forwarding-Schleife a) Pakete/s b) Byte/s

Die Übertragungsrate des UDP-Verkehr entspricht in der Forwarding-Schleife, wie auch aus Abschnitt 4.4.2.2. bekannt, ~6.000 Pakete/s und 390.000 Byte/s. In unserem Testfall sank diese Transferrate auf 3.400 Packete und 210.000 Byte pro Sekunde, was einer Verminderung von ~46% entspricht. Nachdem das Absenden der UDP-Pakete in die Rounting-Schleife unterbrochen wird, regeneriert sich die TCP-Verbindung umgehend und erreicht wieder die anfänglichen Werte.

Abschließend folgt noch die Untersuchung der Wechselwirkung mit 1442 Byte großen UDP-Paketen in einer Forwarding-Schleife. Hier sinkt die Übertragungsrate des TCP-Verkehrs von 60 Paketen auf 48 Pakete in der Sekunde. Das Datenvolumen der gleichen Verbindung sinkt von 87.000 Byte/s auf 68.000 Byte/s. Dies wiederum entspricht einer Verminderung von etwa 20%.

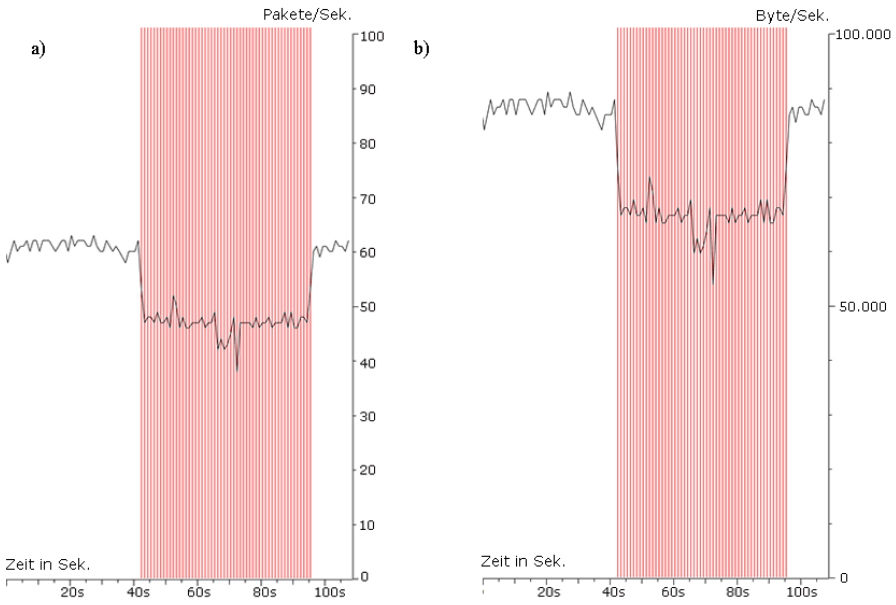


Abbildung 118: TCP vs. 1442 Byte-UDP-Pakete in Forwarding-Schleife a) Packete/s b) Byte/s

Bei der Übertragungsgeschwindigkeit der UDP-Pakete, welche ja in einer Schleife zirkulieren, verändert sich der Datendurchsatz von ~1.300 Paketen/s auf ~1.000 Pakete/s. Gleichermaßen sinkt das Datenvolumen der UDP-Pakete von ~1.900.000 Byte/s um 20% auf ~1.460.000 Byte/s.

4.5.3 FAZIT[2]

Die Ergebnisse aus diesem Kapitel scheinen etwas überraschend, da aufgrund der Protokolleigenschaften ein viel größerer Einfluss des UDP-Verkehrs auf die Übertragungsraten der TCP-Verbindung zu erwarten wäre. Da das UDP als verbindungsloses Protokoll keine Fluss- oder Staukontrolle besitzt, wäre anzunehmen, dass der UDP-Datenverkehr den TCP-Datenstrom immer weiter verdrängt, da dieser seine Übertragungsgeschwindigkeit bei einer überlasteten Route fortlaufend verringert.

In Abschnitt 4.5.1. haben wir feststellen können, dass selbst bei einer normalen Übertragung von UDP- und TCP-Paketen auf einem Verbindungsweg die Übertragungsraten von UDP-Paketen im Gegensatz zum TCP-Verkehr stärker abfallen. Wird der UDP-Verkehr dabei nicht in eine Forwarding-Schleife geleitet, ist die Auswirkung von großen UDP-Paketen auf einem parallelen Verbindungsweg größer als der von kleinen UDP-Paketen. Dabei ist zu erkennen, dass sich bei etwa gleich großen Paketen für TCP und UDP die Übertragungsraten angleichen und beiden Transportprotokollen gleichwertige Geschwindigkeiten zur Verfügung stehen.

Bei einer Forwarding-Schleife, auf dem parallelen Übertragungsweg der TCP Verbindung, hat die enorm größere Anzahl an UDP-Paketen innerhalb dieser Schleife einen höheren Einfluss auf die TCP-Verbindung. Hier sinkt die Durchsatzrate der TCP-Pakete um ~40% während große UDP-Pakete in der Forwarding-Schleife die Datenrate nur um ~20% verringern. In beiden Fällen ist jedoch auch bei der Übertragungsrate der UDP- Pakete in der Schleife der Verlust mit ~46% bzw. ~20% größer oder mindestens gleich groß.

Zu erklären ist dieses Verhalten mit den Eigenschaften der Puffer-Verwaltung in den Vyatta Routern. Hier kommt das sog. Stochastic Fair Queuing (SFQ) [96] zum Einsatz, welches ein faires Traffic-Shaping Verfahren für parallele Datenströme auf einem Router garantiert. Diese Implementierung bevorzugt tendenziell größere Datenpakete, was auch die größeren Unterschiede zwischen den Verlustraten von kleinen UDP-Paketen in Verbindung mit großen TCP-Paketen erklärt (vgl. Abbildung 115 vs. Abbildung 117).

Dieses Ergebnis zeigt jedoch, das nicht alleine die Transportprotokolle und deren Eigenschaften eine Auswirkung auf die Wechselwirkung der unterschiedlichen Datenströme haben. Neben dem SFQ gibt es noch eine Vielzahl weiterer Implementierung zur Verwaltung der Warteschlangen, darunter etwa das DRR (Deficit Round Robin) [97] oder auch CBQ (Class-based Queuing) [98]. Wünschenswert ist jedoch, dass eine Überlastung der Router bereits vom Transportprotokoll selbst verhindert wird und die Kommunikationspartner selbst ihre Übertragungsgeschwindigkeit durch das dazwischenliegende Netz regulieren.

Nicht zu vergessen ist jedoch, dass die hier aufgezeigten Testergebnisse auf virtuellen Maschinen erzielt wurden. Hardware-basierte Router benutzen recht unterschiedliche Verfahren um eine Überlastung möglichst zu vermeiden und unterschiedliche Datenströme mir gleicher Fairness zu behandeln. Die Ergebnisse zeigen also nur einen möglichen Verlauf

einer parallelen Datenübertragung von UDP- und TCP-Paketen auf dem gleichen Netzwerkpfad.

5 FAZIT[2]

Bei unseren Untersuchungen zu Forwarding-Schleifen haben wir festgestellt, dass die Analyse des Internetverkehrs keine triviale Angelegenheit ist. Um die Protokollverteilung im Internet zu bestimmen, sind detaillierte Aufzeichnungen des gesamten Verkehrs zu erstellen und analysieren. Diese Aufzeichnungen sind aufgrund der riesigen Datenmengen, welche in kurzer Zeit anfallen, nur mit großem Aufwand zu erstellen. Eine anschließende Analyse ist mit einem enormen Rechenaufwand verbunden. Ebenso ist derer Zugang zu solch detaillierten Mitschnitten von Internetverkehr ist auch sehr eingeschränkt, da nur wenige Einrichtungen den öffentlichen Zugang zu diesen Daten gewähren.

Nach der Analyse der uns vorliegenden Aufzeichnungen und anderen Arbeiten ist jedoch eindeutig, dass TCP das dominierende Transportprotokoll ist. Die Verteilung liegt insgesamt bei etwa 79% TCP- und etwa 19% UDP-Verkehr. Sehr deutlich ist aber der steigende Anteil an UDP Datenverkehr in den letzten Jahren, was vor allem auf die zunehmende Anzahl an Streaming-Diensten zurückzuführen ist. Neuere Transportprotokolle wie DCCP oder SCTP waren dabei kaum im Internet zu registrieren.

Bei den Forwarding-Schleifen selbst ist eine Unterscheidung zwischen persistenten und temporären Schleifen zu treffen. Persistente, also dauerhafte Forwarding-Schleifen, entstehen durch fehlerhafte Konfiguration und sind auch nur manuell aufzulösen. Temporäre Schleifen kommen während und auf Grund der Konvergenzzeit von Routing-Protokollen zustande. Temporäre Schleifen haben in den meisten Fällen eine sehr kurze Lebensdauer von unter zehn Sekunden, wodurch das Aufspüren recht schwer fällt. Um eine solche Schleife nachzuweisen, eignet sich daher am besten die Analyse der oben beschriebenen Paketaufzeichnungen. Persistente Schleifen existieren über einen längeren Zeitraum, 20% sogar über 7,5 Stunden. Nachweisbar ist diese Art von Schleifen daher mit Traceroute, eine Messung ist auch von außerhalb der Schleife möglich. Der größte Anteil von Schleifen involviert dabei nur zwei Hops und ist überwiegend in den Zielnetzen zu finden. Forwarding-Schleifen, welche sich über ein autonomes System erstrecken, sind hingegen sehr selten. Insgesamt sind jedoch 2% aller gerouteten IP-Adressen von einer Routing-Schleife getroffen.

Um temporäre Forwarding-Schleifen zu vermeiden, ist eine Verbesserung der Infrastruktur oder eine Erweiterung der Routing-Protokolle hilfreich. Persistente Schleifen werden überwiegend durch Konfigurationsfehler verursacht, welche durch eine Verbesserung der Administration im betroffenen Netzwerk vermieden werden können.

Die Untersuchungen der einzelnen Transportprotokolle haben gezeigt, dass lediglich UDP nicht gegen eine Forwarding-Schleife abgesichert ist. Aufgrund seiner Staukontrolle wird bei TCP die Verbindung unterbrochen, wenn während der Übertragung eine Schleife entsteht. Ähnlich war auch das Verhalten von DCCP und es war immer nur eine kurzfristige Erhöhung des Datenaufkommens nach der Schleifenbildung zu messen. Da UDP keine solche Staukontrolle bietet, ist dabei eine fortlaufend erhöhte Datenrate innerhalb der Forwarding-Schleife zu messen und auch ein automatischer Verbindungsabbruch findet nicht statt. Da UDP jedoch nur zum Datentransfer einer Anwendung genutzt wird, welche aber gleichzeitig noch die Ende-zu-Ende Verbindung mit weiteren Protokollen kontrollieren, ist eine dauerhafte Flutung einer Forwarding-Schleife mit UDP Paketen auch auszuschließen. Ist eine Forwarding-Schleife bekannt, können die involvierten IP Adressen jedoch ganz bewusst mit UDP Paketen adressiert und die beteiligten Komponenten überlastet werden.

Ohne böswillige Absichten sind die Transportprotokolle gut gegen Forwarding-Schleifen abgesichert. Um die wachsende Anzahl an Streaming-Diensten zu verbessern, wäre die Verwendung eines Transportprotokolls mit Staukontrolle empfehlenswert. Das oftmals verwendete TCP bietet für echtzeitkritische Anwendungen einen unnötigen Überfluss an Kontrollpaketen. Zusätzlich ist eine wiederholte Übertragung verlorener Pakete nicht gewünscht. Das von uns untersuchte Transportprotokoll DCCP wäre eine gute Alternative für diese Anwendungen, befindet sich aber leider noch in einer experimentellen Entwicklungsphase.

ANHANG[2]

Im Anhang an diese Diplomarbeit befindet sich eine DVD mit den wichtigsten Elementen in digitaler Form. Der Verzeichnisinhalt ist dabei wie folgt:

/CAPTURES/	enthält die Capture-Dateien der einzelnen Tests (alle in .pcap Dateiformat)
/CAPTURES/PING/	enthält die Aufzeichnungen der TCP-Testreihe
/CAPTURES/TRACEROUTE/	enthält die Aufzeichnungen der TCP-Testreihe
/CAPTURES/TCP/	enthält die Aufzeichnungen der TCP-Testreihe
/CAPTURES/UDP/	enthält die Aufzeichnungen der UDP-Testreihe
/CAPTURES/TCP_UDP/	enthält die Aufzeichnungen der Testreihe TCP vs. UDP
/CAPTURES/DCCP_CCID2/	enthält die Aufzeichnungen der DCCP-Testreihe mit CCID2-Staukontrolle
/CAPTURES/DCCP_CCID3/	enthält die Aufzeichnungen der DCCP-Testreihe mit CCID3-Staukontrolle
/CLIENTSERVER/	enthält die verwendeten Python-Skripte der Client-Server Testumgebung
/DOKUMENTE/	enthält die Diplomarbeit in Digitaler Form (.docx und .pdf) sowie die Präsentation (.pptx)
/VIRTUALBOX/	enthält die im Testszenario verwendeten virtuellen Maschinen R1-R4 als importierbares Image für VirtualBox (.ova)

LITERATURVERZEICHNIS

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, “Internet inter-domain traffic,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 75–86, 2010.

[2] “Exterior-Gateway-Protokoll.” [Online]. Available: http://de.wikipedia.org/wiki/Exterior-Gateway-Protokoll. [Accessed: 20-Oct-2012].

[3] “Interior-Gateway-Protokoll.” [Online]. Available: http://de.wikipedia.org/wiki/Interior_Gateway_Protocol. [Accessed: 20-Dec-2012].

[4] J. Xia, L. Gao, and T. Fei, “Flooding attacks by exploiting persistent forwarding loops,” *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement - IMC ’05*, p. 1, 2005.

[5] AS65000, “BGP Routing Table Analysis Report.” [Online]. Available: http://bgp.potaroo.net/as2.0/bgp-active.html. [Accessed: 10-Dec-2012].

[6] K. Coffman and A. Odlyzko, “Internet growth: Is there a’Moore's Law'for data traffic?,” *SSRN 236108*, 2000.

[7] Wikipedia, “Dotcom-Blase.” [Online]. Available: http://de.wikipedia.org/wiki/Dotcom-Blase. [Accessed: 10-Feb-2013].

[8] Wikipedia, “Internet Traffic.” [Online]. Available: http://en.wikipedia.org/wiki/Internet_traffic. [Accessed: 17-Oct-2012].

[9] CISCO, “Visual Networking Index (VNI).” [Online]. Available: http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html. [Accessed: 13-Oct-2012].

[10] A. Dainotti, A. Pescapè, and K. Claffy, “Issues and future directions in traffic classification,” *Network, IEEE 26.1 (2012)*, pp. 35–40, 2012.

[11] B. Choi, J. Park, and Z. Zhang, “Adaptive packet sampling for accurate and scalable flow measurement,” *GLOBECOM’ 2004 Conference*, vol. M, 2004.

[12] N. Duffield, “Sampling for Passive Internet Measurement: A Review,” *Statistical Science*, vol. 19, no. 3, pp. 472–498, Aug. 2004.

[13] M. Crovella and T. Friedman, “Community-oriented network measurement infrastructure (CONMI) workshop report,” *ACM SIGCOMM Computer Communication Review 36.2*, pp. 41–48, 2006.

[14] J. Apisdorf, K. Thompson, and R. Wilder, “performance statistics collection part I : design and implementation,” 1996.

[15] Endace, “DAG Packet-Capture Cards,” 2008. [Online]. Available: http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html. [Accessed: 23-Nov-2012].

[16] C. Fraleigh, S. Moon, B. Lyles, and C. Cotton, “Packet-level traffic measurements from the Sprint IP backbone,” *Network, ...*, no. December, pp. 6–16, 2003.

[17] H. Jiang and C. Dovrolis, “Passive estimation of TCP round-trip times,” *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, Jul. 2002.

[18] Information Society Technology, “LOBSTER Project.” [Online]. Available: http://www.ist-lobster.org. [Accessed: 13-Feb-2013].

[19] Edgewall Software, “Stager.” [Online]. Available: https://trac.uninett.no/stager/. [Accessed: 20-Feb-2013].

[20] B. Claise, “RFC 5101 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.” [Online]. Available: https://tools.ietf.org/html/rfc5101. [Accessed: 20-Feb-2013].

[21] “MOMENT Project,” 2010. [Online]. Available: http://cordis.europa.eu/projects/index.cfm?fuseaction=app.details&TXT=MOMENT+LOBSTER&FRM=1&STP=10&SIC=&PGA=&CCY=&PCY=&SRC=&LNG=en&REF=85537 . [Accessed: 26-Feb-2013].

[22] “ETOMIC: European Traffic Observatory Measurement Infrastructure.” [Online]. Available: http://www.etomic.org/. [Accessed: 26-Feb-2013].

[23] Y. Shavitt, “The DIMES Project.” [Online]. Available: http://www.netdimes.org. [Accessed: 26-Feb-2013].

[24] CAIDA, “CoralReef Software Suite.” [Online]. Available: http://www.caida.org/tools/measurement/coralreef/. [Accessed: 27-Jan-2013].

[25] T. Karagiannis, A. Broido, N. Brownlee, and U. C. S. Diego, “Is P2P dying or just hiding ?,” 2003.

[26] S. Diego and F. Directions, “Trends in Wide Area IP Traffic Patterns,” no. May 1999.

[27] M. Fomenkov, K. Keys, and D. Moore, “Longitudinal study of Internet traffic in 1998-2003,” 2003.

[28] J. Xu and J. Fan, “Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme,” in *10th IEEE International Conference on Network Protocols*, 2002.

[29] CAIDA, “Data - Overview of Datasets, Monitors, and Reports.” [Online]. Available: http://www.caida.org/data/overview/. [Accessed: 17-Oct-2012].

[30] WAND Group, “Waikato Internet Traffic Storage.” [Online]. Available: http://www.wand.net.nz/wits/. [Accessed: 20-Dec-2012].

[31] University of Minnesota, “Minnesota Internet Traffic Studies.” [Online]. Available: http://www.dtc.umn.edu/mints/. [Accessed: 18-Dec-2012].

[32] NLANR Research Group, “NLANR PMA Data.” [Online]. Available: https://labs.ripe.net/datarepository/data-sets/nlanr-pma-data. [Accessed: 13-Jan-2013].

[33] “18 united states code §2511.” [Online]. Available: http://www.law.cornell.edu/uscode/text/18/2511. [Accessed: 14-Feb-2013].

[34] G. Minshall, “TCPDPRIV.” [Online]. Available: http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html. [Accessed: 14-Mar-2013].

[35] Wireshark-Community, “Wireshark.” [Online]. Available: http://www.wireshark.org/. [Accessed: 17-Sep-2012].

[36] Information Sciences Institute University of Southern California, “RFC 791 - Internet Protocol,” 1981. [Online]. Available: http://tools.ietf.org/html/rfc791. [Accessed: 14-Mar-2013].

[37] IANA, “IANA IPv4 Special Purpose Address Registry.” [Online]. Available: http://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xml. [Accessed: 21-Jan-2013].

[38] Wikipedia, “Regional Internet Registry.” [Online]. Available: http://de.wikipedia.org/wiki/Regional_Internet_Registry. [Accessed: 21-Jan-2013].

[39] T. Jessup, “Protocol Header Images,” 2010. [Online]. Available: http://www.troyjessup.com/headers/. [Accessed: 21-Jan-2013].

[40] IANA, “Protocol Numbers.” [Online]. Available: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml. [Accessed: 14-Mar-2013].

[41] Wikipedia, “Transmission Control Protocol.” [Online]. Available: http://de.wikipedia.org/wiki/Transmission_Control_Protocol. [Accessed: 18-Apr-2013].

[42] J. Postel, “RFC 768 - User Datagram Protocol,” 1980. [Online]. Available: http://xml2rfc.tools.ietf.org/html/rfc768. [Accessed: 14-Mar-2013].

[43] “RFC 4340 - Datagram Congestion Control Protocol (DCCP),” 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt. [Accessed: 01-Jul-2013].

[44] D. Ramakrishnan, K., Floyd, S., & Black, "The Addition of Explicit Congestion Notification (ECN) to IP." 2001.

[45] D. Murray and T. Koziniec, "The state of enterprise network traffic in 2012," *18th Asia-Pacific Conference on Communications (APCC)*, pp. 179–184, Oct. 2012.

[46] "RFC 3448 - TCP Friendly Rate Control." [Online]. Available: http://www.ietf.org/rfc/rfc3448.txt. [Accessed: 23-Jul-2013].

[47] "RFC 4342 - Congestion Control ID 3." [Online]. Available: http://www.ietf.org/rfc/rfc4342.txt. [Accessed: 01-Jul-2013].

[48] BCIX, "Traffic Statistics," 2013. [Online]. Available: http://www.bcix.de/bcix/statistik/. [Accessed: 07-Mar-2013].

[49] W. John and S. Tafvelin, "Analysis of internet backbone traffic and header anomalies observed," *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, p. 111, 2007.

[50] W. John, "On measurement and analysis of internet backbone traffic," Chalmers University of Technology, 2008.

[51] M. Zhang, M. Dusi, W. John, and C. Chen, "Analysis of UDP Traffic Usage on Internet Backbone Links," *2009 Ninth Annual International Symposium on Applications and the Internet*, pp. 280–281, Jul. 2009.

[52] SUNET, "SUNET." [Online]. Available: www.sunet.se. [Accessed: 14-Mar-2013].

[53] C. Lee, D. Lee, and S. Moon, "Unmasking the growing UDP traffic in a campus network," *Passive and Active Measurement*, 2012.

[54] Microsoft, "Skype." [Online]. Available: www.skype.de. [Accessed: 24-Mar-2013].

[55] Sopcast, "Sopcast." [Online]. Available: http://www.sopcast.com. [Accessed: 11-Feb-2013].

[56] H. Sawashima, Y. Hori, H. Sunahara, and Y. Oie, "Characteristics of UDP packet loss: Effect of tcp traffic," in *Proceedings of INET'97: The Seventh Annual Conference of the Internet Society*, 1997.

[57] Y. Wang, "Conditional Drop Probability of RED Mechanism with UDP and TCP Traffic," *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, vol. 1, pp. 813–820, 2005.

[58] J. Postel, "The TCP maximum segment size and related topics," 1983.

[59] J. Xia, L. Gao, and T. Fei, "A measurement study of persistent forwarding loops on the Internet," *Computer Networks*, vol. 51, no. 17, pp. 4780–4796, Dec. 2007.

[60] P. Francois and O. Bonaventure, “Avoiding Transient Loops During the Convergence of Link-State Routing Protocols,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1280–1292, Dec. 2007.

[61] U. Hengartner, S. Moon, R. Mortier, and C. Diot, “Detection and analysis of routing loops in packet traces,” *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 107–112, 2002.

[62] H. Z. Le, F., Geoffrey G. Xie, “Understanding Route Aggregation,” CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2010.

[63] L. Franck, G. G. Xie, and H. Zhang, “Instability Free Routing : Beyond One Protocol Instance,” in *Proceedings of the 2008 ACM CoNEXT Conference*, 2008.

[64] V. Paxson, “End-to-end routing behavior in the Internet,” *Transactions on Networking, IEEE/ACM*, vol. 6692, no. October, pp. 601–615, 1997.

[65] H. Breitbach, “IP Fast ReRoute (IPFRR): Loop-free Alternates,” Universität Koblenz-Landau, 2007.

[66] Z. Zhong, R. Keralapura, and S. Nelakuditi, “Avoiding transient loops through interface-specific forwarding,” *Quality of Service – IWQoS 2005*, pp. 219–232, 2005.

[67] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, “Delayed Internet routing convergence,” *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM ’00*, pp. 175–187, 2000.

[68] I. Van Beijnum and J. Crowcroft, “Loop-freeness in multipath BGP through propagating the longest path,” *2009. ICC Workshops*, 2009.

[69] Wikipedia, “Routing.” [Online]. Available: http://de.wikipedia.org/wiki/Routing. [Accessed: 19-Feb-2013].

[70] Wikipedia, “Tier 1 Network.” [Online]. Available: http://en.wikipedia.org/wiki/Tier_1_network. [Accessed: 10-Jan-2013].

[71] Wikipedia, “Tier 2 Network.” [Online]. Available: http://en.wikipedia.org/wiki/Tier_2_network. [Accessed: 10-Jan-2013].

[72] Universität Oldenburg, “Rechnernetze - Routing Information Protocol.” [Online]. Available: http://einstein.informatik.uni-oldenburg.de/rechnernetze/rip.htm. [Accessed: 19-Feb-2013].

[73] C. Steigner, “Rechnernetze I & II.” Universität Koblenz-Landau, 2006.

[74] F. Bohdanowicz, “Avoidance of Routing Loops Arbeitsberichte aus dem Fachbereich Informatik,” no. 1, 2009.

[75] J. Doyle and C. Jennifer, *Routing TCP/IP*. Cisco Press, 2005.

[76] J. Hawkinson and T. Bates, "RFC1930 - Guidelines for creation, selection, and registration of an Autonomous System (AS)." [Online]. Available: http://tools.ietf.org/html/rfc1930. [Accessed: 18-Jan-2012].

[77] A. Sridharan, S. Moon, and C. Diot, "On the correlation between route dynamics and routing loops," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[78] University of Oregon, "Route Views Project." [Online]. Available: http://www.routeviews.org/. [Accessed: 21-Jan-2013].

[79] Wikipedia, "Traceroute." [Online]. Available: http://de.wikipedia.org/wiki/Traceroute. [Accessed: 20-Dec-2012].

[80] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services," in *Sixth Symposium on Operating Systems Design and Implementation*, 2004.

[81] D. Katz and D. Ward, "RFC 5880 - Bidirectional Forwarding Detection (BFD)," 2010. [Online]. Available: http://tools.ietf.org/html/rfc5880. [Accessed: 12-Jan-2013].

[82] Wikipedia, "PlanetLab." [Online]. Available: http://en.wikipedia.org/wiki/PlanetLab. [Accessed: 08-Jan-2013].

[83] P. Lapukhov, "Anomalies in BGP: Part I." [Online]. Available: http://blog.ine.com/2010/01/30/anomalies-in-bgp-part-i/. [Accessed: 07-Jan-2013].

[84] C. Science, "Activity Report 2008-2012," no. Informatik III, 2012.

[85] Jeremy Grossmann et al., "GNS3." [Online]. Available: http://www.gns3.net. [Accessed: 20-Oct-2012].

[86] Dynagen, "Dynagen." [Online]. Available: http://www.dynagen.org. [Accessed: 29-Nov-2012].

[87] AG-Rechnernetze Uni-Koblenz, "VNUML," 2009. [Online]. Available: http://www.uni-koblenz.de/~vnuml/index.de.php. [Accessed: 20-Nov-2012].

[88] Departamento de Ingeniería de Sistemas Telemáticos, "Virtual Network User Mode Linux." [Online]. Available: http://web.dit.upm.es/~vnuml/. [Accessed: 07-Mar-2013].

[89] Vyatta Inc, "Vyatta." [Online]. Available: http://www.vyatta.org/. [Accessed: 23-Oct-2012].

[90] "Wikipedia - Ping." [Online]. Available: https://de.wikipedia.org/wiki/Ping_(Datenübertragung). [Accessed: 15-Jul-2013].

[91] S. a. Baset and H. G. Schulzrinne, “An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol,” *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–11, 2006.

[92] “Maximum Segment Size.” [Online]. Available: http://de.wikipedia.org/wiki/Maximum_Segment_Size. [Accessed: 01-Jul-2013].

[93] “Compound TCP.” [Online]. Available: http://en.wikipedia.org/wiki/Compound_TCP. [Accessed: 09-Jun-2013].

[94] Linux Magazine, “Developing multimedia applications with DCCP.” [Online]. Available: http://www.linux-magazine.com/Issues/2008/93/DCCP. [Accessed: 13-Jan-2013].

[95] linuxmanpages.com, “Traffic Control (TC).” [Online]. Available: http://www.linuxmanpages.com/man8/tc.8.php. [Accessed: 12-Feb-2013].

[96] P. E. McKenney, “Stochastic fairness queueing,” in *INFOCOM’90: Ninth Annual Joint Conference of the IEEE Computer and Communication Societies*, 1990.

[97] M. Shreedhar and G. Varghese, “Efficient Fair Queuing using Deficit Round Robin,” *IEEE/ACM Transactions on Networking*, pp. 375–385, 1996.

[98] C. Semeria, “Supporting differentiated service classes: queue scheduling disciplines,” *Juniper networks*, pp. 1–27, 2001.