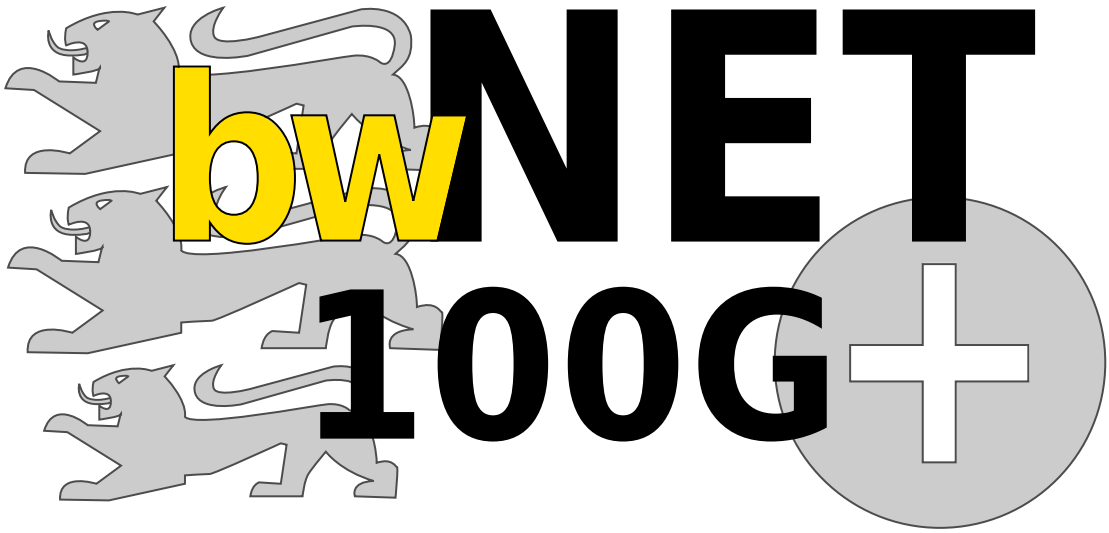
Begleitprojekt zum 100G-Ausbau des Landeshochschulnetzes BelWü



Forschung und innovative Dienste für ein flexibles 100G-Netz in Baden-Württemberg

Zwischenbericht 1: Stand der Technik

Dipl.-Inform. Mario Hock[*], Thomas Lukaseder, M.Sc.[***],
Dipl.-Inform. Mark Schmidt[**], Prof. Dr. Frank Kargl[***],
Prof. Dr. Michael Menth[**], und Prof. Dr. Martina Zitterbart[*]

[*]Karlsruher Institut für Technologie, Institut für Telematik
[**]Universität Tübingen, Lehrstuhl für Kommunikationsnetze
[***]Universität Ulm, Institut für Verteilte Systeme

14. Juli 2015

Koordinator und Ansprechpartner ALWR:
Prof. Dr. Stefan Wesner <stefan.wesner@uni-ulm.de>

Wissenschaftliche Koordination:
Prof. Dr. Martina Zitterbart <zitterbart@kit.edu>

Inhaltsverzeichnis

1 Einleitung

Das Landesforschungsnetz BelWü wird aktuell von 10 Gbit/s auf 100 Gbit/s ausgebaut. Die Verzehnfachung der physischen Datenrate wird aber nicht zwangsläufig zu einem entsprechenden Anstieg der "Quality of Experience" der Dienstnutzer führen. Viele im Gesamtsystem eingesetzten Technologien müssen erst auf die neue Herausforderung angepasst werden. Es ist das Ziel des bwNET100G+ Projektes, durch eine Kooperation zwischen Rechenzentren und Forschungslehrstühlen, Ergebnisse und Vorschläge zu entsprechenden Lösungsansätzen zu unterbreiten. In einem ersten Schritt soll dieser Bericht Herausforderungen und Lösungsansätze aus der Literatur in den Bereichen flexibler Netzbetrieb, Datentransport in Hochleistungsnetzen und Sicherheitskonzepte in Hochleistungsnetzen zusammenfassen, um daran anschließend konkrete Experimente und Forschungsarbeiten zu planen und durchzuführen.

Zur Zeit werden in der Netzwerktechnik viele innovative Technologieansätze untersucht, welche die Netzwerkverwaltung, den effektiven, zuverlässigen und schnellen Netzbetrieb sowie die Sicherheit der Netzwerke verbessern sollen. Viele Technologien an denen geforscht wird stecken aber oft erst in den Kinderschuhen und die heutige Einsetzbarkeit in Produktivnetzen ist entsprechend fraglich. Auch ist der Aspekt der hohen Geschwindigkeitsanforderungen in zukünftigen Netze nicht immer ein explizites Designziel.

Die Flexibilität des Netzwerkbetriebs ist in den letzten Jahren durch die Entwicklung von Software Defined Networking (SDN) enorm in den Fokus gerückt. SDN – der "disruptive Netzwerktechnikansatz, der ändern wird wie fast jedes Unternehmen mit Netzwerken umgeht"[1] – bietet vielfältige Möglichkeiten, um gerade in Hochgeschwindigkeitsnetzen Flexibilität im Netzwerkmanagement zu ermöglichen, auch und gerade was den weiteren Einsatz eigentlich nun zu langsamer Technologie in diesen Netzwerken angeht. Allerdings ist SDN heute nur selten im produktiven Einsatz und kann sein Potenzial daher noch nicht ausspielen. Das bwNET100G+ Projekt wird untersuchen, wie sich SDN in verschiedensten Einsatzszenarien bewährt.

Auch für die Verbesserung der Sicherheit in Netzwerken kann SDN ein Instrument – neben anderen – sein. Die Geschwindigkeiten im Bereich 100 Gbit/s und darüber werden mit aktuellen Sicherheitsmechanismen bei weitem noch nicht erreicht. Ansätze zur Beschleunigung von Firewalls und Intrusion Detection Systemen (IDS) gibt es viele in der Literatur, aber auch hier gilt, dass die Praxistauglichkeit noch nicht bewiesen ist. Hier plant bwNET100G+ umfangreiche Evaluierungs- und Forschungsarbeiten, um BelWue konkrete Empfehlungen für den Ausbau dieser Systeme an die Hand zu geben.

Auch Transportprotokolle wie TCP werden durch die hohen Geschwindigkeiten vor Probleme gestellt. Das TCP-Protokoll, auf dem der Großteil des Internetverkehrs basiert, ist nicht für derart hohe Geschwindigkeiten ausgelegt. Es existieren viele Varianten, die

[1] "We view SDN as a disruptive approach to networking that will change how virtually every company with a network operates." – Open Networking Foundation *All About ONF* `https://www.opennetworking.org/images/stories/downloads/about/onf-what-why.pdf`

unterschiedliche Probleme adressieren. Diese Varianten sind aber nicht zwangsläufig miteinander kompatibel, was die Fairness untereinander beeinträchtigt. Gerade in schnellen Netzen wird das Erreichen einer geringen Pufferauslastung und einer niedrigen Anzahl an Paketverlusten immer wichtiger. Die ist mit den heute eingesetzten TCP-Varianten nur schwer zu erreichen.

Diese Beispiele zeigen, dass die Leistung eines Netzwerks nicht allein von der Datenrate der Glasfaserkabel abhängt, sondern von vielen Faktoren bestimmt werden. In diesem Bericht sollen zunächst existierende Ergebnisse und Technologien zusammen gefasst werden. Zunächst werden hierzu Methoden für den flexiblen und intelligenten Netzbetrieb mittels SDN analysiert, anschließend die Problematik der aktuellen TCP-Varianten erörtert und schließlich Herausforderungen und Lösungsansätze für Sicherheitsmechanismen in Hochgeschwindigkeitsnetzwerken betrachtet.

2 AP1: Methoden für den flexiblen und intelligenten Netzbetrieb

2.1 Software defined Network (SDN) Introduction

SDN stellt einen innovativen Ansatz dar, Netzinfrastrukturen zu flexibilisieren und das Netzwerkmanagement zu vereinfachen. Dabei bricht SDN mit den bisherigen Ansätzen, die auf verteilte intelligente Geräte und Algorithmen setzen. Bei SDN wird das Netzwerk von einem zentralen Controller konfiguriert, der eine globale Sicht auf das Netz hat. Aus Redundanzgründen können auch mehrere Controller zum Einsatz kommen.

Verschiedene Funktionen wie Routing oder Weiterleitung werden als App auf dem Controller realisiert. Dies hat zur Folge, dass die Switches keine eigene Intelligenz mehr benötigen, sondern nur noch einfache Weiterleitungsregeln befolgen, die der Controller ihnen mitgeteilt hat. Damit lassen sich relativ schnell neue Features auf die Switches deployen.

Im folgenden geben wir eine verkürzte Einführung. Ein umfassenderer Überblick über SDN findet sich beispielsweise in unserem [1] sowie in einem weiteren Survey [2].

2.2 Northbound Interface (NBI)

Das Northbound Interface stellt die Programmier-Schnittstellen bereit, über die Dienste und Applikationen mit dem Controller kommunizieren. Ziel der verschiedenen NBIs ist es, von der Netzwerksicht des Controllers zu abstrahieren, so dass beispielsweise nicht mehr einzelne Switches angesprochen werden, sondern nur noch das gewünschte Verhalten des Netzes beschrieben wird. Beispielsweise wird in einer Traffic-Engineering-App nur der Pfad für bestimmte Flüsse festgelegt, ohne dass die App die komplette Topologie und die verwendeten Switches kennen muss. Diese Beschreibung kann in speziellen Hochsprachen erfolgen, welche dann so kompiliert werden, dass das gewünschte Verhalten auf die Controllersicht abgebildet wird. Im Beispiel würden die Pfade anschließend auf konkrete Switch-Konfigurationen übersetzt.

Allerdings existiert bisher noch keine standardisierte Beschreibung für das NBI. Statt dessen kommen verschiedene Programmiersprachen zum Einsatz. Verbreitete SDN-Programmiersprachen für die Erstellung von SDN Apps sind beispielsweise Frenetic [3], und der Nachfolger Pyretic [4] sowie Procera [5]. Diese Sprachen bieten eine deklarative Syntax und basieren auf funktioneller rekursiver Programmierung. Auf diese Weise erlauben es die Sprachen Netzwerkpolicies zusammenzusetzen. Eine weitere Alternative stellt NetIDE [6] dar.

2.3 Southbound Interface

Das Southbound Interface stellt die Schnittstelle zwischen SDN-Controllern und SDN-fähigen Switches dar. Dieses Interface kann offen dokumentiert oder geschlossen und proprietär sein. Das verbreitetste Protokoll ist das offene OpenFlow[7]. Dieses wird in Abschnitt 2.4 genauer betrachtet, da dieses Protokoll für weitere Arbeiten in AP1 verwendet wird.

Cisco hat 2014 OpFlex[8] als Konkurrenz zu OpenFlow vorgeschlagen und will dieses aktuell in der IETF als Standard [9] etablieren. Der Fokus von OpFlex liegt v.a. auf dem Verwalten von Policies. Ein weiteres Protokoll zur Verwaltung der Konfigurationen von Netzwerkkomponenten ist das Network Configuration Protocol (NetConf) [10] welches bereits in der IETF standardisiert wurde und v.a. von Juniper Networks unterstützt wird. NetConf setzt auf ein XML-Format zum Datenaustausch und verwendet YANG [11] als Modellierungssprache.

Darüber hinaus gibt es mit ForCES [12, 13] seit 2004 ein Framework das ebenfalls die Controll- und Dataplane trennt. ForCES verwendet dazu Logical Function Blocks (LFB), die modular kombiniert werden können. und es auf diese Weise ermöglichen, komplexe Forwarding-Mechanismen zu erstellen. Ein LFB realisiert dabei eine bestimmte Funktionalität wie z.B. IP-Routing. Die LFBs werden dann über ein eigenes Protokoll von Kontrollinstanzen konfiguriert. Dieser Ansatz wird von einigen als flexibler als OpenFlow betrachtet [14, 15].

Ebenso schlägt die SoftRouter-Architektur [16] die Trennung zwischen Controll- und Dataplane vor. Allerdings werden hierbei Controll- und Dataplane-Elemente dynamisch miteinander verknüpft.

2.4 OpenFlow

OpenFlow ist ein verbreitetes offenes Protokoll, das zur Kommunikation von SDN-fähigen Switches mit den SDN-Controllern verwendet wird.

Grundidee ist die Spezifikation von sogenannten “Flow Tables”, welche Pakete anhand von “Matching Rules” bestimmten Flows zuordnen und für diese Flows die weitere Verarbeitung festlegen. Dies kann beispielsweise eine Weiterleitung auf einen bestimmten Ausgangsport, das Verwerfen von Paketen oder die Weiterleitung zum Controller sein.

OpenFlow wurde ursprünglich von der Universität Stanford entwickelt und im Jahr 2009 veröffentlicht. Ab Version 1.1 hat die Open Networking Foundation (ONF) die Veröffentlichung und Standardisierung von OpenFlow übernommen. Aktuell ist Version 1.4. Mittlerweile wird OpenFlow von einer Vielzahl von Switch-Herstellern unterstützt, allerdings oft nicht in den aktuellsten Versionen oder mit allen Erweiterungen.

Die Versionen unterscheiden sich vor allem im unterstützen Feature-Set.

So war es bei OpenFlow 1.0 zunächst nur möglich auf bestimmte Felder der Ethernet-, IP- und Transport-Header wie Quell- und Ziel- MAC-/IP-Adresse oder TCP-/UDP-Ports zu matchen und bestimmte Aktion wie “Forwarding” zu veranlassen. Weiterhin kann auf EtherType, VLAN-Tags, DS und ECN-Flags sowie das Protocol-Feld gemacht werden. Zusätzlich können vom Controller Countern erzeugt werden, wodurch vom Switch Statistiken erstellt und abgerufen werden können. Über Queues kann eine Minimalrate für verschiedene Flüsse gesetzt werden und somit bereits einfache QoS realisiert werden.

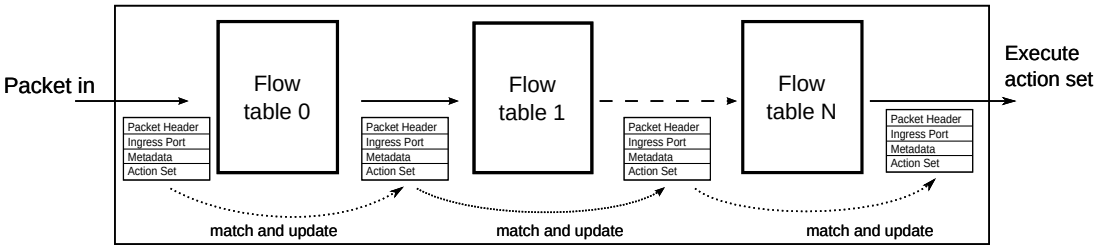


Abbildung 1: Pipelines mit Flow-Tables.

In Version 1.1 wurde das Abarbeiten der Pakete grundlegend überarbeitet und über verschiedene Pipelines von Flusstabellen realisiert. Der Ablauf bei der Verarbeitung wird in Grafik 1 dargestellt. Zusätzlich wurde eine Group-Table eingeführt, die es erlaubt, komplexere Forwarding-Regeln zu definieren. In einer optionalen Erweiterung ist es möglich, auch auf MPLS-Labels und Verkehrsklassen zu matchen. Außerdem wurde der Umfang der Actions erweitert, so dass es beispielsweise möglich ist das TTL-Feld zu dekrementieren.

Mit Version 1.2 erhielt OpenFlow eine Unterstützung für IPv6. Dies erlaubt beispielsweise auf IPv6-Quell- bzw. IPv6-Ziel-Adressen, das Flowlabel und ICMPv6 zu matchen. Zusätzlich wurde mit OpenFlow Extensible Match (OXM) eine Erweiterung eingeführt, die es Herstellern einfacher erlaubt, eigene Felder für Matches zu definieren. Darüber hinaus kann ein OpenFlow 1.2 kompatibler Switch nun auch gleichzeitig mit mehreren Controller verbunden sein. Dazu wird ein Controller als Master definiert, dem eine Reihe von Slaves zur Seite stehen.

In OpenFlow 1.3 wurden die Monitoringmöglichkeiten sowie die Unterstüzung bei Operation and Management (OAM) ausgebaut. Um dies zu bewerkstelligen wurden u.a. Meters in die Switch-Architektur eingeführt. Ein Meter wird direkt über einen Eintrag in der Flow-Tabelle angesprochen und erlaubt es eine maximale Daten- oder Paketrate festzulegen. Verkehr, der diese Rate überschreitet, kann entweder einfach gedroppt werden oder es können entsprechende Werte im DS-Feld gesetzt werden, um eine spätere QoS Behandlung auf anderen Geräten zu initiieren. Zusätzlich wurde die Unterstützung für Multi-Controller-Anwendungen erweitert, so dass nicht mehr nur klassische Failover-Szenarien sondern beispielsweise auch komplexes Loadbalancing umgesetzt werden kann. Außerdem wurde noch die IPv6-Unterstzüng um die Behandlung von Extension-Headers

	OF 1.0	OF 1.1	OF 1.2	OF 1.3 & OF 1.4
Ingress Port	X	X	X	X
Metadata		X	X	X
Ethernet: src, dst, type	X	X	X	X
IPv4: src, dst, proto, ToS	X	X	X	X
TCP/UDP: src port, dst port	X	X	X	X
MPLS: label, traffic class		X	X	X
OpenFlow Extensible Match (OXM)			X	X
IPv6: src, dst, flow label, ICMPv6			X	X
IPv6 Extension Headers				X

Tabelle 1: OpenFlow Match Felder.

	OF 1.0	OF 1.1	OF 1.2	OF 1.3 & OF 1.4
Per table statistics	X	X	X	X
Per flow statistics	X	X	X	X
Per port statistics	X	X	X	X
Per queue statistics	X	X	X	X
Group statistics		X	X	X
Action bucket statistics		X	X	X
Per-flow meter				X
Per-flow meter band				X

Tabelle 2: Statistikerhebung in OpenFlow.

wie Encrypted Secure Payload (ESP) erweitert. Auch wird Provider Backbone Bridge (PBB) unterstützt.

Version 1.4 erweitert die OXM-Fähigkeiten durch neue tabellenbasierte Matches anstelle der bisher hardgecodeten Einträge. Darüber hinaus wurde Unterstzüzung zur Konfiguration optischer Ports integriert. Zusätzlich können Controller-Nachrichten nun in Bundles anstatt wie bisher einzeln an die Switches verschickt werden. Auch wurden viele kleinere Detailverbesserungen und Bugfixes eingepflegt.

In Tabelle 1 sind die Matchmöglichkeiten der einzelnen OpenFlow-Versionen zusammengefasst und Tabelle 2 veranschaulicht die Fähigkeiten zur Statistikerhebung bei den einzelnen Versionen.

2.5 OpenFlow Controller Überblick

Der Openflow Controller stellt die logische zentrale Controllplane eines SDN-Netzes dar. Neben dem ursprünglich in Stanford entwickelten Controller NOX[17], der in C/C++ implementiert ist und OpenFlow 1.0 unterstützt, und seinem Python-Nachfolger POX

Name	Programming language	License	Comment
NOX [17]	C++	GPL	Initially developed in Stanford University.
POX [18]	Python	Apache	Forked from the NOX controller. POX is written in Python and runs under various platforms.
Beacon [22]	Java	BSD	Initially developed in Stanford.
Floodlight [20]	Java	Apache	Forked from the Beacon controller and sponsored by Big Switch Networks.
Ryu [21]	Python	Apache	Python based OpenFlow Controller supporting OpenFlow 1.4.
Maestro [23]	Java	LGPL	Multi-threaded OpenFlow controller developed at Rice University.
NodeFLow [24]	JavaScript	MIT	JavaScript OpenFlow controller based on Node.JS.
Trema [25]	C and Ruby	GPL	Plugins can be written in C and in Ruby. Trema is developed by NEC.
OpenDaylight [19]	Java	EPL	OpenDaylight is hosted by the Linux Foundation but has no restrictions on the operating system.

Tabelle 3: Liste von OpenFlow Controllern.

[18] wurden mittlerweile eine Reihe von weiteren SDN Controllern von verschiedenen Entwicklerteams veröffentlicht. Diese unterscheiden sich sowohl in der verwendeten Programmiersprache, der unterstützten OpenFlow-Version und der Verbreitung in der Community. Tabelle 3 listet die wichtigsten Controller auf. Von diesen hat sich OpenDayLight [19] als derjenige etabliert, der am meisten Unterstützung aus der Industrie erfährt. Deshalb bietet OpenDayLight mittlerweile das größte Featureset und enthält nicht nur OpenFlow-Unterstüzung zur Konfiguration der Switches, sondern auch eine Reihe weiterer Protokolle, die die verschiedenen Hersteller beigesteuert haben. Auf der anderen Seiten gibt es mit dem in Java geschrieben Floodlight[20] und dem in Python geschrieben Ryu[21] Beispiele für Alternativen, die eine niedrigere Einstiegshürde bieten und für OpenFlow basiertes SDNs in der Regel mehr als ausreichend sind.

Die Auswahl eines geeigneten Controllers sollte daher immer Projektspezifisch erfolgen und neben den möglichen Features auch die Kompatibilität mit der eigenen Hardware und die Komplexitätshürde mit berücksichtigen.

2.6 Praxis Tests

Um abschätzen zu können, welche SDN Controller für die Nutzung in bwNET-100G+ in Frage kommen, wurde eine Auswahl in verschiedenen Umgebungen getestet. Die Testumgebungen beinhalten eine Netzwerkemulation auf Basis von Mininet [26], vernetzte

virtuelle Maschinen (VM) auf Basis der in [27] vorgestellten Plattform sowie Tests mit realen Switches des Herstellers HP.

Da sich OpenDayLight mittlerweile als Standart-Controller der Industrie zu etablieren scheint, wurde zunächst geprüft, ob dieser Controller Verwendungen finden kann. In den Tests stellte sich allerdings relativ schnell heraus, dass OpenDayLight eine relativ hohe Einstiegshürde besitzt, was u.a. der umfangreichen Plugin-API und dem verwendeten Buildsystem, sowie dem sehr großen Featureumfang geschuldet ist. Darüber hinaus hat OpenDayLight einen relativ hohen Ressourcenbedarf und startet relativ träge. Ein weiteres Problem stellt außerdem die sehr rege Upstream-Entwicklung dar, so dass es durchaus vorkommen kann, dass die aktuelle Version an manchen Tagen noch nicht einmal kompiliert werden kann.

Das andere Ende der Komplexitätsskala stellt POX dar, welcher allerdings offiziell nur OpenFlow 1.0 unterstützt und einzelne Features von höheren OpenFlow Versionen ergänzt. Es ist mit POX relativ einfach möglich, erste funktionsfähige Controller-Apps zu programmieren. Deshalb eignet sich POX insbesondere auch zum schnellen Prototypen einzelner Features, um die Umsetzbarkeit dieser zu Testen bevor ein kompletter Use Case angegangen wird. Auch der Ressourcenbedarf und die Startdauer halten sich im Rahmen. Zusätzlich gibt es sehr viele und gute Tutorials die den Einstieg weiter erleichtern.

Ryu bietet ebenfalls eine relativ niedrige Einstiegshürde, welche aber etwas höher ist als bei POX. Dies ist aber vor allem auf den größeren Funktionsumfang und die Unterstützung der neueren OpenFlow-Versionen zurückzuführen. Es gibt hier ebenfalls relativ viele Tutorials und Beispiele im Internet zu finden. In POX prototypisch implementierte Features sollten sich relativ leicht auf Ryu portieren lassen, was zum einen an der gleichen Programmiersprache aber auch an ähnlichen APIs liegt.

Unsere Tests zeigen also, dass die Wahl eher auf POX oder Ryu anstatt auf OpenDayLight fallen sollte, falls nicht der volle Funktionsumfang von OpenDaylight benötigt wird. Für Anwendungsfälle die mit OpenFlow 1.0 auskommen bietet sich POX als erste Wahl an. Falls aktuellere OpenFlow-Versionen benötigt werden bietet sich Ryu an.

2.7 Use Cases

In Zusammenarbeit mit dem Rechenzentrum der Universität Tübingen (ZDV) wurden folgende Use Cases erarbeitet, die im Laufe des Projekts realisiert werden sollen. Diese Use Cases entstammen alle praxisnahen Anforderungen und haben als Zielausrichtung die Vereinfachung und Flexibilisierung des Netzmanagements. Diese Use Case sollen im weiteren Projektfortschritt von bwNET100G+ weiter evaluiert und ergänzt werden. Des Weiteren wird angestrebt, einzelne Use Cases nach einer Testphase in den Produktivbetrieb zu übernehmen.

Die Vorstellung der bisher erarbeiteten Use Cases ist dabei stets wie folgt gegliedert:

- Aktueller Stand und Problembeschreibung,

- eine Lösungsidee wie dieses Problem angegangen werden soll und
- mögliche Herausforderungen.

Die weitere Ausarbeitung der Use Cases erfolgt in der weiteren Projektphase.

2.7.1 Firewallentlastung mit SDN

Aktueller Stand Die vorhandene Firewall hat mit $1000 Mbit/s$ nur eine begrenzte Bandbreite und stellt somit ein Bottleneck dar. Für normalen Verkehr, wie z.B. Emails oder Webseiten abrufen, reicht die vorhandene Bandbreite aus. Allerdings reicht die Bandbreite nicht aus, falls große Datenmengen, wie beispielsweise Backupdaten, übertragen werden sollen. Die Anschaffung einer neuer Firewall stellt keine Option dar, da diese sowohl im Erwerb als auch im Service relativ teuer ist.

Lösungsidee "Sicherer" Verkehr muss nicht durch die Firewall geleitet werden, sondern kann mit Hilfe von Bypassing um diese herumgeführt werden. Dazu wird ein Interface zur Definition von "sicherem" Verkehr benötigt. Dieses lässt sich als SDN-Controller-App realisieren. Die an dem Bypass beteiligten Switches werden dann vom Controller durch OpenFlow konfiguriert.

Szenarien Es werden 3 Arten von Flüssen betrachtet:

- Interne Flüsse: Interne Flüsse stellen beispielsweise PC zu PC Verkehr innerhalb einer Abteilung dar und müssen nicht durch die Firewall.
- Externe Flüsse: Flüsse vom oder ins Internet werden als externe Flüsse bezeichnet und stellen unsicheren Verkehr dar, der durch die Firewall muss.
- Sichere Flüsse: Backup-Daten eines PCs auf den Speicher der Rechenzentrums oder Daten vom/zum Rechencluster stellen beispielsweise sicheren Verkehr dar, der um die Firewall herum geleitet werden kann.

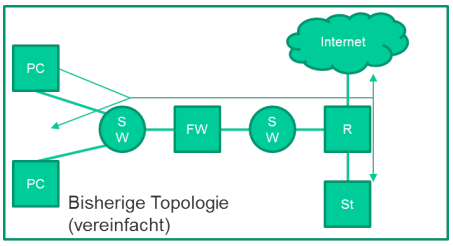


Abbildung 2: Alte Topologie.

Abbildung 2 zeigt die bisherige Netzwerktopologie ohne SDN. Die Firewall stellt hierbei das Default-Gateway dar und ist direkt integriert. In Abbildung 3 wird eine mögliche neue

Architektur gezeigt, die Firewall Bypassing ermöglicht. Dazu wurde ein neuer SDN-fähiger Switch integriert, der unsicheren Verkehr durch die Firewall leitet und sicheren Verkehr an dieser vorbei.

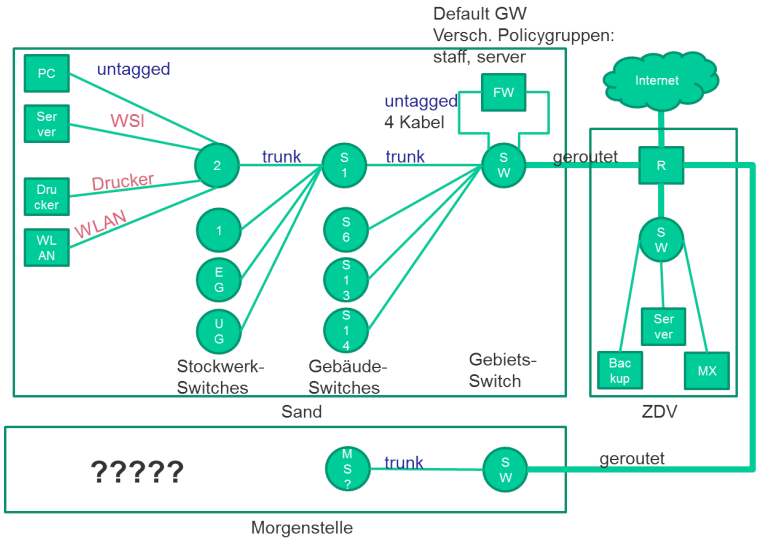


Abbildung 3: Neue Topologie (mit SDN).

2.7.2 Vereinfachtes Netzwerkmanagement mit SDN

Aktueller Stand Jeder Switch muss von Hand vor Ort konfiguriert werden. Das Einspielen der Konfiguration erfolgt dabei per serieller Konsole, SCP oder herstellerspezifischen Tools. Zusätzlich ist das Config-Format (oft) Switch- oder Herstellerspezifisch.

Idee Plug'n'Play für neue Switches. Dies bedeutet dass die Switches nach einem Neustart oder einer Neuinstallation sich automatisch ihre Konfiguration holen und somit ohne händische Interaktion einsatzbereit sind. Die Konfiguration für das Netzwerk wird dabei mit Hilfe eines Zentralen geräte- und herstellerunabhängigen Managementtool erstellt und abgelegt.

Umsetzung Die initiale Konfiguration der Switches erfolgt per BootP und OpenFlow. Dabei wird dem Switch während des Bootvorgangs eine Minimalkonfiguration, welche u.a. den SDN-Controller enthält, mitgeteilt. Die weitere Konfiguration, wie Weiterleitungsregeln, erfolgt dann per OpenFlow entweder proaktiv oder on-demand.

2.7.3 SDN-basierte VLAN-Zuweisung

Aktueller Stand Heute gibt es 2 verbreitete Arten der VLAN-Zuweisung:

- Statisches portbasiertes VLAN
- Dynamische geräte-/nutzerabhängige VLAN-Konfiguration Per 802.1x und Radius

Idee Ziel ist es die VLAN-Zuweisung dynamisch zu realisieren ohne dafür einen Radius-Server und die damit einhergehenden Komponenten zu benötigen. Der Vorteil dabei ist, dass so eine leichtgewichtigere Lösung umgesetzt werden und die VLAN-Konfiguration zentral in einem Managementtool erfolgen kann. Die VLAN-Konfiguration würde dabei per SDN-Controller auf den Switches konfiguriert.

Umsetzung Der SDN-Controller wird um eine App erweitert, die als 802.1X Authenticator agiert und die Benutzer dynamisch in die richtigen VLANs einteilt. Die VLANs werden dynamisch per OpenFlow auf den Switches konfiguriert.

Herausforderung Die sichere Authentifizierung der Nutzer und Endgeräte stellt dabei eine spezielle Herausforderung dar.

2.7.4 Virtuelles Datacenter

Aufbau Der Aufbau eines virtuellen Datacenter ist wie folgt: Ein virtuelles Datacenter besteht üblicherweise aus mehreren Standorten, von denen jeder Standort mehrere Server-Racks beinhaltet. Pro Rack gibt es mehrere Server auf denen wiederum mehrere virtuelle Maschinen laufen. Es gibt mehrere VMs pro physikalischem Host und mehrere Hosts sind in einem Rack zusammengefasst. Diese VMs können von verschieden Kunden gemietet und pro Kunde miteinander vernetzt werden. Die Netze der einzelnen Kunden sollen dabei voneinander isoliert werden und formen jeweils ein eigenes Layer-2-Netz. Bei solchen Netzen müssen die VMs der einzelnen Kunden nicht zwangsweise auf dem gleichen Server liegen, sondern können sich sogar an verschiedenen Standorten befinden.

Migration von VMs Es kann vorkommen, dass die VMs verschoben werden müssen. Dabei soll die logische Layer-2-Struktur für den Kunden unverändert bleiben und die Downtime möglichst minimal sein. Ein SDN-fähiges Netz kann dabei automatisch die virtuellen Netze umstrukturieren.

Herausforderung Eine geringe Downtime während der Migration insbesondere zwischen 2 Standorten stellt dabei eine Herausforderung dar. Die Umstrukturierung des jeweiligen Layer 2-Netzes kann ebenfalls herausfordernd werden, wenn die Kundennetze entsprechend komplex werden.

2.7.5 Anwendungsspezifisches QoS

Verschieden Anwendungen haben spezifische Anforderungen an das Netz. So hat beispielsweise VoIP besondere Anforderungen an Latenzen. Allgemein kann man verschiedene Verkehrsklassen und Ihre Anforderungen unterscheiden. Mögliche Verkehrsklassen sind:

- Echtzeitverkehr
- Backupverkehr
- Daten zum/vom Rechencluster
- Netzwerkdateisystem

Diese Klassen haben Anforderungen wie

- Verzögerung
- Bandbreite

Diese Anforderungen müssen im Netzwerk entsprechend unterstützt werden.

Idee Die Zuordnung von Anwendungen in die verschiedenen Verkehrsklassen und die Zuordnung von Anforderungen zu Klassen lässt sich in einer SDN-Controller-App realisieren. Dabei kann beispielsweise der Admin oder ein entsprechend privilegierter Benutzer temporäre oder permanente Regeln festlegen, welche dann in Netzwerkpolicies übersetzt werden.

Umsetzung Die erstellten Policies legen beispielsweise die Priorität oder Bandbreite von Flüssen fest und werden per OpenFlow entsprechend auf den Switches verteilt. Dabei kann es bisweilen sinnvoll sein, andere Flüsse umzulegen um bestimmte Kriterien erfüllen zu können.

2.7.6 Weitere Optimierung mit SDN

Aktueller Stand In großen Layer2-Netzen kann das Problem auftreten, dass sehr viel ARP- und Broadcast-Verkehr das Netz nahezu lahmlegt. Um dieses Problem in den Griff zu bekommen, gibt es spezielle Netzwerkgeräte, die den Verkehr entsprechend filtern.

Idee Die Reduzierung von unnötigem ARP-Verkehr und weiteren Broadcasts kann auch ohne separates Gerät kostengünstig per SDN realisiert werden. Darüber hinaus ist es möglich, nicht mehr benötigte Forwarding-Einträge frühzeitig zu löschen oder diese umzubiegen falls der Client den Standort gewechselt hat.

Umsetzung ARP-Verkehr lässt sich beispielsweise mithilfe eines Lernenden ARP-Proxy reduzieren und entsprechende Forwardingregeln per OpenFlow auf den Switches installieren. (siehe Abbildung 4) Ebenso lässt sich das Umbiegen von Forwarding-Einträgen aufgrund von mobilen Clients per OpenFlow realisieren und auf den entsprechenden Pfaden im Netz installieren (siehe Abbildung 5)

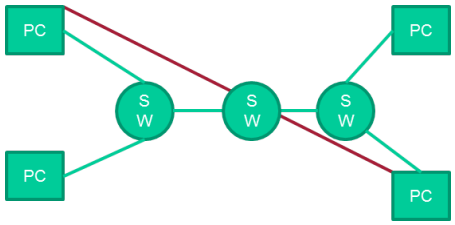


Abbildung 4: Weiterleitungsregeln auf dem kompletten Pfad.

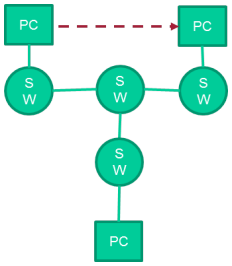


Abbildung 5: Entfernen auf nicht benötigtem Teilbaum.

2.8 Zusammenfassung

Wie man sieht, sind SDN und OpenFlow vielversprechende Technologien im Kontext der in bwNET100G+ betrachteten Szenarien. Wie oben geschildert sehen wir eine Vielzahl von konkreten Szenarien, mit welchen SDN- / OpenFlow-fähige Netzwerkgeräte den Netzwerkbetrieb flexibler und effizienter gestalten können. Diese werden im Verlauf des Projektes auf ihre Umsetzbarkeit getestet. Auch werden konkrete Aussagen zur Nutzbarkeit bestehender OpenFlow Implementierungen erarbeitet.

3 AP2: Datentransport in Hochleistungsnetzen

Das Internet basiert auf dem unzuverlässigen, verbindungslosen *Internet-Protokoll* (kurz *IP*). Es wird zur Zeit in Version 4 (*IPv4*, RFC 791 [28]) und Version 6 (*IPv6*, RFC 2460 [29]) genutzt und bietet einen *„Best Effort“*-Dienst. Das heißt, dass durch IP keinerlei Garantien gegeben werden, ob, wann und auf welchem Weg Datenpakete ihr Ziel erreichen; jedoch versucht das Netz so gut wie möglich diese zeitnah zuzustellen.

Aufbauend auf IP werden *Transportprotokolle* genutzt, um weitere Dienste zur Verfügung zu stellen. Das meist genutzte Transportprotokoll ist das *Transmission Control Protocol (TCP)*, vgl. RFC 793 [30]. Es bietet einen zuverlässigen, verbindungsorientierten Dienst oberhalb von IP.

TCP garantiert somit, dass alle zu versendenden Daten *vollständig*, *unverändert* und in der richtigen *Reihenfolge* beim Empfänger ankommen. Hierzu werden unter anderem *Sequenznummern*, *Prüfsummen* und *Sendewiederholungen* eingesetzt. Diese Garantien gelten allerdings nur, sofern keines der beteiligten Endsysteme ausfällt und das IP-Netz nicht dauerhaft ausgefallen ist. Zeitliche Garantien darüber, *wann* die Daten ihr Ziel erreichen, gibt TCP keine.

Zum Schutz des Empfängers vor Überlast enthält TCP darüber hinaus einen Mechanismus zur *Flusskontrolle*, der dafür sorgt, dass der Sender niemals mehr Daten schickt, als der Empfänger verarbeiten kann.

Außerdem enthält TCP einen Mechanismus zum Schutz des Netzes vor Überlast, die *Staukontrolle*. IP-Netze bieten eine hohe Flexibilität und gute Ressourcenausnutzung, insbesondere für Anwendungen die keinen *konstanten* Bedarf an Übertragungskapazitäten haben. Es findet keine Reservierung von Übertragungskapazitäten statt und es gibt keine Zugangskontrolle, die die Menge der Daten begrenzt, die ein Sender in das Netz einspeisen kann. Dies macht das IP-Netz allerdings anfällig für *Staus*.

Der Bedarf einer Staukontrolle wurde zunächst nicht erkannt; TCP, wie es 1981 in RFC 793 [30] standardisiert wurde, sah keine Mechanismen zum Schutz des Netzes vor Überlast vor. Im *ARPANET*, einem Vorläufer des Internets, in dem TCP/IP zu dieser Zeit hauptsächlich genutzt wurde, führte dies zunächst auch nicht zu Problemen, da das Netz sehr homogen aufgebaut war und für die damaligen Anwendungen ausreichend Übertragungskapazitäten zur Verfügung standen.

Staukollapse

Andere Voraussetzungen bestanden im privaten TCP/IP-Netz der *Ford Aerospace and Communications Corporation*, in dem es 1984 erstmals zu größeren Problemen aufgrund von Staus gekommen ist. John Nagle beschriebt diese in RFC 896 [31] unter dem Namen *„Staukollaps“* (engl. *„congestion collapse“*). Insbesondere kamen in diesem Netz unterschiedliche Übertragungstechnologien zum Einsatz, die ein großes Spektrum an

unterschiedlichen Übertragungsraten boten (dies trifft auf das heutige Internet ebenfalls zu, im ARPANET war das allerdings zunächst nicht der Fall). Darüber hinaus war das Ford Aerospace Netzwerk nach eigenen Angaben deutlich stärker ausgelastet, als das ARPANET.

RFC 896 [31] beschreibt einen einfachen Ansatz zur Vermeidung von Staukollapsen: Bei erhöhter Pufferauslastung (z. B. über 50 %) werden alle Sender mittels einer expliziten *ICMP*-Nachricht vom Gateway aufgefordert, ihre Senderate zu reduzieren. Diese reagieren darauf, indem sie ihr Flusskontrollfenster kurzzeitig auf ein Minimum reduzieren, was die Senderate auf ein niedriges Niveau absenkt, aber die Kommunikation nicht vollständig verhindert. Insbesondere wird darauf hingewiesen, dass das Transportprotokoll (hier TCP) unbedingt in die Behandlung von Stausituationen mit einbezogen werden muss, da IP nicht zwischen Quittungen und Nutzdaten unterscheiden kann. Werden aber Quittungen unterdrückt, ohne dass das Transportprotokoll von der Stausituation unterrichtet wird, werden vermehrt Sendewiederholungen ausgelöst, die das Stauproblem weiter verschlimmern. Der in RFC 896 [31] beschriebene Lösungsansatz hat nach eigenen Angaben im Netz von Ford Aerospace das Auftreten von Staukollapsen effektiv verhindert, basierte aber größtenteils auf Erfahrungswerten mit diesem speziellen Netz und wurde nicht ins ARPANET bzw. Internet übernommen.

3.1 Staukontrollmechanismen

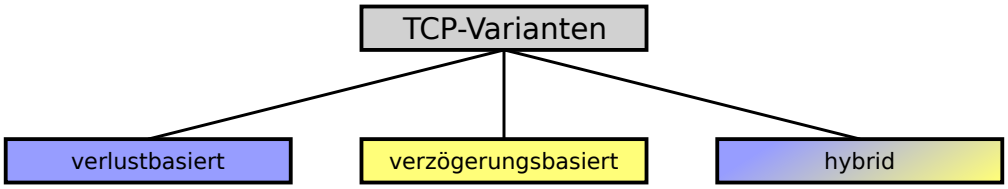
Zwei Jahre später, im Jahr 1986, kam es dann aber auch im Internet zu Staukollapsen, was zu einer wissenschaftlichen Aufarbeitung des Themas führte. Van Jacobson rekapituliert die Problematik in [32], und stellt dort den ersten Staukontrollmechanismus vor, der im Rahmen des Internets größere Beachtung gefunden hat. Dieser Mechanismus ist in die TCP-Implementierung von *4.3BSD-Tahoe (1988)* eingeflossen und wird daher auch als *„TCP Tahoe“* bezeichnet. Zu dieser Zeit gab es bereits andere Arbeiten, die sich mit Staukontrolle in paketvermittelten Netzen beschäftigt haben. Zu erwähnen sind hier insbesondere der *CUTE*-Algorithmus von Raj Jain [33] und eine Übersicht früher Staukontrollkonzepte von Mario Gerla und Leonard Kleinrock in [34].

Seit der Einführung von TCP Tahoe wird Staukontrolle im Internet von den Endsystemen verteilt durchgeführt. Hierbei treffen die Endsysteme jeweils lokal Entscheidungen, wann bzw. mit welcher Rate sie Pakete ins Netz schicken. Hierzu wurde, neben dem in RFC 793 [30] definierten Flusskontrollfenster, ein *Staukontrollfenster* eingeführt. Die Senderate wird somit begrenzt durch das Minimum aus Flusskontrollfenster, Staukontrollfenster und der maximalen Übertragungsrate des Senders. Anders als der Empfänger teilt das Netz dem Sender allerdings nicht mit welche Ressourcen dem Sender zur Verfügung stehen. Die Aufgabe der Staukontrolle ist es somit stets eine adäquate Fenstergröße zu ermitteln. Die jeweils lokal bei den Sendern getroffenen Entscheidungen müssen so konsistent sein, dass durch das Zusammenwirken der betroffenen Endsysteme eine effektive Staukontrolle für das Netz erreicht wird. Die Staukontrollalgorithmen für das Internet sind außerdem so entworfen, dass sich die zur Verfügung stehenden Übertragungskapazitäten zudem

relativ fair auf die einzelnen TCP-Verbindungen verteilen, ohne dass sich die Endsysteme explizit koordinieren. Dieses Konzept ist auch in den später entwickelten TCP-Varianten erhalten geblieben. Eine faire Aufteilung ergibt sich allerdings nur, wenn die genutzten Algorithmen zueinander „kompatibel“ sind (vgl. Diskussionen in RFC 2309 [35] und RFC 2914 [36]). Daher können sie nur sehr eingeschränkt von verschieden Akteuren unabhängig voneinander entwickelt und ausgebracht werden – die Stabilität des Internets und die Fairness unter den teilnehmenden Endsystemen wäre gefährdet (RFC 5033 [37]). Dies hat dazu geführt, dass der heutige *„State-of-the-Art“* der Staukontrolle noch stark mit den frühen Entwicklungen verbunden ist und großer Wert auf Rückwärtskompatibilität und Verträglichkeit mit älteren Varianten gelegt werden muss.

Die Funktionsweise der Staukontrolle im Internet entspricht somit dem Ende-zu-Ende-Prinzip [38], d. h. die Entscheidungen über eine Reaktion werden jeweils in den (sendenden) Endsystemen getroffen. Diese Ansätze haben sich traditionell auf indirekte Anzeichen für das Vorliegen einer Stausituation verlassen, beispielsweise das Auftreten von Paketverlusten. Diese impliziten Annahmen können jedoch falsch sein, beispielsweise treten bei drahtlosen Übertragungen auch häufiger Paketverluste auf, die jedoch nicht staubedingt sind, sondern durch ungünstige Übertragungssituationen hervorgerufen wurden. Eine *explizite* Stauanzeige – wie in RFC 3168 [39] als Explicit Congestion Notification (ECN) definiert – ist daher eine Optimierungsmöglichkeit. Dazu müssen die Zwischensysteme (in der Regel die Router) allerdings eine Markierung im IP-Paket setzen, so dass die vom Stau betroffenen Endsysteme diese explizite Stauanzeige nutzen können. Vorteil hierbei ist, dass die Markierung noch so frühzeitig erfolgen kann, bevor die Pakete verworfen werden müssen und somit seitens des Endsystems früher reagiert werden kann, wodurch Paketverluste und Übertragungswiederholungen vermieden werden können [40]. Allerdings müssen die Zwischensysteme hierfür aktive Warteschlangenmechanismen einsetzen, um drohende Stausituationen frühzeitig zu erkennen und zurück zu melden. Betreiber haben bislang (entgegen den Empfehlungen in RFC 2309 [35]) jedoch überwiegend auf den Einsatz solcher Mechanismen verzichtet, um eine Erhöhung der Komplexität durch deren Einsatz und Konfiguration zu vermeiden. Weitere Vorteile sind, dass die Warteschlangen in den Zwischensystemen im Mittel kürzer bleiben, die Ende-zu-Ende-Latenz damit sinkt und somit auch die Regelschleifen, die auf Rückmeldungen von Endsystemen basieren, schneller ablaufen können, d. h. Endsysteme können auch schneller auf Stausituationen reagieren. Zudem wird eine dauerhafte Operation in der Nähe des Überlastbereichs vermieden, in dem die Zwischensysteme aufgrund vollständig gefüllter Warteschlangen neu ankommende nur noch Pakete verwerfen können.

Da ECN im Internet nicht garantiert zur Verfügung steht, können alle dort eingesetzten TCP-Varianten mit *impliziten* Stausignalen arbeiten. Die Varianten können generell in drei Gruppen unterteilt werden:



- Verlustbasierte Verfahren
  - Nutzen Paketverluste als implizites Stausignal. Die zur Verfügung stehende Kapazität an einer Engstelle kann nur durch gezieltes Überfüllen der Warteschlange erkannt werden.
- Verzögerungsbasierte Verfahren
  - Nutzen eine erhöhte Umlaufzeit oder Verzögerung in Senderichtung als implizites Stausignal. Hierbei kann eine drohende Stausituation bereits frühzeitig erkannt werden, sodass es in der Regel nicht zu Paketverlusten kommt. Die Warteschlangen werden nur leicht gefüllt, was kurzen Umlaufzeiten führt.
- Hybride Verfahren
  - Sollen schnelle Netze besser ausnutzen können, ohne die im Internet verbreiteten Staukontrollverfahren übermäßig zu beeinträchtigen. Sie erhöhen ihre Senderate sehr schnell und nutzen Mechanismen der verzögerungsbasierten Verfahren um zu erkennen, wann sich die Warteschlange einer Engstelle zu füllen beginnt. Danach verhalten sie sich ähnlich wie die verlustbasierten Verfahren.

Im Folgenden werden zunächst ältere Ansätze zur Staukontrolle diskutiert, gefolgt von den heute meist eingesetzten Verfahren, die speziell für schnelle Netze entwickelt wurden. Hierbei werden auch gezielt TCP-Varianten diskutiert, die so nicht im Internet eingesetzt werden, aber die weitere Entwicklung spürbar beeinflusst haben. Neuere Verfahren und alternative Entwicklungen werden zum Schluss des Kapitels vorgestellt.

TCP Tahoe

Der Entwicklung von *TCP Tahoe* lag die Annahme zugrunde, dass Staus in IP-Netzen selten, aber sehr gefährlich für die Funktionsfähigkeit des Netzes sind. Daher werden bei Detektion eines Staus vergleichsweise extreme Maßnahmen getroffen, um das Netz zu schützen. Neben dem Flusskontrollfenster führt TCP Tahoe ein *Staukontrollfenster* ein, das festlegt, wie viele Daten von einem Sender maximal in das Netz eingespeist werden dürfen. Die Größe dieses Fensters wird über zwei verschiedene Mechanismen beeinflusst: *Slow Start* und *Congestion Avoidance*. Slow Start dient dazu, schnell die Kapazität des Netzes zu ermitteln, indem das Staukontrollfenster bei einem kleinen Wert initialisiert und dann exponentiell vergrößert wird, solange kein Stau detektiert wird.

Paketverluste dienen dem Algorithmus als *implizites Stausignal*. Beim Auftreten des ersten Paketverlustes wird ein Schwellenwert festgelegt, der im weiteren Verlauf den Übergang von Slow Start in Congestion Avoidance regelt, genannt *Slow-Start-Threshold*. Er wird auf die Hälfte der zu diesem Zeitpunkt im Transit befindlichen Datenmenge gesetzt.

TCP Tahoe reinitialisiert das Staukontrollfenster nach jedem Paketverlust auf einen kleinen Wert und beginnt dann wieder mit Slow Start. Wie eben erwähnt, bestimmt der Slow Start Threshold, bei welcher Fenstergröße der Slow Start endet und in den *Congestion-Avoidance*-Modus gewechselt wird. In diesem Modus wird das Staukontrollfenster nur noch *linear* erhöht, um ein Paket pro Umlaufzeit.

Unter der Annahme, dass Stausituationen selten auftreten, wird die Verbindung schnell durch das Flusskontrollfenster (oder durch die maximale Übertragungskapazität des Senders) begrenzt. Der erwartete Verlauf der Übertragungsrate pro Zeit entspricht dann etwa der in Abb. 6 dargestellten idealisierten Kurve.

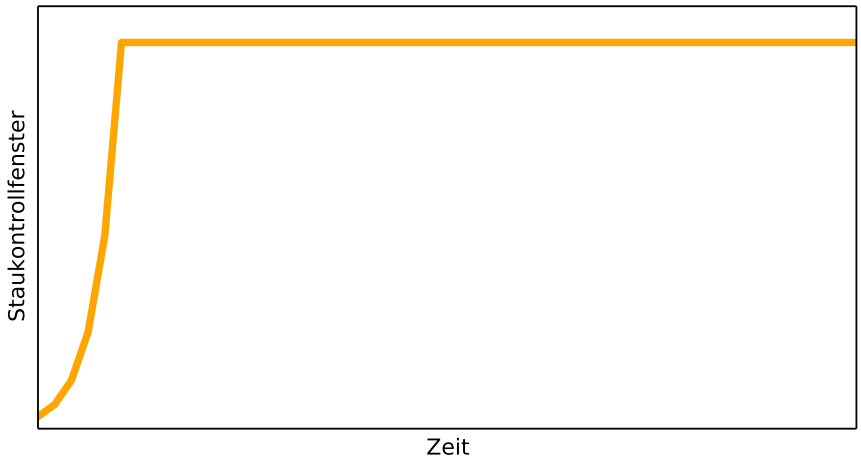


Abbildung 6: Slow Start mit anschließender Begrenzung durch Flusskontrollfenster

Die Staukontrolle von TCP Tahoe hat sich als effektives Mittel erwiesen, um Staukollapse zu verhindern. Einschränkungen beim Netzwerkdurchsatz konnten zunächst vernachlässigt werden, da zu dieser Zeit das Flusskontrollfenster meist relativ klein war und die Umlaufzeiten relativ kurz.

In Fällen, in denen die Senderate aber tatsächlich größtenteils durch die Staukontrolle begrenzt ist, kann der Einsatz von TCP Tahoe allerdings zu einem deutlich verminderten Durchsatz führen. In Abb. 7 wird wiederum ein idealisierter Kurvenverlauf des Staukontrollfensters über die Zeit in einem fiktiven Szenario dargestellt. Hierbei ist die Senderate nur durch das Staukontrollfenster begrenzt. Ab einer bestimmten, festen Größe

des Fensters wird die Pufferkapazität einer Engstelle[2] im Netz überschritten und es kommt zu Paketverlusten.

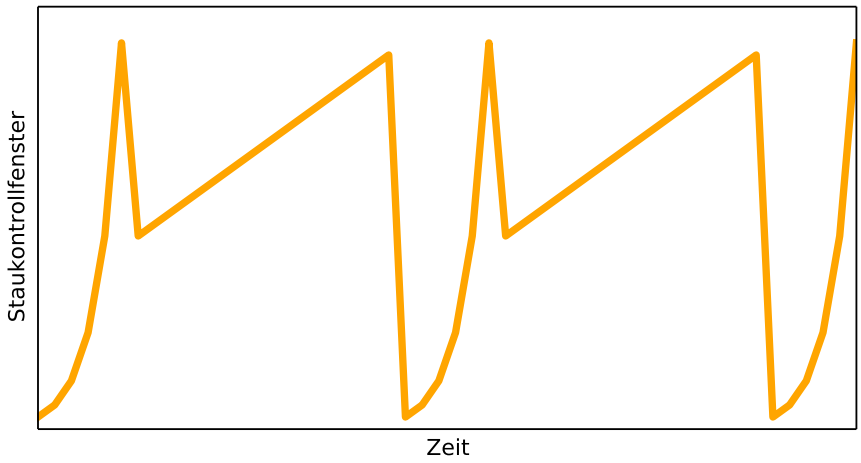


Abbildung 7: Slow Start und Congestion Avoidance

Der Kurvenverlauf unterscheidet sich deutlich von Abb. 6 und zeigt die verminderte Leistung von TCP Tahoe in einer solchen Situation. Ähnliche Situationen sind in heutigen Netzen keine Seltenheit. Endsysteme, also beispielsweise PCs und Server, sind häufig über ein schnelles LAN angebunden, wohingegen die Internetanbindung des Standorts einen geringeren Durchsatz bietet. Dadurch kann es schnell zu einer Situation kommen, in der die Verbindung nur durch die Staukontrolle begrenzt ist und es tritt eine ähnliche Situation auf, wie im eben beschrieben (vgl. Abb. 7).

TCP Reno

Um den Leistungsproblemen von TCP Tahoe zu begegnen, wurde das Staukontrollverfahren weiterentwickelt. Im nachfolgenden BSD Release, *4.3BSD-Reno (1990)*, findet sich eine Weiterentwicklung von TCP Tahoe, die üblicherweise als *TCP Reno* bezeichnet wird, und im Jahr 1999 im RFC 2581 [41] „TCP Congestion Control“ standardisiert wurde. TCP Reno wird daher auch häufig auch als „Standard TCP“ bezeichnet.

Neu ist dabei, im Gegensatz zu TCP Tahoe, die Unterscheidung zwischen *schweren* Staus und *leichten* Staus. Ein schwerer Stau wird angenommen, wenn ein Paket so lange nicht bestätigt wurde, dass der *Sendewiederholungstimer* abgelaufen ist. Dies entspricht der Staudetektion von TCP Tahoe und die Reaktion darauf fällt ebenfalls identisch aus: Reduktion des Staukontrollfensters auf einen kleinen Wert, Anpassung

[2] Eine *Engstelle* bezeichnet einen Kapazitätsengpass in einer Netzkomponente, bei dem Datenpakete längerfristig mit einer höheren Datenrate ankommen als sie weitergeleitet werden können.

des Slow Start Threshold, gefolgt von Slow Start. Ein leichter Stau hingegen wird angenommen, wenn mindestens drei duplizierte Quittungen empfangen wurden. Da TCP kumulative Quittungen einsetzt, bedeutet dies, dass beim Empfänger zwar weitere Daten empfangen wurden, es jedoch eine Lücke im Datenstrom gibt; d. h. mindestens ein Paket ist verloren gegangen, oder es ist zu einer Reihenfolgenvertauschung der Pakete gekommen. Reihenfolgevertauschungen sind in heutigen IP-Netzen möglich, aber selten. Daher werden drei duplizierte Quittungen abgewartet, bevor von einem Paketverlust und somit von einem leichten Stau ausgegangen wird.

Nun folgen zwei Reaktionen: Zum einen setzt der *Fast Recovery* Modus das Staukontrollfenster auf die Hälfte der im Transit befindlichen Daten. Dies bedeutet dass der Slow Start bis zum Slow-Start-Threshold übersprungen wird. Zum anderen wird das verlorene Paket noch vor Ablauf des Sendewiederholungstimers durch den *Fast Retransmit* Mechanismus erneut verschickt. Wurden alle verlorenen Daten empfangen, wird die Verbindung im Congestion Avoidance Modus fortgesetzt. Ist die Verbindung, wie oben, nur durch die Staukontrolle begrenzt, und treten immer ab einer bestimmten Fenstergröße Paketverluste auf, entsteht die für TCP Reno charakteristische Sägezahnkurve (vgl. Abb. 8).

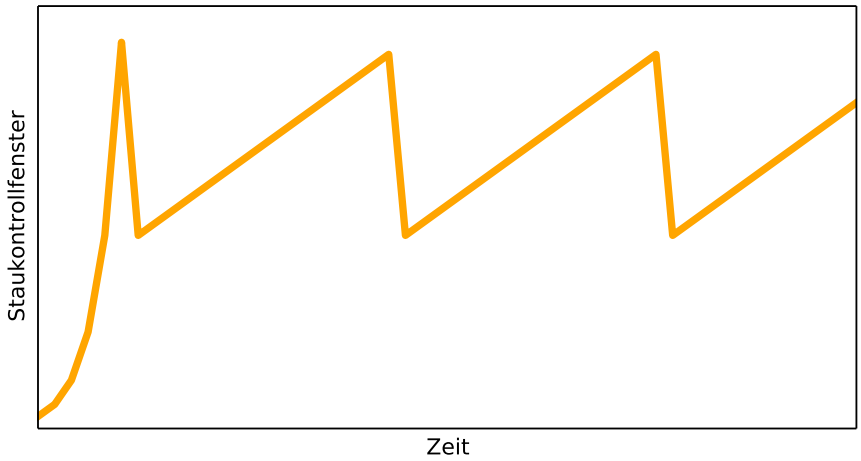


Abbildung 8: Typische „Sägezahnkurve“ von TCP Reno

Im Vergleich zu TCP Tahoe, wird der Netzwerkdurchsatz durch *Fast Retransmit* und *Fast Recovery* in einem solchen Szenario deutlich erhöht und die Leistung von TCP Reno wurde somit lange Zeit als ausreichend angesehen. Dennoch hat auch dieses Staukontrollverfahren verschiedene Nachteile.

Aufgrund der Funktionsweise von TCP Reno werden die Puffer an einer Engstelle üblicherweise bereits von einer einzigen Verbindung regelmäßig bis zum Überlauf gefüllt. Dies führt zu erhöhten und stark schwankenden Umlaufzeiten. Unter Umständen ist es sogar möglich, dass sich für eine bestimmte TCP-Verbindung eine „stehende Warteschlange“

[42] aufbaut, die einen großen Teil der Pufferkapazitäten an der Engstelle verschwendet und die Umlaufzeiten dauerhaft erhöht.

Ein weiteres Problem von TCP Reno ist, dass der eingesetzte Congestion Avoidance Algorithmus nicht für die stetig gestiegenen Übertragungskapazitäten skaliert. Das lineare Wachstum des Staukontrollfensters hängt von der Paketgröße und der Umlaufzeit der Verbindung ab. Die Paketgrößen haben sich seit der Einführung von TCP Reno kaum geändert, und die im Internet üblichen Umlaufzeiten sind durch die weltumspannende Verbreitung des Internets eher gestiegen. Merkliche Reduktionen sind nicht zu erwarten, da hier die Lichtgeschwindigkeit als limitierender Faktor (bei gegebener Leitungslänge) eine untere Schranke festlegt.

Eine gestiegene Übertragungskapazität bedeutet daher, dass (in einem Szenario, wie oben beschrieben) die Perioden zwischen zwei Paketverlusten immer länger werden. In RFC 3649 [43] wurde nachgerechnet, dass bei einer 10 Gbit/s-Verbindung mit einer Umlaufzeit von 100 ms und einer Paketgröße von 1500 Bytes solch eine Periode etwa 1 h 40 min lang ist. Praktisch bedeutet dies, dass TCP Reno unter diesen Voraussetzungen in den seltensten Fällen überhaupt jemals die volle Übertragungskapazität erreicht. Wie in Abschnitt 3.2 beschrieben, ist eine einfache Anpassung der *Steigungsrate* nicht ausreichend.

Verzögerungsbasierte Staukontrolle

Noch bevor die Übertragungsraten im Internet so weit zugenommen haben, dass TCP Reno keinen ausreichenden Durchsatz mehr liefern konnte, wurde bereits am Problem der starken Pufferauslaustung und der periodischen Paketverluste geforscht. Anstelle von Paketverlusten als einzigen Stauindikator nutzen die TCP-Varianten *TCP DUAL (1992)* [44] und *TCP Vegas (1995)* [45] Veränderungen in der Umlaufzeit, um wachsende Puffer zu erkennen, bevor es zu einem Paketverlust kommt.

TCP DUAL versucht dabei, den Füllstand des Puffers an der Engstelle zu schätzen und reduziert sein Staukontrollfenster um $\frac{1}{8}$, sobald der Puffer mehr als 50 % gefüllt ist. Ansonsten nutzt TCP DUAL ebenfalls Slow Start und einen linearen Anstieg des Staukontrollfensters, ähnlich zu Congestion Avoidance.

Einen etwas anderen Ansatz verfolgt TCP Vegas. Es versucht nicht, den generellen Füllstand des Puffers an der Engstelle zu schätzen, sondern die Anzahl *eigener* Pakete in diesem Puffer, indem es die geschätzte Warteschlangenverzögerung mit der geschätzten eigenen Datenrate multipliziert (vgl. Formel 1). Ist diese Zahl kleiner als `zwei`, erhöht TCP Vegas das Staukontrollfenster linear. Ist sie größer als `vier`, wird das Fenster entsprechend reduziert. Dazwischen befindet sich die sogenannte *„Dead Zone“*, bei der die Übertragungsrate als optimal angenommen wird und das Staukontrollfenster

unverändert bleibt. Solch eine Phase besitzt kein anderes der bisher beschriebenen Staukontrollverfahren.

$$\textit{[Pakete im Puffer]} \approx (\mathrm{RTT} - \mathrm{RTT}_{\min}) \cdot \frac{\mathrm{cwnd}}{\mathrm{RTT}} \tag{1}$$

RTT := geschätzte Umlaufzeit
$\mathrm{RTT}_{\min}$:= geschätzte Umlaufzeit, bei leerem Puffer
cwnd := Größe des Staukontrollfensters in Paketen

Das Ergebnis sind geringe Pufferauslastung, kaum (bis keine) Paketverluste, eine sehr gleichmäßige Übertragungsrate und ein höherer Durchsatz als TCP Reno. Allerdings hängt das korrekte Verhalten stark von einer korrekten Schätzung der gepufferten Pakte und damit der Variationen der Umlaufzeit ab. Hier gibt es eine Menge Störfaktoren, wie zum Beispiel Stau in der Rückrichtung, Routenänderungen und ungenaue Zeitgeber. Daher ist TCP Vegas weniger robust als TCP Reno, welches zwar erst sehr spät auf Staus reagiert, dafür aber nur von zuverlässigeren Stausignalen abhängt.

Unabhängig von Implementierungsdetails und Möglichkeiten die Pufferauslastung zuverlässiger zu ermitteln, weist TCP Vegas jedoch auch ein schwerwiegenderes, konzeptionelles Problem auf. Ein Staukontrollverfahren, dass nur geringe Pufferauslastungen verursacht, kann (ohne spezielle Unterstützung im Netz) nicht mit einem aggressiveren Verfahren wie TCP Reno konkurrieren. Werden TCP Reno und TCP Vegas an einer gemeinsamen Engstelle parallel genutzt, zieht sich TCP Vegas bis auf ein Minimum zurück und TCP Reno belegt einen Großteil der verfügbaren Übertragungskapazität.

Verzögerungsbasierte Staukontrollverfahren, insbesondere TCP Vegas, bieten somit einige Vorteile gegenüber verlustbasierten Verfahren, wie TCP Reno, haben aber wiederum auch Nachteile. Da die *Robustheit* aber eine der wichtigsten Eigenschaften ist, die von den im Internet eingesetzte Protokollen gewährleistet werden muss, und die Nachteile von TCP Reno in der Vergangenheit als wenig problematisch angesehen wurden, konnten sich verzögerungsbasierte Verfahren nie durchsetzen. Und da darüber hinaus ein Parallelbetrieb dieser beiden Verfahren, wie oben beschrieben, im Internet nicht möglich ist, ist auch eine langsame, graduelle Einführung von verzögerungsbasierten Algorithmen nicht ohne Weiteres möglich.

In den frühen Jahren der Internet-Staukontrolle lag der Fokus somit insbesondere auf der *Funktionalität* der Staukontrolle und der Fähigkeit Staukollapse effektiv zu verhindern. Metriken, mit denen sich verschiedene Konzepte evaluieren lassen, und eine differenzierte Abwägung, welche Eigenschaften für die Staukontrolle im Internet wichtig sind, waren daher eher zweitrangig. TCP Reno war als Weiterentwicklung von TCP Tahoe auch ohne spezielle Metriken leicht als klare Verbesserung zu erkennen.

Heute gilt das Problem der Staukollapse im Wesentlichen als gelöst und der Fokus liegt nun klar auf der *Leistungsfähigkeit* der Staukontrolle. Da hier allerdings verschiedene Ziele wie Robustheit, Durchsatz, Fairness, Pufferauslastung, etc. gegeneinander abgewogen werden müssen, mussten zunächst *Ziele und Metriken* für Staukontrolle definiert bzw. entwickelt werden. RFC 5166 [46] *„Metrics for the Evaluation of Congestion Control Mechanisms“* (S. Floyd, 2008) und RFC 5033 [37] *„Specifing New Congestion Control Algorithms“* (S. Floyd, M. Allman, 2007) befassen sich mit diesen Themen.

Fairness

Neben dem Verhindern von Staukollapsen ist auch die *faire* Aufteilung von Übertragungskapazitäten innerhalb der Best-Effort-basierten Dienste ein wichtiges Ziel, das von der Staukontrolle im Internet erfüllt werden muss. Für diese Fairness gibt es jedoch keine allgemeingültige, von allen akzeptierte Definition. Da TCP Reno, wie oben beschrieben, lange Zeit als „Standard TCP“ galt, sind die Fairness-Eigenschaften von TCP Reno weitgehend akzeptiert; RFC 2914 [36] stellt an neuere TCP-Varianten die Forderung, dass sie konkurrierende TCP-Reno-Verbindungen nicht wesentlich stärker beeinträchtigen, als es eine andere TCP-Reno-Verbindung tun würde, um eine Art „Wettrüsten“ zu vermeiden.

Zwei konkurrierende TCP Reno Verbindungen bekommen im Mittel einen etwa gleichen Anteil an der zur Verfügung stehenden Übertragungskapazität, wenn sie die gleiche *Umlaufzeit (RTT)* und die gleiche *Paketgröße* haben. Bei größerer Umlaufzeit, bzw. kleinerer Paketgröße bekommt eine Verbindung einen proportional kleineren Anteil. Dies ergibt sich implizit durch die *additive* Vergrößerung des Staukontrollfensters in der Congestion Avoidance Phase und die *multiplikative* Reduktion bei Detektion eines Staus. Dieses Konzept wird in der englischsprachigen Literatur als *„Additive Increase, Multiplicative Decrease“* (kurz: *AIMD*) bezeichnet.

TCP Vegas nutzt kein AIMD, sondern erreicht seine Fairness-Eigenschaften auf andere Weise: Generell gilt, dass zwei Verbindungen die (im Schnitt) gleich viele Daten in einem Puffer haben, den selben Anteil an der Übertragungskapazität bekommen, sofern alle Daten im Puffer gleich behandelt und der Reihe nach gesendet werden werden. Da jede TCP Vegas Verbindung, wie bereits beschrieben, versucht eine geringe Anzahl an Paketen im Puffer der Engstelle zu halten, haben alle konkurrierenden TCP Vegas Verbindungen etwa die gleiche Menge an Daten in diesem Puffer und bekommen somit auch etwa den gleichen Anteil an der Übertagungskapazität. Anders als bei TCP Reno hat hier die Umlaufzeit keinen direkten Einfluss auf diese Aufteilung. Diese Eigenschaft wird als *RTT-Fairness* bezeichnet. Das Fairness-Modell von TCP Vegas unterscheidet sich somit von dem von TCP Reno.

Beide Modelle haben allerdings gemeinsam, dass Fairness *pro TCP-Verbindung* angestrebt wird; nutzt ein Endsystem mehrere TCP-Verbindungen zum selben Ziel, bekommt es gegenüber einem konkurrierenden Endsystem mit nur einer Verbindung einen größeren

Anteil der zur Verfügung stehenden Übertragungskapazität. RFC 2309 [35] stellt neben dieser Herangehensweise noch zwei weitere Möglichkeiten zur Diskussion:

1. Fairness sollte, unabhängig von der Anzahl der genutzten TCP-Verbindungen, *pro Sender-Empfänger-Paar* definiert sein.
2. Fairness sollte *pro Sender* (bzw. pro Empfänger) definiert sein.

Der RFC favorisiert Fairness auf Basis von Sender-Empfänger-Paaren zu definieren, regt aber weitere Diskussionen in der IETF an. Im Internet hat sich allerdings die Fairness pro TCP-Verbindung, wie eben für TCP Reno und TCP Vegas beschrieben, durchgesetzt.

3.2 Verlustbasiertes TCP für schnelle Netze

Wie bereits erwähnt, skaliert TCP Reno nicht für höhere Datenraten. Inzwischen wurde es daher unter Linux von *CUBIC TCP (2006/2008)* [47] abgelöst, unter Windows kommt *Compound TCP (2005)* zum Einsatz. Dennoch ist TCP Reno immer noch vielerorts im Einsatz. Eine Untersuchung, welche TCP-Varianten von den 5000 beliebsten Webservern verwendet werden, findet sich in [48]. Den größten Anteil hat demnach CUBIC TCP (oder Varianten wie BIC TCP) mit etwa 45 %, TCP Reno (oder Varianten, wie TCP New Reno) kommt auf 17 %–26 % und Compound TCP auf 10 %–19 %[3]. Da die jeweiligen Anpassungen nur auf der Senderseite nötig sind, ist diese Heterogenität aber kein Problem für die Interoperabilität.

High-Speed TCP

Bis zur Entwicklung dieser beiden Varianten wurden erst eine Reihe anderer Ansätze untersucht, die sich aber als weniger geeignet herausgestellt haben. *High-Speed TCP (kurz: HS-TCP)* wurde 2003 von Sally Floyd in RFC 3649 [43] vorgestellt und verfolgt das Ziel, effizient in schnellen Netzen mit hohem Bandbreitenverzögerungsprodukt (z. B. 10 Gbit/s, 100 ms RTT) zu sein, ohne dabei konkurrierende TCP-Reno-Verbindungen zu stark zu beeinträchtigen. Da TCP Reno in schnellen Netzen keinen guten Durchsatz erreicht, wird diese Anforderung wie folgt umgesetzt: Bei langsamen Übertragungsraten verhält sich HS-TCP identisch zu TCP New Reno (einer Variante von TCP Reno mit annähernd identischen Fairness-Eigenschaften, vgl. RFC 2582 [49]). Konnte die Übertragungsrate ohne Paketverlust über ein festgelegtes Maß gesteigert werden, passt HS-TCP die Steigungsrate im Congestion Avoidance Modus an. Im Falle einer leichten Stausituation (drei duplizierte Quittungen), wird das Staukontrollfenster bei hohen Geschwindigkeiten nicht auf die Hälfte reduziert (vgl. TCP Reno), sondern auf einen Faktor, der von aktuellen Größe des Staukontrollfensters abhängt.

[3] Die verwendete TCP-Variante ist nicht immer zweifelsfrei zu erkennen. Insbesondere konnten nicht immer zwischen TCP Reno und Compound TCP unterschieden werden.

Bei höheren Übertragungsraten wird also gezielt auf Fairness mit TCP Reno verzichtet. Die *Intra-Protokoll-Fairness* zwischen zwei (oder mehr) HS-TCP Instanzen ist hingegen auch bei höheren Geschwindigkeiten immer noch gegeben, sofern die Paketgrößen und Umlaufzeiten identisch sind. RTT-Unfairness ist zwar bereits von TCP Reno bekannt, allerdings fällt es bei HS-TCP aufgrund der veränderlichen Steigungs- und Reduktionsparameter sehr viel intensiver aus (vgl. [50]).

Das Konzept des exponentiellen Wachstums im Slow Start stößt bei hohen Geschwindigkeiten ebenfalls an seine Grenzen. Da sich die Datenmenge, um die sich das Staukontrollfenster jeweils vergrößert, in jedem Schritt verdoppelt, ist hier schnell ein Wachstum erreicht, das den Algorithmus wirkungslos macht. Sofern die Verbindung nicht durch die Flusskontrolle limitiert wird, endet Slow Start immer mit einem Paketverlust. Bei einer 10 Gbit/s Verbindung kann dieser Paketverlust, laut [50], bis zu 83.000 Pakete (120 MByte) betreffen. HS-TCP enthält daher einen *Limited Slow Start* (RFC 3742 [51]), bei dem die Fenstervergrößerung auf maximal 100 Pakete pro Umlaufzeit beschränkt ist. Bei einer Umlaufzeit von 100 ms wird eine Übertragungsrate von 1 Gbit/s somit nach ca. 8 s erreicht, was in [50] als akzeptabel bezeichnet wird. RFC 3649 [43] beschreibt Limited Slow Start als einfach umzusetzende Lösung, empfiehlt für die Zukunft aber *explizites Feedback* zu nutzen, um einen schnellen und sicheren Startvorgang zu gewährleisten.

Scalable TCP

Ebenfalls im Jahr 2003 wurde *Scalable TCP* [52] entwickelt. Anders als TCP Reno und HS-TCP setzt es nicht auf das AIMD-Konzept, sondern auf *MIMD*. Das heißt, dass das Staukontrollfenster in Abwesenheit von Paketverlusten nicht linear vergrößert wird, sondern *multiplikativ*. Bei einem leichten Stau wird das Staukontrollfenster ebenfalls multiplikativ reduziert, aber um einen anderen Faktor; typische Faktoren sind `0.01` für die Vergrößerung des Fensters und `0.125` für die Verkleinerung. Die Fenstergröße wird somit mit `1.01`, bzw. mit `0.875` multipliziert.

Daher wird das Staukontrollfenster grundsätzlich exponentiell vergrößert, bis es zu einem Paketverlust kommt. Durch die starke Vergrößerung des Staukontrollfensters kommen Paketverluste allerdings *sehr* häufig vor. Im Gegensatz zu AIMD bietet MIMD des weiteren keinerlei Fairness, auch dann nicht, wenn alle Verbindungen die gleiche Umlaufzeit haben.

Scalable TCP zeigt daher gut, dass Staukontrollverfahren mit exponentiellem Wachstum für das Internet ungeeignet sind.

BIC TCP

Ein Jahr nach der Entwicklung von HS-TCP und Scalable TCP wurde der *Binary Increase Congestion Control*-Algorithmus (kurz *BIC*) vorgestellt ([53], 2004). Hierbei wird das Finden einer geeigneten Größe für das Staukontrollfenster als *binäre Suche* aufgefasst.

Fenstergrößen, bei denen es zuletzt zu einem Paketverlust gekommen ist, werden als obere Schranke interpretiert; Fenstergrößen, bei denen keine Paketverluste aufgetreten sind als untere Schranke. Somit wird das Staukontrollfenster jeweils auf den Mittelwert dieser beiden Schranken gesetzt. Um einen zu starken Anstieg zu vermeiden, wird, ähnlich zum Limited Slow Start, die maximale Vergrößerung des Staukontrollfensters begrenzt.

Unter den verlustbasierten Staukontrollmechanismen reduziert BIC somit erstmals die Stärke des Anstiegs des Staukontrollfensters, wenn es nahe des vermuteten Maximums ist. Dadurch kann der nächste Paketverlust hinausgezögert werden und lange Zeit mit einer hohen Rate gesendet werden.

Da in einem IP-Netz die für eine Verbindung zur Verfügung stehende Übertragungskapazität nicht konstant ist, sondern auch zunehmen kann, benötigt der BIC-Algorithmus eine zweite Phase. Ansonsten könnte das Staukontrollfenster nie über einen einmal gemessenen Maximalwert steigen. Sind die obere und die untere Schranke zu nah zusammengerückt, wird daher erneut ein Limited Slow Start initiiert. Dieser startet dann nicht bei einer geringen Fenstergröße, sondern beim aktuellen Wert des Staukontrollfensters.

CUBIC TCP

CUBIC TCP [47] (2006/2008) ist eine direkte Weiterentwicklung von BIC und greift die Idee auf, das Staukontrollfenster nur leicht zu vergrößern, wenn es nahe des vermuteten Maximums ist. Im Gegensatz zu einer binären Suche, gefolgt von Limited Slow Start, wird dieses Verhalten bei CUBIC TCP aber durch eine *kubische Funktion* erreicht.

Motiviert war die Weiterentwicklung insbesondere dadurch, dass BIC in manchen Fällen eine schlechte RTT-Fairness bietet und sich auch unfair gegenüber anderen verbreiteten TCP-Varianten verhalten kann (vgl. [50]). Mit der Beschreibung der Staukontrollfenstergröße als Funktion, kann eines der Kernkonzepte von *Hamilton TCP* ([54], 2004) in CUBIC genutzt werden. Hierbei wird das Staukontrollfenster nicht in Abhängigkeit der Umlaufzeit angepasst, sondern in Abhängigkeit von der vergangenen *Zeit* seit der letzten Stausituation.

Um nicht in bestimmten Situationen von TCP-Reno-Verbindungen unterdrückt zu werden, berechnet CUBIC außerdem, welche Fenstergröße die eigene Verbindung (voraussichtlich) hätte, wenn TCP Reno statt CUBIC verwendet worden wäre. Ist dieser Wert größer als das aktuelle Fenster, wird es auf diesen Wert erhöht.

Nach [50] ist CUBIC seit 2006 die Standard-TCP-Variante im Linux Kernel und zeigt dort ein zufriedenstellendes Verhalten in schnellen Netzen. Trotz der nur langsamen Erhöhung des Staukontrollfensters um das vermutete Maximum, führt CUBIC als verlustbasiertes Verfahren dennoch zu einer hohen Pufferauslastung und zu regelmäßigen Paketverlusten. Die RTT-Fairness, die meist als positive Eigenschaft angesehen wird, wird allerdings u.a. in RFC 5166 [46] kontrovers diskutiert. Das Hauptargument gegen RTT-Fairness ist, dass Verbindung mit größerer Umlaufzeit oft auch mehr Netzwerkressourcen beanspruchen

(z.B. über eine größere Anzahl von Routern laufen), als Verbindungen mit kleinerer Umlaufzeit. Daher, so die Argumentation, sollten diese Verbindungen nur einen kleineren Anteil an der zur Verfügung stehenden Übertragungskapazität bekommen.

3.3 Verzögerungsbasierte Staukontrolle in schnellen Netzen

Aufgrund der bereits beschriebenen konzeptionellen Probleme, die einen Parallelbetrieb von TCP Vegas mit verlustbasierten TCP-Varianten (wie TCP Reno) unmöglich machen, wurden *rein verzögerungsbasierte* TCP-Varianten kaum noch weiterentwickelt. Mit *FAST TCP* [55] (2005) und *TCP New Vegas* [56] (2005) existieren dennoch zwei solcher Algorithmen, welche die andere Schwäche von TCP Vegas adressieren, nämlich die Nutzbarkeit in schnellen Netzen.

TCP New Vegas

TCP New Vegas setzt dabei auf einen Mechanismus, der das Staukontrollfenster in der initialen Slow Start Phase näher an den für die optimale Übertragungsrate benötigten Wert heranbringen soll. Während TCP Vegas die Slow Start Phase verlässt, sobald die Umlaufzeit der Verbindung merklich zunimmt, setzt TCP New Vegas sie mit verminderter Intensität fort, sodass in der darauf folgenden Congestion Avoidance Phase ein geringer „Abstand“ zur optimalen Fenstergröße überbrückt werden muss. An der Congestion Avoidance Phase selbst ändert TCP New Vegas allerdings nichts. Wird durch einen temporären Stau der Slow Start frühzeitig beendet, skaliert TCP New Vegas somit nicht besser als TCP Vegas oder auch TCP Reno.

FAST TCP

Treten keine Paketverluste auf, werden bei *FAST TCP* Verringerung und Erhöhung des Staukontrollfensters simultan durchgeführt. Die Erhöhung ist ein konstanter Wert (α), die Verringerung hängt proportional vom Verhältnis der minimal gemessenen Umlaufzeit (RTT_{min}) und der aktuellen Umlaufzeit ($RTT_{aktuell}$) ab. Leicht vereinfacht wird das Staukontrollfenster bei FAST TCP durch folgende Gleichung gesetzt:

$$w_{\text{neu}} = w_{\text{alt}} * \frac{\text{RTT}_{\text{min}}}{\text{RTT}_{\text{aktuell}}} + \alpha \tag{2}$$

Hinzu kommt noch eine Glättung nach dem bekannten *Exponential Moving Average (EMA)* Verfahren. Außerdem darf sich das Staukontrollfenster in einem Schritt maximal verdoppeln, d. h. maximal so stark steigen wie im Slow Start.

Schritte sind bei FAST TCP allerdings nicht die Umlaufzeiten, sondern ein konstanter Wert, üblicherweise `20 ms`. Dies führt ähnlich wie bei CUBIC zu (Intra-Protokoll-)RTT-Fairness.

Das Verhalten von FAST TCP hängt laut [50] stark vom Parameter α ab. Große Werte von α erhöhen die *Skalierbarkeit* in schnellen Netzen, dies geht aber auf Kosten der *Stabilität*. Geeignete Werte für α sind eine immer noch offene Frage.

FAST TCP wurde nicht von der IETF standardisiert, sondern von der Firma *"FastSoft"* (inzwischen aufgekauft von *"Akamai Technologies"*) als kommerzielles Produkt vertrieben. Der Algorithmus ist patentrechtlich geschützt [57].

3.4 Hybride Staukontrollverfahren

Hybride Staukontrollverfahren setzen sowohl Mechanismen der verlustbasierten Verfahren als auch Mechanismen der verzögerungsbasierten Verfahren ein. Die meisten Ansätze lassen sich in eine der folgenden beiden Kategorien einteilen: Zum einen gibt es Verfahren, die im Wesentlichen verlustbasiert sind, aber Mechanismen der verzögerungsbasierten Verfahren nutzen, um eine bessere Leistung zu erreichen. Auf der anderen Seite gibt es Verfahren, die zunächst rein verzögerungsbasiert arbeiten, aber auf aggressiveres verlustbasiertes Verhalten zurückfallen können, falls sie mit anderen verlustbasierten Verfahren konkurrieren und daher sonst unterdrückt würden. Für beide Kategorien gibt es bereits frühe Umsetzungen.

Den zuletzt genannten Weg geht *TCP Vegas+* ([58], 2000). Ziel bei der Entwicklung war, es einen Migrationspfad weg von TCP Reno hin zu TCP Vegas zu ermöglichen. Daher verhält sich *TCP Vegas+* zunächst wie das normale *TCP Vegas*. Wird über eine Heuristik festgestellt, dass verlustbasierte Verfahren die eigene Verbindung zu unterdrücken drohen, wechselt *TCP Vegas+* sein Verhalten und verhält sich fortan wie TCP Reno.

Den anderen Weg geht *TCP Veno* ([59], 2003). Hier wird mit den Mechanismen von TCP Vegas die Pufferauslastung an der Engstelle geschätzt. Nimmt sie zu, wird aber nicht (wie bei TCP Vegas) das Staukontrollfenster verkleinert, sondern nur die *Steigungsrate* reduziert. Treten drei duplizierte Quittungen auf, was von TCP Reno als leichter Stau gewertet wird, reduziert auch TCP Veno das Staukontrollfenster, allerdings ebenfalls abhängig von der geschätzten Pufferauslastung an der Engstelle. Falls der Puffer als leer angenommen wird, reduziert TCP Veno das Staukontrollfenster nur um 20 %, ansonsten verhält es sich wie TCP Reno und reduziert um 50 %.

Für schnelle Netze sind beide Verfahren nicht geeignet, da das Staukontrollfenster bei beiden Verfahren maximal so stark steigen kann, wie bei TCP Reno.

TCP Africa, Compound TCP und TCP Illinois

Gerade der zuletzt genannte Ansatz scheint aber für schnelle Netze besonders geeignet zu sein. Ähnlich zu TCP Veno, setzt *TCP Africa* ([60], 2005) ebenfalls auf das Schätzen der Pufferauslastung an der Engstelle um die Steigungsrate des Staukontrollfensters anzupassen. Anstelle von einem Reno-artigen Anstieg bei leeren Puffern und „weniger-als-Reno“ bei gefüllten Puffern, nutzt TCP Africa bei leeren Puffern einen zu HS-TCP vergleichbaren Anstieg, und verhält sich identisch zu TCP Reno, wenn eine entsprechende Pufferauslastung feststellbar ist. Somit kann schnell eine adäquate Senderate erreicht und auch über längere Zeit gehalten werden, bevor es zu einem Paketverlust kommt. Trotzdem ist gewährleistet, dass TCP-Africa-Verbindungen nicht von TCP-Reno-Verbindungen unterdrückt werden können. Die Vorteile von rein verzögerungsbasierten Verfahren, wie TCP Vegas, hat TCP Africa dadurch allerdings nicht. TCP Africa wurde bisher nur simulativ evaluiert und z. B. noch nie im Linux-Kernel implementiert.

Fast gleichzeitig mit TCP Africa wurde *Compound TCP*, ebenfalls 2005, von *“Microsoft Research”* vorgestellt. Es nutzt ein ähnliches Konzept. Wird eine geringe Pufferauslastung angenommen, vergrößert Compound TCP das Staukontrollfenster ebenfalls ähnlich stark wie HS-TCP. Nimmt die Pufferauslastung zu, wird die Vergrößerungsrate proportional zur Pufferauslastung reduziert. Sie bleibt jedoch mindestens so groß wie bei TCP Reno. Daher unterscheidet sich das Verhalten von TCP Africa und Compound TCP vor allem in dieser Übergangsphase. Während bei TCP Africa die Steigungsrate des Staukontrollfensters auf einen Schlag reduziert wird, nimmt bei Compound TCP diese Rate nur langsam ab, bis schließlich die gleiche Rate bei bei TCP Reno erreicht ist. Compound TCP hat, wie bereits erwähnt, TCP Reno als Standard-Staukontrollverfahren in Windows Server abgelöst. Der Algorithmus wurde allerdings von Microsoft patentiert [61] und kann nicht frei verwendet werden.

Ein ähnliches Verhalten wie Compound TCP liefert *TCP Illinois* ([62], 2006), welches im Linux Kernel implementiert und frei verwendbar ist. Der Algorithmus selbst unterscheidet sich intern jedoch deutlich von Compound TCP.

Alle drei eben genannten Verfahren (TCP Africa, Compound TCP und TCP Illinois) nutzen somit Mechanismen der verzögerungsbasierten Verfahren, um das Staukontrollfenster schnell öffnen zu können, ohne direkt einen Paketverlust zu produzieren. Geringe Pufferauslastung ist jedoch kein erklärtes Ziel dieser Varianten und Paketverluste können nicht verhindert werden.

YeAH TCP

Hierin unterscheidet sich *YeAH TCP* ([63], 2007) von den drei eben diskutierten Verfahren. Eines der Ziele von YeAH TCP ist es, Netze mit geringen Pufferkapazitäten zu

unterstützen. Dazu werden zwei Mechanismen genutzt. Zum einen wird mittels *Precautionary Decongestion* die Pufferauslastung generell niedrig gehalten, zum anderen reagiert YeAH TCP vergleichsweise schwach auf einen Paketverlust.

Precautionary Decongestion schätzt mit den Methoden von TCP Vegas wie viele eigene Pakete im Puffer der Engstelle enthalten sind (im Folgenden auch als $\mathcal{Q}$ bezeichnet). Ist diese Zahl größer als ein Schwellenwert, wird das Staukontrollfenster genau um diesen Wert reduziert. Theoretisch wird somit innerhalb der nächsten Umlaufzeit der Puffer an der Engstelle geleert, ohne dass die Empfangsdatenrate abnimmt[4]. Ist dieser Schwellenwert niedrig gesetzt, trägt eine einzelne YeAH TCP Verbindung nur schwach zur Pufferauslastung bei und erhöht somit auch die Umlaufzeit auf dem Pfad nur leicht.

Sowohl um nicht-staubedingte Paketverluste besser zu verkraften als auch um Staus effektiv und schnell zu lösen, nutzt YeAH TCP wiederum die eben genannte Schätzung $\mathcal{Q}$ der eigene Pakete im Puffer der Engstelle. Bei drei duplizierten Quittungen wird, wie seit TCP Reno üblich, ein leichter Stau angenommen. YeAH TCP verringert sein Staukontrollfenster daraufhin ebenfalls um $\mathcal{Q}$, mindestens jedoch um $\frac{1}{8}$ des aktuellen Wertes (und maximal um die Hälfte des aktuellen Wertes):

$$w_{neu} = 1 - min\{max\{\frac{w_{alt}}{8}, \mathcal{Q}\}, \frac{w_{alt}}{2}\}$$

Theoretisch trägt die Verbindung nun nicht mehr zum Stau bei, ohne jedoch einen starken Einbruch der Übertragungsrate hinnehmen zu müssen. Dieses Konzept entstammt *TCP Westwood* ([64], 2001), welches auf Umgebungen mit hohen, nicht-staubedingten Paketverlustraten ausgelegt ist, wie z. B. WLAN.

Ein Kompatibilitätsmodus sorgt dafür, dass YeAH TCP nicht von verlustbasierten TCP-Varianten unterdrückt werden kann. Dazu erkennt eine Heuristik, ob YeAH TCP mit solchen Verbindungen konkurriert und schaltet daraufhin den *Precautionary Decongestion*-Mechanismus ab.

Abgesehen von den eben beschriebenen Mechanismen, setzt YeAH TCP auf ein ähnliches Konzept wie TCP Africa. Ist die geschätzte Pufferauslastung gering, erhöht sich das Staukontrollfenster nach den Regeln von Scalable TCP, ansonsten nach Regeln von TCP Reno. Hierzu kommt der bereits beschriebene Wert $\mathcal{Q}$, in Kombination mit einer Stauschätzung nach dem Vorbild von TCP DUAL, zum Einsatz. Anstelle von Scalable TCP kann außerdem, laut [63], jede (verlustbasierte) High-Speed TCP-Variante genutzt werden.

Durch das explizite Ziel der geringen Pufferauslastung und durch die gute Skalierbarkeit in schnellen Netzen scheint YeAH TCP vielversprechend zu sein, und soll im Rahmen des *bwNET100G+*-Projektes noch genauer untersucht werden. Bei Geschwindigkeiten von 100 Gbit/s könnte die Nutzung der Scalable TCP Regeln jedoch zu großen Paketverlusten

[4] Da in der darauffolgenden Umlaufzeit voraussichtlich keine eigenen Pakete mehr gepuffert sind, liegt es jedoch nahe, dass das Staukontrollfenster dann zu gering ist um die Übertragungskapazität voll auszunutzen.

führen, noch bevor eine gestiegene Umlaufzeit ermittelt werden kann. Ebenso sollten die Nachteile des, auf TCP Reno optimierten, Kompatibilitätsmodus untersucht werden.

3.5 Hintergrunddienste

Ziel der bisher beschriebenen TCP-Varianten ist es, auf Basis eines *„Best-Effort“*-IP-Netzes, die zu sendenden Daten möglichst schnell zu übertragen. Daneben gibt es aber auch einen Bedarf nach einem *„Lower-than-Best-Effort“*-Transportprotokoll für Hintergrunddienste. Dies sind beispielsweise Backups, welche zwar früher oder später abgeschlossen sein sollen, allerdings möglichst ohne den sonstigen Datenverkehr zu beeinträchtigen. Ein weiteres Beispiel für einen Hintergrunddienst ist das *opportunistische Prefetching* von Web-Inhalten, also das Laden von Websites auf Verdacht noch bevor ein entsprechender Link angeklickt wurde. Dieses *Prefetching* kann das Surfen im Web zwar beschleunigen, aber nur wenn dadurch das Laden von Websites hinter tatsächlich angeklickten Links nicht beeinträchtigt wird.

TCP Nice

TCP Nice ([65], 2002) ist ein solches *„lower-than-best-effort“* Transportprotokoll, für Hintergrunddienste. Wie die anderen hier vorgestellten TCP-Varianten muss auch für TCP Nice nur der Staukontrollalgorithmus auf Senderseite angepasst werden. Es basiert auf TCP Vegas und macht sich dessen vermeintlichen Nachteil explizit zu nutze: In Konkurrenz mit den etablierten (verlustbasierten) TCP-Varianten, wie TCP Reno, zieht sich TCP Vegas stark zurück und bekommt nur noch einen kleinen Anteil an der zur Verfügung stehenden Übertragungskapazität. Durch verschiedene Anpassungen setzt TCP Nice dieses Verhalten noch konsequenter um, mit dem Ziel sich explizit zurückzuziehen, wenn es gegen „reguläre“ TCP-Verbindungen konkurriert. Ist das nicht der Fall können TCP Vegas wie auch TCP Nice die freie Übertragungskapazität größtenteils nutzen.

TCP LP

Etwa zeitgleich ist *TCP LP (Low Priority)* ([66], 2003) entstanden, das im Wesentlichen die gleichen Ziele verfolgt. Im Gegensatz zu TCP Nice basiert es jedoch auf TCP Reno und TCP DUAL. Die Schätzung der belegten Pufferkapazität nach dem Vorbild von TCP DUAL dient einer *frühen Stauerkennung*. TCP LP kann sich somit zurückziehen bevor konkurrierender Verkehr sich aufgrund von Paketverlusten zurückzieht. Bei geringer Pufferauslastung verhält sich TCP LP identisch zu TCP Reno.

LEDBAT

Aus einem weiteren Anwendungsfall für *„Lower-than-Best-Effort“*-Transportprotokolle heraus hat sich *LEDBAT („Low Extra Delay Background Transport“)* ([67], 2010) entwickelt. Die Entwicklung geschah hier im Kontext des *Micro Transport Protocols*, das von dem Bittorrent-Client *„µTorrent“* eingesetzt wird und wird zur Zeit von der IETF standardisiert. Anders als bei den beiden eben beschriebenen Verfahren ist bei LEDBAT die höchste Priorität, dass nur wenig *zusätzliche Verzögerung* entsteht. Kann dies nicht gewährleistet werden, zieht sich LEDBAT, ähnlich TCP Nice und TCP LP größtenteils zurück.

Somit ist LEDBAT in den gleichen Szenarien nutzbar wie TCP Nice und TCP LP, aber auch noch in einem weiteren: Voice-over-IP-Telefonate, Videotelefonie und beispielsweise auch Online-Spiele sind auf geringe Verzögerungen angewiesen. Sie selbst benötigen aber oft nicht die gesamte zur Verfügung stehende Übertragungskapazität. Daher kann Verkehr durch einen Hintergunddienst toleriert werden, sofern dieser die Verzögerung nicht aufgrund hoher Pufferauslastung zu stark erhöht. LEDBAT versucht daher zu garantieren, dass die, die durch Pufferung bedingte, Verzögerung einen festen Wert (üblicherweise zwischen `25ms` und `100ms` nicht überschreitet.

Wie bei anderen verzögerungsbasierten Staukontrollverfahren kann die Schätzung der Pufferauslastung jedoch fehlerhaft sein. Trifft eine LEDBAT-Verbindung niemals auf einen komplett leeren Puffer, wird die Grundverzögerung falsch geschätzt und daher eine zu hohe Verzögerung indiziert. Dies fällt besonders ins Gewicht, wenn mehrere LEDBAT-Verbindungen über die selbe Engstelle laufen, hier kann es zum sogenannten *„late-comer advantage“* Problem kommen: Eine später gestartete LEDBAT-Verbindung schätzt dabei die Pufferauslastung falsch ein und unterdrückt die bereits existierende LEDBAT-Verbindung. Generell enthält LEDBAT *keinerlei Fairness-Mechanismen*, weswegen hier weiterer Forschungsbedarf besteht.

3.6 Verwandte Themen

In diesem Abschnitt werden abschließend einige weitere Ansätze und Problemstellungen behandelt, die im Rahmen der Staukontrolle wichtig sind.

Multipath TCP

Die „MPTCP“-Working-Group[5] der IETF beschäftigt sich momentan mit der Entwicklung einer TCP-Variante namens *Multipath TCP* ([68], 2012), die mehrere Pfade gleichzeitig nutzen kann. Anders als bei den bisher beschriebenen TCP-Varianten sind bei Multipath TCP größere Änderungen am Protokoll sowohl beim Sender als auch beim Empfänger nötig. Es wird bei der Entwicklung jedoch versucht, möglichst kompatibel für

[5] https://datatracker.ietf.org/wg/mptcp

im Internet verbreitete *Middleboxes* zu sein, also für Geräte, die im Weiterleitungspfad zwischen zwei Endsystemen sitzen, aber nicht rein auf der IP-Schicht arbeiten, sondern gewisse Annahmen über das genutzte Transportprotokoll treffen, (z. B. Firewalls, NAT-Gateways, verschiedene Formen von Verkehrsoptimierern, uvm.). Um Fairness erreichen zu können, aber auch für optimalen Durchsatz, sind für Multipath TCP spezielle Staukontrollverfahren nötig. Oft basieren diese zwar auf bestehenden Staukontrollverfahren für normales TCP, sind aber für die Nutzung mehrerer paralleler Pfade angepasst. Standardmäßig kommt bei Multipath TCP die *LIA*-Staukontrolle (RFC 6356 [69]) zum Einsatz, welche auf TCP Reno basiert. Weitere implementierte Verfahren sind das auf TCP Vegas basierende *wVegas* und *OLIA* [70], welches schneller auf Änderungen im Netz reagieren soll. In künftigen Versionen der Multipath-TCP-Implementierung soll darüber hinaus *BALIA* [71] hinzukommen, welches die Funktionsweise der früheren Verfahren aufgreifen und generalisiert. Ein Multipath-TCP-Staukontrollverfahren für schnelle Netze, bzw. Netze mit großem Bandbreitenverzögerungsprodukt, gibt es zur Zeit jedoch nicht. Von Multipath TCP existiert eine Implementierung für Linux, sie ist jedoch noch nicht in den *offiziellen* Kernel Quellen enthalten. Außerdem wird Multipath TCP von Apple für den Sprachassistenten „Siri“ genutzt. Diese Implementierung ist aber nicht öffentlich verfügbar.

Datacenter TCP

Speziell für den Einsatz in Rechenzentren wurde *Datacenter TCP* ([72], 2010) entwickelt. Die besondere Herausforderung ist hier, dass in Rechenzentren sehr kleine Umlaufzeiten üblich sind. Andere TCP-Varianten sind darauf nicht optimiert und haben beispielsweise unpassende Schwellenwerte und Timereinstellungen. Vor allem aber wirken sich hohe Pufferauslastungen bei kleinen Umlaufzeiten, relativ gesehen, besonders stark aus. Datacenter TCP setzt daher auf *Explicit Congestion Notification (ECN)* und versucht darüber, mit Hilfe von speziellen Algorithmen, die Schwere des Staus zu ermitteln und entsprechend zu reagieren. Üblicherweise wird ECN nur eingesetzt, um zu erkennen, dass ein Stau vorliegt, aber es wird nicht ermittelt wie stark dieser ist. Datacenter TCP ist im Linux Kernel enthalten und ist ebenfalls für Windows verfügbar.

RemyCC

Eine neue Herangehensweise an die Entwicklung von Staukontrollverfahren gehen Keith Winstein und Hari Balakrishnan in *“TCP ex Machina: Computer-Generated Congestion Control”* [73] (2013). Anstelle selbst die Regeln für das Erhöhen und Verringern des Staukontrollfensters zu definieren, werden bei diesem Ansatz nur die *Optimierungsziele* (z. B. geringe Verzögerungen) festgelegt und das Netz modelliert, in dem der Staukontrollmechanismus genutzt werden soll (Umlaufzeiten, Bandbreiten, erwarteter Verkehr, etc.). Die Regeln des Staukontrollmechnismus entwickelt dann ein Computerprogramm namens *„Remy“*, welches für einen Regelsatz (auch *Staukontroll-Schema* oder *RemyCC* genannt)

auf einem 48 Kern Server mehrere Stunden Rechenzeit benötigt. Diese Optimierung läuft aber ausschließlich offline. Ist ein RemyCC fertig berechnet, kann er so auf die Endsysteme übertragen werden und wird dort nicht weiter optimiert. Erste Auswertungen zeigen sehr gute Ergebnisse. Ist das Netz ausreichend gut modelliert, übertreffen die resultierenden RemyCCs die Leistung bisheriger Staukontrollverfahren deutlich. Weichen die Eigenschaften des Netzes aber stark vom ursprünglichen Modell ab, sinkt auch die Leistung der RemyCCs ab. Ob dieser Ansatz somit für das ganze Internet funktionieren würde, ist zur Zeit noch unklar. Naheliegender wäre es, RemyCC in Teilnetzen, z. B. direkt auf Schicht-2 der Internetprotokollfamile (Ethernet, WLAN, etc.), einzusetzen. Die automatisierte Entwicklung von Staukontrollmechanismen steht allerdings noch ganz am Anfang, und es sind noch viele offene Punkte zu klären, bevor dieser Ansatz in der Praxis eingesetzt werden kann. Die Regeln eines RemyCC sind meist sehr komplex und von Menschen nicht nachvollziehbar, d. h. es kann nur experimentell überprüft werden, ob und wie gut ein RemyCC seine Optimierungsziele erfüllt und in welchen Fällen es zu Problemen kommt. In [73] wurden diese Experimente nur simulativ durchgeführt, es lässt sich daher nicht mit Sicherheit sagen, ob dieser Ansatz in der Praxis (außerhalb von Simulation oder Laborbedingungen) überhaupt funktioniert.

XCP

Die bisher in diesem Kapitel vorgestellten Staukontrollverfahren folgen alle einem Ende-zu-Ende-Ansatz. Obwohl es die Hauptaufgabe der Staukontrolle ist, das Netz zu schützen, ist das Netz kaum in diese Verfahren involviert. Üblicherweise werden Paketverluste bzw. eine erhöhte Umlaufzeit als implizite Stauindikatoren genutzt. Mit ECN haben die Router zwar die Möglichkeit einen Stau aktiv zu signalisieren, aber die Aussagekraft von ECN ist sehr begrenzt (1-bit Information).

Dina Katabi et al. haben daher bereits 2002 in [74] einen *„Clean-Slate“*-Ansatz untersucht, wie sich das Netz aktiv in die Staukontrolle einbringen kann. Clean-Slate bedeutet in diesem Zusammenhang, dass bei ihrem Design keine Rücksicht auf die bereits ausgebrachte Infrastruktur genommen wird, so können auch Ideen untersucht werden, die zunächst schwer umsetzbar scheinen. Ihr Ergebnis ist das *eXplicit Control Protocol (XCP)*, welches Modifikationen an Sender, Empfänger und an den Routern erfordert.

Bei XCP informieren die Sender in ihren Paketen alle auf dem Pfad befindlichen Router über die Größe ihres Staukontrollfensters und über ihre geschätzte Umlaufzeit. Erhöhung und Verringerung der Staukontrollfenster wird von den Routern auf dem Pfad kontrolliert. Dazu kodiert jeder Router die gewünschte Änderung einmal pro *Regelungsinterval* in die gesendeten Pakete ein. Der Empfänger muss diese Information dann zurück an den Sender übermitteln. Router, die später auf dem Pfad liegen, können diesen Änderungswert nach unten korrigieren, so dass das Staukontrollfenster effektiv von dem Router kontrolliert sind, der die engste Stelle auf dem Pfad darstellt. Jeder Router bestimmt ein spezielles Regelungsinterval für alle Verbindungen, die durch das selbe Ausgangsinterface des

Routers gehen. Als Wert wird die durchschnittliche Umlaufzeit dieser Verbindungen gewählt, die der Router aus den Informationen der Sender berechnen kann.

Des weiteren trennt XCP die beiden Ziele „maximierung der Linkauslastung“ und „Fairness“ voneinander. Ein *Efficiency Controller* bestimmt, wie stark sich der *aggregierte* Durchsatz der Verbindungen auf einem Ausgangsinterface ändern soll und ein separater *Fairness Controller* verteilt diese Änderung auf die entsprechenden Verbindungen. Falls der aggregierte Durchsatz bereits optimal ist, kann der Fairness Controller trotzdem die Senderaten der einzelnen Verbindungen beeinflussen, um unabhängig vom Efficiency Controller eine faire Verteilung der Übertragungskapazität zu erreichen. XCP erzielt in Simulationen eine sehr gute Leistung.

Der Ansatz die Staukontrolle vom Netz selbst bzw. von den Routern darin kontrollieren zu lassen ist sehr interessant und wurde bei seiner Veröffentlichung als richtungsweisend angesehen. Es gibt aber auch einige Nachteile. Wie die, zur Zeit immer noch andauernde, Einführung von IPv6 in das Internet zeigt, ist es sehr schwierig neue Protokolle im gesamten Internet zu etablieren, insbesondere wenn dazu Änderungen an den Routern erforderlich sind. Selbst wenn die Einführung von XCP gelänge, stellt dies eine große Gefahr dar, dass die erfolgte Umsetzung für unabsehbare Zeit „fest zementiert“ ist und sich Änderungen und neue Ideen nur noch schwer umsetzen lassen. *Software Defined Networking* (kurz *SDN*) versucht diese langen „Updatezyklen“ für neue Features im Netz drastisch zu reduzieren. Die aktuell vielversprechendste SDN-Technologie ist *OpenFlow*[6]. Routerbasierte Staukontrolle wie XCP lässt sich damit aber konzeptionell nicht umsetzen, da OpenFlow keine Programmierbarkeit der Verarbeitung *pro Paket* vorsieht. Ein weiterer Nachteil von XCP ist, dass es nicht auf Staus in niedrigeren Schichten (z. B. in einem Ethernet-Switch) reagieren kann.

Laut [74] muss XCP nicht für jede einzelne Verbindung Zustand halten, was für die Skalierbarkeit im Internet entscheidend ist. Außerdem sei es möglich, Verbindungen zu erkennen, die sich nicht korrekt verhalten. RFC 6077 [75] weist allerdings darauf hin, diese beiden Eigenschaften nicht gleichzeitig umzusetzen sind. Möchte man also, aus Skalierbarkeitsgründen, keinen Zustand für die einzelnen Verbindungen halten, muss auch bei XCP darauf vertraut werden, dass sich die Endsysteme fair verhalten. Somit ist das Netz nicht wesentlich stärker geschützt als bei der aktuell genutzten Ende-zu-Ende-Staukontrolle.

3.7 Konsequenzen für “100G+”

Die nun zur Verfügung stehenden sehr hohen Datenübertragungsraten sollen dazu genutzt werden, neue Anwendungsgebiete zu ermöglichen und bestehende Aufgaben besser zu lösen. Projekte wie bwVisu oder auch hochqualitative Videokonferenzen haben die Anforderung große Datenmengen mit möglichst geringer Verzögerung zu übertragen.

[6] https://www.opennetworking.org/

Geringe Verzögerungen und möglichst wenige Paketverluste werden aber auch für Web-Anwendungen und Cloud-Dienste immer wichtiger. Da bei sehr großen Übertragungsraten viele Datenobjekte (Bilder, Webseiten, etc.) in einer oder wenigen Umlaufzeiten übertragen werden können, fallen Warteschlangenverzögerungen und Sendewiederholungen besonders ins Gewicht [76]. Daher sollte Staukontrolle in „100G+“-Netzen besonderen Wert auf eine geringe Auslastung der Warteschlangen legen. Solche Verfahren können allerdings nicht parallel zu den heute genutzten TCP-Varianten betrieben werden, ohne ihre positiven Eigenschaften zu verlieren. Daher müssen zunächst Wege gefunden werden neue TCP-Varianten von den bestehenden Varianten im Netz effektiv zu trennen.

Der Slow Start Mechanismus ist für die eben genannten Ziele ebenfalls nicht gut geeignet. Kurze Verbindungen, die den Slow Start nie verlassen, werden durch die anfangs kleinen Staukontrollfenster unnötig verzögert. Bei den restlichen Verbindungen führt das exponentielle Wachstum des Staukontrollfensters zu einer Überlastung der Pufferkapazitäten und, gerade bei hohen Geschwindigkeiten, zu einem massiven Paketverlust. RFC 3649 [43] empfiehlt daher explizites Feedback aus dem Netz zu nutzen um die Initialisierung des Staukontrollfensters beim Verbindungsaufbau zu optimieren. Wie bereits im Kontext von XCP diskutiert, kann das Netz generell viele wertvolle Informationen zur Verfügung stellen, um eine effiziente Staukontrolle zu ermöglichen. Dies gilt auch dann, wenn die Staukontrollmechanismen, anders als bei XCP, weiterhin in den Endsystemen implementiert werden. Daher soll im weiteren Projektverlauf auch untersucht werden, welche Informationen aus dem Netz für die Endsysteme hilfreich sind, und wie sie diese nutzen können.

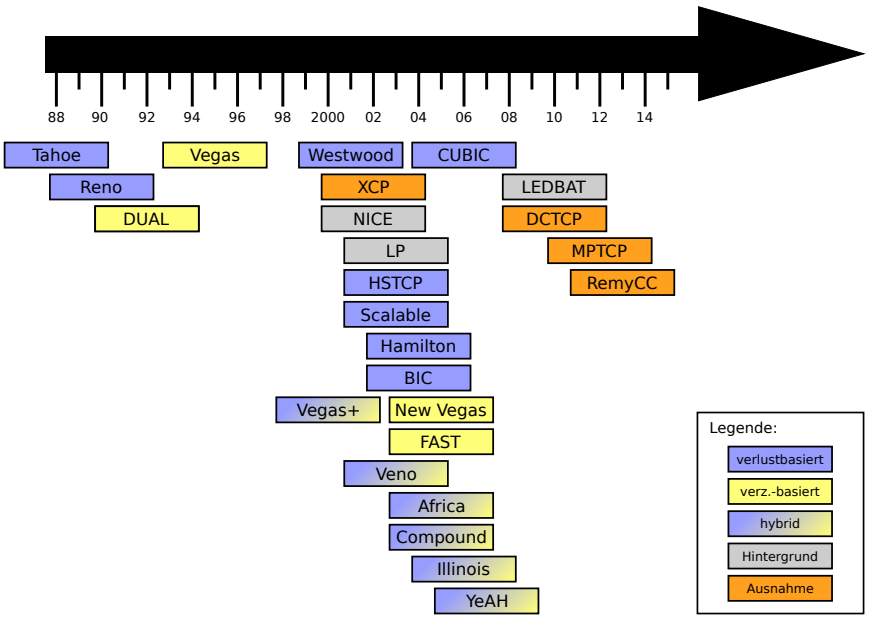


Abbildung 9: Zeitstrahl der verschiedenen TCP-Varianten

Überblick TCP-Varianten

Name	Jahr	Typ	Ziel, Staukontrolle, Kommentar
TCP Tahoe	1988	verlust	· Verhinderung von Staukollapsen · AIMD · Nicht für schnelle Netze geeignet
TCP Reno	1990	verlust	· Bessere Performance als Tahoe · AIMD · Nicht für schnelle Netze geeignet
TCP DUAL	1992	verzög.	· „Sägezähne“ verkleinern · AIMD · Nicht für schnelle Netze geeignet
TCP Vegas	1995	verzög.	· Geringe Verzögerung, wenig Paketverluste · AIAD · Nicht für schnelle Netze geeignet
TCP Vegas+	2000	hybrid	· Migration Reno → Vegas · Vegas / Reno (je nach Konkurrenz) · Nicht für schnelle Netze geeignet
High-Speed TCP	2003	verlust	· Schnelle Netze (hohes BDP[7]) · AIMD mit veränderlichen Parametern · Besonders schlechte RTT-Fairness
Scalable TCP	2003	verlust	· Schnelle Netze (hohes BDP) · MIMD · Nahezu konstanter Stau, keine Fairness
TCP Veno	2003	hybrid	· Weniger Paketverluste als Reno · Reduziert CA[8]-Steigungsrate wenn Stau droht · Nicht für schnelle Netze geeignet
BIC TCP	2004	verlust	· Schnelle Netze (hohes BDP) · Binäre Suche · Abgelöst von CUBIC
Compound TCP	2005	hybrid	· Schnelle Netze (hohes BDP) · Summe aus HS-TCP und Reno; nur Reno (wenn hohe verzög.) · Standard TCP in Windows, patentiert, Fensterreduktion nur durch Verlust
TCP New Vegas	2005	verzög.	· Schnelle Netze (hohes BDP) · Veränderter Slow Start

[7]BDP = *Bandbreitenverzögerungsprodukt*
[8]CA = *Congestion Avoidance*

			· Nur Änderung beim Slow Start, keine Änderung bei CA
FAST TCP	2005	verzög.	· Schnelle Netze (hohes BDP) · Tradeoff zwischen Skalierbarbeit und Stabilität · Patentiert, aufwändiges Tuning
TCP Africa	2005	hybrid	· Schnelle Netze (hohes BDP) · HS-TCP; Reno (wenn hohe verzög.) · Nicht implementiert
CUBIC TCP	2006	verlust	· Schnelle Netze (hohes BDP) · Kubische Funktion · Standard TCP in Linux
TCP Illinois	2006	hybrid	· Schnelle Netze (hohes BDP) · Invers-/proportional zu verzög. · Fensterreduktion nur durch Verlust
YeAH TCP	2007	hybrid	· Schnelle Netze (hohes BDP), geringe verzög. · Scaleable TCP; New Reno (wenn hohe verzög.) · *Precautionary Decongestion*, Kompatibilitätsmodus (wenn unterdrückt)

Tabelle 4: Überblick TCP-Varianten

4 AP3: Sicherheitskonzepte für Hochleistungsnetze

Das Lahmlegen von Diensten, das illegitime Nutzen von Ressourcen, Spionage, Betrug oder Diebstahl sind häufig die Ziele von Angriffen auf Computernetzwerken. Jeder Anbieter eines Netzwerkes hat ein Interesse daran, diese Angriffe zu unterbinden und so Schäden von sich selbst oder von Kunden abzuwenden. Computersysteme auf dem neuesten Stand zu halten ("patchen") kann gegen viele aber nicht alle Angriffe schützen. Auch ist es oft nicht möglich Systeme auf den neuesten Stand zu bringen und Sicherheitslücken sind oft kurz nach ihrem Bekanntwerden noch nicht geschlossen. In diesen Fällen muss der Schutz der Systeme durch Mechanismen der Netzwerksicherheit erfolgen. Die zwei wesentlichen Ansätze versuchen hier entweder, durch Perimeter Defense den Zugriff zum Netzwerk und auf die verwundbaren Systeme zu verhindern oder die Mechanismen sollen Angriffe erkennen, um die Angreifer aus dem Netzwerk auszuschließen zu können oder den Administratoren der Systeme eine zeitnahe Reaktion zu ermöglichen. Firewalls als präventives System und Intrusion Detection und Prevention Systeme als reaktive Systeme erfüllen die Aufgabe der Erkennung und Abwehr solcher Angriffe.

Firewalls beschränken den eingehenden und ausgehenden Verkehr in einem Netzwerk nach festgelegten Regeln [77]. Hauptsächlich werden Firewalls zum Unterbinden schadhaften Netzwerkverkehrs eingesetzt, jedoch können sie auch genutzt werden, um Policy Regeln etwa in Firmennetzwerken durchzusetzen, etwa um den Zugriff auf Social Media Seiten während der Arbeitszeit zu verhindern [78]. Firewalls sind proaktive Systeme, die Angriffe erschweren und bestenfalls verhindern sollen. Intrusion Detection Systeme dienen der Überwachung des Verkehrs innerhalb des Netzwerks und betreffen somit nicht ausschließlich den an der Netzwerkgrenze aus- und eingehenden Verkehr. IDS melden erkannte Angriffe, reagieren aber nicht auf diese während Intrusion Prevention Systeme aktiv eingreifen sobald ein Angriff erkannt wird. IPS sind daher reaktive Systeme, die sich einschalten, wenn eine Attacke entdeckt wird und setzen auf IDS auf. Trotz dieser Unterschiede sind Firewalls, IDS und IPS artverwandte Systeme und nutzen ähnliche Basismechanismen. Pakete werden von allen drei Systemen auf bestimmte Eigenschaften untersucht und sie reagieren auf Unregelmäßigkeiten, etwa durch Meldung oder Zugangsbeschränkung. Der Übergang zwischen den Systemen ist fließend, was dazu führen kann, dass zum Beispiel Intrusion Prevention Systeme auch als Firewalls eingesetzt werden können [79].

Software Defined Networking (SDN) ist eine weitere Technologie, die in diesem Projekt auch hinsichtlich ihrer Sicherheitsaspekte untersucht werden soll. SDNs sind der Versuch, die Data Plane (das System, das Pakete weiterleitet) und die Control Plane (das System, das entscheidet wohin Pakete weitergeleitet werden) von Netzwerken voneinander zu trennen. Dies soll der einfacheren Verwaltung der Netzwerktopologie und der Netzwerkadministration an sich dienen, bietet aber auch jenseits der Vereinfachung des Bestehenden neue Möglichkeiten, aus Sicherheitssicht aber auch neue Gefahren. Auch hierauf soll im Folgenden eingegangen werden.

4.1 Angriffsvektoren

Es gibt eine Reihe verschiedener Angriffsvektoren auf Netzwerke und darin befindliche Rechner, welche durch in den folgenden Kapiteln genannten Technologien erkannt und verhindert werden sollen und hier vorgestellt werden [80].

4.1.1 Denial of Service Angriffe

Das Ziel von DoS-Attacken ist es, das Netzwerk oder spezielle Dienste im Netz unzugänglich zu machen. Sie können daher in zwei Gruppen aufgeteilt werden: Attacken auf Betriebssysteme, welche Fehler ausnutzen, die sich durch Patches beheben lassen und Netzwerkattacken, die inhärente Ressourcenlimits von Netzwerkprotokollen und der Netzwerkinfrastruktur ausnutzen. Ein typisches Beispiel sind SYN Flooding Attacken, welche den für den Verbindungsaufbau notwendigen Three-Way Handshake ausnutzen. Unter Angabe von gefälschten Absender-IP-Adressen sendet der Angreifer SYN-Pakete an das Opfer, um eine Verbindung aufzubauen. Das Opfer erstellt einen Eintrag für die Verbindung im eigenen Speicher und antwortet mit einer SYN/ACK Nachricht an die vom Angreifer angegebene Adresse, empfängt aber die finale ACK-Nachricht nicht, da das Gerät mit der gefälschten Adresse nicht antwortet. Der Eintrag im Speicher des Opfers wird erst nach einer bestimmten Zeit gelöscht. Genug Zeit also für den Angreifer den Angriff zu wiederholen und mit genügend Wiederholungen den Speicher des Opfers zu füllen und ihn so außer Gefecht zu setzen.

4.1.2 Probing

Probing wird eingesetzt, um die Netzwerkinfrastruktur auszuspähen, etwa die vergebenen IP-Adressen, welche Services angeboten werden und welche Betriebssysteme in Benutzung sind. Diese Informationen dienen Angreifern dazu, mögliche Angriffsvektoren im Netzwerk ausfindig zu machen und dienen meist der Vorbereitung für Angriffe auf Dienste im Netzwerk.

4.1.3 Kompromittierung

Die Kompromittierung eines Systems funktioniert durch Ausnutzung von dem Angreifer bekannten Sicherheitslücken im Zielsystem. Man unterscheidet hier zwischen Insider und Outsider Attacken und es kann ferner unterteilt werden in Remote to Local Attacken und User to Root Attacken. Ersteres bedeutet, dass Angreifer über das Netzwerk auf einen Rechner Zugang erlangen, etwa durch Passwörter raten oder durch das Ausnutzen von Softwarelücken. Zweiteres setzt voraus, dass Angreifer in einem System Zugang haben – entweder durch einen legitimen Account oder durch einen Angriff der ersten Kategorie – aber keine Rootrechte, die nun durch diesen Angriff erlangt werden sollen.

4.1.4 Viren, Würmer und Trojaner

Viren, Würmer und Trojaner sind Programme, die sich auf Computern replizieren und durch das Netzwerk verbreiten. Viren infizieren andere Programme und benötigen typischerweise menschliche Interaktion zur Replikation (etwa durch das Öffnen eines infizierten E-Mail-Anhangs). Würmer breiten sich selbstständig in Netzwerken aus. Trojaner maskieren sich selbst als legitimes Programm, welches eventuell sogar die versprochene Funktionalität aufweist.

4.2 Firewalls

Es gibt zwei grundsätzlich unterschiedliche Systeme die unter den Begriff Firewall fallen. Zum einen Softwarefirewalls die auf persönlichen Computern installiert werden und zum anderen Netzwerkfirewalls. Beiden gemein ist der grundsätzliche Ansatz, unautorisierte Zugriffe auf ein System zu verhindern. Im Rahmen dieses Projekts relevant ist dabei nur die Netzwerkfirewall, weswegen im Folgenden nur auf diese eingegangen werden soll. Netzwerkfirewalls werden dabei meist auf den Routern eingesetzt, die das Netzwerk abgrenzen und überwachen so den gesamten ein- und ausgehenden Netzwerkverkehr.

4.2.1 Einordnung verschiedener Firewalltechnologie

Bei Firewalls wird grundsätzlich unterschieden zwischen stateless und stateful Firewalls. Stateless Firewalls entscheiden bei jedem Paket von neuem wie sie mit dem Paket umgehen sollen während stateful Firewalls sich Verbindungen und vorher getroffene Entscheidungen merken. Stateful Firewalls sind also komplexer während stateless Firewalls einen höheren Rechenaufwand benötigen, da jedes Paket analysiert werden muss [77].

4.2.2 Geschwindigkeit aktueller Firewalltechnologie

Aktuelle Firewalltechnologien, etwa von HP oder Cisco erreicht Geschwindigkeiten von 40 Gbit/s [81] [82], es ist zu erwarten, dass auch 100Gbit/s bald erreicht werden, weswegen es in diesem Projekt nicht darum gehen soll, Firewalls zu beschleunigen sondern eher um den Umgang mit bestehender Firewallinfrastruktur und wie diese weiter in schnelleren Netzwerken eingesetzt werden kann. Ein möglicher Ansatz unter Einbeziehung von SDN wurde etwa in Kapitel 2.7.1 beschrieben.

4.3 Intrusion Detection / Prevention Systeme

Wie auch bei Firewalls gibt es auch bei dem Begriff Intrusion Detection System zwei grundsätzlich unterschiedliche Systeme. Zum einen wieder die hostbasierte Lösung für persönliche Rechner, die zum Beispiel Log-Files auswertet und so unerlaubte Zugriffe

entdeckt oder auch – dann eher unter dem Begriff Anti-Virus Programm oder Malware Detection – aktiv gegen Malware im System vorgeht, zum Anderen aber auch wieder die netzwerkbasierte Lösung, die Network Intrusion Detection Systeme. Auch hier soll wieder nur auf die Netzwerktechnologie eingegangen werden, gerade da Hostbasierte Intrusion Detection Systeme weniger Probleme haben mit der Geschwindigkeit der Netzwerkkarte mitzuhalten, während Network Intrusion Detection Systeme (NIDS) mit immer schneller werdenden Netzen mithalten müssen und oft zentral in Backbones eingesetzt werden, wo ein Vielfaches der Datenrate anfällt.

Network Intrusion Detection Systeme verwenden verschiedene Methoden, um Angriffe zu erkennen. Grob lassen sie sich in zwei Klassen aufteilen. Zum Einen signaturbasierte Systeme und zum Anderen anomaliebasierte Systeme.

4.3.1 Signaturbasierte NIDS

Signaturbasierte Systeme gleichen den Netzwerkverkehr mit den Signaturen von bekannten Angriffsszenarien ab. Diese Systeme sind sehr zuverlässig in der Erkennung bekannter und insbesondere automatisierter Angriffe, haben aber keine Möglichkeit unbekannte Angriffsmethoden zu erkennen.

Die Signaturen können dabei auf unterschiedliche Art und Weise im System abgebildet werden [83].

Zustandsbasierte Erkennung erfolgt etwa durch Modellierung von Zustandsübergänge. Hier kann immer nur eine bestimmte Reihenfolge abgedeckt werden und sollte ein Zustand nicht erkannt worden sein, wird der gesamte Angriff nicht erkannt. Alternativ können auch Petrinetze verwendet werden, damit kann eine kompliziertere Baumstruktur abgebildet werden.

String Matching bedeutet, es wird nach bestimmten Symbolfolgen in den Paketen gesucht. String Matching ist einfach zu realisieren, allerdings sehr unflexibel. Die Schwierigkeit hier besteht darin, Zeichenfolgen auch dann zu finden, wenn sie über mehrere Pakete verteilt sind, insbesondere wenn die Pakete nicht in der richtigen Reihenfolge durch das IDS geleitet werden. Außerdem verursachen auch bereits kleine Abweichungen in den Zeichenfolgen eine Nicht-Erkennung eines Angriffes.

Expertensysteme sind Systeme, die auf Basis von Regeln, die das Eindringen von Angreifern ins Netzwerk beschreiben, ein solches Eindringen erkennen. Oft werden Techniken wie Vorwärtsverkettung zur Beschreibung eingesetzt. Expertensysteme eignen sich besonders, wenn neue Daten regelmäßig in das System aufgenommen werden sollen. Diese Systeme sind sehr flexibel und mächtig, sind dadurch aber auch langsamer als einfachere Systeme.

Einfache regelbasierte Systeme sind eine vereinfachte Form der Expertensysteme, die schneller aber auch weniger mächtig sind.

4.3.2 Anomaliebasierte NIDS

Anomaliebasierte Systeme überwachen den Netzwerkverkehr und versuchen, auffälliges Verhalten zu melden. Die große Schwierigkeit bei diesen Systemen ist es die False Positive Rate auf ein praktikables Maß zu senken. Da im Allgemeinen nur ein sehr geringer Anteil des Netzwerkverkehres zu Angriffen gehört, kann auch eine auf den ersten Blick als gering erscheinende False Positive Rate von 1% bereits ein IDS weitgehend unbenutzbar machen. Angenommen 0.001% des Netzwerkverkehres wären Angriffe und Angriffe würden sicher erkannt werden. Wenn 1% des Netzwerkverkehres bereits als False Positive erkannt wird, ist nur jede tausendste Angriffsmeldung ein echter Angriff.

Anomaliebasierte Intrusion Detection Systeme lassen sich in verschiedene Subgruppen einteilen. Die folgende Unterteilung folgt der Einordnung in [84]. Teilt man die Systeme ein nach der eingesetzten Technik bilden sich die drei Gruppen statistische Modelle, wissensbasierte Modelle und maschinelles Lernen.

Auf Statistik basierende Modelle Die statistischen Modelle sehen in einer Lernphase die Erstellung eines Netzwerkprofil auf Basis mitgeschnittenen Netzwerkverkehres vor. Die Metriken für diese Profil sind beispielsweise Übertragungsrate, die Anzahl der Pakete eines Protokolls, die Verbindungsrate oder die Anzahl verschiedener IP-Adressen. Befindet sich das IDS nun im Betrieb, also in der Erkennungsphase, wird ein Profil des laufenden Netzwerkverkehres erstellt und gegen das vorher erstellte Profil abgeglichen. Abweichungen über einem bestimmten Schwellwert werden als Alarm signalisiert.

Die ersten statistischen Verfahren verwendeten univariante Modelle mit unabhängigen Gaußsche Zufallsvariablen [85]. Später wurden multivariante Systeme mit Abhängigkeiten zwischen den Zufallsvariablen eingeführt [86]. Weitere Ansätze sind das Verwenden der empirische Verteilungsfunktionen, Markovmodelle um TCP-Verbindungen abzubilden oder die Modellierung der Verbindung mit Grauwertmatrizen [87]. Eine weitere Methode, die aus der Informationstheorie stammt, ist es die Entropie des normalen Netzwerkverkehres zu bestimmen und mit dem aktuellen Netzwerkverkehrs zu vergleichen. Die Idee besteht darin, dass Angriffe eine andere Redundanz aufweisen als gewöhnlicher Netzwerkverkehr [88].

Ein großer Vorteil statistischer Verfahren ist, dass kein Vorwissen über den Netzwerkverkehr benötigt wird sondern das Wissen direkt über Beobachtung des Netzwerkverkehres erlangt werden kann. Es kann allerdings nicht jedes Angriffsszenario mit statistischen Erhebungen erkannt werden.

Wissensbasierte Modelle Auch bei anomaliebasierten NIDS gibt es Expertensysteme. Diese stellen eine der meistgenutzten Erkennungsarten bei A-NIDS dar. Diese klassifizieren die Auditdaten in drei Schritten. Zuerst werden verschiedene Eigenschaften und Klassen von Daten in den Trainingsdaten identifiziert, dann werden Regeln, Parameter oder Prozeduren daraus abgeleitet. Die Auditdaten werden schließlich den entsprechenden Regeln, Parametern und Prozeduren zugeordnet.

Spezifikationsbasierte Anomalieerkennung nennt sich dabei ein Modell, bei dem menschliche Experten das Modell erstellen auf Basis des spezifizierten Systemverhaltens. Mit einer vollständigen Spezifikation und wenn legitimes Verhalten jenseits des spezifizierten ausgeschlossen werden kann, ist diese Methode sehr effektiv. Lediglich Angriffe, die der Spezifikation folgen, können von einem solchen System nicht erkannt werden. Problematisch ist allerdings, dass viele Systeme – gerade die in diesem Projekt betrachteten – keine einfache Beschränkung auf klare Spezifikationen ermöglichen. Ein weiteres Problem dieser Systeme ist die geringe Flexibilität. Um eine Erweiterung des Systems vorzunehmen muss die Spezifikation, die Implementierung und auch das IDS angepasst werden. Je stärker die Spezifikation erweitert wird, desto höher das Risiko, dass Angriffe im Rahmen der Spezifikation möglich werden. Etwa können spezifikationsbasierte Anomalieerkennungssysteme in abwärtskompatiblen Systemen schwer Downgrade Attacken erkennen.

Spezifikationen werden hier in unterschiedliche Weise angegeben, etwa mit einem endlichen Automat, in einer Beschreibungssprache (z.B. n-Gramme [89] [90] [91] oder UML) oder mit einer regelbasierten Klassifikation.

Schemata auf Basis maschinellen Lernens Maschinelles Lernen erstellt Wissen auf Basis von gesammelten Erfahrungen eines technischen Systems. Ein solches System kann in einer Einlernphase erst die Umgebung selbst kennen lernen und sich so auf die speziellen Gegebenheiten dieses Systems anpassen. Änderungen im System erfordern dann eine neue Einlernphase aber eventuell keine Änderung am IDS selbst. Praktische Probleme entstehen hierbei dadurch, dass Trainingsdaten erstellt werden müssen, die garantiert keine Angriffe erhalten und den realen Netzwerkverkehr möglichst gut abbilden. Diese Technologien haben aber als einzige den Vorteil, dass auch ein Lernen während des Betriebs grundsätzlich möglich ist, dies aber auch die Gefahr birgt, dass ein Angreifer durch ein langsames Ändern des Netzwerkverkehres das IDS an den geänderten Zustand gewöhnt und so unentdeckt bleiben kann.

Es gibt sehr viele unterschiedliche Herangehensweisen, wie das Vorgehen einer lernenden Maschine aussehen kann. Die bekanntesten und am meisten verbreiteten Methoden werden im Folgenden erklärt.

- Bayes'sche Netze bilden probabilistische Verhältnisse zwischen Variablen durch einen azyklischen Graphen ab. Die Knoten stellen dabei Zufallsvariablen und die Kanten bedingte Abhängigkeiten dar [92].

- Markov Modelle teilen sich auf in Markovketten und Hidden Markov Modelle. Markovketten sind Zustandsautomaten, bei denen die Übergänge mit Wahrscheinlichkeiten belegt sind. Durch diese kann die Topologie und Fähigkeiten des Systems modelliert werden. Das Hidden Markov Model sieht das System als eine Markovkette an, dessen Eigenschaften aber nicht beobachtbar sind. Es gibt zahlreiche IDS-Modelle auf Basis von Markov Modellen [93] [94] [95].
- Künstliche neuronale Netze, eingesetzt in IDS unter anderem von Miikkulainen et al. [96], simulieren die Vorgehensweise biologischer neuronaler Netze im menschlichen Gehirn. Neuronale Netze sind sehr flexibel und können sich Änderungen in der Umgebung vergleichsweise leicht anpassen. Neuronale Netze bieten allerdings keine Möglichkeit getroffene Entscheidungen nach zu empfinden.
- Fuzzylogik stellt eine Verallgemeinerung der booleschen Logik dar, indem Wahrheitsaussagen mit einer Unschärfe belegt werden. Bei IDS kann die Fuzzylogik zur Modellierung des Netzwerkverkehres zum Einsatz kommen, da die Eigenschaften sich gut damit abbilden lassen [97]. Ein Verhalten wird dann als normal angesehen, wenn es sich in einem bestimmten Bereich bewegt. Fuzzylogik wird hier oft in Zusammenhang mit Data-Mining Techniken eingesetzt [98].
- Genetische Algorithmen sind eine besondere Form der evolutionären Algorithmen, die verschiedene von der Evolutionsbiologie inspirierte Verfahren wie Vererbung, Mutation, Selektion und Rekombination verwenden. In IDS eingesetzt können genetische Algorithmen zur Erstellung von Klassifikationen dienen [99], zur Findung verschiedener Parameter und Eigenschaften der Detektionsprozesse [97] oder zur Filterung des Netzwerkverkehres [100].
- Clusterbildungstechniken ordnen die beobachteten Daten Gruppen zu. Diese Gruppen werden oft durch eine repräsentativen Punkt in dieser Gruppe vertreten. Datenpunkten wird dann eine Distanz zu diesen Gruppen zugeordnet, etwa mit der euklidischen Distanz oder der Mahalanobis-Distanz. Überschreitet die Distanz einen bestimmten Grenzwert wird eine Anomalie erkannt. Mananadhar und Aung entwickelten ein IDS System auf dieser Basis, welches sich auf die TCP Header und andere einfach zugreifbare Daten beschränkt [101].

4.3.3 Eingesetzte Software

Tabelle 5 gibt einen Überblick über verschiedene IDS Systeme, die – laut Herstellerangaben – auch anomaliebasiert arbeiten. Zwei Open Source Lösungen sind hier besonders hervor zu heben, da diese besonders in der Forschung weit verbreitet sind. Snort [102] ist ein signaturbasiertes IDS, das aber mit Erweiterungen auch rudimentäre anomaliebasierte Erkennungsmöglichkeiten bietet, etwa mit dem mittlerweile eingestellten SPADE (Statistical Packet Anomaly Detection Engine) Plug-in. Der Bro Network Security Monitor [103] ist ein Framework für Intrusion Detection Systeme. Beide Systeme setzen auf

Name	Manufacturer	H	R	Anomaly-related techniques
AirDefense Guard	AirDefense & Inc.	•	•	Context-aware detection, correlation and multi-dimensional detection engines
Barbedwire IDS Softblade	BarbedWire Technologies	•	•	Protocol analysis, pattern matching
BreachGate WebDefend	Breach security	•		Behaviour-based analysis, statistical analysis, correlation
Bro	Lawrence Berkeley National Laboratory	•	•	Application level semantics, event analysis, pattern matching, protocol analysis
Checkpoint IPS-1	NFR Security	•	•	Confidence indexing
Cisco Intrusion Prevention System	Cisco Systems	•	•	Behaviour analysis, statistical analysis
DeepNines BBX Intrusion Prevention (IPS)	DeepNines Technologies	•		Multi-Method Inspection (MMI), behaviour analysis, protocol analysis, data correlation
EMERALD	SRI	•	•	Rule-based inference, Bayesian inference
FireProof	Radware Ltd.	•		Protocol anomalies
Firestorm NIDS	Gianni Tedesco	•		Protocol anomalies
Mazu Profiler	Mazu Networks & Inc.			Behaviour analysis (heuristics)
ModSecurity	Ivan Ristic	•		Event correlation
Network at Guard (NG)	C-DAC (formerly National Centre for Software Technology)	•	•	Protocol anomaly detection, statistical analysis
Next Generation Intrusion Detection Expert System (NIDES)	SRI	•		Statistical analysis
Nitro Security IPS	Nitro Security	•		Behaviour analysis
nPatrol	nSecure			Statistical analysis (profiles)
Portus (PAD)	Livermore Software Laboratories, Inc.	•	•	Protocol anomaly detection
Prelude IDS	Yoann Vandoorselaere et al.			Open platform/multiple anomaly-based modules available (3rd party)
SecureNet IDS/IPS	Intrusion Inc.	•	•	Protocol decoding, protocol anomalies
Siren	Penta Security	•	•	Abnormal user behaviour
Snort IDS	Marty Roesch	•		Open platform/multiple anomaly-based modules available (3rd party)
Snort_inline	Rob McMillen	•	•	Open platform/multiple anomaly-based modules available (3rd party)
Sourcefire ETM	Sourcefire Inc.	•	•	Network behaviour analysis
SPADE	Silicon Defense			Statistical analysis
StealthWatch	Lancope	•	•	Network behaviour analysis, “concern index”
Strata Guard IDS/IPS	StillSecure	•	•	Behaviour analysis, protocol anomalies
Symantec Intrusion Protection	Symantec	•	•	Behaviour-based
TippingPoint Intrusion Prevention System	3COM/TippingPoint Technologies	•		Statistical analysis, profiles
Toplayer Attack Mitigator IPS	Top Layer Networks	•	•	Statistical analysis, profiles

Tabelle 5: Übersicht über anomaliebasierte NIDS-Systeme aus [84]. H = Hybrid, R = Response

die libpcap Bibliothek auf. Eine dritte Open Source Plattform ist Suricata, die von der Open Information Security Foundation entwickelt wird [104].

Auffällig ist, dass auch die anomaliebasierten Systeme im Kern auf einem signaturbasierten Detektionsmodell aufsetzen. Die meisten Systeme setzen also auf eine hybride Lösung um die Vorteile beider Modelle nutzen zu können. Einige der Systeme sehen eine Antwort auf Gefahren vor, etwa durch Einbindung der Firewall, das Zurücksetzen von TCP-Verbindungen oder das Einsetzen eines Honeypots. Die meisten Hersteller treffen keine Aussagen darüber, wie umfangreich die anomaliebasierte Erkennung ihrer Systeme ist oder gar welche Techniken eingesetzt werden [84].

4.3.4 Umgehung von Intrusion Detection Systemen

Um IDS Systeme zu umgehen, werden verschiedene Techniken eingesetzt, die zum Beispiel von Chaboya et al. zusammen gestellt wurden [105]. Etwa kann ausgenutzt werden, dass das IDS mit Netzwerkverkehr eventuell anders umgeht als das Zielsystem, da das IDS oft ein anderes Betriebssystem nutzt als das Zielsystem und sich oft auch in einem anderen Teil des Netzwerkes befindet. Das IDS kann auch überladen werden, wodurch die Angriffspakete eventuell vom IDS nicht mehr analysiert werden oder in einer Flut von Angriffsmeldungen unter gehen. IDS stimulators werden gerade dafür eingesetzt möglichst viele Alarme auszulösen. Außerdem können Angriffe maskiert werden. Es kann zum Beispiel sein, dass das Zielsystem `0x2f` als `/` interpretiert, das IDS jedoch nicht. Dies kann dann eine directory traversal Attacke am NIDS vorbei ermöglichen. Zero-Day-Attacken sind zumindest für signaturbasierte Systeme unmöglich zu erkennen, für anomaliebasierte ist eine Erkennung zumindest nicht garantiert, da Tests hierfür unmöglich sind.

Um signaturbasierte Systeme zu umgehen, die pattern-matching einsetzen, können Polymorphismen verwendet werden, das heißt, dass der Schadcode regelmäßig verändert wird, beispielsweise indem er jedes mal mit einem anderen Verschlüsselungsalgorithmus verschlüsselt wird [106]. Ein Signaturbasiertes IDS kann auf den Schadcode nicht vorbereitet, eine Erkennung somit ausgeschlossen werden.

Eine weitere Methode die Erkennung auszuschließen ist es, den Inhalt der Pakete zu verteilen und so dafür zu sorgen, dass ein IDS erst den kompletten Paketstrom empfangen und zusammen setzen muss um den Inhalt analysieren zu können [107].

Die Analyse von IDS-Daten wird meist von Administratoren von Hand vorgenommen. Eine Möglichkeit dafür zu sorgen, dass eine vom IDS erkannte Attacke dennoch erfolgreich ist, ist den Administrator glauben zu lassen, dass die Attacke nicht erfolgreich war. Dies kann zum Beispiel dadurch erreicht werden, dass direkt nach Entdecken einer Hintertür in einem System kein Versuch unternommen wird sich mit dem System zu verbinden. So ist zwar das Scannen des Netzwerks erkennbar, der Erfolg im Finden einer Lücke nicht. Auch kann ein Angreifer einer übernommenen Maschine vorgeben, diese würde Verbindungsversuche des Angreifers nicht zulassen.

4.3.5 Problematik bei hohen Geschwindigkeiten

Die Leistungsfähigkeit von Netzwerksicherheitssystemen hängt stark mit der Leistungsfähigkeit der Hardware zusammen auf der diese eingesetzt werden. Laut Moore's Gesetz verdoppelt sich diese Leistungsfähigkeit alle 18 Monate. Gilder's Gesetz hingegen besagt, dass die Bandbreite von Netzwerken sich alle sechs Monate verdoppelt. Die wachsende Diskrepanz zwischen Systemleistung und Netzwerkleistung muss mit immer besser durchdachten und effizienteren Systemen begegnet werden [108]. Im Folgenden sollen einige Ansätze vorgestellt werden.

4.3.6 Techniken zur Beschleunigung von IDS

Die Ansätze lassen sich dabei grob in zwei Kategorien zusammenfassen [109]. Zum Einen das gezielte Weglassen von Daten um der Datenrate Herr zu werden und zum Anderen das Verteilen des Netzwerkstromes auf mehrere Instanzen und dadurch Parallelisierung der Verarbeitung.

Datenfilterung kann etwa durch Stichprobenanalysen anstelle der vollständigen Analyse des gesamten Netzwerkverkehres erreicht werden. Brauckhoff et al. untersuchten die Auswirkungen der hierdurch fehlenden Daten und kamen dabei zu dem Schluss, dass bei volumenbasierten Analysemethoden kein negativer Effekt feststellbar ist, flow count Metriken aber beeinträchtigt werden [110]. Mai et al. überprüften mehrere Stichprobenverfahren und stellten Beeinträchtigungen von anomaliebasierter IDS fest, sowohl bezüglich der false positive als auch der false negative Rate [111], dies gilt insbesondere bei der Erkennung von Portscans [112]. Entgegen der klassischen Stichprobenanalyse gehen die aktuellen Datenfilteralgorithmen nicht zufällig vor sondern betrachten gezielt bestimmte Eigenschaften des Netzwerkverkehres. Stichprobenanalyse unter Berücksichtigung der IP-Flows reduziert deutlich den durch die Datenfilterung verursachten Fehler. In diesen Algorithmen werden Stichproben eher aus kurzen Verbindungen entnommen als aus langen [113]. Eine weitere Methode, Stichprobenanalysen zu verbessern, ist es sicher zu stellen, dass die maximale Anzahl an Stichproben entnommen wird, die das System bewältigen kann. Eine solche Methode wurde von Braun et al. vorgestellt [114].

Die Heavy Tail Eigenschaft von TCP-Verbindungen kann auch zur Datenfilterung verwendet werden. Heavy Tail bedeutet hier, dass nur die ersten Pakete einer Verbindung wichtig für die Sicherheitsanalyse sind. Ist etwa nach dem Aufbau einer SSH-Verbindung klar gestellt, dass der Nutzer die Berechtigung für diese Verbindung hat, können weitere Pakete in dieser Verbindung bedenkenlos durchgeleitet werden und beispielsweise eine eventuell umfangreiche Sicherungskopie muss nicht vollständig durch das IDS analysiert werden. Ein Vorfilter, der diese Vorsortierung ermöglicht, wurde auf einer FPGA-Plattform implementiert [115] und mit dem Bro IDS verbunden [116]. Ein ähnlicher Ansatz wurde auch mit einer FPGA-Erweiterung für Snort verfolgt [117].

Parallelisierung ist der zweite Ansatz, der weite Verbreitung gefunden hat. Vasiliadis et al. [118] etwa setzen mit ihrer auf dem Open Source Programm Snort aufgesetzten Entwicklung gnort auf die besonderen Fähigkeiten von Grafikprozessoren zur parallelen Datenverarbeitung [118]. Sie erreichten hierbei eine Verdopplung der Geschwindigkeit gegenüber der unmodifizierten Variante von Snort. Eine Verteilung der Last auf mehrere Rechner ist ebenfalls ein weit verbreiteter Ansatz. Der Bro Network Security Monitor etwa ist darauf ausgelegt, mit mehreren Rechnern im Netzwerk verwendet zu werden. Neben einer zentralen Instanz gibt es mehrere sogenannte Worker, welche die Analyse des Netzwerkverkehres übernehmen. Schwierigkeiten, die bei einem solchen System zu beachten sind, wurden von Vallentin et al. beschrieben [119] und für Bro gelöst [120]. Zum einen muss der Datenverkehr gleichmäßig auf alle Analyseknoten in einer Weise verteilt werden, welche die notwendige Kommunikation zwischen den Instanzen minimiert. Außerdem müssen die Knoten ihre Analyse auf unterster Ebene koordinieren und es musste validiert werden, dass gute Ergebnisse produziert werden. Es können auch mehrere Bro Instanzen auf einem Rechner laufen. So kann trotzdem die gesamte Rechenleistung des Rechners voll ausgenutzt werden, obwohl jeder Worker nur einen CPU-Kern verwendet. Die Netzwerkkarte verteilt dann die verschiedenen Verbindungsströme auf die verschiedenen Instanzen [120]. Wie ein Datenstrom auf verschiedene Instanzen aufgeteilt werden kann, wurde zum Beispiel von Schneider et al. beschrieben [121]. Cisco patentierte 2003 eine Methode, wie der Netzwerkverkehr von einem Load Balancer auf viele Instanzen aufgeteilt und schließlich von parallelen signaturbasierten IDS Instanzen analysiert werden kann [122].

FPGAs eignen sich sehr gut für parallele Datenverarbeitung, daher werden auch Ansätze verfolgt, Teile des IDS in diesen umzusetzen. Jiang et al. etwa haben Regeln des Snort IDS auf einen Xilinx-5 implementiert [123]. Auch die Paketklassifikation kann auf einem FPGA vorgenommen werden [124] oder auch die Analyse des Internetverkehrs [125]. Pontarelli et al. verfolgen den Ansatz, Teile der Regeln auf mehrere FPGAs aufzuteilen und so spezialisierte Hardwareimplementierungen zur Verfügung zu haben, die diesen Teil der Regeln besonders schnell umsetzen können [108].

Um etwa statistisch basierte NIDS Systeme zu parallelisieren, ist es nötig auch die statistischen Verfahren verteilen zu können, ein solches Modell wurde von Amann et al. entwickelt [126].

Parallelisierung kann nur dann eingesetzt werden, wenn verschiedene Aktionen nicht voneinander abhängig sind, also sequenziell abgearbeitet werden müssen. Die String Matching Algorithmen, die für die Erkennung von den IDS-Systemen eingesetzt werden, sind streng sequenziell und arbeiten in der naiven Implementierung jedes Byte nach dem anderen ab, da diese mit finiten endlichen Automaten arbeiten. Eine Möglichkeit, das Parsen zu beschleunigen, ist mehrere Bytes gleichzeitig zu parsen. Ohne den vorherigen Zustand der Maschine zu kennen ist es jedoch schlecht möglich, dies zu bewerkstelligen. Luchaup et al. [127] schlagen daher vor, einen Zustand zu schätzen und später gegebenenfalls die Ergebnisse zu korrigieren falls die Schätzung sich als falsch heraus

stellt. Die Schätzungen werden durch statistische Analysen gestützt um möglichst hohe Trefferzahlen zu bewerkstelligen [127].

4.4 Sicherheit in Software Defined Networking

Mit Software-Defined Networking erlangen Netzwerke stärkere Flexibilität, aber auch eine neue Angriffsfläche. Nicht nur Systeme im Netzwerk sondern auch das Netzwerk selbst kann nun attackiert werden wenn die Sicherheit vernachlässigt wird. Bei der Entwicklung von SDNs wurde die Sicherheit nur eingeschränkt betrachtet [128]. Jedoch gab es bereits früh Ansätze SDN für Security zu benutzen und viele Möglichkeiten die Netzwerksicherheit durch SDN zu verbessern wurden entwickelt. Die folgende Zusammenfassung basiert zu einem großen Teil auf den Arbeiten von Kreutz et al. [129] [130] und Scott-Hayward et al. [131], die jeweils eine ausführliche Sammlung von Maßnahmen zur Sicherung von SDN aber auch Sicherung von Netzwerken durch SDN zusammen getragen haben.

OpenFlow ist das Standardprotokoll für Software Defined Networking Anwendungen, weswegen auch im Speziellen die Sicherheit von OpenFlow betrachtet werden soll.

4.4.1 Sicherheitslücken und Gefahren von SDN

Der OpenFlow Standard 1.0 sah verpflichten für alle Verbindungen zwischen Controller und SDN Devices eine verschlüsselte Verbindung mit TLS vor. Dies bedeutete, dass für jedes Gerät ein Zertifikat anzulegen war und dies zu einem enormen Verwaltungsaufwand beim Einrichten eines Netzwerkes führte. Nachfolgende Versionen des Standards sahen keine Verpflichtung für TLS mehr vor. Dies führte dazu, dass viele Hardwareanbieter vollständig auf die Implementierung von Verschlüsselung verzichteten. Die einzige Möglichkeit, ein solches Netzwerk gegen Man-in-the-middle Attacken abzusichern besteht darin, ein physikalisch getrenntes Managementnetzwerk einzurichten. Einen Angriff auf eine solche Managementstruktur zu erkennen gestaltet sich dabei durch die fehlende Authentisierung als enorm schwierig. Auch werden Angriffe dadurch vereinfacht, dass zwischen dem Controller und den SDN Devices viel mehr Netzwerkverkehr besteht als bei einem herkömmlichen Netzwerk. Muss bei einem herkömmlichen Netzwerk gewartet werden, bis sich ein Administrator mit einem unsicheren Protokoll an einem der Geräte anmeldet, müssen in einem SDN Managementnetz nur einige wenige Pakete abgehört werden. Wurde das Managementnetz übernommen, ist das Produktivnetz nicht mehr absicherbar [132].

Viele Switches bieten einen Listener Mode, der es ermöglicht ohne Authentifizierung neue Regeln festzulegen und Daten auszulesen um die Fehlersuche zu vereinfachen. Ein vorkonfigurierter TCP Port kann dann hierfür genutzt werden. Wird dieser Modus versehentlich oder vorsätzlich nicht deaktiviert, kann das betroffene Switch problemlos übernommen werden [132].

Grundsätzlich ist ein übernommenes Switch im Netzwerk ein erhebliches Risiko für die Verfügbarkeit des Netzwerkes. Das Switch könnte selektiv Pakete verwerfen oder abbremsen, Pakete duplizieren oder Pakete umleiten und damit auch die Vertraulichkeit gefährden. Auch Denial of Service durch eine Flut gefälschter Pakete ist denkbar. Mögliche Lösungsansätze sind etwa Trust Management, bei dem unzuverlässige Switche Vertrauen verlieren und beim Routing nicht mehr berücksichtigt werden, oder der Einsatz anomaliebasierter Intrusion Detection Systeme [130].

Manche Controller überprüfen das Zertifikat der Switche nicht, wodurch es möglich ist, das Netzwerk auszuspähen. Auch werden die flow-tables oft nicht verifiziert. Das heißt ein fehlerhaft agierendes Switch kann auch bei eingeschalteten und gut konfiguriertem TLS zu unerwarteten Verhalten im Netzwerk bis hin zu einem kompletten Ausfall führen [132].

Die Einführung eines zentralen Controllers sorgt auch dafür, dass das Netzwerk einen zentralen Angriffspunkt etwa für DoS-Attacken bieten aber auch Sicherheitslücken im Controller gefährden das ganze Netzwerk [130]. Eine Möglichkeit diese Gefahr abzumildern besteht darin, mehrere redundante Controller im Netzwerk zu betreiben [132] oder regelmäßig die Controller auf einen vertrauenswürdigen Stand zurück zu setzen [130].

Gefälschte Verbindungen können dazu genutzt werden, einzelne Geräte im Netzwerk außer Betrieb zu setzen (Denial of Service, etwa durch Überlastung des TCAM oder der Controller Ressourcen). Authentifizierung kann das Risiko erheblich senken, sollte aber ein Gerät im Netzwerk übernommen worden sein, stellt auch dies keine Sicherheit her. Eine mögliche Lösung in diesem Fall ist ein anomaliebasiertes Intrusion Detection System einzusetzen [130].

Sollte die Sicherung durch TLS ausgehoben worden sein, etwa durch eine kompromittierte CA oder durch schlechte Konfiguration, ist es möglich DoS-Attacken etwa im Managementnetz zu fahren. Es wäre auch möglich ein Black Hole Netzwerk zu nutzen, um Daten an nicht Berechtigte weiter zu leiten [130]. Dieses Problem besteht auch bei der Kommunikation zwischen Management Applikationen und der Controller Software [130].

4.4.2 Verbesserung der Sicherheit von SDN

Ein Ansatz der zur Verbesserung der Sicherheit in Software Defined Networks führen soll zeigen Kreutz et al. in [130] auf. Sie nennen dabei verschiedene Methoden, wie ein solches System administriert werden kann und was bezüglich der bestehenden SDN Technologie verbessert werden muss:

- Replikation

  Durch redundante Controller und redundante Anwendungen kann ein einzelner Controller bei Ausfall – ob durch Angriff oder auch durch einen technischen Defekt – ersetzt werden. Die angreifende Partei muss dann mehrere Controller außer Gefecht setzen was ungleich schwerer ist.

- Vielfalt

  Der Einsatz verschiedener Controller ermöglicht es, die Gefahr, dass Controller durch Sicherheitslücken angreifbar sind, zu minimieren, da das Auftreten der selben Sicherheitslücke bei mehreren Geräten weniger wahrscheinlich ist.

- Selbstheilung

  Das Wiederherstellen eines vertrauenswürdigen Zustandes im Falle einer Kompromittierung oder auch proaktiv in regelmäßigen Abständen kann selbst eventuell noch unentdeckte Attacken beenden.

- Dynamische Gerätezuordnung

  Wenn ein Switch fest einem Controller zugeordnet wird, führt ein Ausfall des Controllers unweigerlich zu einem Ausfall dieses Teils des Netzwerks. Hier muss daher sichergestellt werden, dass dynamisch dem Switch ein anderer Controller zugeordnet werden kann und das Netzwerk so weiter aktiv bleiben kann. Mit genügend Redundanzen kann hier auch ein System erstellt werden, bei dem ein einzelner kompromittierter Controller keine Gefahr für die Verfügbarkeit des Netzwerks darstellt (byzantinische Fehlertoleranz).

- Vertrauen zwischen Geräten und Controllern

  Um auf der Steuerungsebene Vertrauen herzustellen ist es essentiell Vertrauen zwischen den Geräten und Controllern herzustellen. Geräte und Controller sollten dynamisch miteinander kooperieren können, dies sollte aber immer mit der selben Sicherheit und Zuverlässigkeit funktionieren. Eine Möglichkeit wäre hier etwa eine White List der bekannten, vertrauenswürdigen Geräten zu erstellen, welche die Controller vorhalten. Dies kann allerdings die Flexibilität des Netzwerks sehr einschränken.

- Vertrauen zwischen Anwendungen und Controller Software

  Applikationen verändern ihr verhalten durch Bugs, Angriffe oder auch durch Alterung, so dass ein dynamisches Vertrauenssystem wichtig ist. Dadurch können Anwendungen oder auch die Controller Software dynamisch Vertrauen verlieren aber auch zurückgewinnen, wenn zum Beispiel ein DoS-Angriff beendet oder ein Bug gepatcht wurde.

- Security Domänen

  Das Abgrenzen von Bereichen in einem System je nach Zugriffslevel, etwa in Form von Sandboxes ist ein weit verbreitetes Konzept in der Informatik. Auch in einer SDN Control Plattform sind diese Möglichkeiten einsetzbar. Abgegrenzte Zuständigkeiten einzelner Geräte, die über klar abgegrenzte Interfaces die möglichst eingeschränkt sind kommunizieren, kann zur Sicherheit beitragen. Die Dynamik des Systems weiter beizubehalten stellt dabei allerdings eine große Herausforderung dar.

- Security Bausteine

  Security Bausteine wie die Trusted Computing Bases können verwendet werden, um Vertraulichkeit sicher zu stellen. Damit kann selbst bei einem kompromittierten System sicher gestellt werden, dass die Vertraulichkeit gewährleistet bleibt.

- Software Updates

  Schnelle, zuverlässige Software Updates sind in jedem Computersystem unerlässlich um die Sicherheit zu gewährleisten. Eine Kontrollplattform sollte daher vorhanden sein, die Updates an die Systeme verteilt.

Es gibt Entwicklungen, die sich dem Ziel verschrieben haben SDNs sicherer zu machen und diese Sicherheit dabei auch möglichst einfach konfigurierbar zu machen. Zum Einen sei FortNOX zu nennen, eine Erweiterung für den NOX Controller. Diese Entwicklung ermöglicht es, OpenFlow Regeln in Echtzeit zu analysieren [133]. Eine weitere Entwicklung ist FRESCO, ein OpenFlow Security Framework auf Basis von POX [134]. FRESCO soll die Möglichkeit bieten, mit Hilfe von SDN und IDS Systemen, wie zum Beispiel Snort, Attacken im Netzwerk abzufangen und den Attacken entgegen zu wirken.

4.4.3 Verbesserung der Sicherheit durch SDN

Es gibt verschiedene Bereiche bei denen SDN hilfreich sein kann bei der Absicherung von Netzwerken. Zum Beispiel beim Durchsetzen von Richtlinien wie Zugangskontrolle oder Firewallsystemen [135] [136] [137] [138] [139]. Auch können sie eingesetzt werden um DoS-Attacken zu Erkennen und den Schaden zu begrenzen [140] [139]. Durch Random Host Mutation, das heißt ständig wechselnde IP-Adresse der Server, lässt sich die Annahme der Angreifer, dass Server eine feste IP-Adresse haben, aushebeln. Dies lässt sich mit SDN bewerkstelligen [141]. Auch Intrusion Detection Systeme lassen sich mit SDN erweitern. Eine Möglichkeit ist etwa, verdächtigen Netzwerkverkehr umzuleiten um vom IDS betrachtet zu werden, etwa durch Deep Packet Inspection [142]. Eine andere Möglichkeit ist auch die Anomalieerkennung im Netzwerkverkehr durch SDN zu bewerkstelligen [143] [140] [144]. Mit FlowNAC wurde eine Möglichkeit vorgestellt, wie flussbasierte Netzwerkzugangsberechtigungen mit SDN funktionieren kann [145].

4.5 Zusammenfassung

Alles in allem zeigen sich viele Verbesserungsmöglichkeiten bezüglich der Geschwindigkeit von IDS und IPS Systemen. Den bisherigen Ansätzen ist gemein, dass die Geschwindigkeit dieser Ansätze jedoch nicht ausreicht die Anforderungen der Zukunft zu erfüllen. Eine Verbindung der verschiedenen Ansätze wird bisher nicht verfolgt. Bei SDN zeigt sich, dass Sicherheit nicht der zentrale Fokus der Entwickler war und hier Nachholbedarf in der Konzeption besteht. Es bieten sich aber auch viele Einsatzmöglichkeiten von SDNs zur Absicherung von Netzwerken, die bisher aber nur konzeptionell erforscht wurden.

5 Zusammenfassung

Dieser Bericht stellt erste Ergebnisse des bwNET100G+ Projekts vor. Auf Basis einer umfassenden Literaturrecherche wurde der Stand der Technik bei Software-defined Networking / OpenFlow, Transportprotokollen und Netzwerksicherheit im Hinblick auf Netzwerke mit Datenraten von 100G+ betrachtet.

Wie wir darstellen konnten, bietet SDN vielversprechende Möglichkeiten zur Vereinfachung und Flexibilisierung des Netzwerkbetriebs in Hochgeschwindigkeitsnetzen. Im weiteren Verlauf des Projekts werden wir untersuchen, inwieweit sich derartige Ansätze tatsächlich heute schon produktiv nutzen lassen und wo ggf. die besonderen Probleme liegen.

Weiterhin werden wir untersuchen, wie sich Hochgeschwindigkeitsnetze effizient durch Transportschichtverbindungen ausnutzen lassen, sodass hohe Übertragungsraten bei niedrigen Latenzen erreicht werden können.

Schließlich werden wir im Bereich der Sicherheitstechnologien die durch SDN entstehenden Probleme analysieren, aber auch durch SDN ermöglichte Sicherheitsmechanismen untersuchen. Weiterhin werden wir analysieren, mit welchen Ansätzen sich Firewalls und Intrusion Detection Systeme für Geschwindigkeiten von 100 Gbit/s und darüber nutzbar machen lassen.

Diese Ergebnisse werden in zukünftigen Projektberichten veröffentlicht.

Literatur

[1] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014. [Online]. Available: http://www.mdpi.com/1999-5903/6/2/302

[2] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," http://arxiv.org/abs/1406.0440, 2014.

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *In Proceedings of the ACM SIGPLAN International Conference on Functional Programming*. Tokyo, Japan, Sep. 2011.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software-Defined Networks," in *In Proceedings of the*. Lombard, USA, 2013, pp. 1–14.

[5] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High-Level Reactive Network Control." Helsinki, Finland, Oct. 2012, pp. 43–48.

[6] F. M. Facca, E. Salvadori, H. Karl, D. R. Lopez, P. A. A. Gutierrez, D. Kostic, and R. Riggio, "NetIDE: First Steps towards an Integrated Development Environment for Portable Network Apps." Berlin, Germany, 2013, pp. 105–110.

[7] Open Networking Foundation members, "OpenFlow Switch Specification," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow, The Open Networking Foundation, oct 2013.

[8] Cisco and/or its affiliates, "OpFlex: An Open Policy Protocol," Cisco Whitepaper, Cisco, Whitepaper, 2014.

[9] M. Smith, R. Adams, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "OpFlex Control Protocol," Internet-Draft (Informational), Internet Engineering Task Force, Nov. 2014.

[10] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.

[11] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020 (Proposed Standard), Internet Engineering Task Force, Oct. 2010.

[12] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification," RFC 5810 (Proposed Standard), Internet Engineering Task Force, Mar. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5810.txt

[13] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," RFC 3746 (Informational), Internet Engineering Task Force, Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3746.txt

[14] S. Hares, "Analysis of Comparisons between OpenFlow and ForCES," Internet Draft (Informational), Jul. 2012. [Online]. Available: http://tools.ietf.org/html/draft-hares-forces-vs-openflow-00

[15] E. Haleplidis, S. Denazis, O. Koufopavlou, J. Halpern, and J. H. Salim, "Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface." Darmstadt, Germany, Oct. 2012, pp. 91–96.

[16] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter Architecture," in *In Proceedings of the.* San Diego, USA, Nov. 2004.

[17] Amin Tootoonchia et al., "NOX," http://www.noxrepo.org/nox/versions-downloads/, 2012.

[18] James McCauley et al., "POX," http://www.noxrepo.org/pox/versionsdownloads/, 2014.

[19] The Linux Foundation, "Opendaylight," http://www.opendaylight.org/, 2014.

[20] Project Floodlight, "Floolight," http://www.projectfloodlight.org/floodlight/, 2014.

[21] Ryu SDN Framework Community, "Ryu," http://osrg.github.io/ryu/, 2014.

[22] D. Erickson, "The Beacon OpenFlow Controller," in *In Proceedings of the.* Hong Kong, China, 2013, pp. 13–18.

[23] Z. Cai, A. L. Cox, and T. S. Eugene Ng, "Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane," Rice University, Tech. Rep., 2011.

[24] "NodeFLow OpenFlow Controller," https://github.com/gaberger/NodeFLow, (accessed on November 2013). [Online]. Available: https://github.com/gaberger/NodeFLow

[25] "Trema: Full-Stack OpenFlow Framework in Ruby and C," http://trema.github.io/trema/, (accessed on November 2013). [Online]. Available: http://trema.github.io/trema/

[26] B. Lantz, B. O'Connor, and C. Burkard, "Mininet," http://mininet.org/, 2014.

[27] M. Schmidt, F. Heimgaertner, and M. Menth, "A virtualized testbed with physical outlets for hands-on computer networking education," in *Proceedings of the 15th Annual Conference on Information Technology Education*, ser. SIGITE '14. New York, NY, USA: ACM, 2014, pp. 3–8. [Online]. Available: http://doi.acm.org/10.1145/2656450.2656451

[28] J. Postel, "Internet Protocol," RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[29] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Internet Engineering Task Force, Dec. 1998, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[30] J. Postel, "Transmission Control Protocol," RFC 793 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[31] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, Internet Engineering Task Force, Jan. 1984. [Online]. Available: http://www.ietf.org/rfc/rfc896.txt

[32] V. Jacobson, "Congestion avoidance and control," in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM '88. New York, NY, USA: ACM, 1988, pp. 314–329. [Online]. Available: http://doi.acm.org/10.1145/52324.52356

[33] R. Jain, "A timeout-based congestion control scheme for window flow-controlled networks," *Selected Areas in Communications, IEEE Journal on*, vol. 4, no. 7, pp. 1162–1167, Oct 1986.

[34] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 553–574, Apr 1980.

[35] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309 (Informational), Internet Engineering Task Force, Apr. 1998, updated by RFC 7141. [Online]. Available: http://www.ietf.org/rfc/rfc2309.txt

[36] S. Floyd, "Congestion Control Principles," RFC 2914 (Best Current Practice), Internet Engineering Task Force, Sep. 2000, updated by RFC 7141. [Online]. Available: http://www.ietf.org/rfc/rfc2914.txt

[37] S. Floyd and M. Allman, "Specifying New Congestion Control Algorithms," RFC 5033 (Best Current Practice), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5033.txt

[38] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end Arguments in System Design," *ACM Transactions in Computer Systems*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://doi.acm.org/10.1145/357401.357402

[39] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001, updated by RFCs 4301, 6040. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[40] G. Fairhurst and M. Welzl, "The Benefits of using Explicit Congestion Notification (ECN)," Internet-Draft draft-ietf-aqm-ecn-benefits-04 (Work in progress), Internet Engineering Task Force, May 2015, https://tools.ietf.org/html/draft-ietf-aqm-ecn-benefits.

[41] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr. 1999, obsoleted by RFC 5681, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[42] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[43] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Internet Engineering Task Force, Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[44] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-tahoe bsd tcp congestion control algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 9–16, Apr. 1992. [Online]. Available: http://doi.acm.org/10.1145/141800.141801

[45] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: ACM, 1994, pp. 24–35. [Online]. Available: http://doi.acm.org/10.1145/190314.190317

[46] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," RFC 5166 (Informational), Internet Engineering Task Force, Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5166.txt

[47] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1400097.1400105

[48] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "Tcp congestion avoidance algorithm identification," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 310–321.

[49] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582 (Experimental), Internet Engineering Task Force, Apr. 1999, obsoleted by RFC 3782. [Online]. Available: http://www.ietf.org/rfc/rfc2582.txt

[50] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for tcp," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 3, pp. 304–342, Third 2010.

[51] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," RFC 3742 (Experimental), Internet Engineering Task Force, Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3742.txt

[52] T. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/956981.956989

[53] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (bic) for fast long-distance networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, March 2004, pp. 2514–2524 vol.4.

[54] D. Leith and R. Shorten, "H-tcp: Tcp for high-speed and long-distance networks," in *Proceedings of PFLDnet*, vol. 2004, 2004.

[55] C. Jin, D. Wei, S. Low, J. Bunn, H. Choe, J. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W. chun Feng, "Fast tcp: from theory to experiments," *Network, IEEE*, vol. 19, no. 1, pp. 4–11, Jan 2005.

[56] J. Sing and B. Soh, "Tcp new vegas: Improving the performance of tcp vegas over high latency links," in *Network Computing and Applications, Fourth IEEE International Symposium on*, July 2005, pp. 73–82.

[57] C. Jin, S. Low, and X. Wei, "Method and apparatus for network congestion control," Jul. 5 2011, uS Patent 7,974,195. [Online]. Available: https://www.google.com/patents/US7974195

[58] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between tcp reno and vegas for deployment of tcp vegas to the internet," in *Network Protocols, 2000. Proceedings. 2000 International Conference on*, 2000, pp. 177–186.

[59] C. P. Fu and S. Liew, "Tcp veno: Tcp enhancement for transmission over wireless access networks," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 2, pp. 216–228, Feb 2003.

[60] R. King, R. Baraniuk, and R. Riedi, "Tcp-africa: an adaptive and fair rapid increase rule for scalable tcp," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, March 2005, pp. 1838–1848 vol. 3.

[61] K. Tan and Q. Zhang, "Compound transmission control protocol," Aug. 18 2009, uS Patent 7,577,097. [Online]. Available: https://www.google.com/patents/US7577097

[62] S. Liu, T. Başar, and R. Srikant, "Tcp-illinois: A loss and delay-based congestion control algorithm for high-speed networks," in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, ser. valuetools '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1190095.1190166

[63] A. Baiocchi, A. P. Castellani, and F. Vacirca, "Yeah-tcp: yet another highspeed tcp," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42.

[64] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "Tcp westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01. New York, NY, USA: ACM, 2001, pp. 287–297. [Online]. Available: http://doi.acm.org/10.1145/381677.381704

[65] A. Venkataramani, R. Kokku, and M. Dahlin, "Tcp nice: A mechanism for background transfers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844159

[66] A. Kuzmanovic and E. Knightly, "Tcp-lp: a distributed algorithm for low priority data transfer," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, March 2003, pp. 1691–1701 vol.3.

[67] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: The new bittorrent congestion control protocol," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, Aug 2010, pp. 1–6.

[68] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228338

[69] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356 (Experimental), Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6356.txt

[70] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal: Performance issues and a possible solution," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413178

[71] Q. Peng, A. Walid, J. Hwang, and S. Low, "Multipath tcp: Analysis, design, and implementation," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[72] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[73] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486020

[74] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: ACM, 2002, pp. 89–102. [Online]. Available: http://doi.acm.org/10.1145/633025.633035

[75] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe, "Open Research Issues in Internet Congestion Control," RFC 6077 (Informational), Internet Engineering Task Force, Feb. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6077.txt

[76] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: The virtue of gentle aggression," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 159–170, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2486014

[77] K. Ingham and S. Forrest, *A History and Survey of Network Firewalls*, ser. The University of New Mexico Computer Science Department Technical Report 2002-37., 2002.

[78] Bluewolf Studie: 44% der befragten Unternehmen erlauben keinen Zugriff auf Social Media während der Arbeitszeit. [Online]. Available: http://www.bluewolf.com/assetdownload/Infographic_HowSocialReadyAreYou.pdf

[79] M. H. Alsafasfeh and A. I. Alshbatat, "Configuring snort as a firewall on windows 7 environment," *JUSPN*, pp. 73–77, 2011.

[80] A. Lazarevic, V. Kumar, and J. Srivastava, "Intrusion detection: A survey," in *Managing Cyber Threats*, 2005, pp. 19–78.

[81] Cisco Data Sheet on ASA 5500 Series. [Online]. Available: http://www.cisco.com/c/dam/en/us/products/collateral/security/asa-5500-x-series-next-generation-firewalls/data_sheet_c78_459036.pdf

[82] Hp Data Sheet on Firewall Series. [Online]. Available: http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA3-7159ENW.pdf

[83] S. Axelsson, "Intrusion detection systems : A survey and taxonomy," in *Computer Engineering*, 2000, pp. 1–27. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.3043&rep=rep1&type=pdf

[84] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1, pp. 18–28, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404808000692

[85] D. Denning and P. Neumann, "Requirements and model for IDES – a real-time intrusion detection system," Computer Science Laboratory, SRI International, Tech. Rep. 83F83-01-00, 1985.

[86] N. Ye, S. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on Computers*, vol. 51, no. 7, pp. 810–820, 2002.

[87] C. Callegari, S. Giordano, and M. Pagano, "New statistical approaches for anomaly detection," *Security Comm. Networks*, vol. 2, no. 6, pp. 611–634, 2009. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/sec.104/abstract

[88] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '05. ACM, 2005, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/1080091.1080118

[89] K. Wang, J. Parekh, and S. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection*, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11856214_12

[90] K. Rieck and P. Laskov, "Detecting unknown network attacks using language models," *Detection of Intrusions and Malware & Vulnerability Assessment*, pp. 74–90, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11790754_5

[91] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A close look on n-grams in intrusion detection: Anomaly detection vs. classification," in *AISec '13 Proceedings of the 2013 ACM workshop on Artificial intelligence and security.* ACM New York, NY, USA ©2013, 2013, pp. 67–76.

[92] J. Xu and C. R. Shelton, "Intrusion detection using continuous time bayesian networks," 2014. [Online]. Available: http://arxiv.org/abs/1401.3851

[93] N. Ye, "A markov chain model of temporal behavior for anomaly detection," in *In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, 2000, pp. 171–174.

[94] D. yan Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, pp. 229–243, 2003.

[95] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, G. Vigna, C. Kruegel, and

E. Jonsson, Eds. Springer Berlin Heidelberg, 2003, no. 2820, pp. 220–237. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-45248-5_13

[96] J. R. Miikkulainen, M.-J. Lin, and Risto, "Intrusion detection with neural networks," vol. 10, 1998. [Online]. Available: http://nn.cs.utexas.edu/?ryan:nips97

[97] S. M. Bridges, R. B. Vaughn, A. Professor, and A. Professor, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in *Proceedings of the National Information Systems Security Conference (NISSC)*, 2000, pp. 16–19.

[98] J. E. Dickerson and J. A. Dickerson, "Fuzzy network profiling for intrusion detection," in *19th International Conference of the North American Fuzzy Information Processing Society NAFIPS*, 2000, pp. 301–306.

[99] W. Li, "Using genetic algorithm for network intrusion detection," in *In Proceedings of the United States Department of Energy Cyber Security Group Training Conference*, 2004, pp. 24–27.

[100] M. S. Hoque, M. A. Mukit, and M. A. N. Bikas, "An implementation of intrusion detection system using genetic algorithm," *International Journal of Network Security & Its Applications*, vol. 4, no. 2, pp. 109–120, 2012. [Online]. Available: http://arxiv.org/abs/1204.1336

[101] P. Manandhar and Z. Aung, "Towards practical anomaly-based intrusion detection by outlier mining on TCP packets," in *Database and Expert Systems Applications*, ser. Lecture Notes in Computer Science, H. Decker, L. Lhotská, S. Link, M. Spies, and R. R. Wagner, Eds. Springer International Publishing, 2014, no. 8645, pp. 164–173. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-10085-2_14

[102] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. USENIX Association, 1999, pp. 229–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=1039834.1039864

[103] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 1999, pp. 2435–2463.

[104] The Suricata Open Source IDS/IPS/NSM Engine. [Online]. Available: http://suricata-ids.org

[105] D. Chaboya, R. Raines, R. Baldwin, and B. Mullins, "Network intrusion detection: Automated and manual methods prone to attack and evasion," in *IEEE Security & Privacy*, vol. 4, no. 6, 2006, pp. 36–43.

[106] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Superbus von Underduk. Polymorphic shellcode engine using spectrum analysis. [Online]. Available: http://phrack.org/issues/61/9.html

[107] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 4, pp. 1011–1020, 2012.

[108] S. Pontarelli, G. Bianchi, and S. Teofili, "Traffic-aware design of a high-speed FPGA network intrusion detection system," in *IEEE Transactions On Computers*, vol. 62, 2013, pp. 2322–2334.

[109] S. Campbell and J. Lee, "Intrusion detection at 100G," in *State of the Practice Reports*, ser. SC '11. ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2063348.2063367

[110] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. ACM, 2006, pp. 159–164. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177101

[111] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. ACM, 2006, pp. 165–176. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177102

[112] J. Mai, A. Sridharan, C.-N. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2285–2298, Dec 2006.

[113] Q. Pan, H. Yong-feng, and Z. Pei-feng, "Reduction of traffic sampling impact on anomaly detection," in *2012 7th International Conference on Computer Science Education (ICCSE)*, 2012, pp. 438–443.

[114] L. Braun, C. Diekmann, N. Kammenhuber, and G. Carle, "Adaptive load-aware sampling for network monitoring on multicore commodity hardware," in *IFIP Networking Conference, 2013*, 2013, pp. 1–9.

[115] N. Weaver, "The shunt: An FPGA-based accelerator for network intrusion prevention," in *In FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international*. ACM Press, 2007, pp. 199–206.

[116] J. M. Gonzalez, V. Paxson, and N. Weaver, "Shunting: A hardware/software architecture for flexible, high-performance network intrusion prevention," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. ACM, 2007, pp. 139–149. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315264

[117] F. Engelmann, T. Lukaseder, B. Erb, R. van der Heijden, and F. Kargl, "Dynamic packet-filtering in high-speed networks using NetFPGAs," in *Third International Conference on Future Generation Communication Technology (FGCT)*, 2014.

[118] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds. Springer Berlin Heidelberg, 2008, no. 5230, pp. 116–134. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-87403-4_7

[119] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware," in *Proceedings of the 10th international conference on Recent advances in intrusion detection, series*, 2007, pp. 107–126.

[120] V. Paxson, R. Sommer, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *IEEE Sarnoff Symposium*, 2007, pp. 1–7.

[121] F. Schneider, J. Wallerich, and A. Feldmann, "Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware," in *Passive and Active Network Measurement*, ser. Lecture Notes in Computer Science, S. Uhlig, K. Papagiannaki, and O. Bonaventure, Eds. Springer Berlin Heidelberg, 2007, vol. 4427, pp. 207–217. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71617-4_21

[122] S. Shanklin and G. Lathem, "Parallel intrusion detection sensors with load balancing for high speed networks," 2003, US Patent 6,578,147. [Online]. Available: http://www.google.com/patents/US6578147

[123] W. Jiang and V. K. Prasanna, "Field-split parallel architecture for high performance multi-match packet classification using FPGAs," in *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '09. ACM, 2009, pp. 188–196. [Online]. Available: http://doi.acm.org/10.1145/1583991.1584044

[124] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, ser. FPGA '05. ACM, 2005, pp. 238–245. [Online]. Available: http://doi.acm.org/10.1145/1046192.1046223

[125] F. Khan, M. Gokhale, and C.-N. Chuah, "FPGA based network traffic analysis using traffic dispersion patterns," in *2010 International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 519–524.

[126] J. Amann, S. Hall, and R. Sommer, "Count me in: Viable distributed summary statistics for securing high-speed networks," in *Research in Attacks, Intrusions and Defenses*, ser. Lecture Notes in Computer Science, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Springer International Publishing, 2014, no. 8688, pp. 320–340. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-11379-1_16

[127] D. Luchaup, R. Smith, C. Estan, and S. Jha, "Multi-byte regular expression matching with speculation," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds. Springer Berlin Heidelberg, 2009, no. 5758, pp. 284–303. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-04342-0_15

[128] R. Hinden. (2014) SDN & security: Why take over the hosts when you can take over the network. [Online]. Available: http://www.rsaconference.com/events/us14/agenda/sessions/1021/sdn-security-why-take-over-the-hosts-when-you-can#sthash.jW4dxGYQ.dpuf

[129] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *CoRR*, vol. abs/1406.0440, 2014. [Online]. Available: http://arxiv.org/abs/1406.0440

[130] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. ACM, 2013, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491199

[131] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1–7.

[132] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. ACM, 2013, pp. 151–152. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491222

[133] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 121–126. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342466

[134] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *ISOC Network and Distributed System Security Symposium*, 2013.

[135] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. USENIX Association, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267336.1267346

[136] H. Jamjoom, D. Williams, and U. Sharma, "Don't call them middleboxes, call them middlepipes," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. ACM, 2014, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620760

[137] K. Wang, Y. Qi, B. Yang, Y. Xue, and J. Li, "LiveSec: Towards effective security management in large-scale production networks," in *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2012, pp. 451–460.

[138] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," in *19th IEEE International Conference on Network Protocols (ICNP)*, 2011, pp. 7–12.

[139] G. Stabler, A. Rosen, S. Goasguen, and K.-C. Wang, "Elastic IP and security groups implementation using OpenFlow," in *Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing*, ser. VTDC '12. ACM, 2012, pp. 53–60. [Online]. Available: http://doi.acm.org/10.1145/2287056.2287069

[140] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *2010 IEEE 35th Conference on Local Computer Networks (LCN)*, 2010, pp. 408–415.

[141] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proceedings of the first workshop on hot topics in software defined networking*, 2012.

[142] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *20th IEEE International Conference on Network Protocols (ICNP)*, 2012, pp. 1–6.

[143] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds. Springer Berlin Heidelberg, 2011, no. 6961, pp. 161–180. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-23644-0_9

[144] K. Giotis, G. Androulidakis, and V. Maglaris, "Leveraging SDN for efficient anomaly detection and mitigation on legacy networks," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*, 2014, pp. 85–90.

[145] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*, 2014, pp. 79–84.