Software Defined Networking mit OpenFlow

Josias Montag
Betreuer: Daniel Raumer, Florian Wohlfart
Hauptseminar: Innovative Internettechnologien und Mobilkommunikation
WS 2012/2013
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Mail: montag@in.tum.de

KURZFASSUNG

Diese Ausarbeitung behandelt das Software Defined Networking (SDN) mit OpenFlow. Das SDN lagert die Kontrollinstanz des Netzwerkes aus und ermöglicht die Virtualisierung auf Netzwerkebene, welche langfristig durch sich veränderte Anforderungen an die Netzwerk Infrastrukturen nötig wird und Einschränkungen von klassischen Netzwerken entgegenwirkt. Technisch gesehen handelt es sich bei OpenFlow um ein Layer 2 Protokoll. Für die Nutzung von OpenFlow ist ein vom Hersteller angepasster Switch nötig, welche inzwischen zahlreiche Hersteller anbieten. Praktische Anwendungen, die die Vorteile des SDNs ausnutzen, sind Experimente in Produktivnetzwerken, Load Balancing, Clouds, transparente Dienste oder große Datencenter. Für die Programmierung solche virtueller Netzwerke bietet sich die dafür spezialisierte Programmiersprache Frenetic an. Obwohl das SDN und OpenFlow viele praktische Anwendungszwecke bietet, kommt es bisher nur in sehr speziellen Szenarien zum Einsatz.

Schlüsselworte

Software Defined Networking, OpenFlow, Netzwerk Virtualisierung, Frenetic

1. EINLEITUNG UND DEFINITION

„Virtualisierung“ und „Cloud“ sind momentan die Schlagwörter der IT-Industrie und haben sich von einem Trend zu einem nicht mehr wegzudenkenden Standard im Bereich von IT-Infrastrukturen entwickelt. [10]

Unter Virtualisierung versteht man „Technologien, die entworfen wurden, um eine Ebene der Abstraktion zwischen den Hardware Systemen und der darauf laufenden Software zu bereitzustellen“ [20]. Im Bereich der Server Virtualisierung bedeutet dies, dass die physisch vorhandenen Ressourcen der Hardware in verschiedene virtuelle Pools aufgeteilt oder zusammengefasst werden. Dadurch wird eine Abstraktion erreicht, die es ermöglicht die einzelnen virtuellen Maschinen unabhängig von der eigentlichen Hardware zu betreiben, zu duplizieren oder zu verschieben.

Wendet man die obige Definition von Virtualisierung auch auf Netzwerk Infrastrukturen an, ergeben dich die Grundzüge des „Software Defined Networking“ (SDN). Im Deutschen könnte SDN am ehesten als „durch Software gesteuerter Netzwerkbetrieb“ übersetzt werden. Unter Software Defined Networking versteht man das Entkoppeln der Kontroll-Ebene von der Daten-Ebene in der Netzwerk Architektur. Durch das Verschieben der Kontrollinstanz wird eine Abstraktion und Virtualisierung des Netzwerkes erreicht [12, S. 7].

Ziel der SDN Architektur ist es, unabhängig vom Hersteller der einzelnen Netzwerkgeräte, Netzwerk Ressourcen mittels dynamischer und automatisierter Software einzurichten, zu steuern, abzusichern und zu optimieren [12, S. 8].

Diese Ausarbeitung wird zunächst die Möglichkeiten des SDN gegenüber klassischen Netzwerken sowie die Funktionen, Ziele und Absichten von OpenFlow erklärt. Dabei werden unterschiedliche praktische Anwendungsfälle für die Nutzung SDN Architekturen dargelegt. Außerdem wird die Programmierung von SDN-Controllern mit der Programmiersprache Frenetic kurz aufgezeigt. Abschließend setzt sich diese Ausarbeitung kritisch mit den Limits und der Zukunft von OpenFlow auseinander.

2. EINSCHRÄNKUNGEN VON KLASSISCHEN NETZWERKEN

Die Anforderungen an die Netzwerk Infrastrukturen haben sich, besonders in den letzten Jahren, stark verändert. Gründe dafür sind unter anderem neue „On-Demand“ Geschäftsmodelle, die zunehmende Anzahl von Geräten, wie zum Beispiel durch Smartphones, und das Verlangen nach immer größeren Bandbreiten. Klassische Netzwerkstrukutren, dessen grundsätzlichen Strukturen noch aus den frühen 90er Jahren stammen, sind dafür teilweise nicht ausgelegt worden.

Verändertes Nutzerverhalten und größere Bandbreiten: Klassische Netzwerke sind starr und wurden für Client-Server Verbindungen konzipiert, die für eine bestimmte Zeit aufgebaut werden und anschließend wieder getrennt werden. Diese Architektur passt teilweise nicht mehr zu modernen Anwendungen, die auf viele verschieden Server gleichzeitig zugreifen und beispielsweise „Push“-Dienste nutzen. Der Trend geht zu „Always-On“ und der Nutzer möchte von überall auf der Welt jederzeit auf seine Daten zugreifen können. Um diese Bedürfnisse zu befriedigen, muss das Netzwerk flexibel werden und fähig sein sich in Echtzeit an das Nutzerverhalten anzupassen: Beispielsweise müssen die immer größer werdenden Bandbreiten dynamisch Anwendungen oder Nutzern zugewiesen werden können oder das Netzwerk muss selbstständig Engpässe erkennen

doi: 10.2312/NET-2013-02-1_09

und darauf reagieren können. [12, S.3]

Komplexität und Verwaltbarkeit: Die zunehmende Anzahl der Hosts in den, meist historisch gewachsenen, Netzwerken führt zu einer hohen Komplexität bei der Verwaltung. Durch die zunehmende Server-Virtualisierung vervielfacht sich zusätzlich die Anzahl der virtuellen Hosts. Diese Hosts in die unterschiedlichen Konfigurationsinterfaces der verschiedenen Switches und Router für beispielsweise ACLs (Access Control Lists), VLANs (Virtuelle Subnetze) oder QoS (Quality of Service) einzutragen ist ein hoher Verwaltungsaufwand. Diese Aufgaben können durch SDN automatisiert werden und damit die OPEX Kosten (Betriebsausgaben) gesenkt werden. [12, S.4,5]

Abhängigkeit von den Geräteherstellern: Hersteller von Switches oder Routern bieten oft zentrale Verwaltungsmöglichkeiten für ihre Geräte an. Diese funktionieren jedoch nicht herstellerübergreifend, wodurch man von einem bestimmten Hersteller, dessen Support und dessen Produktzyklen abhängig ist. Setzt man auf einen einheitlichen Standard wie OpenFlow ist man unabhängig vom Hersteller und kann beliebige Geräte kombinieren. Zudem können erhebliche Kosten für die Erneuerung von Switches eingespart weden, da die eigene Software auf auf den Switches gegebenenfalls aktualisiert oder angepasst weden kann, ohne dass neue Hardware gekauft werden muss. [12, S.6]

Skalierungs- und Anpassungsmöglichkeiten: Durch „On-Demand“ Geschäftsmodelle muss sich das Netzwerk ebenfalls „On-Demand“ an die Anforderungen anpassen können. Das Hinzufügen und die Konfiguration von neuen Netzwerkgeräten kann durch SDN automatisiert werden und daher in viel kürzerer Zeit erfolgen. [12, S.6]

3. SOFTWARE DEFINED NETWORKING MIT OPENFLOW

OpenFlow ist ein Open Source Projekt hat sich als die Standardschnittstelle für SDN-Architekturen etabliert. Die ursprüngliche Spezifikation von OpenFlow wurde 2008 an der Stanford Universität verfasst. Seit 2011 wird OpenFlow von der Open Networking Foundation (ONF) betreut. Die ONF wurde von der Deutschen Telekom, Facebook, Google, Microsoft, Verizon, und Yahoo! gegründet und zählt inzwischen über 80 Mitglieder. [1, 9]

Abbildung 1: OpenFlow Logo [1]

3.1 OpenFlow Unterstützung

Grundvoraussetzung für die Nutzung von OpenFlow sind mit OpenFlow kompatible Switches bzw. Router. Dabei unterscheidet man zwischen dedizierten OpenFlow Switches („Type 0“) und OpenFlow-Enabeld Switches („Type 1“). Die Type-0 Switches funktionieren nur im OpenFlow-Modus, das heißt sie bieten sonst keine klassischen Switch Funktionalitäten. Bei den OpenFlow-Enabeld Switches dahingegen ist OpenFlow eine Zusatzfunktion. Der Hersteller muss das vorhandene geschlossene System nicht offenlegen und OpenFlow kann trotzdem auf die Funktionalitäten des Switches zurückgreifen. (Kapitel 3.2) [16, S. 71]

Alle größeren Hersteller bieten solche Geräte bzw. passende Firmware-Updates an: IBM galt als Vorreiter und bot als einer der Ersten OpenFlow enabled Switche und sogar OpenFlow Controller an [7]. HP ist Mitglied der Open Networking Foundation und bietet derzeit 16 OpenFlow fähige Switche an [4]. Cisco setzt auf die eigene Lösung „Cisco Open Network Environment“, einem Framework mit verschiedenen APIs zur Steuerung von Cisco-Geräten, das jedoch ebenfalls eine OpenFlow Schnittstelle anbietet [2].

Alternativ besteht die Möglichkeit OpenFlow auf Software Routern zu betreiben, wofür Images für beispielsweise Debian und sogar ein Modul für die Heim-Router Software DD-WRT und OpenWRT zur Verfügung stehen [6].

3.2 Funktionsweise von OpenFlow

Bei OpenFlow handelt es sich um ein Layer 2 Protokoll mit der grundsätzlichen Zielsetzung, die Steuerung des Netzwerkes von den einzelnen Netzwerkgeräten von unterschiedlichen Herstellern in einen zentralisierten Software Controller zu verschieben. Abbildung 2 zeigt diese Architektur:

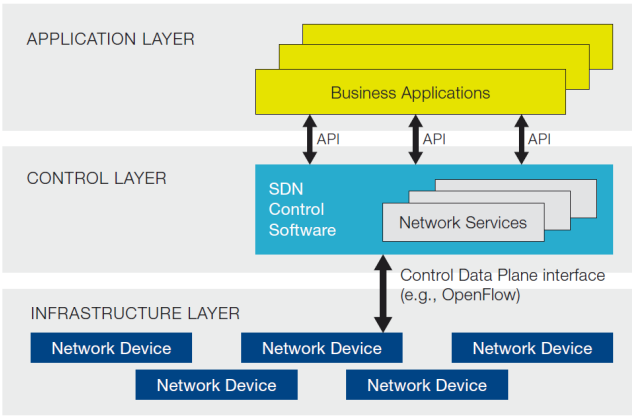

Abbildung 2: SDN Architektur [12, S. 7]

Switche und Router im Netzwerk (die Infrastructure Layer) leiten nur noch die Datenpakete weiter. Die eigentlichen Entscheidungen im Netzwerk trifft ein zentralisierter und programmierbarer SDN-Controller (die Control Layer). Der SDN-Controller erlangt so eine Gesamtsicht über die Netzwerk Infrastruktur. Applikationen und Dienste (die Application Layer) im Netzwerk können nun über eine API des SDN-Controllers Echtzeitinformationen über das Netzwerk bekommen oder sogar die Steuerung der Datenflüsse im Netzwerk direkt beeinflussen. Das OpenFlow Protokoll dient hierbei als Vermittler zwischen Netzwerkgerät und SDN-Controller.

OpenFlow identifiziert einzelne Datenströme („Flows“) und behandelt diese entweder nach statisch definierten oder auch nach dynamisch programmierten Regeln. Ein Flow könnte zum Beispiel eine TCP-Verbindung, alle Pakete von einer

doi: 10.2312/NET-2013-02-1_09

bestimmten IP oder MAC Adresse oder Pakete mit einem bestimmten VLAN-Tag sein. Die OpenFlow Switche prüfen zunächst für jeden neuen Flow, ob sich eine Regel für diesen Datenstrom in der lokalen Flow Table des Switches befindet. Falls für den Datenstrom noch keine Regel vorhanden ist, wird bei der zentralen SDN-Controller-Software nachgefragt, was mit diesem Flow geschehen soll. Die Verbindung zwischen Switch und Controller erfolgt über den Secure Channel mittels des OpenFlow-Protokolls. Diese, so vom OpenFlow Controller erlangte Regel, wird nun in die lokale Flow Table des Switches eingetragen und auf diesen Datenstrom, sowie alle zukünftigen auf diese Regel passenden Flows, angewendet [12, S. 9] [16, S. 71].

Abbildung 3 zeigt ein vereinfachtes Beispiel einer Flow Table eines OpenFlow Gerätes.

OpenFlow Enabled Network Device

MAC src	MAC dst	IP src	IP dst	TCP dport	Action	Count
*	10:20:*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	port 2	300
*	*	*	*	25	drop	892
*	*	*	192.*	*	local	120
*	*	*	*	*	controller	11

Abbildung 3: Flow Table [12, S.9]

Für die definierten Actions in der Flow-Table gibt es verschiedene Möglichkeiten: [16, S. 71]

1. Weiterleiten der Pakete an einen oder mehrere Ports.
2. Verkapseln des Paketes um es dann über den Secure Channel an den Controller zu senden. Normalerweise geschieht dies mit dem ersten Paket eines Flows, um eine Regel für diesen Flow zu erlangen. Außerdem ist es auch möglich das Paket im Controller weiterzuverarbeiten und zum Beispiel zu verschlüsseln oder zu komprimieren.
3. Verwerfen von Pakten, um zum Beispiel Angriffe abzuwehren.
4. Die Pakete an den normalen Layer 2 und Layer 3 Verarbeitungsprozesse des Switches weiterleiten. Dies ermöglicht es bei OpenFlow-Enabeld Switches nur unter bestimmten Umständen die Paketverarbeitung mit OpenFlow zu nutzen und ansonsten auf die reguläre Paketverarbeitung des Switches zurückzugreifen.

3.3 OpenFlow in der Praxis

Um den praktischen Nutzen von OpenFlow zu demonstrieren werden im Folgenden einige Anwendungsfälle in praktischen Szenarios genauer erläutert.

3.3.1 Experimente und Innovation in bestehenden Netzwerken

Forschung und Entwicklung in Netzwerken ist eine schwierige Herausforderung: Einerseits gehören die Netzwerke an Universitäten und in Unternehmen zu der geschäftskritischen Infrastruktur die auf keinen Fall gestört werden darf. Andererseits ist ein Fortschritt in der Netzwerkinfrastruktur nur möglich, wenn man mit neuen Technologien unter realen Bedienungen im echten, Netzwerk experimentiert und anschließend auf diese umstellt. Dazu gehören neue Routing-Protokolle oder der Umstieg auf IPv6. Des Weiteren bieten sich auch Stress-Tests oder Simulationen von Angriffen an, um das Verhalten des realen Netzwerks in diesen Situationen zu untersuchen.

Der Nutzen von OpenFlow für solche Zwecke lässt sich am besten an einem Beispiel aufzeigen: Ein Forscher einer Universität hat das fiktives Routing-Protokoll „X-OSPF“ entwickelt, das langfristig das Routing-Protokoll Open Shortest Path First (OSPF) in dem Universitätsnetz ersetzen soll. Der Forscher will das Protokoll nun von seinem Rechner aus in dem echten Netzwerk ausprobieren, ohne den regulären Netzwerkbetrieb zu beeinträchtigen. Dazu installiert er im OpenFlow Controller des Netzwerkes das „X-OSPF“-Protokoll und hinterlegt die Regel, alle Pakete die von der MAC Adresse seines Rechners stammen zunächst an den SDN Controller weiterzuleiten. Erreicht nun das erste Paket von seinem Rechner einen OpenFlow-Switch, wird das Paket an den Controller weitergeleitet. Der Controller verarbeitet das Paket und sendet die entsprechenden Regeln für das Routing in den Flow-Tables der Switche, die sich auf dem Pfad des Paketes befinden. Dadurch wird das „X-OSPF“-Protokoll nun nur auf die Pakete angewendet, die vom Rechner des Forschers stammen. Dieser Setup ermöglicht nun das Testen des „X-OSPF“-Protokolls im realen Netzwerk ohne den Produktivbetrieb zu beeinträchtigen. [16, S. 72]

Das Netzwerk der Stanford Universität ist inzwischen komplett auf OpenFlow umgestellt, um Experimente dieser Art zu ermöglichen. Dabei kommt, der für solche Zwecke speziell entwickelte, „FlowVisor“ zum Einsatz. Der FlowVisor ist ein spezieller OpenFlow Controller, der als Proxy zwischen den OpenFlow Switches und Controller fungiert und die Verteilung auf verschiedene Controller ermöglicht. Dadurch wird das Netzwerk in sogenannte „Slices“ unterteilt. Ein Slice ist das Produktivnetzwerk und die anderen Slices sind voneinander unabhängige und abgeschirmte Experimente. Jeder Slice hat seinen eignen OpenFlow Controller. Dadurch werden die einzelnen Experimente isoliert und es wird sichergestellt, dass die einzelnen Experimente das Produktivnetzwerk nicht stören können. [19, S. 129]

Beispiel für eines dieser „Slices“ an der Stanford Universität ist das „OpenPipes“-Experiment: Dabei werden Frames von Video-Streams im Netzwerk automatisch erkannt und mit einem bestimmten Filter versehen. Ein weiterer Versuch trägt den Namen „OpenRoads“: Man versucht eine nahtlose Übergabe (Hand Over) zwischen den WLAN und WIMAX Endpunkten der Universität zu ermöglichen. Dabei erkennt das Netzwerk schon im Voraus, wann es zu einer Übergabe kommen kann und leitet die bestehenden Verbindungen an den neuen Endpunkt um. [19, S. 130]

3.3.2 Load Balancing

Load Balancer haben die Aufgabe, den ankommenden Datenverkehr möglichst effektiv zu verteilen und dabei möglicherweise ausgefallene Server zu berücksichtigen. Normaler-

doi: 10.2312/NET-2013-02-1_09

weise werden dazu in regelmäßigen Abständen Anfragen an die einzelnen Server gesendet um deren Status zu prüfen. Die momentane Auslastung des Netzwerkes an sich wird dabei meistens nicht berücksichtigt.

Ein auf OpenFlow basierter und, ebenfalls an der Stanford Universität entwickelt und eingesetzter, Load-Balancer mit dem Namen „Plug-n-Serve“ hat das Ziel die Antwortzeit von HTTP Anfragen zu minimieren und neue Webserver dynamisch hinzuzufügen. Die Web Server sind verteilt im „Gates Computer Science“ Gebäude der Stanford Universität und es kommen verschiedene OpenFlow-Enabled Switche von Cisco, HP und NEC zum Einsatz.

Abbildung 4 zeigt die Architektur des „Plug-n-Serve“ Systems.

Flow Manager: Das Modul innerhalb des OpenFlow-Controllers das dafür sorgt, dass alle Pakete des gleichen Flows auf den gleichen Webserver umgeleitet werden. Dadurch bleiben Sessions bestehen.

Net Manager: Dieses Modul überwacht den Status des Netzwerkes. Dazu werden die OpenFlow Switche regelmäßig nach aktueller verfügbarer Bandbreite und Latenz befragt.

Host Manager: Der Host Manager überwacht die Auslastung der einzelnen Webserver. Außerdem werden neue Webserver automatisch erkannt und zum System hinzugefügt.

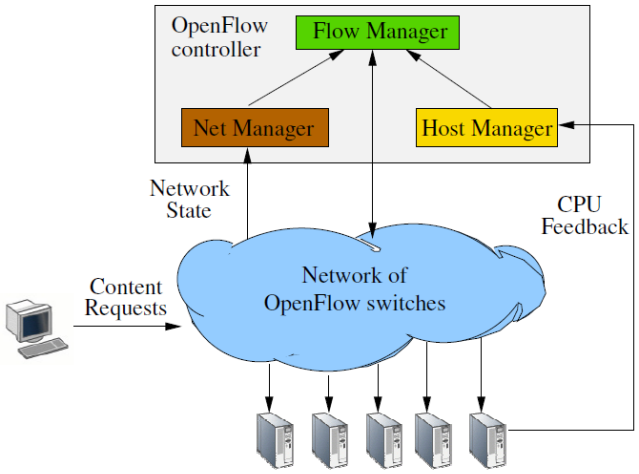

Abbildung 4: Plug-n-Serve Architektur [14, S.2]

Alle Webserver nutzen den gleichen IP-Alias. Wird vom OpenFlow Controller nun ein neuer Flow von einem Client an diese IP erkannt, wird ein passender Webserver aus dem Pool ausgewählt. Dabei werden alle im Controller verfügbaren Daten wie Latenz, verfügbarer Bandbreite, durchschnittliche Antwortzeit und Auslastung der einzelnen Webserver berücksichtigt. Der ausgewählte Webserver wird nun in den Flow-Tables der Switche hinterlegt und alle Pakete des Flows erreichen nun direkt den Webserver. [14]

3.3.3 Virtualisierung und Clouds

Cloud Computing und die Virtualisierung von IT Ressourcen ist inzwischen Standard und wird sicher auch die Zukunft weiter prägen. Dabei muss auch die darunterliegende Netzwerk Infrastruktur an die neuen Gegebenheiten angepasst werden. Um den Anforderungen des Cloud Computings gerecht zu werden, bietet sich eine Virtualisierung der Netzwerk Infrastruktur an:

Größere Datencenter umfassen über 100.000 Server, die je nach Hardware und Anforderungen in bis zu 20-30 virtuelle Hosts eingeteilt werden können. Dies resultiert in 2-3 Millionen MAC und IP-Adressen, dessen Verwaltung eine große Herausforderung ist. [15, S.672] Für die Netzwerk Infrastrukturen ist es wichtig, das Geschäftsmodell „Infrastructure as a Service“ (IaaS) umzusetzen: Das Netzwerk muss in Echtzeit umkonfigurierbar sein und skalieren können; virtuelle Hosts müssen zwischen physischen Hosts ohne Verwaltungsaufwand migriert werden können; Das virtualisierte Netzwerk muss in abgetrennte Subnetze für die einzelnen Nutzer unterteilbar sein. [15, S.672]

Auf der „IEEE CloudCom 2011“, einer internationalen Konferenz rund um Cloud Computing, wurde ein auf OpenFlow basierendes Konzept vorgestellt, dass diesen Aufgaben gerecht werden soll:

Eine Möglichkeit der Unterteilung bzw. Virtualisierung von Netzwerken sind VLANs (Virtual Local Area Networks). Jedes Paket wird mit einem spezifischen VLAN-Tag gekennzeichnet, um die Pakete dem virtuellen Teilnetzwerk zuzuordnen. Diese Methode hat zum einen den Nachteil, dass die Komplexität durch die Verwaltung des VLANs weiter erhöht wird und zum anderen wird durch das Mitsenden der VLAN Tags zusätzlicher Overhead erzeugt.

Die von dem Konzept bevorzugte Unterteilung in Teilnetzwerke erfolgt über die MAC Adressen auf Layer 2: Die MAC Adressen können nach dem IEEE 802.3 Standard lokal zugewiesen und verwaltet werden. Das ermöglicht es, die 48 Bits einer MAC Adresse in einen Host Anteil zur Identifizierung und einer Netz ID zur Einteilung in Subnetze (ähnlich dem VLAN Tag) zu unterteilen. [15, S.674]

Die MAC Adresse von virtuellen Maschinen wird normalerweise zufällig generiert (z.B. bei VMWare, Xen oder VirtualBox) und lässt sich beliebig ändern. Dies ermöglicht es VMs ohne großen Aufwand durch das Ändern des Netz Anteils der MAC Adresse in andere Teilnetzwerke zu verschieben. Da die 48 Bit MAC Adresse nach dem Ethernet Standard in jedem Fall übertragen wird, entsteht kein weiterer Overhead und auch die Verwaltung bleibt übersichtlich und einfach. [15, S.675]

OpenFlow unterstützt ab Version 1.1 auf MAC-Präfixen basierende Regeln. Dadurch kann der OpenFlow Controller anhand des MAC-Präfixes (der Netz ID) sicherstellen, dass der Netzwerkverkehr der einzelnen Teilnetzwerke isoliert wird. So werden z.B. Pakete an die MAC Broadcast Adresse nur an die virtuellen Maschinen mit der gleichen MAC-Präfix zugestellt. Außerdem ist es denkbar, für jedes Teilnetzwerk einen eigenen OpenFlow Controller mit eigenen Regeln einzurichten. [15, S.676]

doi: 10.2312/NET-2013-02-1_09

3.3.4 *Transparente Schaltung von Diensten*

Wedge Networks, eine Sicherheitsfirma aus den USA, hat sich mit der transparenten Einschiebung von zusätzlichen Diensten im Netzwerk mittels OpenFlow befasst. Konkret sollte ein Virenscanner, Anti-Spam System und ein Web Filter in ein Netzwerk integriert werden. Dabei sollten diese unabhängig von der Konfiguration der Clients und Server funktionieren.

Zur Umsetzung wurde ein OpenFlow Switch, ein OpenFlow Controller und ein Sicherheitsserver für den Virenscanner und die Filter verwendet. Im OpenFlow Controller wurden Regeln hinterlegt, die den Datenverkehr auf den Ports für HTTP, HTTPS, IMAP, POP3 und SMTP an den Sicherheitsserver umgeleitet haben. Zusätzlich wurden Regeln aufgesetzt, die die geprüften und gefilterten Datenströme anschließend wieder an das ursprüngliche Ziel weiterleiten.

Das Experiment funktioniere anschließend verlässlich und ohne nennenswerte Einbußen bei der Performance. Das Netzwerk wurde dadurch unabhängig von Server und Client-Konfiguration vor Viren geschützt. Ähnliche Anwendungszwecke zur Filterung, Monitoring und Absicherung des Datenverkehrs bieten sich beispielsweise in großen Campus Netzwerken. [8]

3.3.5 *Big Data Centers*

Das beste Beispiel für den Einsatz von OpenFlow bei riesigen Datenzentren ist Google. Dienste wie die Google Suche, Gmail, Google Maps oder YouTube tauschen täglich Unmengen von Daten zwischen den verschiedenen Google Datenzentren aus.

Googles Netzwerk ist in das Internet-Backbone, das den Traffic zu den Nutzern transportiert, und dem internen Backbone, das für den Traffic zwischen den Datenzentren verantwortlich ist, unterteilt.

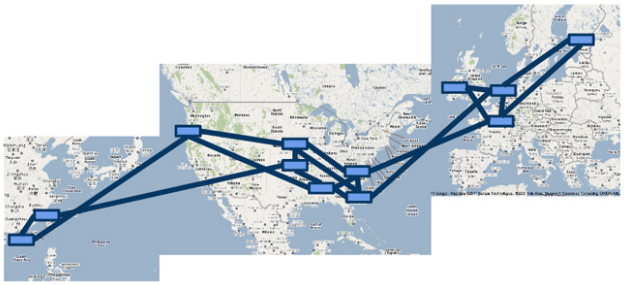

Abbildung 5: Googles internes Backbone [3]

Die Anforderungen für das interne Backbone von Google wurden mit steigendem Traffic immer größer: Der Datenverkehr muss zwischen den einzelnen Links intelligent verteilt werden, neue Dienste müssen schnell und einfach realisierbar sein und das Netzwerk muss auf ausfallende Links schnell reagieren [3].

Als Google deswegen im Jahr 2010 anfing, das interne Datencenter auf OpenFlow umzustellen, waren auf dem Markt kaum Geräte verfügbar, die den Anforderungen von Google gerecht wurden. Deswegen hat Google zunächst eigne Switche und Controller gebaut, die bis heute im Einsatz sind. Um eine hohe Fehlertoleranz zu gewährleisten, werden mehrere OpenFlow Controller eingesetzt. Zusätzlich nutzt Google einen zentralisierten „Traffic Engineering“ (TE) Dienst. Dieser sammelt die Daten von den OpenFlow Controllern und erhält so eine Gesamtsicht über das Netzwerk, um dynamisch Bandbreiten für die einzelnen Dienste zuteilen zu können. Inzwischen ist 95% des Google Netzwerkes auf OpenFlow umgestellt und trotz des großen Datenaufkommens kommt es sehr selten zu Engpässen oder gar Ausfällen. [5]

> „Das Software Defined Networking ist ein pragmatischer Denkansatz um die Komplexität und den Verwaltungsaufwand von Netzwerken zu reduzieren. Obwohl es immer noch zu früh ist um den abschließenden Erfolg zu verkünden, zeigen Googles Erfahrungen mit SDN und OpenFlow im Netzwerk, dass es bereit ist für den Einsatz in der realen Welt.“

(Urs Hölzle, Vizepräsident der technischen Infrastruktur von Google, Übersetzt aus dem Englischen [5])

3.4 Programmierung in SDN-Architekturen mit der Programmiersprache Frenetic

Die meisten OpenFlow Controller basieren auf dem Betriebssystem NOX, das für die Verteilung der verschiedenen Regeln an die einzelnen Switches verantwortlich ist. [11, S.1] Die Programmierung des OpenFlow Controllers ist aus verschiedenen Gründen eine Herausforderung:

- Der Controller enthält nur die Pakete bzw. Flows, für die noch keine Regel im Switch hinterlegt ist. Regeln müssen daher mit Bedacht installiert und gegebenenfalls auch wieder gelöscht werden.
- Die Regeln können sich, beispielsweise durch Wildcards, gegenseitig beeinflussen beziehungsweise überschreiben: Für die einzelnen Regeln müssen Prioritäten in den Flow Tables hinterlegt werden.
- Verschiedene Module, wie Routing oder Monitoring, müssen aufeinander abgestimmt sein, wenn sie auf die gleichen Flows zugreifen.

Ein Ansatz um die Programmierung von OpenFlow Controllern zu vereinfachen ist Frenetic. Frenetic verwendet eine Abstraktion, die eine paketbasierte Programmierung erlaubt. Bei Frenetic handelt es sich um eine höhere Programmiersprache, die auf den Ideen der funktionalen Programmierung basiert. Die verschiedenen Regeln werden dann von Frenetic zur Laufzeit erzeugt und auf den Switches und Controller verteilt. Technisch handelt es sich bei Frenetic um eine Sammlung von Python Bibliotheken. [11, S.1]

doi: 10.2312/NET-2013-02-1_09

Die Funktionsweise von Frenetic wird im Folgenden an einigen Beispielen demonstriert.

Der Frenetic Programmcode in Listing 1 implementiert die einfache Funktion eines Switches, alle Pakete die den Switch an einem Port erreichen an den anderen Port weiterzuleiten.

Listing 1: Einfacher Repater

```
def switch_join(switch):
    # Repeat Port 1 to Port 2
    p1 = {in_port:1}
    a1 = [forward(2)]
    install(switch, p1, DEFAULT, a1)
    # Repeat Port 2 to Port 1
    p2 = {in_port:2}
    a2 = [forward(1)]
    install(switch, p2, DEFAULT, a2)
```

Wenn ein neuer Switch an das Netzwerk angeschlossen wird, wird die Funktion `switch_join` aufgerufen und zwei Regeln installiert: Die Variablen `p1` und `p2` beschreiben die Muster (Patterns), wann die einzelnen Regeln angewendet werden sollen. In diesem Fall das Eintreffen von Paketen an Port 1 bzw. 2. `a1` und `a2` stehen für die darauf folgenden Aktionen (Actions) zum Weiterleiten der Pakete an den jeweils anderen Port. Mit `install` werden die Regeln in der Flow Table das Switches hinterlegt. Der dritte Parameter, in diesem Fall `DEFAULT`, beschreibt die Priorität der Regel. [11, S.2] [18, S.5]

Eine erweitertes Programm in Listing 2 soll den kompletten Traffic auf Port 80 beobachten.

Listing 2: Überwachung von Port 80

```
def switch_join(switch)):
    # Web traffic from Internet
    p = {inport:2,tp_src:80}
    install(switch, p, DEFAULT, [])
    query_stats(switch, p)
def stats_in(switch, p, bytes):
    print bytes
    sleep(30)
    query_stats(switch, p)
```

Zunächst wird eine Regel installiert, die bei eingehenden Traffic an Port 2 des Switches und TCP Port 80 ausgeführt wird. Die Funktion `query_stats` zählt den Traffic in Bytes und die Funktion `stats_in` gibt die übertragene Datenmenge alle 30 Sekunden aus [11, S.2] [18, S.6].

3.5 Kritische Betrachtung und Ausblick

OpenFlow und SDN wird oft als die Zukunft der Routing Protokolle angepriesen. Klassische Routing Protokolle stimmen sich nicht aufeinander ab, sie berücksichtigen nicht die Auslastung des Netzwerkes, sie sind statisch und können nicht in Echtzeit verändert und kontrolliert werden. All diese Nachteile können durch ein OpenFlow basiertes Netzwerk ausgeräumt werden. Trotzdem sollte man nicht unberücksichtigt lassen, dass sich die klassischen Routing Protokolle in den letzten 20 Jahren bewährt haben: sie sind verlässlich, geprüft, selbst-heilend, autonom und skalierbar. [17, S.8]

Bei OpenFlow handelt es sich trotz allem um Software, welche Fehler enthalten kann, anfällig ist oder fehlerhaft programmiert werden kann. Man sollte daher sehr sorgfältig abwägen, ob OpenFlow im spezifischen Anwendungsfall wirklich sinnvoll ist und auch einen Mehrwert bietet. Obwohl es viele erfolgreiche experimentelle und durchaus marktreife Projekte mit OpenFlow gibt, hört man recht wenig von der praktischen Nutzung von OpenFlow Netzwerken in der freien Wirtschaft. Zum einem ist die Skepsis groß, Teile der kritischen Netzwerkinfrastruktur Software anzuvertrauen. Zum Anderem lassen sich viele Probleme bereits durch bewährte Techniken wie beispielsweise VLANs oder QoS lösen. Wirklich bewährt hat sich OpenFlow bisher nur bei riesigen Hostern und Universitäten mit speziellen Anwendungszwecken - oder eben bei Google.

Nicht unbedeutend für die Zukunft von SDN und OpenFlow ist die Unterstützung und die Integration von den Hardware Herstellern. Diese läuft relativ schleppend, obwohl die meisten Hersteller inzwischen OpenFlow unterstützen. Oft wird nur die alte OpenFlow Version 1.0 unterstützt. Dies liegt daran, dass die Hersteller wie Cisco nur ungern Software auf ihren Geräten erlauben, die die eigene Software verdrängen könnte. Es besteht die berechtigte Sorge, dass die bisher teuer verkauften Produkte durch günstige OpenFlow Switches mit Open Source Software ersetzt werden. Cisco entwickelt zudem ihre eigenen SDN Lösungen („Cisco One“), bei denen sie an den Lizenzen verdienen möchten [13]. Diese Blockadehaltung gegenüber OpenFlow aber durch den Konkurrenzdruck zwangsläufig weiter aufweichen.

Auf längere Sicht wird sich die Virtualisierung von Netzwerken und damit das Software Defined Networking sicher weiter durchsetzen. Doch ähnlich wie bei der Virtualisierung von Server Infrastrukturen wird dieser ein langjähriger Prozess sein und sehr langsam voranschreiten. Im Gartner Hype Cycle 2012 für Netzwerk und Kommunikation wird SDN und OpenFlow noch im Bereich „Technology Trigger“ kurz vor dem Übergang zum Gipfel der überzogenen Erwartungen („Peak of Inflated Expectations“) angesiedelt und mit einer Marktreife in 5 bis 10 Jahren eingeordnet. [10]

4. LITERATUR

[1] About openflow. `http://www.openflow.org/wp/about/`, September 2012.

[2] Cisco open network environment. `http://www.cisco.com/web/solutions/trends/open_network_environment/indepth.html`, Oktober 2012.

[3] Going with the flow: Google's secret switch to the next wave of networking. `http://www.wired.com/wiredenterprise/2012/04/going-with-the-flow-google/all/1`, November 2012.

[4] Hp simplifies networking with broadest choice of openflow-enabled switches. `http://www.hp.com/hpinfo/newsroom/press/2012/120202a.html`, Oktober 2012.

[5] Inter-datacenter wan with centralized te using sdn and openflow. `https://www.opennetworking.org/images/stories/downloads/misc/googlesdn.pdf`, Oktober 2012.

doi: 10.2312/NET-2013-02-1_09

[6] Openflow wiki. http://www.openflow.org/wk/index.php/Main_Page, September 2012.

[7] Openflow – next-generation networking for a smarter planet. https://www-304.ibm.com/connections/blogs/VMstg/tags/openflow?lang=en_us, November 2012.

[8] Wedge networks whitepaper: Transparent service insertion in sdns using openflow, September 2012.

[9] C. Berndtson. Open networking foundation: About. https://www.opennetworking.org/about/, September 2012.

[10] N. R. Bjarne Munch, Katja Ruud. Hype cycle for networking and communications, 2012. *Gartner*, Juli 2012.

[11] N. Fostery, R. Harrison, M. L. Meola, M. J. Freedman, J. Rexford, and D. Walker. Frenetic: a high-level language for openflow networks. *PRESTO '10 Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, (6), 2010.

[12] O. N. Foundation. White paper: Software-defined networking: The new norm for networks, April 2012.

[13] M. Fratto. Prediction: Openflow is dead by 2014; sdn reborn in network management, Mai 2012.

[14] N. Handigol, S. Seetharaman, N. McKeown, and R. Johari. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, August 2009.

[15] J. Matias, E. Jacob, D. Sanchez, and Y. Demchenko. An openflow based network virtualization framework for the cloud. *Cloud Computing Technology and Science (CloudCom)*, pages 672–678, December 2011.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[17] I. Pepelnjak. Openflow and sdn: hype, useful tools or panacea?, 2012.

[18] J. Rexford. Frenetic: A programming language for openflow networks. ttp://www.frenetic-lang.org/, Oktober 2012.

[19] R. Sherwood, G. Gibb, and M. Kobayashi. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130, January 2010.

[20] J. K. Waters. Virtualization definition and solutions, März 2007.

doi: 10.2312/NET-2013-02-1_09