Northbound API Requirements for SDN

Christian Fuchs
Betreuer: Daniel Raumer
Seminar Innovative Internettechnologien und Mobilkommunikation SS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: fuchs@in.tum.de

KURZFASSUNG

Software Defined Networking (SDN) hat über die letzten Jahre eine starke Verbreitung erfahren. An der Schnittstelle zwischen den Netzwerkgeräten und den Controllern, dem sogenannten Southbound Interface, hat sich das Protokoll OpenFlow als ein Standard durchgesetzt, der von großen Teilen des Markts akzeptiert und unterstützt wird. An der Schnittstelle zwischen den Contoller Frameworks und den Applikationen, dem Northbound Interface (NBI), verhält sich dies jedoch grundlegend anders, da hier jedes Controller Framework eine eigene API (Application Programming Interface) besitzt. Aufgrund des Umstands, dass die Anwendungsgebiete von SDN sehr vielseitig sind, unterscheiden sich diese APIs teilweise stark voneinander. Das bringt einige Probleme mit sich, weshalb der Wunsch nach einem Standard für diese Northbound API immer stärker wird. Doch auch der Prozess der Standardisierung ist mit Schwierigkeiten verbunden, was zur Folge hat, dass derzeit noch kein Standard existiert und bis jetzt auch noch nicht abzusehen ist, ob und in welcher Form ein solcher Standard etabliert werden könnte.

Schlüsselworte

Software Defined Networking, Northbound API, Northbound Interfaces, Southbound Interface, NBI Standardisierung, Pyretic, SDN Controller, OpenFlow

1. EINLEITUNG

2008 wurde von einer Gruppe von Forschern der Universitäten Stanford, Princeton, Washington, dem MIT und der UC California im Artikel „OpenFlow: Enabling Innovation in Campus Networks“ [15] das Protokoll OpenFlow vorgestellt. Dieses sollte, aufbauend auf einem Netz aus OpenFlow fähiger Hardware, ermöglichen, Experimente im universitätseigenen Netzwerk durchzuführen, ohne den normalen Netzbetrieb zu stören oder großen Konfigurationsaufwand betreiben zu müssen. Das Konzept, das dieser vorgestellten Technik zugrunde lag, in anderer Form bereits durch andere Projekte, wie zum Beispiel RFC 5810 „Forwarding and Control Element Separation“ behandelt und unter dem Namen Software Defined Networking bekannt wurde, bot jedoch auch einen Lösungsansatz für Probleme, die weit über die zu Anfang adressierte Problematik hinausging, was großes Interesse vonseiten der Netzwerkindustrie hervorrief. Die darauffolgende Verbreitung von SDN wird häufig als wahrer „Hype“ beschrieben und führte dazu, dass SDN schon wenige Jahre später in die meisten Gebiete der Netzwerktechnik Einzug gehalten hat, was, wenn man die Dauer anderer Innovationen im Netzwerkumfeld bedenkt, wie beispielsweise der Einführung von IPv6, durchaus bemerkenswert ist. Es ist aber auch klar, dass diese rasante Entwicklung herausfordernde Probleme mit sich bringt. Eines dieser Probleme, welches aktuell viele Akteure im SDN Umfeld beschäftigt, ist die Diskussion um die Standardisierung des NBIs. Bevor jedoch auf diese Problematik eingegangen wird, soll in dieser Arbeit zunächst in Kapitel 2 das Konzept und die Anwendungsgebiete des Software Defined Networkings kurz vorgestellt werden, um eine Grundlage für die folgenden Abschnitte zu schaffen. Im dritten Abschnitt sollen daraufhin die Begriffe Northbound und Southbound Interface eingeführt, unter Punkt 3.1 kurz auf das Southbound Interface OpenFlow eingegangen und unter Punkt 3.2 die Probleme am NBI thematisiert werden. Punkt 4 beinhaltet das Kernthema dieser Arbeit. Es sollen dort insbesondere mögliche Anforderungen an eine Northbound API, die Vor- und Nachteile einer Standardisierung, sowie die Anforderungen an einen NBI Standard herausgearbeitet werden. Abschnitt 4.6 beschäftigt sich kurz mit den Unterschieden im Standardisierungsprozess von Northbound und Southbound Interface. Im letzten Abschnitt soll abschließend als Beispiel für ein relativ abstraktes und mächtiges NBI Pyretic beschrieben werden. Hier konzentriert sich diese Arbeit auf Modularisierung und Kompositionsoperatoren, das abstrakte Paket Modell, die High-Level Policy Funktionen und die Grenzen, denen Pyretic unterworfen ist.

2. SOFTWARE DEFINED NETWORKING - KONZEPT UND ANWENDUNGEN

Das grundlegende Modell von SDN ist unkompliziert und wurde bereits in vielen wissenschaftlichen Arbeiten ausführlich beschrieben. Einen guten Einstieg bietet ein Whitepaper der Open Networking Foundation (ONF) [18] oder der Artikel [10]. Sehr gut werden die Komponenten und Schlüsselprinzipien von SDN auch in [21], einem weiteren Dokument der ONF, beschrieben. Hier ist auch das SDN Grundprinzip prägnant zusammengefasst: „The control plane is (1) logically centralized and (2) decoupled from the data plane“ ([21], Seite 5). Dabei bezeichnet die Control Plane die Ebene in der Paketverarbeitung, auf der die Entscheidungen bezüglich der Weiterleitung von Paketen getroffen wird. Ein Teil dieser Control Plane kann beispielsweise die Routingtabelle sein. Die Data Plane ist im Gegensatz dazu konkret für die Weiterleitung der Pakete verantwortlich. Kommt also ein Paket an einem Netzwerkgerät an, wird die Information wie es behandelt werden soll aus der Control Plane geholt und die Data Plane verarbeitet das Paket dann gemäß dieser In-

doi: 10.2313/NET-2014-08-1_13

formationen, passt also zum Beispiel Header Felder an und gibt es auf einem bestimmten Interface wieder aus, droppt es oder leitet es an eine bestimmte Applikation weiter. Der Begriff **Software Defined** Networking ist darauf zurückzuführen, dass der Controller, in welchen die Control Plane ausgegliedert wird, in der Regel in Software realisiert ist. Dies hat zur Folge, dass sämtliche Kontrollinformation, also die gesamte Netzintelligenz, in Software vorliegt. Damit kontrolliert die Software das gesamte Netzwerkverhalten, das Netzwerk ist also durch Software „definiert".
Doch inwiefern lässt sich dieses Konzept auf reale Systeme übertragen? Eine in der Realität häufig auftretende Topologie besteht aus einer Menge an Hosts, die durch miteinander konkatenierten, programmierbaren Switches verbunden sind. In traditionellen Netzwerken ohne SDN muss zur Beeinflussung des Netzwerkverhaltens jeder Switch einzeln konfiguriert werden, da hier innerhalb eines Switches die Control Plane fest mit der Data Plane verbunden ist. Dies bringt diverse Probleme mit sich: Zum Beispiel ist selbst bei sorgfältig durchgeführter Konfiguration die Gefahr von Inkonsistenzen sehr hoch, was besonders in sicherheitsrelevanten Systemen, wie Firewalls oder Access Control Mechanismen, sehr gefährlich sein kann. Man hat auch keine globale Sicht auf das Netzwerk, was beispielsweise die Realisierung von komplexeren Quality of Service (QoS) Mechanismen stark erschwert. Ein weiteres Problem sind die Konfigurationssprachen an sich. Diese sind zum einen meist sehr hardwarenah, bieten also nur wenig Abstraktion, und sind zum andern häufig herstellerspezifisch, was den herstellerübergreifenden Tausch von Hardware bei vorhandener Konfiguration wesentlich verkompliziert. Folglich sind solche Topologie mit hohem Aufwand und hohen Kosten bei Entwicklung und Administration der Netzwerke verbunden, insbesondere da auch Skalierbarkeit eine große Problematik darstellt. Denn bei steigender Anzahl an Netzwerkgeräten verschärfen sich die aufgeführten Probleme zunehmend.
Hier schafft SDN Abhilfe. Durch die zentrale Konfigurationsschnittstelle am Controller, welche meist eine globale Sicht auf das Netzwerk und, abhängig von der Gestaltung der Northbound API, oft auch ein höheres Abstraktionslevel bietet, lösen sich viele der beschriebenen Probleme, wie zum Beispiel die Skalierbarkeitsproblematik, von selbst. Bei Einsatz von standardisierten Protokollen wie OpenFlow, ist sowohl der herstellerübergreifende Wechsel der Hardware, als auch die Realisierung von Multivendor-Umgebungen problemlos möglich. Diese und weitere Vorteile von SDN sind insbesondere bei der Verwirklichung von Virtualisierungstechniken, welche ständig weiter an Bedeutung gewinnen, überaus wichtig. Und der Bedarf an mächtiger und dennoch einfach zu konfigurierender Virtualisierung ist aktuell schon sehr groß, bedenkt man zum Beispiel all die Produkte im Feld der mobilen Endgeräte, von Cloud Services, von Cloud Computing und von Infrastrukture as a Service (IaaS) Diensten, bei denen Endkunden virtuelle Maschinen in virtuellen Netzen verschiedenster Art und Form angeboten werden. Virtualisierung und die SDN Lösungen, auf denen diese basiert, sind schon heute in vielen Bereichen unverzichtbar geworden und werden folglich auch in sämtlichen Bereichen der Netzwerkwelt verwendet. Einige populäre Beispiele hierfür sind das SDN basierte Wide Area Network (WAN) G-Scale von Google, welches weltweit Datenzentren miteinander verbindet [7], Googles interne Virtualisierungstechnik Andromeda [27], die seit April 2014 auch Googles Cloud Service Nutzern zur Verfügung steht, oder Microsofts Hyper-V Technologie, die Virtualisierungstechnik in Microsoft Server 2012 [22].
Die vielseitige Verwendung von SDN hatte nicht nur eine starke Verbreitung des Konzepts zur Folge, sondern auch, dass sich die Bedeutung des Begriffs „Software Defined Networking" immer weiter in verschiedene Richtungen erweiterte und veränderte. Somit findet sich der Begriff SDN heute auch in Technologien wieder, die mit dem eigentlichen Konzept nur noch wenig zu tun haben. Martin Casado, ein renommierter Wissenschaftler im SDN Umfeld, der auch stark an der Entwicklung der ersten verbreiteteren SDN Technologien, wie OpenFlow oder der Controllerplattform NOX, an der Universität von Stanford beteiligt war, drückte diesen Umstand folgendermaßen aus: „I actually don't really know what SDN means anymore, to be honest. [...] The term was coined in 2009. At that time it meant somthing fairly specific, but now it is just being used as a general term for networking, like all networking is SDN. [...] I think it's an umbrella term for cool stuff in networking" ([1]). Aktuelle Versuche, SDN konkret zu definieren, wie beispielsweise in RFC 7149, beziehen diese Terminologieveränderungen mit ein und fassen SDN viel weiter, als das vorgestellte Konzept von der Trennung von Control und Data Plane.

3. NORTH- UND SOUTHBOUND API

Die Begriffe North- und Southbound Interface existieren auch abseits von SDN. Die Logik hinter der Begriffsbildung ist eingängig: Norden ist oben, Süden ist unten. Hat man nun eine Komponente in einem hierarchisch aufgebauten System, dann ist das Northbound Interface folglich die Schnittstelle dieser Komponente nach oben und somit die Schnittstelle zur in der Hierarchie nächst höheren Instanz, also ein Satz von aufrufbaren Funktionen, die der nächst höheren Instanz von der Komponente präsentiert wird, damit diese die Funktionalität jener nutzen kann. Analog dazu ist das Southbound Interface der Komponente die Schnittstelle nach unten und somit die Schnittstelle zur in der Hierarchie nächst niedrigeren Instanz. Damit ist auch klar, dass North- und Southbound Interface in einem System keine fest definierten Schnittstellen, sondern abhängig von der betrachteten Komponente sind. Konkret auf SDN bezogen, ist diese betrachtete Komponente meist der Controller, wie auch (Abb. 1) veranschaulicht. Das NBI ist folglich die Schnittstelle zwischen dem Controller und den SDN Applikationen. Sie bietet typischerweise eine abstrakte Sicht auf das Netzwerk und Funktionalitäten, mit denen man das Verhalten des Netzes festlegen und Serviceinformationen auslesen kann. Da diese Schnittstelle meist in irgendeiner Form programmierbar ist, spricht man oft anstatt von einem Interface, von einer API (Application Programming Interface). Im Gegensatz dazu bezeichnet man das Southbound Interface nur selten als API, da an der Schnittstelle zwischen dem Controller und den von diesem kontrollierten Netzwerkkomponenten, den Switches, keine Konfigurationsschnittstelle im herkömmlichen Sinne definiert wird. Es handelt sich hier vielmehr um das Protokoll, mit dem der Controller mit den Komponenten kommuniziert, die Menge an von den Switches zu unterstützenden Aktionen, wie zum Beispiel FORWARD, DROP, FLOOD oder Weiterleitung an den Controller, und die Menge an Informationen, die vom Switch bereitgestellt werden müssen. Solche Informationen sind beispielsweise QoS Metainformationen wie Byte- und Packetcounter. Man muss hierbei präzise auf die Terminologie achten. Es gibt auf dem

doi: 10.2313/NET-2014-08-1_13

Markt etliche SDN Lösungen, bei denen verschiedene Controller Frameworks aufeinander aufbauen. Das heißt, dass dort das NBI des hardwarenahen Controllers gleichbedeutend mit dem Southbound Interface des abstrakteren Controllers ist, und die Trennung von Applikation und Controller verschwimmt. Angelehnt daran muss auch beachtet werden, dass in dieser Hierarchie der Controller kein abgeschlossenes, in Software realisiertes, System darstellt, in welchem sämtliche Kontrollinformation, die das Netz steuert, fest implementiert ist. Es muss als Controller Framework betrachtet werden, also als eine Bibliothek, ein Set von Funktionen, die von der Kommunikation am Southbound Interface abstrahiert und Funktionalitäten wie die Verhinderung von Inkonsistenzen bietet. Die eigentlichen Anweisungen, die das gewünschte Verhalten des Netzwerks spezifizieren, kommen in dieser Hierarchie über das NBI des Controllers aus den Applikationen [10] [21].

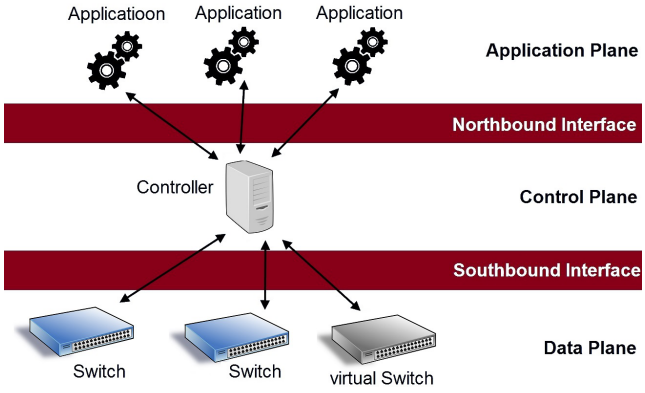

Abbildung 1: Überblick über die SDN Hierarchie

3.1 Southbound Interface OpenFlow

Wie bereits eingangs in der Einleitung beschrieben, wurde das Protokoll OpenFlow von Wissenschaftlern um Nick McKeown von der Universität Stanford zunächst für den Zweck entwickelt, im eigenen Campusnetzwerk Experimente durchführen zu können, wie zum Beispiel ein selbst definiertes Routing Protokoll in einem realen Netz zu testen. Man suchte nach einer Lösung, die mit möglichst wenig Administrationsaufwand sicherstellt, dass der normale Netzbetrieb nicht gestört wird. Das Paper, in dem OpenFlow 2008 vorgestellt wurde, trug dementsprechend auch den passenden Namen „OpenFlow: Enabling innovation in campus networks“. Wie im letzten Absatz festgehalten, definiert OpenFlow als Southbound Interface nicht nur das Kommunikationsprotokoll an sich, das heißt die Abfolge und den Aufbau der Pakete, die zwischen Controller und Switches ausgetauscht werden, sondern auch eine Menge an von den Switches zu unterstützenden Informationen und Aktionen. Bei Einsatz von Hardware Switches ist klar, dass diese das OpenFlow Protokoll explizit unterstützen müssen, weswegen die OpenFlow Entwickler schon von Beginn an eng mit Hardwareherstellern zusammenarbeiten mussten. Das robuste und auf viele Anwendungsgebiete anwendbare und in gewissem Maße auch erweiterbare Design von OpenFlow, in Verbindung mit der stetig ansteigenden Anzahl OpenFlow fähiger Hardware, stellte ein Erfolgskonzept dar und leitete die lawinenartige Verbreitung von OpenFlow basiertem SDN, den „SDN-Hype“, ein.
Einen großen Anteil daran hatte auch die Open Networking Foundation (ONF), eine 2011 gegründete Non-Profit Organisation, welche sich durch eine starke Marktmacht seiner Mitglieder auszeichnet, zu denen unter anderem die Deutsche Telekom, Facebook, Google, Microsoft, Verizon, Yahoo, Netgear, Oracle, HP, Samsung, Intel und IBM gehören. Sie hat sich zum Ziel gesetzt, die Verbreitung und Weiterentwicklung von SDN zu fördern, offene Standards zu schaffen und als erstes Etappenziel OpenFlow als Standard des Southbound Interfaces zu etablieren. Das Design von OpenFlow ist dabei einfach gehalten. Jeder OpenFlow fähige Switch verfügt über eine Pipeline an Flow Tables, in welcher jeder Eintrag einen bestimmten Flow, eine Menge von Paketen mit gleichen Charakteristika, wie beispielsweise gleiche Source- und Destination-IP, repräsentiert. Zu jedem Flow enthält ein Flow Table Eintrag neben den Matchings, welche eben jene Charakteristika spezifizieren, auch eine bestimmte Aktion, die ausgeführt werden soll, wenn ein zum jeweiligen Flow gehörendes Paket am Switch ankommt, und Metainformationen, wie zum Beispiel Timeouts, die Anzahl an bisher empfangenen Bytes oder die Anzahl an bisher empfangenen Paketen. Diese können vom Controller abgefragt oder, im Falle von Informationen wie Timeouts, gesetzt werden. OpenFlow hatte in den letzten Jahren einen großen Einfluss auf die Entwicklung und Weiterentwicklung von SDN. Fast alle namhaften Controller basieren auf OpenFlow, oder unterstützen es zumindest. Und als fester, vom Markt akzeptierter Standard der Schnittstelle am unteren Ende des Controllers ist OpenFlow auch für die folgenden, das NBI behandelnde, Abschnitte von zentraler Bedeutung. Eine ausführlichere Beschreibung von OpenFlow findet sich in den „OpenFlow Switch Specifikations“ der ONF [20]. Ein guter Überblick wird in [3] gegeben.

3.2 Das Northbound Interface

Im Gegensatz zum Southbound Interface hat sich am Northbound Interface bisher noch keine Schnittstelle als Standard durchsetzen können. Weshalb dies so ist und welche Folgen sich daraus ergeben, soll im nachfolgenden Abschnitt erläutert werden. Da diese Überlegungen sehr aktuell sind, basieren viele der folgenden Informationen nicht auf etablierten wissenschaftlichen Arbeiten, sondern sind häufig aus Blogs, Forenbeiträgen und Internetartikeln erfahrener und kompetenter Wissenschaftler und Forscher entnommen. Wichtige Anlaufstellen sind dabei die Websites „www.sdncentral.com“ und „searchsdn.techtarget.com“. Wichtige Artikel, aus welchen ein Großteil der in Abschnitt 3.2 und 4 vorgestellten Informationen entnommen wurde, sind [24], [9], [16], [12], [5] und [6]. Es sollte dem Leser deshalb bewusst sein, dass einerseits, auch wenn in dieser Arbeit versucht wird die mit Stand Q2/2014 aktuell allgemein vorherrschende Meinung korrekt wiederzugeben, der Inhalt dennoch wesentlich von der Sichtweise dieser Experten abhängt, und dass andererseits zu vielen der untersuchten Themen aktuell noch kein wissenschaftlicher Konsens gefunden ist.
Die aktuelle Situation gestaltet sich wie folgt: Es befinden sich derzeit mehr als 30 gängige Controller Frameworks auf dem Markt [25]. Alle diese Controller wurden von verschiedensten Instanzen mit jeweils sehr individuellen Zielen zur Realisierung eines ganz spezifischen Zwecks geschaffen. Es gibt proprietäre Controller, Controller von Open Source Ent-

doi: 10.2313/NET-2014-08-1_13

wicklern, Controller die an Universitäten geschaffen wurden, einige bieten die Basis für eine Palette von kommerziellen Netzwerkprodukten, andere sind eher von experimenteller Natur, einige sind ausgefeilt, bieten großen Funktionsumfang oder sind auf möglichst hohe Effizienz oder Ressourcensparsamkeit getrimmt, während andere nur spezifische Aufgaben erfüllen oder mit dem primären Ziel geschaffen wurden, möglichst einfach zu bedienen zu sein. Dementsprechend arbeiten diese Controllerplattformen auch auf völlig verschiedenen Abstraktionslevels, sowohl bezüglich Abstraktion von der Topologie, als auch im Hinblick auf den Abstraktionsgrad der Konfigurationsschnittstelle. Zudem sind die Schnittstellen der Controller in die verschiedensten Programmiersprachen eingebettet. Folglich sind aktuell mehr als 30 verschiedene NBIs im Umlauf. Diese Situation kann zu dem Schluss führen, dass die in Kapitel 2 angesprochene Portabilitätsproblematik, die das herstellerübergreifende Übertragen von Konfiguration erschwert und von SDN in Verbindung mit OpenFlow als standardisiertem Southbound Interface eigentlich beseitigt werden sollte, von den Switches hin zu den Controllern verlagert wurde. An dieser Stelle setzen nun die Überlegungen bezüglich der Schaffung eines Standards für das NBI an. Denn wie OpenFlow am Southbound Interface, würde ein Standard diese Probleme beseitigen [9][19][5].

4. NBI STANDARISIERUNG

Wie bereits angedeutet ist aktuell noch kein konkreter Standard in Sicht. Es gibt aber durchaus Projekte, aus denen ein Standard hervorgehen könnte. Darüber hinaus wird in Fachkreisen eine lebhafte Diskussion über die Vorteile und Probleme einer Standardisierung, sowie über die Anforderungen an NBI und NBI Standard geführt. Diese Vorgänge sollen in diesem Abschnitt zusammengefasst werden.

4.1 Aktueller Stand der Standardisierung

Erfolgsversprechende Bestrebungen gehen aktuell von der ONF aus. Diese hatte bereits 2012 eine Diskussionsgruppe zum Thema NBI Standard ins Leben gerufen. Man kam zu dem Schluss, dass der Zeitpunkt für eine Standardisierung noch nicht reif sei, und beschloss zunächst nur Use-Cases zu erfassen und diese Arbeit in eine andere Arbeitsgruppe, die „Architecture and Framework Working Group“, einzugliedern. Aufgrund des immer stärker werdenden Drucks, den einige der Mitglieder innerhalb der ONF ausgeübt haben, wurde nun im Oktober 2013 auch eine offizielle „Northbound Interfaces Working Group“ gegründet [5][8]. Die Leitung wurde an Sarwar Raza, Direktor von Cloud-Networking und SDN in der Advanced Technology Group for Networking bei Hewlett-Packard, vergeben. Die Arbeitsgruppe hat bis jetzt außer einem Charter [19], in welchem kurz ihre Ziele, Aufgaben und Vorgehensweisen beschrieben werden, zwar noch keine Ergebnisse veröffentlicht, dies ist bei der ONF aber auch nicht untypisch, da diese nur selten Resultate präsentiert solange der Arbeitsprozess noch nicht abgeschlossen ist. In dem zu Anfang veröffentlichten Charter wird klar festgehalten, dass sich die NBI Working Group nicht das Ziel gesetzt hat, zwingend einen Standard zu etablieren. Vielmehr soll festgestellt werden, inwiefern Standardisierung sinnvoll, beziehungsweise durchführbar ist, also ähnliche Überlegungen angestellt werden, wie sie in den folgenden Abschnitten beschrieben werden. Die ONF verfolgt dabei einen auf der Evaluation von Use-Cases aufbauenden Ansatz. Es wird in dem Charter ebenfalls betont, dass, falls die NBI Working Group schließlich doch zu der Entscheidung finden sollte, selbst einen Standard etablieren zu wollen, vor konkreten Schritten in Richtung Standardisierung unbedingt zuerst eine lauffähige Implementierung des zu standardisierenden Konzepts geschaffen und in den Markt eingeführt werden muss. Die ONF will dabei nicht nur mit Entwicklungsabteilungen ihrer Mitglieder, sondern unter Umständen auch mit externen Entwicklern, auch mit Entwicklern aus Open Source Projekten, zusammenarbeiten. Die Bestrebungen der ONF werden auch in einigen Artikeln, wie [5], [8], [13] und [14] beschrieben und bewertet.

Neben der ONF bemüht sich derzeit keine andere Institution, wie beispielsweise IEEE oder IETF, um einen Standard am NBI. Sollte die ONF jedoch zu keinen Ergebnissen kommen und der Wunsch nach einem Standard vonseiten des Marktes weiter steigen, ist es auch nicht ausgeschlossen, dass sich diese Institutionen zukünftig intensiver auf diesem Feld betätigen werden [8]. Neben den Bemühungen der ONF sehen einige Experten das OpenDaylight Projekt [26] als erfolgversprechendsten Ausgangspunkt für eine Standardisierung [5]. OpenDaylight ist ein Open Source Projekt, welches von der Linux Foundation verwaltet wird und auf der Unterstützung namhafter Unternehmen im Netzwerksektor, wie den Platin Mitgliedern Cisco, HP, IBM, Jupiter Networks, Redhat und Microsoft, aufbaut. Es wurde im April 2013 ins Leben gerufen und hat zum Ziel, ein breit gefächertes Framework für Anwendungen im Bereich des Software Defined Networkings und des Network Functions Virtualizations zu erschaffen. Erster Code wurde im Februar 2014 unter dem Namen Hydrogen veröffentlicht. OpenDaylight baut auf einer eigenen Java Virtual Machine (JVM) auf, bietet Unterstützung für das auf Java basierende OSGi-Framework, welches die Verwaltung von Softwarekomponenten und Modularisierung vereinfacht, und nutzt das Konzept der RESTful APIs [26]. Dieses Konzept findet in vielen modernen API Konzeptionen Anwendung. Es basiert auf der Dissertation von Roy Fielding [4] und definiert 6 Richtlinien für Representational State Transfer. Eine genaue Einführung in das Innenleben von OpenDaylight oder eine genaue Beschreibung der dort verwendeten APIs übersteigt den Rahmen dieser Arbeit bei Weitem, und ist auch nicht sonderlich sinnvoll, da Hydrogen den ersten Release repräsentiert, und sich das Framework daher in naher Zukunft stark verändern wird. Das OpenDaylight Projekt ist noch jung und hauptsächlich an der Produktion von Code, nicht am Etablieren eines Standards, interessiert. Daher ist eine offizielle Standardisierung vonseiten OpenDaylights trotz der Marktmacht seiner Unterstützer unwahrscheinlich. Es ist aber durchaus möglich, dass es sich in den kommenden Jahren auf dem Markt als De-facto-Standard durchsetzt [5].

4.2 Vorteile einer Standardisierung

In Abschnitt 3.2 wurden bereits Probleme angesprochen, die von einem NBI Standard gelöst werden könnten. Um die Bedeutung der Diskussion um die Standardisierung, beziehungsweise den wachsenden Wunsch nach einem Standard, vollständig verstehen zu können, ist es nötig, sich über sämtliche Vorteile einer Standardisierung klar zu werden. Ein schon behandelter Vorteil ist die Verbesserung der Nutzerfreundlichkeit durch Steigerung der Portabilität. Baut eine Anwendung auf einer standardisierten Schnittstelle auf, ist es möglich diese auf verschiedenen Controllern, die die-

doi: 10.2313/NET-2014-08-1_13

se standardisierte Schnittstelle nutzen, zu übertragen. Dies kann für die Nutzer einen enormen Mehrwert darstellen, denkt man beispielsweise wieder an einen Wechsel der Controllerplattform, den Aufbau von Multi-Vendor Umgebungen oder die Entwicklung einer Applikation, die auf verschiedenen Controllern ordnungsgemäß funktionieren sollen. Letzteres ist eine sehr zentrale Anforderung, da es sich bei den Nutzern des NBIs auch um die Entwickler von Applikationen handelt. Möchte man eine Applikation entwickeln die größere Teile des Marktes erreicht, muss diese auf verschiedenen, konkurrierenden Controllern lauffähig sein, und dazu ist es derzeit unumgänglich das eigene Programm für jeden einzelnen Controller, beziehungsweise dessen NBI umzuschreiben. Dies stellt auch eine Verschwendung von Ressourcen dar, da Ressourcen, die derzeit für die Portierung der Applikationen auf verschiedene Systeme und die Pflege der Programme auf den verschiedenen Systemen, bei Existenz eines Standards sinnvolleren Verwendungen, wie beispielsweise der Steigerung der Performance oder der Usability zugunsten kommen könnten. Sowohl auf standardisierten Schnittstellen aufbauende Applikationen, als auch standardisierte Schnittstellen unterstützende Controller erreichen folglich mehr Endnutzer. Dies bringt weitere Vorteile, wie zum Beispiel, dass auch vergleichsweise kleine Unternehmen größere Chancen auf dem SDN Markt haben. Dadurch, und dadurch, dass verschiedene Controllerhersteller nun nicht mehr hauptsächlich über die Gestaltung der Northbound API, beziehungsweise Anwendungsentwickler nicht mehr hauptsächlich über den zugrundeliegenden Controller konkurrieren, sondern Faktoren wie Performance, Stabilität und Sicherheit sehr viel größere Bedeutung erhalten würden, könnte die veränderte Konkurrenzsituation den Markt auch stark beleben. Man sollte an dieser Stelle auch bedenken, dass das Wichtigste im SDN Umfeld die Applikationen sind. Denn auch wenn die Controller und die Netzwerkgeräte auf der Data Plane wichtige Bestandteile der SDN Hierarchie sind, stellen sie letztendlich nur die Werkzeuge dar, um auf dem Netzwerk ausgefeilte und mächtige Applikationen realisieren zu können. Damit sind Bestrebungen, die die Anwendungsentwickler in ihrer Arbeit unterstützen, besonders wichtig und, da eine Standardisierung des NBIs natürlich insbesondere den Anwendungsentwickler zugute kommen würde, gilt es den Wert der vorgestellten Vorteile besonders stark zu gewichten [12][6][19].

4.3 Anforderungen an ein NBI

Um erfolgreich einen neuen Controller und dessen NBI entwickeln zu können, muss der Entwickler zu Beginn genau spezifizieren, welche Anforderungen er an jene stellen möchte. Dies ist selbstverständlich auch bei der Entwicklung eines standardisierten NBIs, beziehungsweise eines NBI Standards, nötig. Hier gestaltet sich diese Aufgabe jedoch deutlich komplexer, da ein Standard möglichst viele der unterschiedlichen und sich teilweise auch widersprechenden Anforderungen abdecken soll: Eine sinnvolle Anforderung ist beispielsweise ein für den Anwendungsentwickler angemessener Funktionsumfang, angemessener Abstraktionsgrad und angemessene Komplexität. Doch wer ist der Entwickler und was ist für ihn angemessen? Entwickler bei Internet Service Providern oder in Rechenzentren werden meist nicht durch hohe Komplexität beeinträchtigt. Sie sind unter Umständen sogar auf einen großen Funktionsumfang und ein niedriges Abstraktionslevel angewiesen. Netzwerkadministratoren

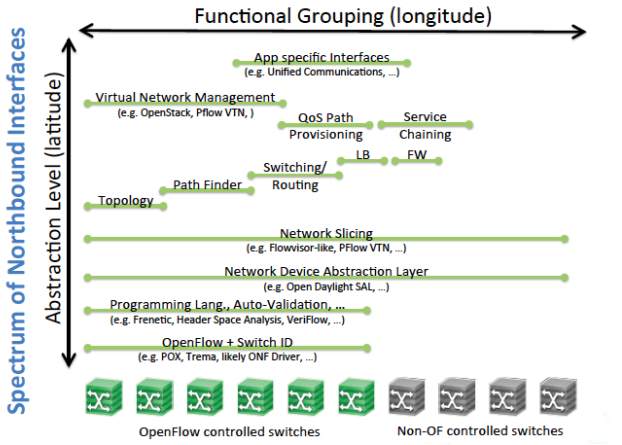

Abbildung 2: Abstraktionslevel am Northbound Interface, entnommen aus [19]

oder kleine Unternehmen, die SDN Apps entwickeln wollen, deren Schwerpunkt möglicherweise gar nicht auf SDN liegt, hingegen benötigen dies nur selten und favorisieren geringe Komplexität und einen etwas höheren Grad an Abstraktion bezüglich der eigentlichen Kommunikation zwischen Controller und Switches.
Eine Übersicht über verschiedene Ebenen, auf welchen SDN-Anwendungsentwickler operieren können, findet sich in (Abb. 2), einer Abbildung aus dem initialen Charter der NBI Working Group [19]. Wie dargestellt, arbeiten Entwickler von Virtual Network Management Applikationen auf einem viel höheren Level als Low-Level Controller wie POX. Somit erweist sich ein Low-Level NBI wie das von POX für diese Entwickler häufig als ungeeignet. Auch bei der Abstraktion von der tatsächlichen physischen Topologie verhält sich dies ähnlich. Entwickler, die Anwendungen entwickeln, die von der realen Topologie weitgehend unabhängig sind, wollen häufig auch nur möglichst wenig von dieser wahrnehmen, um beispielsweise eine einfache Wiederverwendbarkeit der Module gewährleisten zu können. Entwickler in Rechenzentren hingegen, die ihre Topologie genau kennen und wissen, dass sich diese nur selten verändert, möchten häufig dieses Wissen auch ausnutzen, um zum Beispiel möglichst effizientes Load Balancing betreiben zu können. Einige Entwickler bevorzugen bestimmte Lizenzmodelle, unter welchen Controller und damit auch NBI veröffentlicht werden. Weitere Anforderungen an eine Northbound API können aus Anforderungen hervorgehen, die man an den zugrundeliegenden Controller stellt. Dazu zählt beispielsweise gute Skalierbarkeit oder die Möglichkeit des Betriebs eines verteilten Controllers. Auch solche Anforderungen haben Einfluss auf die Gestaltung der jeweiligen Northbound API und können ebenfalls mit anderen Anforderungen, wie einem möglichst großen Funktionsumfang oder möglichst geringer Komplexität in Konflikt stehen.
Letztendlich lässt sich also festhalten, dass ein abgeschlossener, wohl definierter Satz von Anforderungen an das NBI, der alle Wünsche sämtlicher Entwickler abdeckt, nicht existieren kann. Stattdessen hängen diese immer vom individuellen Verwendungszweck ab, was bereits die Kernproblematik des nächsten Abschnitts darstellt [14][19][6].

4.4 Probleme einer Standardisierung

In Abschnitt 4.2 wurden die Vorteile vorgestellt, die eine Standardisierung mit sich bringen würde. Dennoch existiert wie in Abschnitt 4.1 dargelegt bisher noch kein Standard. Dies liegt zunächst einmal an der Vielzahl an verschiedenen Akteuren in Forschung, Entwicklung und auf dem freien Markt, wovon jeder unterschiedlichste Interessen, Vorstellungen und Visionen über die (optimale) Zukunft von SDN vertritt. Jeder dieser Akteure verwendet SDN für seine individuellen Anwendungszwecke, was wie in Abschnitt 4.3 beschrieben dazu führt, dass jeder ein individuelles Set an Anforderungen mitbringt und daraus die Frage entsteht, welche dieser konkurrierenden Anforderungen der Standard verwirklichen soll. Durch die Vielzahl und Vielfalt an Anforderungen gehen die meisten Experten davon aus, dass eine einzige API nicht ausreichen wird und ein möglicher Standard mehrere APIs standardisieren müsste [5][11]. Ein anderer Ansatz wäre, die Schnittstellen stark erweiterbar zu gestalten. Daraus ergibt sich die Fragestellung wie granular man den Standard aufbauen sollte und die Gefahr, dass bei zu großer Erweiterbarkeit, beziehungsweise der zu starken Fragmentierung des Standards in verschiedene APIs, nach einigen Jahren der Verbesserungen, Weiterentwicklung, Anpassung des Standards an neue Anforderungen und dem daraus resultierenden Aufkommen von nicht kompatiblen Versionen, die Situation am NBI wieder genauso komplex, facettenreich und mit Redundanz versehen sein wird, wie sie zum jetzigen Zeitpunkt schon ist. Ein weiteres Problem sind Akteure, die nicht am Erfolg von auf offenen Standards basierendem SDN interessiert sind, sondern es vorziehen würden, ihre Marktmacht ausnutzen zu können, um mit ihren proprietären Lösungen den Markt zu beherrschen. Generell ist der von Software geprägte Markt am NBI sehr dynamisch und verändert sich mitsamt den gängigen Anforderungen relativ schnell, was an sich schon im Widerspruch zu den meist langwierigen Standardisierungsprozessen steht. Standardisierung von hardwareunabhängigen Softwarekomponenten gestaltet sich meist schwierig. Aufgrund all dieser Probleme stehen einige Akteure einer Standardisierung generell skeptisch gegenüber, sodass die Gefahr besteht, dass jene einen veröffentlichten Standard einfach nicht annehmen. Hieraus folgt auch bereits das größte Hindernis, welches ein möglicher Standard überwinden müsste: Mangelnde Marktakzeptanz. Software-Standards sind bei Weitem nicht so bindend wie Hardware-Standards, und damit ist Marktakzeptanz bei Standardisierung von Softwareschnittstellen viel bedeutender. Ein Standard den niemand benutzt ist nutzlos und im eigentlichen Sinn auch kein Standard [6][13][14].

4.5 Anforderungen an einen NBI Standard

Aus den Hindernissen, die der Standardisierung des NBIs im Wege stehen, lassen sich schließlich auch die Anforderungen an einen NBI Standard ableiten. Man sollte zunächst bei der Gestaltung des Standards vorsichtig mit Inkompatibilität verursachender Erweiterbarkeit umgehen, und stets versuchen, bei der Weiterentwicklung des Standards die entstehenden Versionen kompatibel zu halten. Redundanz ist ebenfalls schädlich und gefährdet jede Form von Standard. Deshalb muss besonders bei der Weiterentwicklung die Einführung redundanter Komponenten in den Standard verhindert werden. Bei Use-Case basierten Ansätzen müssen zukünftige Veränderungen der Use-Cases schon im Vorfeld mit eingeplant und bei der Weiterentwicklung auch neue, gängige Use-Cases mit aufgenommen werden. Dabei sollten nur die wichtigsten Use-Cases miteinbezogen werden. Ein Standard wird nicht sämtliche Anwendungsgebiete bis ins Detail abdecken können, weshalb es wichtig ist die Grenzen des Standards von Anfang an klar abzustecken. Auch hier hängen die konkreten Anforderungen an den Standard wieder von den anvisierten Verwendungszwecken ab [14]. Schließlich sollte ein Standard nach gängiger Meinung (z.B. in [19]) auch unabhängig von konkreten Programmiersprachen gestaltet werden. Auch die ONF will wenn dann nur ein abstraktes Datenmodell standardisieren. Konkrete Implementierungen soll der Markt liefern. Eine Referenzimplementierung sollte jedoch schon vor der eigentlichen Standardisierung in den Markt eingeführt werden, da sich nur so die wichtigste Anforderung an einen Standard erfüllen lässt: Marktakzeptanz erlangen [19][14].

4.6 Unterschied im Standardisierungsprozess von North- und Southbound Interface

Es wurde nun in Abschnitt 3.1 das Southbound Interface OpenFlow und dessen Entwicklung zum Standard behandelt und in Abschnitt 4 ausführlich auf das NBI und dessen Standardisierung eingegangen, sodass nun rekapituliert und die Unterschiede im Standardisierungsprozess von North- und Southbound Interface diskutiert werden kann. Das Southbound Interface definiert Anforderungen an Hardware, Hardwareentwicklungszyklen sind lang und Hardware Entwicklung daher sehr kostspielig. Da diese großen Investitionen meist nur von den großen, traditionellen Netzwerkgeräteherstellern getätigt werden können, haben diese bei Entwicklung am Southbound Interface einen großen Vorteil. Kleine Unternehmer und Newcomer sind benachteiligt. Der Markt ist deshalb verhältnismäßig statisch und beherbergt nur wenige Akteure, weswegen sich Standardisierungsprozesse dementsprechend unproblematisch gestalten. Das NBI hingegen ist eine Softwareschnittstelle. Hier können sich kleine Hersteller einfacher und schneller etablieren und es sind dadurch mehr Akteure auf dem Markt tätig. Er ist diversifizierter, dynamischer und regelt vieles selbst. Die Standardisierung des NBIs in diesem dynamischen Umfeld ist, wie bei Standardisierung von Softwareschnittstellen häufig zu beobachten, sehr schwierig [6][11].

5. NORTHBOUND API PYRETIC

Abschließend soll in dieser Arbeit ein konkretes NBI beschrieben werden. Die Wahl fiel hierbei auf die Open Source Plattform Pyretic. Denn Pyretic bietet eine übersichtliche, einfach zu verstehende und relativ abstrakte Northbound API, was einen starken Kontrast zu den NBIs einiger häufig benutzter Controllerplattformen darstellt, wie beispielsweise der verbreiteten Plattform NOX, welche im engeren Sinn nur Funktionen liefert, die als Wrapper für die OpenFlow Kontrollnachrichten betrachtet werden können. Pyretic wurde vor allem an den Universitäten Cornell und Stanford entwickelt und 2013 in [17], beziehungsweise [23], vorgestellt. Pyretic gehört zu der bereits 2010 vorgestellten Frenetic Familie, verfügt im Gegensatz zu Frenetic allerdings über sequentielle Komposition (siehe Abschnitt 5.2), ist anstelle von Ocaml in die Programmiersprache Python eingebettet und basiert in der aktuellen Implementierung auf dem Controller POX, statt wie die Implementierung von Frenetic auf dem Controller NOX. Durch die von Pyretic gebotene Möglich-

doi: 10.2313/NET-2014-08-1_13

keit in wenigen, leicht verständlichen Zeilen Code mächtige Applikationen zu schreiben, in Zusammenspiel mit der Pyretic Laufzeit, welche inkonsistente Konfiguration der OpenFlow Switches weitgehendst verhindert, wird das Risiko von ungewolltem oder unvorhersagbarem Netzwerkverhalten minimiert und schnelles Prototyping vereinfacht. Ein Lizenzmodell im Stil der Berkeley Software Distribution (BSD) Lizenz erfüllt dabei die Anforderungen von kommerziellen, wie auch von nicht kommerziellen Entwicklern (vgl. [23], S. 1). Einen Blick auf Pyretics NBI zu werfen ist deshalb durchaus sinnvoll, auch wenn Pyretic an sich, wie in Abschnitt 4.4 beschrieben, keinen großen Stellenwert auf dem SDN Markt inne hat.

5.1 Das abstrakte Paket Modell

Aus Sicht Pyretics sind Pakete Wörterbücher, die bestimmte Felder, wie Header Informationen, Paketdaten, Standort, also das Tupel aus der Identität eines virtuellen oder physischen Gerätes und dem jeweiligen Port, aber auch virtuelle, selbst definierte Headerfelder, auf ihre tatsächlichen Werte, beziehungsweise einen Stack von Werten, mappen. Dieser Stack wird dazu benutzt, verschiedene Ebenen der Abstraktion zu realisieren. So kann zum Beispiel der Standort Wert eines Paketes ein Stack sein, der auf der untersten Ebene die Identität des physischen Switches, in dem sich das Paket gerade befindet, und in verschiedenen Ebenen darüber die Identitäten der virtuellen Switches der virtuellen Netze, in dem sich das Paket logisch aufhält, enthalten. Auf diesen Grundkonzepten aufbauend bietet Pyretic ein ausgefeiltes Konzept zur Topologieabstraktion. Dieses basiert auf „Network Objects“, die aus einer Topologie, einer Policy und einem Mapping, welches die Elemente der Topologie mit denen der darunterliegenden Topologie verknüpft, bestehen. Pyretic bietet dabei Funktionalitäten, die es ermöglichen automatisch Mappings zu generieren um Veränderungen einer Topologie auf andere Topologieabstraktionsebenen abzubilden oder semantisch gleiche Policies auf verschiedene Topologien zu übertragen. Mithilfe dieser Network Objects lassen sich auch die Konzepte „information hiding“ und „information protection“, also Zugriffssteuerung auf Informationen, umsetzen [17].

5.2 Modularisierung und Komposition

In modernen SDN-Applikationen ist besonders im industriellen Umfeld und bei Verwendung der gängigen „Low Level NBI Controllern“ hohe Komplexität ein großes Problem. Pyretic versucht sich diesem Problem anzunehmen, indem es dem Entwickler erlaubt seinen Code sehr modular zu gestalten und sämtliche Funktionalität in Modulen zu kapseln. Solche Module lassen sich dann einfach durch zwei verschiedene Kompositionsoperatoren verknüpfen. Die erste Möglichkeit ist die Verknüpfung zweier Module mittels paralleler Komposition, ausgedrückt durch den + Operator. Hierbei werden die Module unabhängig voneinander parallel ausgeführt, was ungewollte Interferenzen zwischen den Funktionen und damit Inkonsistenzen verhindert. Die zweite Möglichkeit ist die sequentielle Komposition, welche durch den << Operator dargestellt wird. Diese sorgt dafür, dass Pakete, die vom ersten Modul zurückgegeben werden dem zweiten Modul als Eingabe dienen. Dieses Modell ermöglicht starke Modularisierung des Codes, wodurch die Portabilität und Wiederverwendbarkeit von Codesegmenten gesteigert, und damit auch Redundanz im Code minimiert werden kann.

5.3 High-Level Policy Funktionen

Die Bewegung und Verarbeitung von Paketen wird in Pyretic durch Funktionen dargestellt, die ein Paket als Eingabe erhalten und eine Menge von Paketen zurückgeben. Diese Menge kann sowohl leer sein, wie zum Beispiel im Falle einer DROP Policy, oder auch ein oder mehrere verschiedene Pakete beinhalten. Die Werte der Felder der abstrakten Paketdarstellung können bei diesen Paketen je nach Definition der Funktion beliebig verändert werden. So wird eine einfache, statische Weiterleitung eines Paketes, welche alle Pakete die an Port A eines Netzwerkgerätes X ankommen unverändert auf Port B wieder ausgeben soll, durch eine Funktion umgesetzt, die bei einem eingehenden Paket mit den Standort Werten „Gerät X, Port A“ ein einzelnes Paket zurückgibt, bei dem der Wert des Standort Feldes, spezifischer des Port Feldes, auf B gesetzt ist und die restlichen Werte vom eingehenden Paket übernommen wurden. Auf diesem Modell aufbauend werden verschiedene einfache High-Policy Funktionen definiert, die als NetCore bezeichnet werden: Es gibt zunächst einige leicht zugängliche Primitive:

A ::= drop | passthrough | fwd(port) | flood | push(h=v) | pop(h) | move(h1=h2)

Diese werden durch Prädikate ergänzt, die sich durch einfache boolesche Operatoren verknüpfen lassen:

P ::= all_packets | no_packets | match(h=v) | ingress | egress | P & P | (P | P) | ∼ P

Damit der Controller Informationen von den Switches abfragen kann, werden Query Policies eingeführt. Diese übergeben die angeforderten Informationen an „Bucket“ genannte Datenstrukturen, auf welche „Listeners“ wie Callback-Funktionen registriert werden können. *packets(n,[h])* übergibt die jeweils ersten n Pakete, die gleiche Werte in den in der Liste *[h]* bestimmten abstrakten Paketfeldern vorweisen, vollständig und unverändert an das Bucket *packet_bucket*. *counts(every,[h])* hingegen ruft alle *every* Sekunden alle registrierten Listener auf und übergibt ihnen über das Bucket *counting_bucket* eine Abbildung von Paketen der gleichen in *[h]* definierten Eigenschaften auf die Anzahl an Paketen, auf die diese Eigenschaften zutreffen.

Q ::= packets(limit,[h]) | counts(every,[h])

Diese einfachen Policies lassen sich dann mittels der bereits vorgestellten sequentiellen und parallelen Komposition verknüpfen, sich durch Anwendung eines Prädikates oder einer if-else Konstruktion konditionieren und sich somit zu komplexeren statischen Policies zusammenfügen.

C ::= A | Q | P[C] | (C | C) | C >> C | if_(P,C,C)

Des Weiteren kann eine Policy ihr eigenes Verhalten selbst modifizieren, wodurch die Möglichkeit geschaffen wird, dynamische, zur Laufzeit veränderliche Policies zu erzeugen. Ausführliche Beispiele und Erklärungen zu allen vorgestellten Inhalten finden sich in [17], [23] und [2].

5.4 Grenzen von Pyretic

Wie bereits erwähnt, bietet Pyretic selbst unerfahrenen Entwicklern die Möglichkeit in kurzer Zeit mit wenig Code mächtige Applikationen zu schreiben. Dennoch soll hier nicht der Eindruck erweckt werden, dass Pyretic revolutionär neue

Ideen umsetzt oder gar einen großen Stellenwert auf dem Markt einnimmt. Pyretic ist ein gutes Beispiel für eine einfache NBI-Sprache, die schöne Konzepte im Hinblick auf Abstraktion bietet und Lösungen für aktuelle NBI-Probleme sucht, aber dennoch klar gesetzten Grenzen unterworfen ist: Pyretic ist, auch durch die Verwendung von POX als Basis, einzig auf OpenFlow ausgerichtet. Unterstützung von anderen Southbound Interfaces ist nicht vorgesehen. Pyretic wurde bisher in keiner veröffentlichten Publikation evaluiert, weder im Hinblick auf Korrektheit, noch im Hinblick auf Usability, Stabilität, Skalierbarkeit oder der besonders wichtigen Performance. Da Pyretic auf POX basiert und dieser in Performanceuntersuchungen sehr schlecht abschneidet [25] und daher für realitätsnahe industrielle Entwicklung als unbrauchbar klassifiziert wird, ist davon auszugehen, dass auch Pyretic, zumindest in der aktuellen Implementierung, keine ausreichend gute Performance bietet, um sich über das Forschungsumfeld hinaus etablieren zu können [17]. Pyretic ist noch jung, weswegen bisher nur wenige Applikationen in Pyretic realisiert wurden. Und letztendlich ist Pyretic auch vergleichsweise stark an Python gebunden, was eine Portierung in andere Sprachen schwierig machen kann. Aufgrund dieser Umstände wird Pyretic selbst kaum große Bedeutung gewinnen, es liefert allerdings Ansätze für neu entwickelte Sprachen und wertvolle Anregungen für einen künftigen Standard.

6. FAZIT

Es wurde in dieser Arbeit das Konzept, die Anwendungsgebiete, der State of the Art und die Bedeutung von SDN erläutert. Es wurde der Einfluss und die Rolle von OpenFlow als Southbound Interface behandelt und detailliert das NBI aus unterschiedlichen Blickwinkeln betrachtet. Im letzten Kapitel wurden einige der etwas abstrakteren Überlegungen anhand des Überblicks über eine konkrete Northbound API veranschaulicht. Aufgrund dieser Ausführungen wurde die derzeit gegebene Situation am NBI, die Probleme, die es aktuell zu bewältigen gilt, und die Fähigkeiten und das Wissen, moderne Trends und Entwicklungen am NBI zumindest grundlegend bewerten zu können, in ausreichendem Maße vermittelt. Dem Leser sollte nun bewusst sein, wie stark die Entwicklungen am NBI sämtliche Bereiche des Software Defined Networkings prägen, und welch großen Einfluss diese Entwicklung dadurch auch auf den gesamten Netzwerksektor hat. Somit bleibt an dieser Stelle nur noch festzuhalten, dass das NBI nicht nur für SDN Entwickler von Interesse ist, sondern in der Tat eine innovative Internettechnologie darstellt, also den Grundgedanken des Seminars, in dessen Rahmen diese Ausarbeitung verfasst wurde, in jeder Beziehung trifft, und daher für alle, die sich mit modernen Netzwerktechnologien auseinandersetzen oder einfach nur aktuelle Innovationen in der Informatik mitverfolgen wollen, ein attraktives Themengebiet verkörpert, und die Empfehlung auszusprechen, die Entwicklung, die das Thema Northbound Interface in den nächsten Monaten und Jahren erfahren wird, ein bisschen im Auge zu behalten.

7. LITERATUR

[1] OpenFlow Inventor Martin Casado on SDN, VMware, and Software Defined Networking Hype. `http://www.enterprisenetworkingplanet.com/netsp/openflow-inventor-martin-casado-sdn-vmware-software-defined-networking-video.html`, aufgerufen am 07.06.2014.

[2] Pyretic Homepage. `http://frenetic-lang.org/pyretic/`, aufgerufen am 07.06.2014.

[3] Z. Bozakov. OpenFlow eine Softwareschnittstelle zur Netzprogrammierung. *ix Magazin für professionelle Informationstechnik*, 8(9):112–115, Juli 2012.

[4] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000.

[5] R. Gewirtz. ONF to standardize northbound API for SDN applications? `http://searchsdn.techtarget.com/news/2240206604/ONF-to-standardize-northbound-API-for-SDN-applications`, aufgerufen am 07.06.2014, Oktober 2013.

[6] I. Guis. The SDN Gold Rush To The Northbound API. `http://www.sdncentral.com/technology/the-sdn-gold-rush-to-the-northbound-api/2012/11/`, aufgerufen am 07.06.2014, November 2012.

[7] U. Holzle. OpenFlow @ Google. `http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf`, aufgerufen am 07.06.2014.

[8] S. Johnson. Do SDN northbound APIs need standards? `http://searchnetworking.techtarget.com/feature/Do-SDN-northbound-APIs-need-standards`, aufgerufen am 07.06.2014.

[9] S. Johnson. A primer on northbound APIs Their role in a software-defined network. `http://searchsdn.techtarget.com/feature/A-primer-on-northbound-APIs-Their-role-in-a-software-defined-network`, aufgerufen am 07.06.2014, Dezember 2012.

[10] K. Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9):16–19, September 2013.

[11] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky und S. Uhlig. Software-defined networking: A comprehensive survey. *CoRR*, Juni 2014.

[12] D. Lenrow. The Most Important Work in SDN: Have We Got It Backward? `http://www.sdncentral.com/education/important-work-sdn-got-backwards/2014/04/`, aufgerufen am 07.06.2014, April 2014.

[13] S. McGillicuddy. ONF launches SDN northbound interface working group. `http://searchsdn.techtarget.com/news/2240207495/ONF-launches-SDN-northbound-interface-working-group`, aufgerufen am 07.06.2014, Oktober 2013.

[14] S. McGillicuddy. ONF Northbound Interface Group chair: No need for standards yet. `http://searchsdn.techtarget.com/news/2240210605/ONF-Northbound-Interface-Group-chair-No-need-for-standards-yet`, aufgerufen am 07.06.2014, Dezember 2013.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker und J. Turner. OpenFlow: Enabling Innovation in Campus

Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, März 2008.

[16] J. Metzler. Where Do We Stand with SDN's Northbound Interface - Webtorials. `http://www.webtorials.com/content/2014/04/where-do-we-stand-with-sdns-northbound-interface.html`, aufgerufen am 07.06.2014, April 2014.

[17] C. Monsanto, J. Reich, N. Foster, J. Rexford und D. Walker. Composing Software-defined Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, Seiten 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[18] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`, aufgerufen am 07.06.2014, April 2012.

[19] Open Networking Foundation. Northbound Interfaces. `https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf`, aufgerufen am 07.06.2014, Oktober 2013.

[20] Open Networking Foundation. OpenFlow Switch Specification Version 1.4.0 (Wire Protocol 0x05). `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`, aufgerufen am 07.06.2014, Oktober 2013.

[21] Open Networking Foundation. SDN Architecture Overview. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf`, aufgerufen am 07.06.2014, Dezember 2013.

[22] B. M. Posey. What You Need to Know About Software Defined Networking in Hyper-V. `http://www.virtualizationadmin.com/articles-tutorials/microsoft-hyper-v-articles/networking/what-you-need-know-about-software-defined-networking-hyper-v-part1.html`, aufgerufen am 07.06.2014, März 2014.

[23] J. Reich, C. Monsanto, N. Foster, J. Rexford und D. Walker. Modular SDN Programming with Pyretic. *USENIX ;login*, 38(5):40–47, Oktober 2013.

[24] SearchSDN. Northbound API guide: The new network application. `http://searchsdn.techtarget.com/guides/Northbound-API-guide-The-rise-of-the-network-applications`, aufgerufen am 07.06.2014.

[25] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov und R. Smeliansky. Advanced Study of SDN/OpenFlow Controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, Seiten 1–6, New York, NY, USA, 2013. ACM.

[26] The OpenDaylight Project. Technical Overview. `http://www.opendaylight.org/project/technical-overview`, aufgerufen am 07.06.2014.

[27] A. Vahdat. Enter the Andromeda zone - Google Cloud Platform's latest networking stack. `http://googlecloudplatform.blogspot.de/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html`, aufgerufen am 07.06.2014, April 2014.

doi: 10.2313/NET-2014-08-1_13