Separating network control from routers with Software Defined Networking

Shpëtim Latifi
South East European University
Bul. Ilindenska 335
Tetovo, FYR of Macedonia
sh.latifi@seeu.edu.mk

Arjan Durresi
IUPUI
723 W. Michigan St., SL 280M
Indianapolis, IN 46202, USA
durresi@cs.iupui.edu

Betim Cico
South East European University
Bul. Ilindenska 335
Tetovo, FYR of Macedonia
b.cico@seeu.edu.mk

ABSTRACT

Data networks have become increasingly complex nowadays. Even though technologies like Ethernet, IP protocol and packet forwarding is rather simple, control mechanisms like middleboxes, Access Control Lists (ACLs), firewalls, traffic engineering, VLANs, etc. have largely contributed to increasing their complexity. Primarily this is due to the lack of basic principles in networking. Networking still remains vertically integrated, where hardware comes with its proprietary software and is not open to innovation.

Software-Defined Networks (SDN) instead decouple the data plane (which is and should remain the job of the physical routers) and control plane. The control plane in SDN is removed from the routers and switches, and instead is done in the edge of the network, thus allowing for third party software, open interface to devices regardless of hardware type and vendor, and easier management of networks. SDN is a new design model in networks rather than a new technology. It is a set of abstractions for the control plane rather than implementation mechanisms;

SDN in essence offers the possibility to network programmers and third party app writers build anything they want on top of both router chips (data plane) and the Network operating system (now through OpenFlow, but it may be something else as well) in the control plane, as well as on top of the Network Operating system due to the open interface it introduces.

Categories and Subject Descriptors

C.2.1 **[Network Architecture and Design]**: Packet-switching Networks; C.2.2 **[Network Protocols]**: Routing protocols

General Terms

Design, Theory, Management, Reliability

Keywords

Software-Defined Networks, abstractions, control plane, packet forwarding

1. INTRODUCTION

Software Defined Networking is not a revolutionary technology, it's an organizing principle in data networks. The rationale behind SDN is more important than its design. In 2008 the SDN elements like the Network Operating System (NOX) and OpenFlow switch interface were defined[4,5,14]. In 2011 the Open Networking Foundation was founded and now it has over 90 companies, among which Google, Cisco, Dell, IBM, Intel, Facebook, Verizon, Arista, Brocade, etc. [3]. Google publicly announced that they will use SDN for their interconnecting their data centers in 2012. SDN is now commercial and in production, although not so widely.

In order to explain SDN and the rationale behind it, it is important to draw a comparison between networks and software systems. A software system is a modular system based on abstractions to solve problems. A modular systems allows for code reuse, change implementation and separate functions. To solve a problem we then should come up with abstractions, which is turn means to decompose the problem into its basic components, and then each components needs to have its own abstraction. These abstractions require an implementation to solve one specific task. Based on the complexity or the hardness of the task, it may again require to go back the step one, until the implementations solve tasks that are easy to implement.

In data networking there are two planes: data plane, which processes packets with local forwarding state. The forwarding decision here is done based on the forwarding state compared to packet header.

The second plane in data networking is the control plane. It puts the forwarding state in the networking device, based on many possibilities and implementations. It can be computed using distributed or centralized algorithm, manually configured, etc, but regardless of this, it is a completely different function as opposed to the forwarding plane[1,6,7].

The abstractions that we have for the data plane are basically known to every network engineer or even computer scientist.

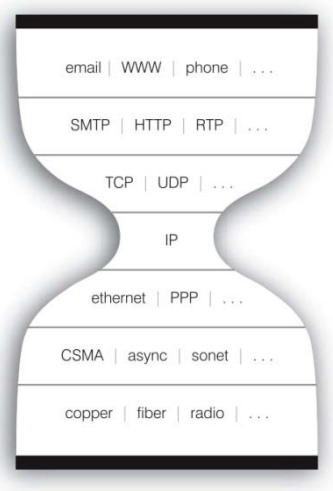

Figure 1. Abstractions and layering of data plane as a reason for Internet success

They are known as the protocol suites and the layering system used in data networking, namely TCP/IP. This layering model has been very successful in decomposing the problem in its basic components, and each implementation in this model solves a specific task. Applications are built on reliable end-to-end transport, built on best effort global packet delivery (network layer), which in turn is again build on best effort local packet delivery (link layer) on top of physical delivery of bits. Each layer is separate on top on the layer below[7].

The control plane on the other side doesn't have abstractions. We have a lot of mechanisms, which serve different goals. For example routing (a family of algorithms), isolations (VLANs, firewalls, traffic engineering, MPLS, etc.), but there is no modularity. And the functionality is limited. The network control plane is what happens when there is mechanisms without abstractions. So there is too many mechanisms without enough functionality. Each problem is solved individually and from scratch. This is not the way problems should be solved. Instead the problem should be first decomposed. That is the main reason that has lead to a great success and acceptance of Software Defined Networking[12,13].

SDN imposes a big change in the industry just like computer industry was changed in the late 80es. At that time the computer industry was based on specialized hardware, specialized OS and specialized apps (usually all from one vendor, namely IBM). The computer industry was vertically integrated, close proprietary and relatively slow to innovations. With the microprocessor, an open interface led to many operating systems and a huge number of apps on top of these Operating systems. Hence, this industry moved from closed and very difficult for innovations, vertically integrated and proprietary, to horizontal, fast innovation and open. Networking too has for a long time worked in a same way, based on specialized hardware, software and features. SDN in essence offers the possibility to network programmers and third party app writers build anything they want on top of both router chips (data plane) and the Network operating system (now through OpenFlow, but it may be something else as well) in the control plane, as well as on top of the Network Operating system due to the open interface it introduces (the control program).

2. THE NETWORK CONTROL PLANE

The Control plane in a network should compute the forwarding state under three constraints[1,7]:

1. The forwarding state should be consistent with specific low level hardware/software,

2. It should be done based on entire network topology, and

3. It should be implemented in every router.

To take care of these constraints, network designers should define specific abstraction for each problem component.

1.The compatibility with specific low level hardware/software needs an abstraction for a general **forwarding model** that hides the details of specific hardware/software;

2. Being able to make **decisions based on entire network** takes another abstraction for the network state, hiding the mechanisms to get it; and

3.Another abstraction that deals with the **actual configuration of each network device**, so they are configured in a much easier and straightforward way.

2.1 The Forwarding Model in SDN

For the forwarding abstraction, we want to hide details of the type of hardware or software the decision in used at. The device itself may by manufactured by any device manufacturer and still the model should work in a same way. **OpenFlow** is the current proposal for that[6,15]. It is a standard interface to a switch so we can access the switch and we can store there flow entries through this protocol. It is a general language which should be understood by any switch. Conceptually this is a pretty easy and straightforward concept, although its practical implementation of design details (like header matching, allowed actions, etc.), may not be so[8]. The forwarding abstraction (implemented through the OpenFlow protocol) exploits the flow table in the routers and populates them with simple rules (if header x, forward to port y, etc.). It does a:

match + action,

in a similar way it is done today in networks. The set of actions in OpenFlow is rather small (forward packet to specific or set of ports, drop the packet, or send it to the control plane, and also define bit rate at which packet will be forwarded. The most

interesting thing to look at with OpenFlow is rather the *action* part of the function, because the desired goal here it to use a minimal set of actions to be a good enough set of actions to do most work on one hand, and offer the possibility to chip vendors to implement it and program writers to offer special features that makes them unique in the market, on the other hand. Eventually the OpenFlow should offer protocol independence to build different types of networks, like Ethernet, VLANs, MPLS, etc, and new forwarding methods which are also backward compatible and technology independent[17,18,19,22].

2.2 The Network State in SDN

In routing, we want to abstract the way how the distributed algorithms get the global network state, and instead only give a global network view, annotating things and information relevant to the network administrator, like delay, capacity, error rate, etc, so the network admin is able to program the switches the way they desire. It is implemented through the Network Operating System in SDN, which runs on servers on the network, and the information flows in both ways, which means the servers get the information about the network state by querying the switches to create the global network view. Based on what policy we want to implement in the switches, we do it in the opposite direction (router configuration). The Network Operating System gets the information from the network routers to create a global network view, and then a control program implemented on top of the Network Operating System implements policies about routing, access control, traffic engineering, etc. This is a major change in the networking paradigm, where the control program implements the policies into the routers [1].

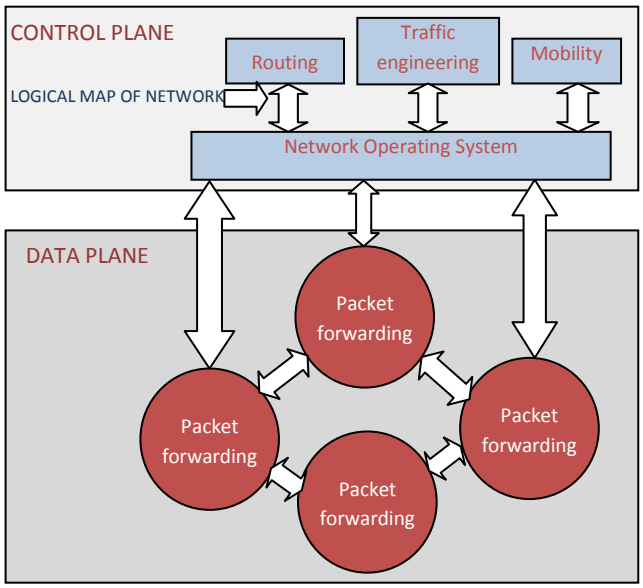

Figure 2. The Control Plane running on the edge, separately from the Data Plane in a SDN

2.3 Router Configuration in SDN

The control program needs to install the policies and flow entries intro each router in the network. But the control program needs to just express desired behavior, and not be responsible for specific statements. Another abstraction deals with writes of specific statements in routers. Rather than the control program dealing with the actual full network topology, it only deals with a virtual layer created on top of the full global network topology. This way each part of the problem is decomposed in a clean way, and each abstraction deals with its own task. That is, the control program expresses the desire for some specific network configuration on one or more routers, and the specification virtualization layer does the actual mechanics of implementing it on the actual routers in the network.

The SDN's achievement is not to eliminate complexity because the layers and the network operating system are still very complex and complicated. Its achievement however is to simplify the interface to the control program so that it has a simple job to specify what we want to do with the network. The hard part is the reusable code, and once it is done right, it will be used by any network programmer without the concern of knowing the details of its implementation. That part is implemented in the operating system and the control program, which is the reusable code. The comparison here is with programming languages and compilers. The programmer needs not to know how the compiler is implemented, not should be familiar with the instruction set, because in programming those two problems have been decoupled a long time ago. This is something that has not yet been done in networks, and that is the major goal SDN accomplishes in the data networking field [5,11,13,15].

3. APPLICATIONS OF SDN

In today's networks we can easily says that topology is policy, meaning the actual physical location of routers, firewalls, etc. dictates how effective the network is, how well the ACLs work, what our broadcast domains are, etc. ***When networks are moved to the cloud, we usually want to keep same policies***, but very few networks operators have an abstract expression of network policy, rather they have a network topology. SDN allows to specify a logical topology to the cloud. The cloud then ignores the physical topology and follows the logical topology based on the policy read based on the topology initially.

The function is evaluated on an abstract network and only the compiler needs to know the actual physical network topology. The major changes that SDN brings to the networking world in general is not the easier network management (which comes as a result of it), but rather primarily decouples the data plane from the control plane. They are now the same in terms of vendor, and place where they are implemented. This changes the business model in networking (hardware bought separately from software,

which can be third party). But it also brings a clean interface which allows for much easier implementation of testing of networks in the split architecture.

3.1 Simplified Network Troubleshooting

SDN allows for implementation of control function in the edge of the network, instead of in the routers, where it actually is, and the core only deals with delivering packets end-to-end. The core may easily remain legacy hardware and the network operators need not to know at all the one is implementing SDN from the own edge of the network. So it simplifies network troubleshooting and also there are no disruption periods during network convergence, due to the fact the one policy is implemented per specific flow, and each packet will be carried either by an old state, or by a new state. In traditional routing packets may well be lost during network convergence, or loops may appear. In SDN the expressions are high level and easily checked and corrected.

3.2 Network customization

The SDN gives an extra benefit to the network operators to more easily customize their networks based on their needs. The policies, traffic engineering, monitoring and security is easier to implement after getting a network state quickly from the Network Operating Systems, and moving the network to a virtual environment without much effort is perhaps the greatest benefit in this specific scenario. An example would be to improve load balancing in a network with many servers in different locations connected though a backbone. The load balancer used in networks today chooses a lightly loaded server to do the job. But since servers are in different location, the load balancer does not take into account to choose the lightly loaded path too. Ideally we would like to choose both, for a best result. SDN not only gives the possibility to do this, but more importantly, because the control plane now resides on the edge, it can be written by anyone and is vendor independent, it can be done in a very short period of time and tested in real time.

3.3 Third Party Apps in Networking

Also, network operators may hire third party people to develop special features for their networks, as well as remove unneeded features from routers. Removing some unused features from routers arguably increases their reliability of routers.

SDN offers a chance to increase the rate of innovation by moving the operation in software, standards will follow the software deployment instead of other way around. Another great opportunity here is experience, technology and innovation between vendors, universities and researchers.

As of now, there are already different domains where SDN has been or is under implementation, including data centers (Google for example), public clouds, university campus networks, cell phone backhauls [16,17,18,19,20,21], but also in enterprise Wi-Fi environments and home networks . There are already over 15 vendors offering SDN products, and probably the number will grow, including new jobs in this field.

SDN is an opportunity to program the network infrastructure in an easier way, by offering network wide visibility as well as direct control through OpenFlow.

Another significant advantage in SDN is that OpenFlow offers the ability to innovate on top of the low level interface that today's controllers provide, by increasing the level of abstraction. Today's controllers do not have a complete network wide view primarily for scaling purposes, whereas in SDN the control program has a complete network wide view, and it can actually use the virtualized information they need for the network state. It is hard to compose different tasks in today's networks (like monitoring, access control and routing).

4. Programming in SDN

SDN and OpenFlow have made possible to program the network. In a SDN there is a logically centralized controller and an arbitrary number of switches under its control. The controller by default is a smart and slow device, as opposed to the fast and dumb switches in the network, which only manage packets (based on the policy coming from the controller). The controller pushes the policies to switches through the OpenFlow API.

There are three aspects which dictate the way how SDN are programmed: 1. The data plane abstraction is very simple and the architecture is centralized with direct control over it. 2. The programming interface of OpenFlow API is relatively low-level with a number of limitations. The functionality when programming through OpenFlow is limited and tied to hardware, and the programmer needs to manage the resources explicitly, which in routers are scarce (similar to doing register allocation in Assembly language coding). 3. Probably the most difficult thing with OpenFlow programming is when combining different modules at the same time (like routing, monitoring, load balancing, etc.). The programmer would be able to do this much easily in a high programming language.

A new programming language, called Frenetic has been developed, allowing for network programming at a higher level of abstraction. Frenetic is a SQL-like query language, which allows composition of different modules possibly at the same time [11]. The following example illustrates how traffic statistics ban be collected:

```
Select(bytes)*
Where(in:1 & scrport:25)*
GroupBy([dstip])*
Every(30)
```

Figure 3. Count number of bytes with TCP port 25 coming in port 1, grouped by destination IP address every 30 seconds.

```
#Repeating between two ports of a repeater
def repeater():
rules=[Rule(in:2, [out:1]),
       Rule(in:1, [out:2])]
register(rules)
```

Figure 4. A repeater forwarding traffic from one port to another

```
#Monitoring web traffic
def traffic_monitor():
q = (Select (bytes)*
Where (in:1 & srcport:25) *
Every (45))
q >> print
```

Figure 5. A traffic monitor colleting incoming traffic data from port 1 with TCP port 25 every 45 seconds

```
#The previous modules composed into one in Frenetic
def main():
repeater()
traffic_monitor()
```

Figure 6. Composition of two modules in Frenetic

Figure 6 illustrates how composition of modules can be done in Frenetic, something which is not possible directly through OpenFlow interface.

5. Future Work

SDN is a new set of abstraction with many unanswered questions related to practical implementations. The issues of mobility, security and privacy will have to be addressed in the future, as new control programs emerge on top of existing physical infrastructures. The third party apps developed by different network programmers will have to be fully validated and checked for security holes before they are implemented in actual networks. But SDN also allows for rapid prototyping at software speeds, not having to wait for vendors to come up with new features in networks. This is where we will focus our research work in the future. We will try to implement new routing policies in existing network environments using SDN and Frenetic in a real network, thus adding extra features to our network that the actual routers do not have. We will also focus on mobility, as most network devices nowadays are mobile.

6. Conclusions

Networking over the years has been vertically integrated, where hardware comes with its proprietary software and is not open to innovation.

Software-Defined Networks (SDN) instead decouple the data plane (which is and should remain the job of the physical routers) and control plane. The control plane in SDN is removed from the routers and switches, and instead is done in the edge of the network, thus allowing for third party software, open interface to devices regardless of hardware type and vendor, and easier management of networks. SDN is a new design model in networks rather than a new technology. It is a set of abstractions for the control plane rather than implementation mechanisms.

SDN is merely a set of abstractions for the control plane. It is not a set of mechanisms. It involves a computing function and the Network Operating System deals with the distribution of state.

SDN in essence offers the possibility to network programmers and third party app writers build anything they want on top of both router chips (data plane) and the Network operating system (now through OpenFlow, but it may be something else as well) in the control plane, as well as on top of the Network Operating system due to the open interface it introduces.

7. REFERENCES

[1] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker. Ethane: Taking Control of the Enterprise. In *Sigcomm 2007.*

[2] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, J. van der Merwe. The Case for Separating Routing from Routers. In *FDNA 2004.*

[3] Members of the Open Networking Foundation. *https://www.opennetworking.org/membership/members*

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. OpenFlow: Enabling Innovation in Campus Networks. In *CCR 2008.*

[5] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker. NOX: Towards an Operating System for Networks. *In CCR 2008*

[6] M. Casado. What OpenFlow is (and more importantly, what it's not). http://networkheresy.com/2011/06/05/what-openflow-is-and-more-importantly-what-its-not/

[7] M. Casado, T. Koponen, S. Shenker, A. Tootoonchian. Fabric: A Retrospective on Evolving SDN. In *HotSDN 2012*

[8] M. Casado. The Scaling Implications of SDN. http://networkheresy.com/2011/06/08/the-scaling-implications-of-sdn/

[9] S. H. Yaganeh, A. Tootoonchian, Y. Ganjali. On the Scalability of Software-Defined Networking. In *IEEE Communications Magazine Feb 2013.*

[10] N. Foster, M. J. Freedman, A. Guha, R. Harrison, N. P. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, D. Walker. Languages for Software-

Defined Networks. In *IEEE Communication Magazine Feb 2013.*

[11] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker. Frenetic: A Network Programming Language. In *ICFP 2011.*

[12] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker. Composing Software-Defined Networks. In *NSDI 2013.*

[13] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker. Abstractions for Network Update. In *Sigcomm 2012.*

[14] B. Lantz, B. Heller, N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *HotNets 2010.*

[15] M. Canini, D. Venzano, P. Perešini, D. Kostić, J. Rexford. A NICE Way to Test OpenFlow Applications. In *NSDI 2012.*

[16] H. Kim, N. Feamster. Improving Network Management with Software Defined Networking. In *IEEE Communications Magazine Feb 2013.*

[17] A. Nayak, A. Reimers, N. Feamster, R. Clark. Resonance: Dynamic Access Control in Enterprise Networks. In *WREN 2009.*

[18] A. Tavakoli, M. Casado, T. Koponen, S. Shenker. Applying NOX to the Datacenter. In *HotNets 2009.*

[19] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, R. Johari. Plug-n-Serve: Load-balancing Web Traffic using OpenFlow. In *Sigcomm 2009 Demo.*

[20] M. Bansal, J. Mehlman, S. Katti, P. Levis. OpenRadio: A Programmable Wireless Dataplane. In *HotSDN 2012.*

[21] L. E. Li, Z. Morley Mao, J. Rexford. Towards Software-Defined Cellular Networks. In *EWSDN 2012.*

[22] List of OpenFlow Software Projects http://yuba.stanford.edu/~casado/of-sw.html