Software-defined networking

Kimmo Ahokas
Aalto University School of Science
kimmo.ahokas@aalto.fi

Abstract

Software-defined networking is a new network architecture that has attracted a lot of attention lately. SDN promises to make network devices simpler while giving network administrators better control over network and increasing network performance. This paper briefly introduces SDN architecture, related network architecture and some recent research related to SDN.

KEYWORDS: software-defined networking, sdn, simulation, experimental evaluation

1 Introduction

As the amount of devices connected to the Internet grows every day and more and more data is transferred from device to device scaling up traditional network architecture gets increasingly difficult. Routing tables in standard routers get too small for global routing, hardware gets too expensive and configuration errors are more common than ever before. It is inevitable that new solutions to these problems must be adopted in the industry.

One of the most-hyped terms in networking technology at the moment is software- defined networking (SDN). Software defined networking aims to separate network control from data-layer devices. The separation allows more fine-grained control while making network devices simpler. This paper introduces software-defined networking and some related research along with different research methodologies applied to software defined networking.

The paper is organized as follows: Section 2 introduces traditional network architecture, section 3 describes software-defined networking in more detail, section 4 briefly introduces other SDN-related research and different research methodologies applied to SDN and finally section 5 presents our conclusions.

2 Background

This section describes those parts of traditional network architecture that are relevant to software-defined networking. This includes network layers, switches, routers, firewalls and basics of routing protocols.

Traditionally network elements are defined by using the Open Systems Interconnection (OSI) model. This conceptual model has 7 layers, each of which can have multiple different protocols. Each layer serves the layer above and is served by the layer below. Ideally layers are independent and interact with adjacent layers using well defined application programming interfaces (API). Thus, it is possible to replace any protocol in single layer with different protocol that offers a similar service. A data unit from a higher layer is encapsulated inside a unit of lower layer. Thus, device working on certain layer only needs to check headers of that layer, without altering upper layer data.[1, chapter 10]

Simple devices usually function only on the lowest layers of OSI model, while more complicated devices handle higher layers as well. End nodes, such as standard desktop computers usually work on all of the layers of OSI model. Traditionally devices operating on higher layers are more complicated than devices operating on lower layers.

The simplest active elements in current networks are layer 2 switches. As the name suggests, these devices are concerned only with headers of layers 1 and 2. These devices relay traffic between different ports based on the packets data link layer destination address and possibly Virtual Local Area Network (VLAN) id [1, chapter 2]. As a layer 2 device switches work only inside one network an can not be used to connect multiple networks. Switches are self- learning devices, which means that forwarding tables are populated automatically, without administrator intervention. Due to simplicity, switches must be installed in spanning tree topology and multiple paths to same destination are not allowed.

Routers are more sophisticated devices, that forward traffic between multiple networks. The forwarding decision is based on information from routing tables stored in the router memory. Routing tables are constructed partly manually by administrators and partly automatically using complicated routing protocols.[1, chapter 7]

Firewalls operate on multiple layers, but usually below layer 5. These devices filter traffic according to multiple rules, such as layer 3 destination and target addresses and layer 4 protocol and port numbers. Firewalls can even inspect application layer data to decide whether traffic should be allowed. Firewalls are mostly configured manually by administrators to only allow traffic that is needed by end users and network services offered to outside network.[1, chapter 30]

The Internet is a collection of interconnected networks that are not managed by single organization. Instead there are multiple autonomous systems (AS) that manage a set of networks [1, chapter 14]. AS can be for example some large organization or Internet service provider (ISP). Since each AS manages their networks independently, routing between multiple autonomous systems changes frequently. Routers use special distributed routing protocols, such as PGB and RIP to calculate new routes when the links inside networks

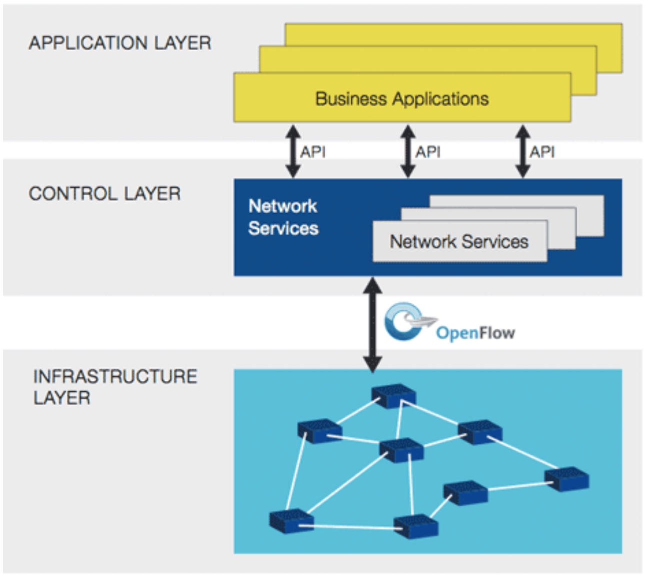

Figure 1: Software-Defined Networking Architecture [7].

change. These systems are complicated and difficult to configure, so routing mistakes are common in the current Internet.[1, chapters 13-15]

3 Software defined networking

This section describes software-defined networking and its advantages and disadvantages over traditional network architecture.

The term software-defined networking was first used in a white paper by Open Networking Foundation [7]. It is a new network architecture which decouples network control from actual data forwarding. It centralizes the control of single network and abstracts the underlying infrastructure. Figure 1 presents the conceptual architecture of software-defined networking. The figure highlights the separation of infrastructure, control and application layers.

At the bottom of SDN architecture is the infrastructure layer which consists of SND-aware network devices, such as SDN switches. Unlike conventional layer 2 switches, SDN switch does not construct the forwarding table automatically, but instead relies on a flow table that is constructed by controller and installed to the switch using shared API. The flow table is more complicated than traditional forwarding table and packets can be classified to different flows based on information from OSI layers 2 to 4.

The most important new feature of software-defined networking is the centralized controller. This facilitates easier network administration, as the configuration information is gathered into single place, instead of distributing it to individual network equipment. Furthermore, having a centralized overview of the network makes building of routing tables easier too, as distributed routing algorithms are not needed inside the network. Instead the controller calculates optimal routes across the whole network, constructs SDN flow tables and installs them into each of the switches in the network.

The application layer contains business applications that utilize the underlying network. Traditionally, applications can not affect the underlying network. However, software-defined networking enables applications to request specific network behavior from network controller. For example, controller can route traffic belonging to certain application differently than other traffic from same hosts.

Because the traffic in network can be classified into different flows based on information from layers 2,3 and 4, the distinction between switch and router is not as clear in software-defined networking. Instead, SDN switches with suitable controller can handle most traffic inside the network without stand-alone routers. It is even possible to partially replace firewalls with SDN switches, as flow rules do not necessarily require target, but traffic can be dropped at the switch as well. Even though SDN switches make switching decisions based on headers from multiple layers of OSI model, it does not necessarily make devices as complicated as traditional network devices working on same layer. This is due the fact that SDN switches only do bit mask matching on headers while the configuration is provided by the controller, not by the internal software.

One of the most used protocols in SDN is OpenFlow [5], promoted by Open Networking Foundation [7]. It is open standard and not tied to single controller or switch vendor. Thus it supports usage in heterogeneous networks. OpenFlow is the protocol used between controller and SDN switches to set up flow tables and to inform the controller of events in the network. These events include but are not limited to link state changes, host joins and switch installations.

The first OpenFlow compatible SDN controller software was NOX [4]. NOX is implemented as a C++ platform, which offers applications view to the network and API for controlling the network. NOX then allows developers to write NOX applications using C++ or Python for controlling the network. One of the newer SDN controllers is OpenDaylight [6], which aims to be more flexible and easier to extend than NOX.

Although the term software-defined networking is quite new, similar ideas have been proposed long before. For example Feamster et al. described architecture for separating routing algorithms from actual routers [3] in 2004, 8 years before the SDN white paper.

Like any new technology, Software defined networking has some shortcomings. Curtis et al. [2] note that in large-scale networks where flows change often the cost of involving controller in the setup process of every new flow can grow intolerable. In the OpenFlow protocol switch must send packets that do not belong to any current flow to the controller, which then generates and installs new rules to the flow tables of involved switches. This can cause significant amount of delay in connection set-up stage, which may be intolerable depending on the application requirements.

According to Tootoonchian and Ganjali [8] current OpenFlow deployments rely on single controller, which has certain drawbacks. First of all, the single controller is a single point of failure, as flow tables in switches can not be updated if the controller fails. Furthermore, the centralized controller might not scale in deployments involving large amounts of events and multiple SDN switches.

4 Related work

This section describes recent software-defined networking related research and different research methods used for studying SDN.

Curtis et al. [2] present DevoFlow as an extension to OpenFlow protocol. DevoFlow is intended for reducing delay of flow set-up and thus reducing the load on SDN control plane. DevoFlow is designed to excessively use OpenFlow wildcard rules and rule cloning to allow the switch to create flow rules independently in simple cases, without involving the controller in the process. The design sifts part of the decision making and statics collection back to the data plane from the control plane.

Authors used computer simulation to analyze performance of DevoFlow and OpenFlow. They developed "a flow-level data center network simulator" and used traffic traces from a cluster of 1500 computers as simulated network load. Their results indicate that DeveFlow could decrease control plane traffic significantly, thus decreasing flow setup latencies and controller load.

Tootoonchian and Ganjali [8] present software called HyperFlow as a solution for distributing logical centralized controller to multiple physical controllers. They implement HyperFlow as an application for OpenFlow controller NOX [4]. HyperFlow allows synchronizing the network information among multiple NOX controllers. Thus, every controller still has view over the whole network. Each controller node controls the closest SDN switches, and in case of controller failure switches must be reconfigured to connect to other controllers in the same network.

HyperFlow is implemented as a real C++ application and evaluated using a testbed of 10 servers. As the used testbed is not large enough to saturate even single NOX controller, authors then proceed to artificially generate network events to measure HyperFlow performance. The experimental evaluation indicates that at this point the HyperFlow slows down NOX, but the failure tolerance may justify the performance loss.

Both simulation and experimental evaluation have proven to be suitable methods for evaluating different software-defined networking solutions. As the SDN architecture includes both software and physical hardware, simulation is essential method at the beginning of development cycle. Simulation allows researchers to identify problems in design before using resources and time to build actual physical design. Simulation also has the advantage of simulating large networks without significant resources. However, simulations always incorporate assumptions about the network and may contain errors that corrupt end results.

Experimental evaluation can give more accurate results, as the actual system is tested in real-world conditions instead of model of the system. Moreover, the software built for experimental evaluation is useful after tests, unlike simulation models. However, experimental evaluation is not practical when developing new hardware. Furthermore, it might be difficult to test the scalability and operation in large-scale deployments with experimental evaluation, as it requires significant amount of hardware and other resources.

5 Conclusions

Even though software-defined networking is a new concept, it has gained massive amounts of attention in a short time span. Many top-level network device manufacturers are including SDN features into their products and it seems extremely likely that software-defined networking approach will be adopted widely in the industry. However, there are still many open problems related to SDN. For example controller redundancy, failure behavior and interoperability between devices from multiple vendors need to be addressed before widespread adoption in the Internet.

References

[1] D. E. Comer. *Internetworking with TCP/IP: Principles, protocols and architecture*, volume 1. Pearson Prentice Hall, Upper Saddle River, NJ, fifth edition, 2005.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[3] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 5–12, New York, NY, USA, 2004. ACM.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[6] J. Medved, A. Tkacik, R. Varga, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pages 1–6, June 2014.

[7] Open Networking Foundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.

[8] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3. USENIX Association, 2010.