ISSN No: 2454-9614

Packet Verification and Packet Forwarding through Controller in Open flow Network

G.K. Sowmya[a], R. Rajagopal[a*]

[a)] *Department Of Computer Science and Engineering, Vivekanandha Institute of Engineering and Technology for Women, Tiruchengode , Tamilnadu, India.*
[b)] *Department Of Computer Science and Engineering, Vivekanandha Institute of Engineering and Technology for Women, Tiruchengode , Tamilnadu, India.*

*Corresponding Author: G.K. Sowmya

E-mail: gksowmya29@gmail.com,
Received: 01/11/2015, Revised: 20/12/2015 and Accepted: 25/02/2016

Abstract

Open Flow is a Software Defined Networking (SDN) protocol that is being deployed in many network systems. SDN application verification takes an important role in guaranteeing the correctness of the application. Through the investigation, process discover that application verification can be very inefficient under the Open Flow protocol since there are many race conditions between the data packets and control plane messages. Furthermore, these race conditions also increase the control plane workload and packet forwarding delay. In the Attendre, an OpenFlow extension, to mitigate the ill effects of the race conditions in Open- Flow networks. To implement the Attendre in NICE (a model checking verifier), Open v Switch (a software virtual switch), and NOX (an OpenFlow controller). Experiments show that Attendre can reduce verification time by several orders of magnitude, and significantly reduce TCP connection setup time.

Keywords: Open Flow, software-defined network, model checking, verification, forwarding delay.

Reviewed by* **ICETSET'16 *organizing committee*

1. Introduction

Networking is the construction, design, and use of a network, including the physical (cabling, hub, bridge, switch, router, and so forth), the selection and use of telecommunication protocol and computer software for using and managing the network, and the establishment of operation policies and procedures related to the network. A process that fosters the exchange of information and ideas among individuals or groups that share a common interest. Networking may fall into one of two categories - social or business. In the latter category, one of the implicit objectives is to form professional relationships that may boost one's future business and employment prospects. A network is a group of two or more computer systems linked together.

1.1 Services and Protocols

The Post defines precisely which types of letters (size, weight, etc) can be delivered by using the Standard Mail service. Furthermore, the format of the envelope is specified (position of the sender and recipient addresses, position of the stamp). Someone who wants to send a letter must either place the letter at a Post Office or inside one of the dedicated mailboxes. The letter will then be collected and delivered to its final recipient. Note that for the regular service the Post usually does not guarantee the delivery of each particular letter, some letters may be lost and some letters are delivered to the wrong mailbox. If a letter is important, then the sender can use the registered service to ensure that the letter will be delivered to its recipient. Networking works by piggybacking a number of different protocols on top of each other. In this way, one piece of data can be transmitted using multiple protocols encapsulated within one another. We will talk about some of the more common protocols that you may come across and attempt to explain the difference, as well as give context as to what part of the process they are involved with. We will start with protocols implemented on the lower networking layers and work our way up to protocols with higher abstraction. TCP stands for transmission control protocol. It is implemented in the transport layer of the IP/TCP model and is used to establish reliable connections. TCP is one of the protocols that encapsulate data into packets. It then transfers these to the remote end of the connection using the methods available on the lower layers. On the other end, it can check for errors, request certain pieces to be resent, and reassemble the information into one logical piece to send to the application layer. The protocol builds up a connection prior to data transfer using a system called a three-way handshake. This is a way for the two ends of the communication to acknowledge the request and agree upon a method of ensuring data reliability. After the data has been sent, the connection is torn down using a similar four-way handshake. TCP is the protocol of choice for many of the most popular uses for the internet, including WWW, FTP, SSH, and email. It is safe to say that the internet we know today would not be here without TCP.

1.2 Goals of Networking

The goals of PC networking have been expanding over the last few years from simple file and printer sharing to access of fax machines, modems, and enterprise wide electronic mail systems. All the while, the essential goals of networking have always been to share resources and to provide a medium for communications. A network resource is either a device or a capability on the network that’s available for use by network users. The computer that the network resources are attached to is called the server. The other computers that access those resources over the network are called clients. The typical PC network user today takes shared file and printer access for granted. But there are now other resources that also can be made available to the user. Among them are fax machines, modems, computer servers, and database servers. Networks connect computers and the users of those computers. Individuals within a building or work group can be connected into local area networks; LANs in distant locations can be interconnected into larger wide area networks (WANs). Once connected, it is possible for network users to communicate with each other using technologies such as electronic mail. This makes the transmission of business (or non-business) information easier, more efficient and less expensive than it would be without the network.

One of the most important uses of networking is to allow the sharing of data. Before networking was common, an accounting employee who wanted to prepare a report for her manager would have to produce it on his PC, put it on a floppy disk, and then walk it over to the manager, who would transfer the data to her PC's hard disk. True networking allows thousands of employees to share data much more easily and quickly than this. More so, it makes possible applications that rely on the ability of many people to access and share the same data, such as databases, group software development, and much more. Intranets and extranets can be used to distribute corporate information between sites and to business partners. Networks facilitate the sharing of hardware devices. For example, instead of giving each of 10 employees in a department an expensive colour printer (or resorting to the “sneaker net” again), one printer can be placed on the network for everyone to share. Small computer networks allow multiple users to share a single Internet connection.

In a business environment, a network allows the administrators too much better manage the company's critical data. Instead of having this data spread over dozens or even hundreds of small computers in a haphazard fashion as their users create it; data can be centralized on shared servers. This makes it easy for everyone to find the data, makes it possible for the administrators to ensure that the data is regularly backed up, and also allows for the implementation of security measures to control that can read or change various pieces of critical information. To be able to deliver these messages, the service provider must be able to unambiguously identify each user. In computer networks, each user is identified by a unique address, we will discuss later how these addresses are built and used. At this point and when considering unicast transmission, the main characteristic obtained. The application layer is responsible for creating and transmitting user data between applications. The applications can be on remote systems, and should appear to operate as if locally to the end user.

2. SDN basics

Software-Defined Networking (SDN) is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures. First, it breaks the vertical integration by separating the network’s control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). Second, with the separation of the control and data planes, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller (or network operating system[1]), simplifying policy enforcement and network (re)configuration and evolution . It is important to emphasize that a logically centralized programmatic model does not postulate a physically centralized system. In fact, the need to guarantee adequate levels of performance, scalability, and reliability would preclude such a solution. Instead, production-level SDN network designs resort to physically distributed control planes the separation of the control plane and the data Plane can be realized by means of a well defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data plane elements via this well defined application programming interface (API), as depicted. The most notable example of such an API is Open Flow. An Open Flow switch has one or more tables of packet-handling rules (flow table). Each rule matches a subset of the traffic and performs certain actions (dropping, forwarding, modifying, etc.) on the traffic. Depending on the rules installed by a controller application, an Open Flow switch can – instructed by the controller – behave like a router, switch, firewall, or perform other roles (e.g., load balancer, traffic shaper, and in general those of a middle box). An important consequence of the software defined networking principles is the separation of concerns introduced between the definition of network policies, their implementation in switching hardware, and the forwarding of traffic. This separation key to the desired flexibility, breaking the network control problem into tractable pieces, and making it easier to create and introduce new abstractions in networking, simplifying network management and facilitating network evolution and innovation. Although SDN and Open Flow started as academic experiments, they gained significant traction in the industry over the past few years. Most vendors of commercial switches now include support of the Open Flow API in their equipment. The SDN momentum was strong enough to make Google, Facebook, Yahoo, Microsoft, Verizon, and Deutsche Telekom fund Open Networking Foundation (ONF) with the main goal of promotion and adoption of SDN through open standards development. As the initial concerns with SDN scalability were addressed in particular the myth that logical centralization implied a physically centralized controller, an issue we will return to later on – SDN ideas have matured and evolved from an academic exercise to a commercial success. Google, for example, has deployed a software-defined network to interconnect its data centres across the globe. This production network has been in deployment for 3 years, helping the company to improve operational efficiency and significantly reduce costs. VMware's network virtualization platform, NSX, is another example. NSX is a commercial solution that delivers a fully functional network in software, provisioned independent of the underlying networking devices, entirely based around SDN principles. As a final example, the world's largest IT companies (from carriers and equipment manufacturers to cloud providers and financial-services companies) have recently joined SDN consortia such as the ONF and the Open Daylight initiative, another indication of the importance of SDN from an industrial perspective.

SDN refers to a network architecture where the forwarding state in the data plane is managed by a remote control plane decoupled from the former. The networking industry has on many occasions shifted from this original view of SDN, by referring to anything that involves software as being SDN. We therefore attempt, in this section, to provide a much less ambiguous definition of software-defined networking. In this paper, we present, to the best of our knowledge, the most comprehensive literature survey on SDN to date. We organize this survey as depicted in Figure 1. We start, in the next two sections, by explaining the context, introducing the motivation for SDN and explaining the main concepts of this new paradigm and how it differs from traditional networking. Our aim in the early part of the survey is also to explain that SDN is not as novel as a technological advance. Indeed, its existence is rooted at the intersection of a series of "old" ideas, technology drivers, and current and future needs. The concepts underlying SDN – the separation of the control and data planes, the flow abstraction upon which forwarding decisions are made, the (logical) centralization of network control, and the ability to program the network – are not novel by themselves. However, the integration of already tested concepts with recent trends in networking – namely the availability of merchant switch silicon and the huge interest in feasible forms of network virtualization are leading to this paradigm shift in networking. As a result of the high industry interest and the potential to change the status quo of networking from multiple perspectives, a number of standardization efforts around SDN are ongoing, is the core of this survey, presenting an extensive and comprehensive analysis of the building blocks of an SDN infrastructure using a bottom-up, layered approach. The option for a layered approach is grounded on the fact that SDN allows thinking of networking along two fundamental concepts, which are common in other disciplines of computer science: a) separation of concerns (leveraging the concept of abstraction) and b) recursion.

Our layered, bottom-up approach divides the networking problem into eight parts: 1) hardware infrastructure, 2) southbound interfaces, 3) network virtualization (hypervisor layer between the forwarding devices and the network operating systems), 4) network operating systems (SDN controllers and control platforms), 5) northbound interfaces (to offer a common programming abstraction to the upper layers, mainly the network applications), 6) virtualization using slicing techniques provided by special purpose.

Computer networks can be divided in three planes of functionality: the data, control and management planes (see Figure 1). The data plane corresponds to the networking devices, which are responsible for (efficiently) forwarding data. The control plane represents the protocols used to populate the forwarding tables of the data plane elements. The management plan includes the software services, such as SNMP-based tools, used to remotely monitor and configure the control functionality. Network policy is defined in the management plane, the control plane enforces the policy, and the data plane executes it by forwarding data accordingly. In traditional IP networks, the control and data planes are tightly coupled, embedded in the same networking devices, and the whole structure is highly decentralized. This was considered important for the design of the Internet in the early days: it seemed the best way to guarantee network resilience, which was a crucial design goal. In fact, this approach has been quite effective in terms of network performance, with a rapid increase of line rate and port densities.

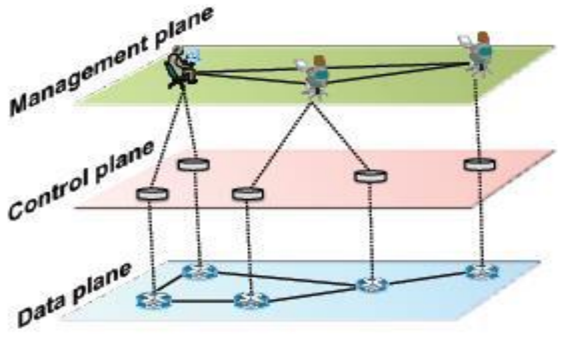

Fig. 1 Layered view of networking functionality.

To support network management, a small number of vendors offer proprietary solutions of specialized hardware, operating systems, and control programs (network applications). Network operators have to acquire and maintain different management solutions and the corresponding specialized teams. The capital and operational cost of building and maintaining a networking infrastructure is significant, with long return on investment cycles, which hamper innovation and addition of new features and services (for instance access control, load balancing, energy efficiency, traffic engineering). To alleviate the lack of in-path functionalities within the network, a myriad of specialized components and middle boxes, such as firewalls, intrusion detection systems and deep packet in current networks.

3. Forwarding and Verification Process

Attendre, an Open Flow extension, to mitigate the ill effects of the race conditions in Open- Flow networks. We have implemented Attendre in NICE (a model checking verifier), Open vSwitch (a software virtual switch), and NOX (an Open Flow controller). Experiments show that Attendre can reduce verification time by several orders of magnitude, and significantly reduce TCP connection setup time. The Open Flow is the mostly commonly used SDN language. In an SDN with a centralized control plane, the Open Flow protocol carries the message between SDN controllers and the underlying network infrastructure, bringing network applications to life. The sender sends the data, to the receiver via switches. If any problem occurring in data packet means, the managed switch route the data packet to the controller node. The controller node intimates the problem to the sender node as a error message. The sender resolving the problem and retransmit the data in same route path.

3.1 System Architecture

SDN separates the data plane and the control plane of a network. The data plane, usually the switch ASIC chip in a hardware switch platform or the kernel module in a software virtual switch, is responsible for fast packet processing, such as forwarding or dropping packets and modifying packet headers. The control plane, on the other hand, configures the packet processing rules on the data plane and handles the packets when the data plane does not know how to process them. OpenFlow is the de facto standard SDN protocol the control plane uses to configure the network data plane.

In fig 2, the sender sends the packet index to the switch. Then the file sends from the sender to the receiver via switch connection. The switch verifies the packet by using the packet index. There is no problem means it can be transmit to the receiver. If any problem occurring in the data packet means the switch router the packet to the controller. The controller forward the ask to the sender. The senders analyze the problem and retransmit the correct data packet via same switch to the receiver.

An Open Flow switch communicates with the controller over a secured TCP channel. In Open Flow, if a packet does not match with a flow table, the packet will be buffered3 at the switch and will be associated with a ***buffer ID***, which can be used to retrieve the packet from the buffer. Then, a ***packet-in*** message containing the ***match***

fields of the packet, buffer ID and the ***table ID*** of the flow table at which the mismatch happens will be sent from the switch to the controller. By processing the packet-in, the controller could add one or more entries to the specified flow tables of switches by sending ***flow-mod*** messages to them. The controller could also issue a ***packet-out*** message containing actions and the buffer ID in the packet-in message back to the switch. The switch will process this packet according to the actions in the packet-out message. The flow mod could also carry a buffer ID, so that a packet-out message is combined with the flow-mod.

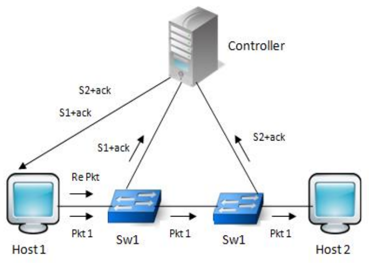

Fig. 2 System Architecture

3.2 Open Flow Network Configurations

Open Flow is a Software Defined Networking (SDN) protocol that is being deployed in many network systems. SDN application verification takes an important role in guaranteeing the correctness of the application. Through our investigation, we discover that application verification can be very inefficient under the OpenFlow protocol since there are many race conditions between the data packets and control plane messages. OpenFlow protocol enables the programmatic and centralized control of a network's behaviour. Software application programs running on a centralized OpenFlow controller can observe virtually any aspects of an OpenFlow enabled network and correspondingly dictate the behaviour of the network by sending command messages to directly configure packet processing rules on the Open Flow enabled network switches. This centralized, user-programmable control plane architecture is generally referred to as Software-Defined Networking (SDN).

3.3 Model Checking And Packet Verification

Model checking is the most powerful one, since it is capable of detecting the widest range of misbehaviour in networks. The checking process is done by the switches when the packets are received. By using the packet index which can be send by the sender host during the transmission process. One of the most important usages of SDN is in fine-grained network security policy enforcement in enterprise networks to combat the unauthorized resource access threat. To realize fine-grained network security policies, a critical feature supported by OpenFlow is the so-called packet-in messages. Using this feature, a traffic flow that is not known to the network switches is directed, via packet-in messages, to the centralized controller for processing.

Among all automated SDN verification approaches proposed (a broader discussion of the various approaches can be found in Section , model checking is the most powerful one, since it is capable of detecting the widest range of misbehaviour (e.g., forwarding loops, reachability failures, policy violations, lack of aliveness) in SDN applications even before these applications are deployed. In model checking, the entire network, including the switches, the hosts and the controller running SDN applications, is modelled as a state machine. The verifier explores the entire state space and checks for network property invariant violations by simulating every possible event ordering inside the entire network.

3.4 Controlling Race Condition

It increased the workload on the switches and the controller. The race conditions can be reduced while eliminating the unnecessary messages, which are the root causes for inefficient verification and increased forwarding delay.

Describe the Ill Effects of the Race Conditions: Using state transition diagrams, we point out the ill effects of the races in model checking based Open Flow application verification. We also describe other ill effects, like increased packet forwarding delay and increased workload on the switches and the controller. 3) Control and Mitigate the Ill Effects of Race Conditions with Attendre: We present Attendre, an extension of the OpenFlow protocol to control the race conditions so as to mitigate the ill effects of races. By controlling these races, i mean that the messaging behaviour of a switch is as if the outcomes of the races are deterministic. By controlling the outcomes of the race conditions, Attended eliminates many necessary network states as well as many unnecessary control plane messages, which are the root causes for inefficient verification and increased forwarding delay. We present the changes to the OpenFlow protocol and the changes to switches in detail.

3.5 Forwarding Delay Acknowledgement

After the verification process, there is no problem means the packet can receive by the end host. In case of packet loss occurs, switch sends the failure acknowledgement to the centralized management for the purpose of correctness the application. By controlling these races, we mean that the messaging behaviour of a switch is as if the outcomes of the races are deterministic. In case of packet loss it forward filler notification to the sender by controlling the outcomes of the race conditions, Attended eliminates many necessary network states.

3.6 Packet Re-forwarding

The verified network has a sender and replier. The sender sends data packets and receives the same number of packets returned by the replier. We run the verification with different numbers of packets and different numbers of switches between the sender and the replier. Send packet again in alternate way. The controller forwards a failure acknowledgement to the sender which can sent by the switch for the correct packet transmission. The following process can be explained in the step by step method in the flow model.

4. Conclusions

Traditional networks are complex and hard to manage. One of the reasons is that the control and data planes are vertically integrated and vendor specific. Another, concurring reason, is that typical networking devices are also tightly tied to line products and versions. In other words, each line of product may have its own particular configuration and management interfaces, implying long cycles for producing product updates (e.g., new firmware) or upgrades (e.g., new versions of the devices). All this has given rise to vendor lock-in problems for network infrastructure owners, as well as posing severe restrictions to change and innovation. The method have identified three types of race conditions present in an Open Flow network, and presented a mechanism called Attendre that mitigates the ill effects of these race conditions. As our results indicate, the benefit of Attendre is potentially very large. More broadly, this work emphasizes the importance of considering race conditions in SDN protocol design and shows that by taking the consequence of race conditions as a first-class protocol design concern, it is possible to gain a deeper understanding of the subtle behaviours of existing protocols, and lead to improved protocol designs.

ACKNOWLEDGMENT

This research was supported by my Head of the Department, Prof.T.R.Srinivasan. We thank Mr.Chandra Mohan for assistance with a technique and Mr.Sathish for comments that greatly improved the manuscript .We thanks famous persons Dr.Karunanithi and Dr.K.C.K.Vijayakumar, for sharing their pearls of wisdom with us during the course of this research, and we thank Mrs.E.Menaka and Mr.R.Rajagopal reviewers for their so-called insights. Mrs.Leema Mathayi helped with some technical stuff. We are also immensely grateful to my family for their comments and support, although any errors are our own and should not tarnish the reputations of these esteemed persons.

References

[1] Benson.T, Akella.A, and Maltz.D.A, “Network traffic characteristics of data centers in the wild,” in Proc. IMC, 2010, pp. 267–280.
[2] Cai.Z, A. Cox.L, and “Maestro: A system for scalable openflow control,” Rice Univ., Houston, TX, USA, Tech. Rep. TR10-08, Dec. 2010.
[3] Canini.M, Venzano.D, Pere˘síni.P, Kosti´c.D, and Rexford.J, “A nice way to test openflow applications,” Proc. NSDI, 2012, pp. 10:1–14.
[4] Casado.M et al., “ETHANE: Taking control of the enterprise,” in Proc. SIGCOMM, 2007, pp. 1–12.
[5] Casado.M et al., “Rethinking enterprise network control,” Trans. Netw., vol. 17, no. 4, pp. 1270–1283, Aug. 2009.
[6] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, “Intelligent design enables architectural evolution,” in Proceedings of the 10th ACM Workshop on Hot Topics in Networks, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 3:1–3:6.
[7] Gude.N et al., “NOX: Towards an operating system for networks,” SIGCOMM Comput. Commun. vol. 38, no. 3, pp. 105–110, Jul. 2008.
[8] Hinrichs.T, Mitchell.J, Gude. N., Shenker .S, and Casado.M, “Expressing and enforcing flow-based network security policies,” Univ. Chicago, Chicago, IL, USA, Tech. Rep., 2008.
[9] H. Kim and N. Feamster, “Improving network management with software defined networking,” Communications Magazine, IEEE, vol. 51, no. 2, pp. 114–119, 2013.
[10] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” Communications Surveys Tutorials, IEEE, vol. 16, no. 3, pp. 1617–1634, Third 2014.
[11] MehdiS.A, Khalid.J, and KhayamS.A, “Revisiting traffic anomaly detection using software defined networking,” in Proc. RAID, 2011, pp. 161–180.
[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: a distributed control platform for largescale production networks,” in Proceedings of the 9th USENIX conference on Operating systems design and implementation, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.