Software Defined Networks – OpenFlow Channel Security and Efficiency

Matthew West
Illinois State University
School* of IT
Normal, IL 61790-5150

Jihad Qaddour
Illinois State University
School of IT
Normal, IL 61790-5150

ABSTRACT

This paper assesses the resource costs of TLS in OpenFlow and puts forth a header format for channel communication that is more efficient and partially backwards-compatible. Resource usage is shown to be reduced by up to 19.36% with a TLS flag added to the OpenFlow header.

General Terms

Software Defined Networking, Control Plane, Protocols, Transport Layer Security, Encryption

Keywords

SDN, OpenFlow, channel, security, efficiency, TLS, header, southbound, SSL, PKI, DS, xid, Mininet, encryption

1. INTRODUCTION

In software-defined networking (SDN), communication between the controller and hardware devices forwarding network traffic is described as southbound traffic. While there are more options for this medium besides OpenFlow, this paper will deal only with the OpenFlow channel (southbound) and switch specification (run on each forwarding device), as well as resource usage of the controller. As with the selection of OpenFlow above, many controllers exist, but Ryu was used for this study due to using a low-overhead Python environment.SDN and network function virtualization (NFV) are a natural match. A quick test of an SDN virtualized environment finds line speeds upwards of 18 gigabits per second. Once switches running OpenFlow are run within an NFV framework, concerns about bandwidth between virtual devices on the same physical server are reduced greatly. The limiting factor of switched network functionality (the data link layer of SDN) in these environments then becomes processing and memory. Memory usage is outside the scope of this paper but is worth examining. As the code driving new networking technologies goes through consecutive version releases, it is possible that some of the code becomes memory inefficient due simply to the number of people and the span of time on which it has been worked.The OpenFlow Switch Specification version 1.5.1 documentation from the Open Networking Foundation (ONF) defines the southbound interface of SDN when it pertains to OpenFlow communications as the OpenFlow channel [1]. In Section 6 of the documentation, channel messages are specified to be sent either directly via TCP (in the clear) or encrypted using Transport Layer Security (TLS). This version of OpenFlow details TLS encryption under subsection 6.3.6: connection Uniform Resource Identifiers (URI) may specify TLS as the protocol to be used for communication when a switch first connects to the controller, but there are mechanisms to opt for TLS encryption which may be initiated by either the switch or controller. Optional to TLS is the use of one or more Certificate Authorities (CA) and Certificate Revocation Lists (CRL), an approach which is recommended to minimize some of the common attack surfaces of SDN southbound traffic. Among the configurations not recommended by the ONF are self-signed certificates and pre-shared keys, which are generally accepted to be less secure than using a CA [2]. While some of the OpenFlow channel traffic necessitates encryption, such as table updates or flow requests (either, if sent in the clear, can provide critical information about the operation of the network), other messages such as hellos and metadata may simply require a digital signature, reducing processing overhead on both controllers and network devices, as well as utilizing a system already implemented in the public-key infrastructure (PKI) framework suggested by the ONF.Research regarding Secure Sockets Layer (SSL) and TLS are discussed in the Related Work section. SSL was officially deprecated by the IETF in 2015 in RFC 7568, and all secure network communication should use TLS version 1.1 or higher (1.0 was designed to be backwards-compatible with SSL, opening it up to some of the same vulnerabilities). Establishing a TLS/SSL connection involves a set of hellos, followed by a session key exchange using PKI which can be implemented in a variety of ways [3]. Ongoing communication is done using the more efficient symmetric key exchanged during session setup. TLS session packets sent over a network connection are encrypted, meaning an eavesdropper should be unable to distinguish any data contained within.With the move from HTTP (uses TCP) to HTTPS (uses SSL/TLS) to reduce some common browser-based attacks on users, some research was done on the increase in network overhead created by the increased size of SSL packets [4][5][6]. While the increase in bandwidth was negligible from the user's end (and for the server's end for most websites), the increase in processor usage on webservers depended heavily on how the website was used. As more web traffic was determined to necessitate secure connections, Session Initiation Protocol (SIP) IP phone provider hardware was studied to determine the resource usage impact of TLS/SSL [7][8]. This type of study can be used for a baseline in SDN implementations because they involve consistent streams of packets, with a session handshake at the beginning utilizing PKI.Whereas in the past, the overhead of TLS/SSL encryption on an individual device would not be of concern unless it was a large portion of that device's total resource usage (the argument being "what else is it going to do with those cycles?"), with SDN moving increasingly towards NFV environments, the amount of hardware that can be virtualized on a server is limited by the shared resource usage of that hardware plus anything else running on the device such as the hypervisor.

2. PREDICTIONS

2.1 Research Questions

So, if controllers and hardware are sharing the same server resources and bandwidth has become less of a concern due to the gains in performance from SDN and NFV, then a main concern of traffic between them becomes how can network engineers reduce resource usage while maintaining secure connections? This paper seeks to answer the following questions:

1. What is the approximate processing overhead of TLS in SDN?
2. What traffic needs to be secured in the OpenFlow channel?
3. If TLS has a noticeable processing overhead and not all traffic needs to be encrypted, the research question becomes
4. What is a possible solution to improve OpenFlow channel efficiency without sacrificing security?

2.2 Hypotheses

H_0: There will be a negligible difference between using TLS encryption on all OpenFlow channel traffic versus encryption only on necessary traffic (less than 5% processing overhead).H_1: There will be a difference between using TLS encryption on traffic as in H_0 (5% overhead or more).

3. RELATED WORK

Previous work has been done on encryption overhead in network traffic, most of it focused on network overhead (additional bandwidth usage). A portion of this work relates to SSL, and a large portion of that relates to HTTPS traffic and servers, as mentioned in the introduction. Cornett et al. [9] identified increases in processor (CPU) efficiency for larger buffer sizes when network security functions were offloaded to another device. This implies some sort of performance cost when using security functions. Unfortunately, if the purpose of SDN is to have network hardware forward traffic and little else, it cannot expect to offload cryptographic functions to those devices (which is not to mention the security issues involved in such an approach).Researchers Lal and Garg [6] also delved into the possibility of offloading SSL in a different manner. Citing slowdowns of HTTPS servers due to TLS processing, they identified one of the large resource usage sources in encryption to be the passing of data between the software (user space) and operating system (kernel), described as system call overhead. Their proposed solution gave the cryptographic software engine direct access to the security engine (SEC) driver, bypassing the kernel. Until companies start equipping servers with SEC hardware to offload cryptographic processing to, their findings are useful to SDN insofar that they pinpoint one of the larger consumers of system resources: the kernel.

Callegari et al. [7] studied the block cipher and hash functions used in TLS and DS. Of the block ciphers, only AES is worth considering, as 3DES requires twice as many cycles per byte as AES-128, which is still considered to be a secure algorithm. AES-256 took roughly 33% more processing effort (747 versus 562) for double the key length (the numbers in AES designations telling the length of the key). Hash functions perform a different role than block ciphers and are more efficient because they do not need to be reversible. SHA-1 was found to use about 16 cycles per byte. SHA-3 is the newest version of the hash and is known to use more processing power per security bit. Even with conservative estimates for SHA-3, it requires ten times fewer cycles than AES-128.Finally, Shen et al. [8] studied the impact of implementing TLS on a SIP server. Their tests started with UDP and ended with TLS using 3DES with proxy authentication. Since OpenFlow operates using TCP only, the TCP NoAuth, TCP Auth, and TLS 3DES Auth are the most interesting points of the article. Each was broken down into the total number of CPU events caused by each part of the SIP forwarding process. The proportions of these and analysis of the data are discussed in Findings.

4. METHODOLOGY & RESOURCES

Research was performed using Mininet to create a virtual SDN. Within the virtual environment, a Ryu controller was deployed using a traffic monitor that gathered data from each virtual switch every 10 seconds and collated the data in a simple table. Data on TCP and TLS efficiency were taken from work by [8], described in Related Work. Data were gathered in 3 parts. First, a simple analytics monitor tied to the controller to simulate nonessential traffic. Next, the proportion of traffic that could be sent via TCP was analyzed for both a case with the monitor and without. Finally, CPU load differences between TCP and TLS transmissions were compared for both test cases.Mininet was installed in a Linux environment with default settings, and a small-scale network was created using a tree topology with a depth of 3 and a fan-out of 3, totaling 27 switches. The example traffic monitor explained in Ryu Documentation 1.0 was implemented with minimal alterations and made to run on top of the normal Ryu SDN controller.After the controller and switches were configured, Wireshark was used to capture packets during events (such as populating flows) and over periods of time (to measure traffic generated by the monitor). The two main tests both captured packets for 10 minutes, with switches populating flows 30 seconds after the start of the test. One test was done with a basic Ryu controller, while the other was done with a Ryu controller that prompted for metadata updates every 10 seconds. These data are analyzed in Findings.

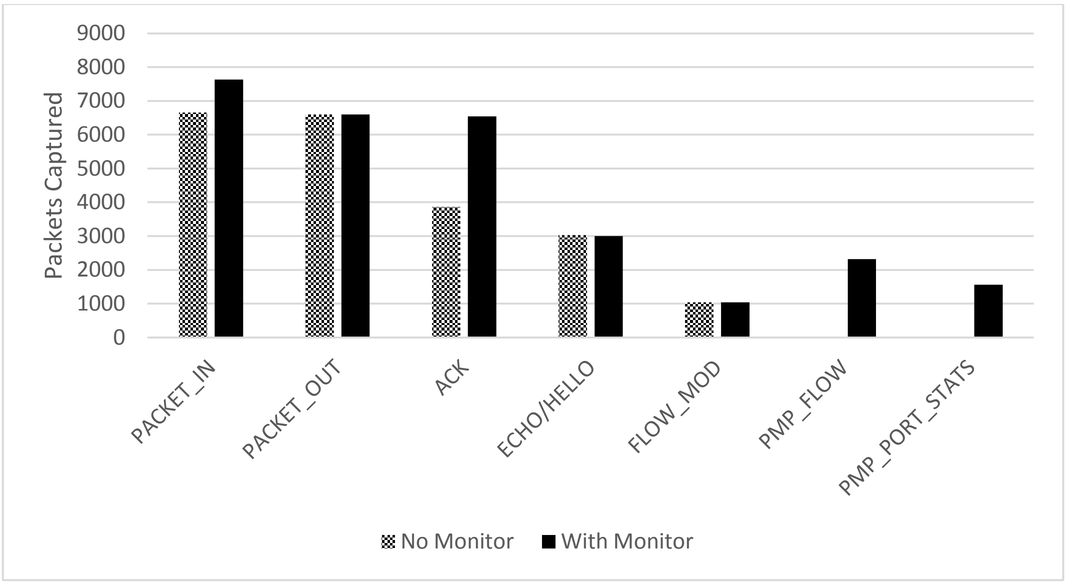

Figure 1: Packets captured by message type

5. FINDINGS

The total number of packets of each type was tallied after each 10-minute test and can be found in Figure 1. PMP messages involve flow hit messages and port usage messages, and therefore only appeared in the test with a simple OpenFlow switch monitor. FLOW_MOD and PACKET_OUT messages were identical in both tests, and ECHO and HELLO messages differed by 0.0083%, which can be considered statistically insignificant. Message types with fewer than 100 occurrences were not included in the data. These types were FEATURES_REPLY (13 and 21), SET_CONFIG (13 each), PMP_PORT_DESC (13 and 52) and PORT_STATUS (7 and 11).

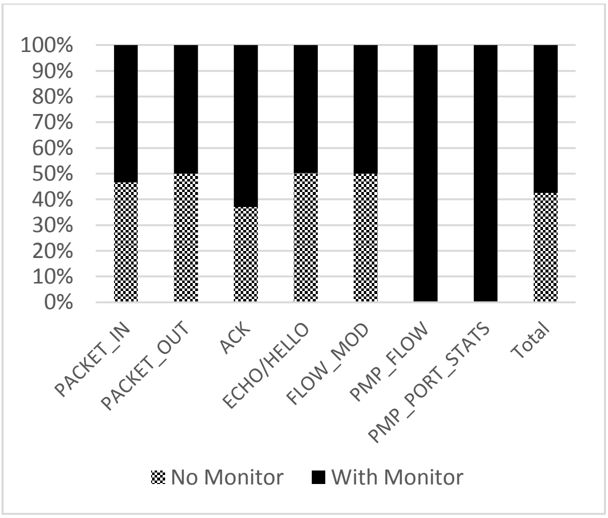

Figure 2: Packet proportions captured across tests by message type

The test with monitor generated 14.6% more PACKET_IN messages than the test without a monitor, as well as 69.65% additional ACK messages. With all message types combined, the controller with monitor caused 35.43% additional OpenFlow channel traffic compared to the controller without over the same period. There were 3874 additional messages of PMP_PORT_STATS and PMP_FLOW types combined, comprising slightly more than half of the extra southbound traffic.Figure 2 compares message types by percentage. One can use this and Figure 1 to conclude PACKET_OUT, ECHO/HELLO, and FLOW_MOD messages are not affected by additional controller monitoring features where metadata is involved. PACKET_IN messages are a way for a switch to request forwarding instructions from the controller, and simply contain the packet necessitating a flow update. These messages do not need to be encrypted, as the encapsulated packet will already have been encrypted if it needs to be. Notably, the reply message (FLOW_MOD) from the controller will require TLS, as it can contain forwarding table rules or other information relevant to the operation of the network and requires the requesting switch have a trusted certificate. Additionally, one can assume that the additional ACK messages generated were due to the PMP messages sent to the controller when using the monitor. If PMP messages can be sent in the clear, these ACKs can be sent in the same manner, but this is not the case for all ACK messages. Therefore, the focus of the following discussion on TLS efficiency in the OpenFlow channel will focus on PACKET_IN, PACKET_OUT, and PMP messages. HELLO/ECHO messages in OpenFlow currently only have a header, so cannot be encrypted in a meaningful way.

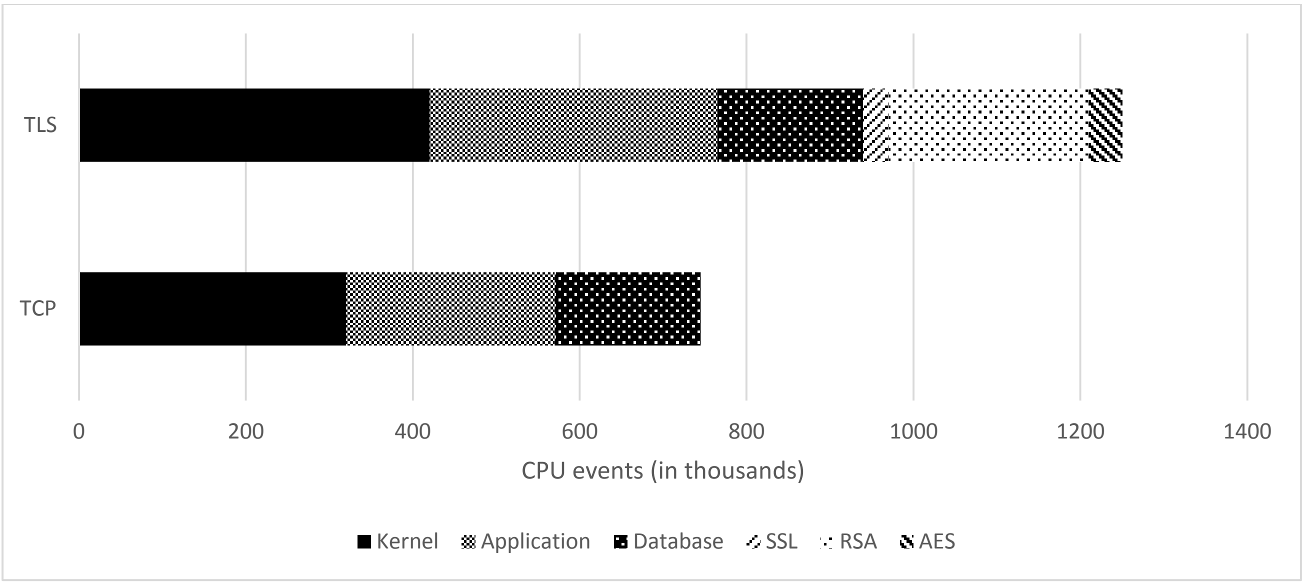

Figure 3: Reference [8] outbound proxy CPU events with SIP filtered

Setting aside the above tests, Figure 3 uses data from [8], which were introduced in the Related Work section. These data support the findings of Lal and Garg involving increased usage of the kernel in operating systems performing encryption functions. Application core and modules were combined from [8]'s data, with cryptographic functions left as separate entities on the Figure. Proxy, database, and other system-specific processes were also removed. The difference between TCP and TLS in this instance is 67%. Although SIP shares some attributes with the OpenFlow channel, the specific differences in both application performance and traffic behavior require caution. Therefore, it is assumed the difference in performance is two thirds that of the SIP system, or 43%.At this point, concise answers to the first two research questions are available. The approximate CPU usage increase of TLS over TCP in SDN is 43%, and FLOW_MOD, PACKET_OUT, and ACK messages need to be encrypted. While PACKET_IN messages can be sent in the clear, the PACKET_OUT response should have some form of authentication, lest an attacker use it to insert packets into a pipeline to which they should not have access (packets from PACKET_OUT use the input port ID, rather than the port receiving the message from the controller). A possible alternative to full encryption for PACKET_OUT will be discussed later. This solution is unlikely to be helpful for sending ACK messages without opening the OpenFlow channel up to man-in-the-middle and replay attacks.Assuming the above, the 10-minute test without monitor in Figure 1 would need 23.11% of its messages to be encrypted with TLS, while the test with a monitor would need 26.43% encryption. This shows that as metadata sent to the controller for administrator use increases, proportion of traffic that requires TLS increases due to a higher volume of ACK packets. This is true even if the packets being generated due to the monitoring function do not need to be encrypted.Using the 10-minute test without monitor, the reduction in CPU usage between encrypting all traffic and only the necessary packets is 13.75% (or 86.25% of the full encryption usage). This test reflects only the most fundamental operation of the OpenFlow channel and offers a good baseline to build upon. Considering digital signatures as an alternative to TLS where authentication but not confidentiality is necessary, PACKET_OUT messages could be sent via TCP with DS. This improves CPU usage reduction to 23.12%. Using data from [7], the best-case scenario for using hashes for authentication over TLS is a reduction of 90% CPU usage. Assuming hashing takes twice as many resources due to the new SHA-2 and SHA-3 algorithms, hashing will take 20% of the resources of TLS. This yields a 19.36% resource usage reduction over always-on TLS, assuming no digital signing of HELLO/ECHO messages. Thus, H_1 is supported due to a meaningful difference in resource usage. This approach has the added benefit of adding authentication and integrity to messages (PACKET_IN, PMP_FLOW, PMP_PORT_STATS) that could otherwise be sent without.

Version 8 bits		Type 8 bits	Length 16 bits
TLS 1 bit	xid 31 bits		

Figure 4: Proposed OpenFlow header with TLS flag

Figure 4 contains a proposed header format for OpenFlow messages to indicate whether the message is sent via TLS and requires decryption, or via TCP with DS and requires hashing. This header borrows 1 bit from the xid (transaction identifier) field and uses it as a TLS flag. This reduces the number of possible xid values by half, to 2,147,483,648. Older versions of OpenFlow would still be able to process these headers while also allowing devices using the new header format to operate in the same environment. Combined with the version field, anything at the current OpenFlow version of 1.5.1 (0x06) or below would see the TLS flag field as simply an xid of 2,147,483,648 or higher, while newer versions (0x07 and onward) would treat it properly as a flag, ensuring full backwards-compatibility for TLS. DS would not operate for versions 1.5.1 or earlier, however. The best approach due to the above attributes would be for newer versions of OpenFlow to default to always-on TLS when operating on networks with older versions.

6. CONCLUSION

The OpenFlow channel mandates security to prevent attackers from gaining an understanding of or altering the function of the network. Which message types need to be encrypted by the current method of TLS is up for discussion. Current OpenFlow specifications allow for either always-on TCP or always-on TLS, with no middle ground between the two options. Some type of flag (Figure 4) that specifies whether TLS is used could improve southbound efficiency by 19.36% and be partially backwards compatible. The way to authenticate and ensure integrity of packets sent via TCP would be to use a DS.

Table 1: Resource usage of OpenFlow TLS options

Packet Encryption Type	Security	Percent TLS encryption	CPU usage
TCP Only	Poor	0%	100%
TLS Only	Good	100%	143%
TLS flag *(Figure 4)*	Good	23.11%	115.3%

Findings are noted in Table 1. An overall reduction in block cipher usage leads to a decrease in CPU usage, even when a DS is added to messages sent via TCP to allow for authentication and integrity verification. Security with ACK and TABLE_MOD messages encrypted is considered to be good, as all other messages aside from HELLO/ECHO have a DS built from the shared session key established during TLS handshaking.

7. LIMITATIONS

Some assumptions were made in this study, most of which were noted in the Findings section. Data on TLS efficiency over TCP were gathered from a SIP implementation rather than an SDN environment. These data were also from 2010, and CPU efficiency regarding block cipher encryption may have improved compared to unencrypted data over the past 8 years more than the 33% performance increase controlled for (this is different than overall available CPU resource, which have certainly increased).The worse of the two 10-minute tests was used as the baseline for the findings on efficiency, but this is only for traffic on a particular network size and structure. Other SDN constructions and implementations may yield different proportions of packets when compared to this data. Additionally, as FLOW_MOD traffic increases (more than 1 table update every 10 minutes) efficiency of the proposed solution decreases. This is unlikely in most networks; newer controllers may have dynamic table updating features, however, which create additional OpenFlow channel overhead (again reducing overall efficiency). Table 2 contains an analysis of the data without the assumptions listed in Findings. The differences on this table are mainly that TLS takes 66% additional resources over TCP and hashing is expected to use 10% of the resources of block cipher encryption (6.6% more than TCP in this case.

Table 2: Resource usage of OpenFlow TLS options best case

Packet Encryption Type	Security	Percent TLS encryption	CPU usage
TCP Only	Poor	0%	100%
TLS Only	Good	100%	166%
TLS flag *(Figure 4)*	Good	23.11%	119.39%

8. REFERENCES

[1] A. Nygren et al., “OpenFlow switch specification version 1.5.1 (protocol version 0x06),” Open Networking Foundation, March 2015.

[2] S. Gowda, “Public certificates vs private certificates vs self-signed certificates, in MSDN Blogs, August 2017, https://blogs.msdn.microsoft.com/shreyasgowda/2017/08/18/public-certificates-vs-private-certificates-vs-self-signed-certificates/

[3] B. Laurie, “Certificate Transparency,” in Communications of the ACM, vol. 57, iss. 10, pp. 40-46, October 2014.

[4] A. Boldberg, “A comparison of HTTP and HTTPS performance,” Computer Measurement Group, CMG98 8, 1998.

[5] B. Jackson, “Analyzing HTTPS performance overhead,” keycdn, January 2017, https://www.keycdn.com/blog/https-performance-overhead/

[6] N. Lal and V. Garg, “Accelerating OpenSSL operations offloaded to hardware crypto accelerator,” in International Journal of Knowledge Based Computer Systems, vol. 4, iss. 1, pp. 30-33 Publishing India Group, 2016.

[7] C. Callegari, R. G. Garroppo, S. Giordano, and M. Pagano, “Security and delay issues in SIP systems,” in International Journal of Communications Systems, vol. 22, pp. 1023-1044, May 2009.

[8] C. Shen, E. Nahum, H. Schulzrinne, and C. Wright, “The impact of TLS on SIP server performance,” in IEEE/ACM Transactions on Networking, vol. 20, no. 4, pp. 1217-1230, August 2012.

[9] L. Cornett, K. Grewal, M. Long, M. Miller, and S. Williams, “Network security: challenges and solutions,” Intel Technology Journal, vol. 13, iss. 2, pp. 112-129, Intel Corporation, June 2009.