OpenFlow: Why Latency Does Matter

Kévin Phemius and Mathieu Bouet
Thales Communications & Security, Paris, France
{kevin.phemius, mathieu.bouet}@thalesgroup.com

***Abstract*—In the OpenFlow framework, packet forwarding (data plane) and routing decisions (control plane) run on different devices. OpenFlow switches are in charge of packet forwarding, whereas a Controller, which can be situated very far from a networking point of view from the switches its manages, sets up switch forwarding tables on a per-flow basis. The connection between a switch and its Controller is thus of primary importance for the performances of the network. In this paper, we study the impact of the latency between an OpenFlow switch and its Controller. We show that UDP and TCP have different effects depending on the available bandwidth on the control link. Bandwidth arbitrates how many flows the Controller can process, as well as the loss rate if the system is under heavy load, while latency drives the overall behavior of the network, that is the time to reach its full capacity. Finally, we propose solutions to mitigate the phenomenons we outline.**

I. Introduction

In the Software Defined Network(SDN) architecture, the control and data planes are decoupled, network intelligence and state are handled by controllers running on commodity hardware, and the underlying network infrastructure is abstracted from the applications. OpenFlow, standardized by the Open Networing Foundation (ONF), is one of the more mature initiatives. OpenFlow started in early 2008 when McKeown et al. [1] proposed a new protocol for SDN. Thereafter, the protocol was actively evaluated on aspects such as application programming [2], maximum supported message rate [3], and maximum handled flows per second [4]. The pivotal element of the protocol, the Controller, was implemented through parallel initiatives such as NOX, Beacon and Floodlight. Researchers mainly focused on improving the performances of a specific Controller like Maestro [5] and NOX [6] or demonstrating the improvement offered by OpenFlow against a classic L2 switch [7].
Nevertheless, the link used by an OpenFlow switch and its Controller to communicate is of primary importance when dimensioning a network or evaluating its performances [8]. Indeed, in many cases the Controller consists in a separate machine that manages tens or hundreds of switches. It can thus be distant, in terms of latency, from the elements it controls, especially when OpenFlow is envisioned for wireless mesh networks [9] or datacenters interconnection. In this paper, we study how the bandwidth and most of all the latency between an OpenFlow switch and its Controller affect the performances of the whole network. We also propose solutions to mitigate the phenomenons we highlight.

II. The OpenFlow Protocol

OpenFlow aims at replacing, or at least extend, current network equipment by a new type of "dumb switches" where the decision making is entirely assumed by Controller(s), giving the switches only a basic set of instructions: *Forward* the packet, *Drop* the packet, *Send* the packet to Controller (after encapsulation) and *Overwrite* part of the packet header. OpenFlow switches only need to look at their Flow Table(s) which contains the action(s) associated to a flow. To identify a flow, a switch can rely on a function which can match various fields in the frame (inbound port, VLAN ID, data layer or network address, transport protocol header, etc.).
To register to a Controller, an OpenFlow switch goes through a procedure called a Handshake. During this exchange of messages, the two parties gather information about one another, such as the Data-path ID to uniquely identify the switch, the maximum capacity of the buffer and how many bytes of a packet to send to the Controller in case of an unknown flow.
Once the switch is registered, it relies on the Controller to handle the management of the flows. When an inbound packet arrives, the switch goes through its Flow Table(s) to try and match the different headers of the packet to an action. If one is matched, it carries the corresponding action. If not it sends the packet (or part of it depending on the configuration) to the Controller with a PACKET_IN message. The Controller then replies back the final decision about the packet, whether it is to forward it with a PACKET_OUT message or drop it entirely. It possibly writes an action in the switch's Flow Table with a FLOW_MOD message in case another packet from the same flow comes up.
Several open source implementations of OpenFlow controllers exist. The ONF released a minimal OpenFlow Reference Controller to allow researchers and network administrators to test OpenFlow. In parallel, Nicira Networks developped NOX and Stanford University Beacon. FloodLight [10] is a fork of the Beacon Controller. Like Beacon, and contrary to NOX, it is written in a 64-bits environment, making use of the much larger registries of these systems and is multi-threaded, allowing much better performances. We use FloodLight as the OpenFlow Controllers in our study since it is the more mature of the four Controllers and gathers a very large community.

III. Experimentations

The Controller is crucial to make the system work. Both its performances and the capacities of its links with the switches of primary importance. Our goal is to show how the Controller and the network perform with bandwidth and latency issues on the control link.

A. Testbed

Our testbed is illustrated in Fig. 1. We decided to put the Controller, that is Floodlight, and the network, provided by *cbench* [4] and Mininet [11], on two separate virtual machines to have a finer control on the link connecting these two entities. The bandwidth and the latency can thus be

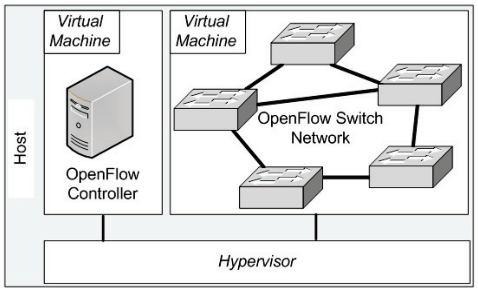


Fig. 1. Testbed used for the tests.

customized in the Hypervisor. By default, since we are in an emulated environment, the testbed has a network-to-controller sub-millisecond latency on a 10Gbit/s Ethernet link which are very ideal conditions. The network formed by the switches relies on 3Gbit/s Ethernet links and, depending on the quantity of switches and their connectivity, a low latency (less than 0.800ms end-to-end with 128 switches in a linear topology). Four our evaluation we adjust the bandwidth with a traffic shaper and the latency by increasing the time a packet has to wait in the egress queue to reach the controller.

B. *Controller's performance calibration*

FloodLight is a high performing OpenFlow Controller. It is able to handle a large amount of equipment while maintaining a high level of service. Before evaluating the impact of the latency of the switch-to-controller link on the performances of the whole network, we study the capacity of treatment of FloodLight in ideal conditions.
To calibrate the Controller and evaluate its processing capabilities, we elaborate a test of the network using *cbench*. It creates a number of switches and hosts and then sends traffic through those switches at the maximum possible capacity. The switches send PACKET_IN messages to the Controller which responds with PACKET_OUT messages. *cbench* measures the amount of packets processed per seconds by the Controller while *ifstat* evaluates the bandwidth consumption. The goal of this test is to put the most possible stress on the Controller. After reception of the messages, the packets are discarded by the switches and there are no flow tables in the switches so every packet is sent to the Controller.
Even if our hardware configuration was not optimal[1], FloodLight was capable of dealing with 900,000 flows per second on a 32 switches network with 1000 hosts per switch (Fig. 2). To maintain this effectiveness, FloodLight handled around 600 Mb/s of PACKET_IN requests while responding with PACKET_OUT messages at a rate of 200 Mbit/s. The rate is asymmetrical because a PACKET_IN message contains a part of the packet as well as a buffer reference in the switch. The Controller only sends back the buffer reference (a small integer) in the PACKET_OUT message.

C. *Increasing switch-to-controller latency*

When subjected to a lower bandwidth in the control link, the switches cannot send as many PACKET_IN messages to the Controller which in return responds with less

[1]Intel Dual Core @ 3 GHz-32 bits, 1 x 4096 MB DDR2, Ubuntu 10.04, Linux kernel 2.6.38-8-generic, Java 1.6, gcc 4.5.2, FloodLight 0.82, cbench 1.2.2, Mininet 1.0.0, ifstat 1.1, iperf 2.0.5

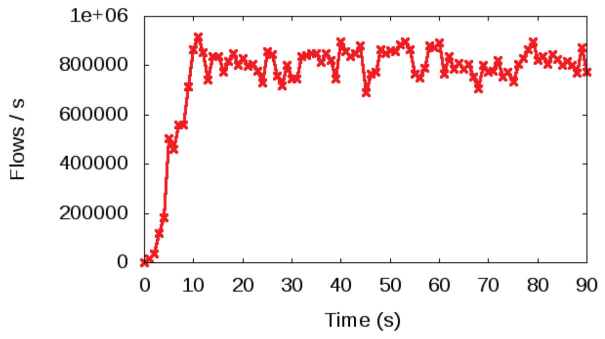


Fig. 2. Flows processed per second by Floodlight (32 switches and 1000 hosts per switch).

PACKET_OUT. The Controller cannot do anything about the control link capacity as it cannot treat messages faster than it receives them. Increasing the bandwidth allows for a more reactive network due to the Controller working at full capacity. Conversely, a low bandwidth will cause some packet losses as the egress queues fill up.
The latency has a very different effect on the performances. As each switch has to do a handshake with the Controller before being able to send requests, there is a time window during which no operation is possible. The handshake generates a burst of traffic as seen in Fig. 3 for a latency of 5ms. As the latency increases, the time to complete the handshake phase increases as well topping at 7s for a 100ms latency in a 32 switches network. With higher latencies (over 300ms) the handshake phase is not clearly distinguishable as the curve follows a sawtooth pattern. *cbench* "throughput" mode is designed to stress the Controller with a stream of requests, thus achieving a nominal phase where the link's maximum capacity is obtain. The relatively low latencies reach the cap fairly quickly with 3s, 5s and 7s for a latency of 5ms, 50ms and 100ms respectively. With 300ms latency it takes over 20s and the 500ms test spent 3/4 of the allotted time to achieve normality. This "slow start" phase, albeit normal when booting the network is heightened by a higher latency and will have to be executed over again in case of temporary failure whether it is a link or a Controller failure during full operation.

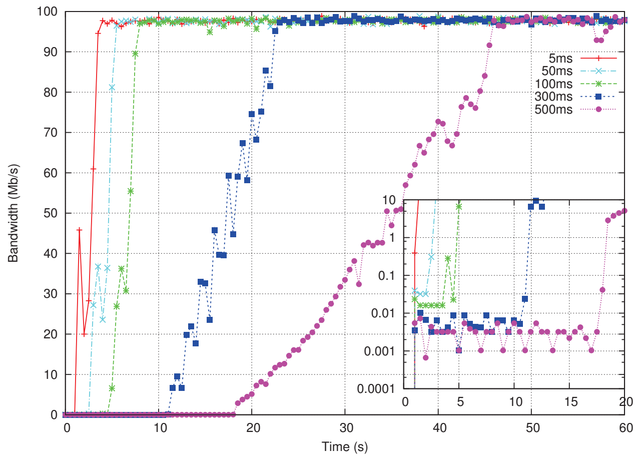


Fig. 3. Bandwidth use for various latencies - Bandwidth: 100Mb/s 32 switches and 1000 hosts per switch.

IV. Analysis

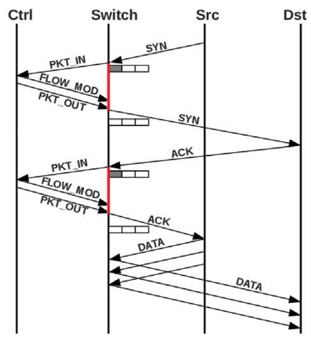
While studying the effects of latency on an OpenFlow network, we saw that the problem is two-sided.

A. TCP

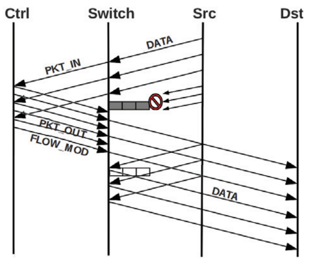
In the case of TCP, the effect of latency consists in a longer delay between the SYN and SYN/ACK segments during the TCP handshake. The forwarding mechanism of Floodlight establishes paths based on the source and destination of the message. When the SYN segment hits the first switch of the path, it is subjected to the high latency of the control link. The Controller computes the route and pushes it along the path on all the concerned switches, these messages are also subjected to the latency. When the destination node sends back the ACK segment, the procedure has to be done over again because the source and the destination are now different (they have been swapped).
The delay encountered by the flow at the establishment of the connection is a double penalty (in red in Fig. IV-B), as the Controller is queried twice and is detrimental to short-lived flows.

B. UDP

UDP uses a different approach than TCP. There is no handshake between the nodes and the source directly sends its packets to the destination. There are two important issues we need to account for.
First, while the connections are established between the switches and the Controller, each packet with no match in the flow table is put in a buffer. If the buffer is full, the default action of OpenFlow is to send the whole packet to the Controller (even if only the header is sufficient), overcharging the already degraded link. If the link cannot support this additional charge, the packets are lost. As TCP only sends one segment (SYN) before sending its data, there is virtually no loss. On the contrary, UDP sends datagram after datagram with no restraint. This bring us to the second point. A high latency forces the buffer to fill up too fast if the throughput of the sender is too high. This leads to packet loss until the buffer is cleared out (Fig. IV-B).
The switch's buffer limits the amount of packets that can be stored while waiting for a decision by the Controller. With a high latency, the packets have to wait longer thus increasing the probability of packet loss. Moreover an OpenFlow buffer is currently measured in packet, not in bytes. This means that a small packet of a few bytes and a big packet at the MTU size will take the same amount of space in the buffer. Albeit allowing a much simpler control of the buffer, there is a clear lack of optimization.



(a) Flow chart of a TCP flow.



(b) Flow chart of a UDP flow.

Fig. 4. Flow charts for TCP and UDP.

V. Proposed Solutions

In this section, we elaborate on solutions that can mitigate the phenomenons we observed for TCP and UDP with a high latency on the control link.

A. Double contact with the Controller

These issues can be prevented on the Controller side of the network. By implementing some simple routing algorithm using the Topology Discovery and Link State applications that most of OpenFlow Controllers possess, we can forward packets based on the topology of the network and pre-load routes in the switches. For example when the Controller receives a SYN segment, it could after computing the route send two FLOW_MOD messages, one for each direction. The flow will thus suffer from the latency on the control link only once (Fig. 5).

B. Buffer overflow

The first seemingly evident solution would be to increase the buffer size. As the switch can store more packets, there will be a lower loss rate.
By lowering the source node's throughput we can limit the effect of high latency. Less packets will fill the buffer which then will have less chance to overflow. TCP has built-in mechanisms that control the rate at which the data are sent, whereas a fixed rate UDP like a CBR application does not. We could ask the source to lower its throughput because we saw that it is beneficial on a high latency situation. But it is not something we can really control. However, an application with a high throughput will cause the buffer to overflow, causing a loss of packets not only for itself but for the other flows (Fig. 6(a)). By using an *Ingress Limiter* (Fig. 6(b)) we can restore some fairness by dropping packets of the incriminated flow at the port level instead of buffer level.
Another solution would be to use the Explicit Congestion Notification (ECN) [12] field on the packet's header. ECN was defined in 2001 and is rarely used since a lot of hardware

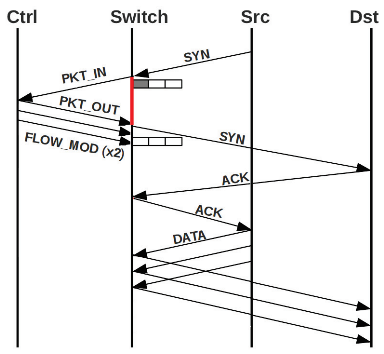


Fig. 5. Flow chart of a TCP flow with route pre-loading.

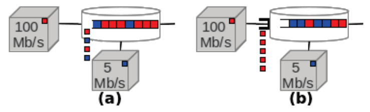


Fig. 6. Switch without (a) and with (b) an Ingress Limiter.

equipment does not support it at all. By enabling it on OpenFlow switches, we could lower the throughput by notifying the sources when a buffer overflow is impending. TCP/IP can natively use ECN while on UDP the congestion protocols must be at the application level.

Finally, we could improve the buffer algorithm which just store-and-forward (or drop) packets. When the first packet of an unknown flow arrives to the switch, it generates a PACKET_IN message. If another packet from the same flow arrives while the first is waiting for a response, it will generate a message too. We could instead scan the buffer before and check if another packet already provoked a message. If not, then the normal procedure would follow. If there is already a packet from the same flow, the switch would not send another message but would keep track of all the packets from that specific flow in the buffer so that when the response turns up, the appropriate actions would be applied to all the packets from the flow. This would limit the quantity of messages exchanged between the switches and the Controller on an already impaired link.

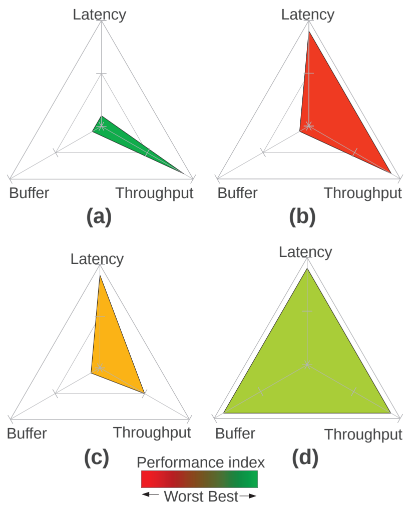


Fig. 7. Performance evaluation of an OpenFlow-based network w.r.t. network latency, buffer size and controller's troughput.

VI. Discussion and Conclusion

In this paper, we saw that the network-to-controller link is critical in establishing the Controller's and the network's performances. With good conditions, an OpenFlow Controller like FloodLight is capable of processing close to a million flows per seconds. It is thus not a bottleneck for the performances of the whole network. We showed that the bandwidth and the latency have a negative impact on the overall performances. We located some issues with the default behavior of OpenFlow and presented solutions to advert them. The bandwidth arbitrates how many flows the Controller can process, as well as the loss rate if the system is under heavy load while the latency drives the overall behavior of the network. Even if the proposed solutions can be implemented easily, an upstream evaluation of the use of the network is necessary.

As we can see in Fig. 7(a) an ideal situation would allow a high throughput and if the latency is low enough, the buffer characteristics are not important. However, keeping an elevated throughput in a high latency situation causes disastrous performances (cf. Fig. 7(b)). One of the solution is to limit the throughput by different means. But it still reduces the overall performance of the network (cf. Fig. 7(c)). The more fitting solution would be to act on the switch's buffer, whether it is by increasing its size or improving its design. It would be easier to increase the size of the buffer but we can quickly fall into *bufferbloat* [13]. If the buffer is too large, the packet will spend more time waiting inside. This can cause TCP to not react the way it is supposed to because the congestion control mechanisms require at least some packets to be lost in order to kick in. In the same way, time sensitive applications will be subjected to high latency and jitter in the network. So the buffer size must be carefully weighted depending on the needs in the network.

ECN can be used seamlessly with TCP, while using it over UDP requires the applications to deal with it, which is not viable. Finally altering the behavior of the buffer necessitates to go deep inside the code of the switch. Some other solutions to be considered are an out-of-band control plane with dedicated high performing links to the Controller, a traffic engineering mechanism to prioritize the control messages or reserve a small fraction of the bandwidth for the OpenFlow control mechanisms.

References

[1] 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.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] M. Canini, D. Venzano, P. Pereini, D. Kostic, and J. Rexford, "A nice way to test openflow applications," in *Proc. of NSDI 2012*, 2012.

[3] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. of PAM*, March 2012.

[4] R. Sherwood and K. Yap, "Cbench: an open-flow controller benchmarker," http://www.openflow.org/wk/index.php/Oflops.

[5] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow control," *Structure*, 2010.

[6] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. of Hot ICE 2012*, 2012.

[7] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," in *Proc. of IEEE ICC*, may 2010.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, Aug. 2011.

[9] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks," in *Proc. of ICCCN*, 31 2011-aug. 4 2011.

[10] "Floodlight openflow controller," http://floodlight.openflowhub.org/.

[11] "Mininet: rapid prototyping for software defined networks," http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet.

[12] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168 (Proposed Standard), IETF, Sep. 2001, updated by RFCs 4301, 6040.

[13] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, vol. 55, jan 2012.