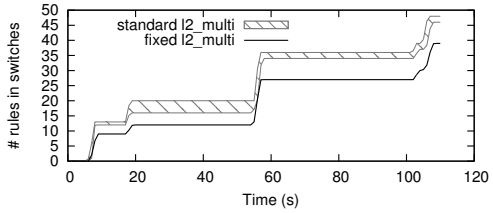
Consistent Packet Processing – Because Consistent Updates Are Not Enough

Peter Perešíni[†*], Maciej Kuźniar[†*], Nedeljko Vasić[†], Marco Canini[⋆], and Dejan Kostić[‡]
[†]*EPFL* [⋆]*TU Berlin / T-Labs* [‡]*Institute IMDEA Networks*
[†]{*name.surname*}*@epfl.ch* [⋆]*m.canini@tu-berlin.de* [‡]*dkostic@imdea.org*

With the abstraction of a logically centralized, global view of the network, Software-Defined Networking (SDN) is poised to dramatically streamline network management and enable extensibility and customization. However, as in every software system, there is always the risk of software faults (or bugs). OpenFlow, currently the standard SDN platform, is particularly subject to such risk because all popular controller frameworks (*e.g.*, NOX, FloodLight, *etc.*) offer an API that closely matches the low-level OpenFlow interface—so exposing developers to all the idiosyncrasies of the underlying asynchronous and distributed collection of switches [1].

To address several of these issues, Monsanto *et al.* [2] propose programming language techniques that raise the abstractions for managing SDNs and Reitblatt *et al.* [3] propose general abstraction for managing network updates. We agree with these authors this is stepping in the right direction. In our ongoing work, we identify an additional abstraction that previous work has so far not considered. In particular, through the example scenario below of a real bug we uncovered in POX[1], we demonstrate the need for *consistent packet processing in SDN controllers*—the abstraction that guarantees that every packet traversing the network (and its duplicates due to flooding) is processed by one consistent state of the controller rather than a mixture of controller states. This abstraction is a natural complementary to the per- packet consistency abstraction [3] which considers the network states *but not the controller state*.

Harmful "amplification" bug. Here, we show that performance and scalability issues can arise even in a straightforward OpenFlow deployment. We examine the use of `l2_multi`, a multi-switch version of a learning switch that maintains the mapping (host, port) for every switch in the network. Initially, the location of the destination host d is not known, and the controller instructs the first switch that encounters a packet to d to flood it. Once host d is reached and replies, the controller sees the reply packet and learns d's location. However, the remains of the flood could still be traversing the network. When these duplicate packets reach switches that (still) do not know d's location these packets are once again sent to the controller. If the switch application running



in the controller is not carefully designed, the controller will route these duplicate packets towards d. Depending on network size, this may potentially generate a packet storm. Moreover, the controller will install unnecessary additional rules in the switches. Combined, these effects ultimately hurt the performance of other flows in the network, as well as the overall network scalability.

The general problem this bug exposes is the lack of consistency of controller state when these packets are processed—duplicate packets that arrive after the controller learns d's location are processed according to the new controller state. To demonstrate the need for *consistent packet processing*, we show the effect of rule amplification described in the previous paragraph. We measure the total number of rules installed in switches forming a small fat-tree topology (20 switches) during a replay of several flows and repeat the experiment 3 times (reporting avg. and std. dev.). The network is controlled by the POX controller using `spanning_tree`, `arp_responder` and `l2_multi` modules. We compare against a manually fixed version of `l2_multi`. The results in the Figure indicate that even in this small topology, the controller installs more rules than needed, *i.e.*, the *inconsistent packet processing* happens.

To address the problem of *consistent packet processing* we propose to use transactional semantic within the controller. In more detail, the first time a specific packet is sent to the controller, the controller starts a new transaction. When the same packet is later seen by the controller, the controller will reuse the context of the previously-started transaction. Thus, the packet avoids observing a new state.

References

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

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

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

[*] Student author. A student author will offer a demo.

[1] This bug also affects other controller frameworks.