Brief Announcement: Towards Distributed and Reliable Software Defined Networking*

Marco Canini[1] Petr Kuznetsov[2] Dan Levin[1] Stefan Schmid[1]
[1] TU Berlin / T-Labs [2] Télécom ParisTech

Software-defined networking (SDN) is a novel paradigm that out-sources the control of packet-forwarding switches to a set of software controllers. The most fundamental task of these controllers is the correct implementation of the *network policy*, *i.e.*, the intended network behavior. In essence, such a policy specifies the rules by which packets must be forwarded across the network. This paper initiates the study of the SDN control plane as a distributed system.

We consider a *distributed* SDN control plane which accepts *policy updates* (*e.g.*, routing or access control changes) issued *concurrently* by different controllers and whose goal is to *consistently* compose these updates. One of our contributions is precisely the notion of consistency of concurrent policy composition. We introduce a formal model for SDN under fault-prone concurrent control. In particular, we seek to ensure *per-packet consistency* [3]. Informally, this property ensures that every packet is processed at every switch it encounters in the data plane according to just one and same policy, which is the composition of policy updates installed by the time when the packet entered the network.

We present the abstraction of *Consistent Policy Composition (CPC)* which offers a *transactional* interface. A policy-update request returns *commit* if the update is successfully integrated in the current network policy or *abort* if the update cannot be installed. Our correctness property informally requires that the abstraction, regardless of the actual interleaving of concurrent policy updates and data packets' arrivals, *appears* sequential to every data packet, as though all the committed requests (and possibly a subset of incomplete ones) are applied atomically and no data packet is in flight while an update is being installed.

We show that it is generally impossible to implement the CPC abstraction in the presence of a single controller's crash failure. The requirement of per-packet consistency allows us to introduce an interesting variant of the bivalency argument [2], where the valency of an algorithm's execution accounts for all possible *paths* a packet may take in all extensions of the execution. Since typically the controllers do not have influence on the network traffic workload, our impossibility proof is able to exploit the intertwined combination of two kinds of concurrency: overlapping policy updates arbitrarily interleaving with traffic.

Accordingly, we investigate stronger model abstractions which enable fault-tolerant CPC implementations. We find that a slightly more powerful SDN switch interface supporting an atomic read-modify-write allows for a *wait-free* CPC solution, and we investigate the tag complexity of such a solution.

* A full version of this paper can be found at [1].

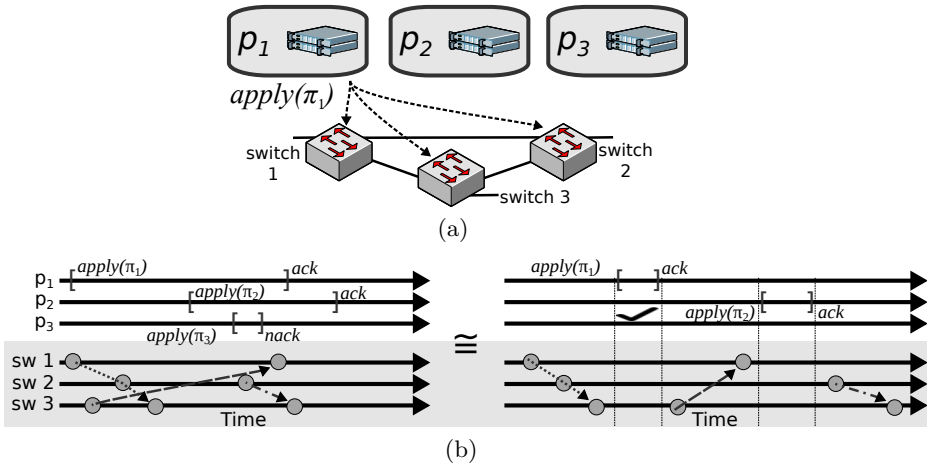

Fig. 1. Example of a policy composition: (a) 3-process control plane and 3-switch data plane, (b) a concurrent history H and its sequential equivalent H_S.

Policy Composition in SDN: Example. Consider a network consisting of three switches `sw1`, `sw2` and `sw3` (Figure 1(a)), and controlled by p_1, p_2, and p_3. The controllers' function is to try to install policy-update requests on the switches. An example of a concurrent history H is presented in Figure 1(b). The three controllers try to concurrently install three different policies π_1, π_2, and π_3. Imagine that π_1 and π_2 are applied to disjoint fractions of traffic (*e.g.*, π_1 affects only `http` traffic and π_2 only `ssh` traffic) and, thus, can be installed independently of each other. In contrast, let us assume that π_3 is conflicting with both π_1 and π_2 (*e.g.*, it applies to traffic from `address` 1.2.3.4). In this history, π_1 and π_2 are committed (returned *ack*), while π_3 is aborted (returned *nack*).

While the concurrent policy-update requests are processed, three packets are injected to the network (at switches `sw1`, `sw2`, and `sw3`) leaving three *traces* depicted with dotted and dashed arrows. Each trace is in fact the sequence of ports which the packet goes through while it traverses the network. For example, in one of the traces (depicted with the dotted arrow), a packet arrives at `sw1`, then it is forwarded to `sw2`, and then to `sw1`. Next to H we present its "sequential equivalent" H_S. The traffic on the data plane is processed *as though* the application of policy updates is *atomic* and packets cross the network *instantaneously*.

The full paper [1] presents a formal specification of concurrent policy composition and explores its implementation costs. We believe that this work opens a new and exciting problem area with a number of unexplored concurrency issues.

References

1. M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. The case for reliable software transactional networking. Technical report, arXiv TR 1305.7429, 2013.
2. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.
3. M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, 2012.