On the Necessity of Time-based Updates in SDN

Tal Mizrahi, Yoram Moses[*]
Technion — Israel Institute of Technology

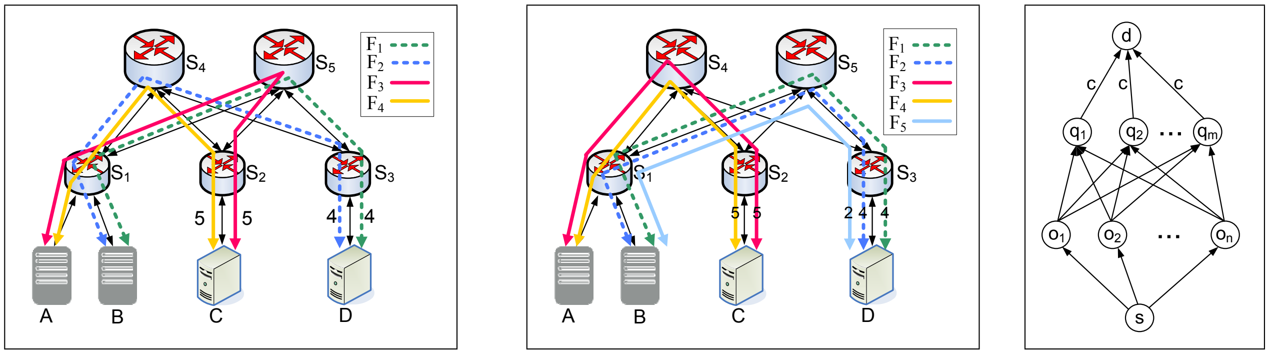
1 Introduction

The usage of accurate time to schedule updates in software defined networks was recently proposed in [1]; time can be a powerful tool for applying network updates in a relatively simple manner and with a very brief period of inconsistency during the update. In the current paper we introduce the flow-swapping scenario, which demonstrates the necessity of time-based updates. We show that while traditional update approaches result in temporary packet loss, the time-based approach allows a smooth reconfiguration procedure. We introduce the *lossless flow allocation* problem, and formally show that given the online nature of resource allocation in SDN, scenarios that require simultaneous changes at multiple switches are bound to occur, with clock-based near-simultaneous scheduling offering an advantageous solution.

2 Flow Swapping

One of the most attractive features of the SDN approach is the ability to dynamically perform traffic engineering decisions (e.g., [2, 3]), and to optimize network performance by assigning routes based on load balancing criteria from a network-wide perspective. We now present the *flow-swapping scenario*, in which optimal traffic engineering requires two flow routes to be swapped.

Consider a network with two servers, C and D, and two storage devices, A and B. All links connecting the servers or storage devices to the switches and between the switches have an equal capacity of 10 Gbps. Assume the servers may run applications of five different types, where the connection of a type i application to a storage device requires i Gbps, for $i \in \{1, 2, \ldots, 5\}$. The need for a new application instance arises at the servers. When the controller identifies a new application of a given type i (we assume that the type is specified in the header information), it needs to configure the switches with forwarding rules that allocate a path for the corresponding flow.



(a) Server C initializes two 5 Gbps flows with storage device A. Server D initializes two 4 Gbps flows with storage device B.

(b) F_2 and F_3 must be rerouted (swapped) to allow F_5 to be installed. The controller updates S_2 and S_3 simultaneously to avoid packet loss.

(c) The lossless flow allocation problem

Figure 1: Lossless Flow Allocation

Flow swapping. In our example (Fig. 1a), server D initializes two type 4 applications, each requiring a session with B, resulting in flows F_1 and F_2. Similarly, server C initializes two type 5 applications with A, giving rise to F_3 and F_4. The controller assigns four different paths to the four new flows, allowing traffic to be load balanced between S_4 and S_5. At this point links $S_4 \leftrightarrow S_1$ and $S_5 \leftrightarrow S_1$ are both loaded with 9 Gbps of traffic, each leaving only 1 Gbps for additional traffic. Now server D initializes a new type 2 application, requiring a session with B. However, in the current network configuration (Fig. 1a) there is no available path that can accommodate a 2 Gbps flow. In order to accommodate the new 2 Gbps flow, the controller must reconfigure the network to the configuration illustrated in Fig. 1b, thus allowing the new flow, F_5, to be forwarded via S_3 and S_5. The transition from Fig. 1a to Fig. 1b requires the controller to update S_2 and S_3 with the new paths for F_2 and F_3, i.e., to *swap* F_2 and F_3.

Lossless updates. Existing approaches for consistent network updates would have the flows at servers S_2 and S_3 updated at two different times. This is true, in particular, of the version approach of [4], as well as for the sequential approach of [5]. If, for example, S_2 is updated before S_3, then in the transient time between the two updates S_4 receives F_2, F_3 and F_4 from S_2

[*]The Israel Pollak academic chair at Technion; this work was supported in part by the ISF grant 1520/11.

and S_3, totaling a rate of 14 Gbps, and transmits to S_1 at 10 Gbps. Hence, during the transition the excess 4 Gbps of traffic is either lost or buffered at S_4. We note that [3] observed the temporary congestion during flow reallocation procedures, and reserved a costly $10-30\%$ of the capacity on each link for this purpose, with the excess capacity allowing a multi-step congestion-free update procedure. It is a key observation that the updates at S_2 and at S_3 should be performed as close as possible to simultaneously to prevent packet loss, and to minimize the amount of resources needed to absorb transient congestion, while preventing the need for excess capacity or for a multi-step update procedure. Using synchronized clocks, as suggested in [1] can help minimize the period of inconsistency and inefficiency.

3 The Lossless Flow Allocation Problem

We now show that the need for flow-swapping is inherent for flow allocation in the SDN framework, i.e., that flow-swapping may be needed regardless of the controller's flow allocation protocol. We introduce a special case of the multi-commodity flow (MCF) problem called the *lossless flow allocation* problem; it is not presented as an optimization problem, but as a game between two players: a source that generates traffic flows (commodities) and a controller that configures the network forwarding rules.

The model. The network in our model is represented by a directed graph (Fig. 1c) with a source s, a destination d, and two layers of intermediate nodes. Each of the edges connected to d has a capacity c, whereas the rest of the edges are assumed to have an infinite capacity. Each flow (commodity) represents a session between s and d; a flow has a constant rate, and cannot be split between two paths. s progressively adds or removes flows (commodities). The controller controls the forwarding policy of nodes $\mathbb{O} = \{o_1, o_2, \dots, o_n\}$, allowing to balance the traffic between the m edges leading to d. When s adds a flow to one of the nodes in $\mathbb{O}$, the controller responds by reconfiguring the forwarding rules of nodes in $\mathbb{O}$ so as to guarantee that all flows are forwarded without packet loss. The sequence of forwarding rule updates allows the controller to rearrange the flows in a way that allows the new flow to be accommodated. Each update in the sequence is either a *reroute* or a *swap*; we define a *reroute* as an update that modifies the forwarding rule of a single flow, whereas a *swap* is an update that simultaneously modifies the forwarding rules of two or more flows. We define a *k-way swap* as a swap in which the controller updates k nodes from $\mathbb{O}$. Each update in the sequence *must* be performed such that at the end of the update none of the edges exceeds its capacity. At the end of the sequence, the controller installs a forwarding rule for the new flow. It is assumed that only after the controller completes the update sequence does the source add or remove another flow.

The game. The *lossless flow allocation game* is played between two players, the *source* and the *controller*. Given a set of flows, F, from s to d, the controller's goal is to configure the forwarding policy of the nodes in $\mathbb{O}$ in a way that allows all flows to be forwarded to d without packet loss, i.e., without exceeding the capacity of any of the edges connected to d. The source's goal is to progressively add flows without exceeding the network's capacity in a way that forces the controller's update sequence to include a *swap*.

Strategies. Given a set of active flows, F, and a new flow f that is added by s, the controller's strategy, $\mathbb{S}_{con}$, determines the update sequence and the forwarding rules of $\mathbb{O}$ at the end of the update sequence. Given the set of flows F, and the set of forwarding rules of $\mathbb{O}$, the source's strategy, $\mathbb{S}_s$, defines the next flow to be added to F or removed from F. We can show:

Theorem 1 *For the setting of Figure 1c, there exists a strategy, $\mathbb{S}_s$, for the source that forces every strategy, $\mathbb{S}_{con}$, for the controller to perform an n-way swap.*

Both the strategy $\mathbb{S}_s$ and the theorem's proof are short, albeit beyond the space constraints of this note. Theorem 1 reveals the significance of time-based updates. It shows that every routing strategy can be forced to swap flows among paths. Since guaranteeing lossless forwarding requires simultaneous swapping, time and synchronized clocks can provide a genuine advantage for network updates.

4 Future Work

We have defined an extension to the OpenFlow wire protocol that allows time-based network updates; a controller can include a *scheduled-time* parameter in OpenFlow messages sent to switches. A preliminary version of this extension is presented in [6]; this proposal is under discussion and is continuing to evolve in the Extensibility working group of the Open Networking Foundation (ONF). An implementation of this proposal is currently in progress, and will allow extensive experimental evaluation of time-based network updates. Our future work will also include evaluation of the scheduling accuracy, i.e., how accurately switches can invoke scheduled updates. Reliability aspects of the time-based approach will also be explored, i.e., how to guarantee a coordinated update when the delivery of update messages is not guaranteed, or when some of the switches are unable to schedule an upcoming update.

References

[1] T. Mizrahi and Y. Moses, "Time-based updates in software defined networks," in *hot topics in software defined networks (HotSDN)*, 2013.

[2] C. -Y. Hong, et al., "Achieving high utilization with software-driven wan," in *Proceedings of ACM SIGCOMM 2013*, pp. 15–26, ACM, 2013.

[3] S. Jain, et al., "B4: Experience with a globally-deployed software defined wan," in *Proceedings of ACM SIGCOMM 2013*, pp. 3–14, ACM, 2013.

[4] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of ACM SIGCOMM*, 2012.

[5] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *Trans. on Networking*, 2007.

[6] T. Mizrahi and Y. Moses, "Time-based Updates in OpenFlow: A Proposed Extension to the OpenFlow Protocol," technical report, CCIT Report #835, July 2013, EE Pub No. 1792, Technion – Israel Institute of Technology, `http://tx.technion.ac.il/~dew/OFTimeTR.pdf`, 2013.