Minimizing Transient Congestion during Network Update in Data Centers

Jiaqi Zheng[†], Hong Xu[‡], Guihai Chen[†§], Haipeng Dai[†]
[†] Nanjing University, Nanjing, China
[‡] City University of Hong Kong, Hong Kong
[§] Shanghai Jiao Tong University, China

ABSTRACT

Data centers are increasingly relying on software defined networking (SDN) to orchestrate data transmission. To maximize network utilization, the SDN controller needs to frequently update the data plane as the network conditions change. Due to its asynchronous nature, data plane update may result in serious transient congestion and packet loss. Prior work strives to find a congestion-free update plan with multiple stages, each of which guarantees that there will be no congestion independent of the update order. This approach prevents the network from being fully utilized and requires solving a series of LP with scalability challenges.As each switch updates its flow table independently and asynchronously, the transition of data plane state—if done directly from the initial to the final stage—may result in serious transient congestion and packet loss.

In this paper, we study the general problem of *minimizing* transient congestion during network update, given the number of intermediate stages. This exposes the tradeoff between update speed and transient congestion, which may be absorbed by switch buffers, and allows the operator to navigate a broader design space. We formulate the *m*inimum *c*ongestion *u*pdate *p*roblem (MCUP) as an optimization program, and propose heuristics to find the update sequence efficiently. Preliminary results show that our approach increases link utilization by 20% and reduces update time by 50% compared to prior work.

Keywords

SDN, Data Centers, Network Update

1. INTRODUCTION

In a software defined network, the controller needs to frequently update the data plane by modifying switch flow tables so as to dynamically optimize performance. This process is not *atomic* [8]: each switch is updated independently and asynchronously. Thus network update may result in serious congestion during the transient period, even though the initial and final configurations are most likely uncongested [4, 7].

CoNEXT Student Workshop'14, December 2, 2014, Sydney, Australia.

http://dx.doi.org/10.1145/2680821.2680823.

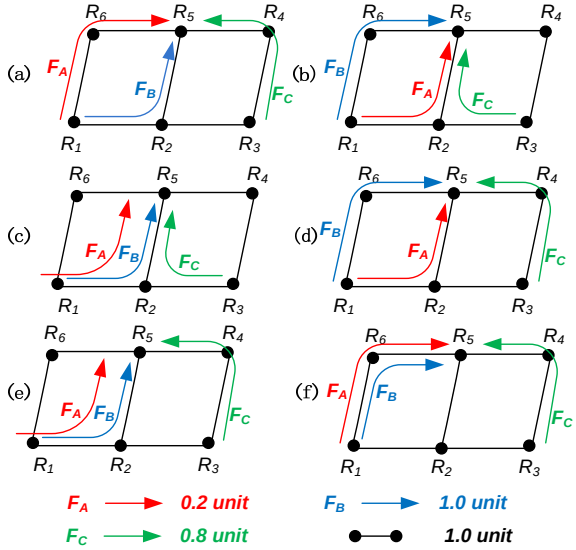


Figure 1: A motivating example.

Prior work addressed this problem by introducing intermediate stages, each of which guarantees that there will be no congestion independent of the update order [4, 7]. This approach, however, has several drawbacks. First, to guarantee that a congestion-free update plan always exists, a portion (10%–50% [4]) of the network capacity has to be left vacant at all times, leading to reduced bandwidth utilization. Second, multipath routing is required in order to make the best use of the vacant capacity during update, which stresses switches with limited flow table entries [3], especially in large-scale data center networks. Third, a series of linear programs need to be solved to find the routing at each stage of the update plan [4, 7]. This is too slow for the operator to react to failures.

In this paper, we take a different approach. We try to find an update plan that *minimizes* the transient congestion, given the number of intermediate stages within which the update has to be done. We argue that, since switches are typically deep buffered, it is worthwhile to explore solutions with a small extend of transient congestion. This problem is more general than congestion-free update in the sense that it allows us to navigate a broader design space, where one may trade off update speed—the number of intermediate stages—for transient congestion. Thus our approach provides a new

flexibility for the operator to specify different tradeoff points for network update depending on the scenario.

As a motivating example, consider Figure 1 where there are six switches $R_1, \ldots, R_6$. The link capacity is one. F_A and F_B are two flows from R_1 to R_5, whose demands are 0.2 and 1.0, respectively. F_C is a flow from R_3 to R_5, whose demand is 0.8. The initial routing is illustrated in Fig. 1(a). At this point, suppose a new flow appears from R_3 to R_4 with a demand of one. The controller then wants to change routing to Fig. 1(b) by updating the forwarding rules of switches. Due to the different update order, the three flows may be temporarily routed as in Fig. 1(c) during transition. In this case congestion occurs at the link from R_2 to R_5, which is overloaded with twice its capacity, and results in severe packet loss.

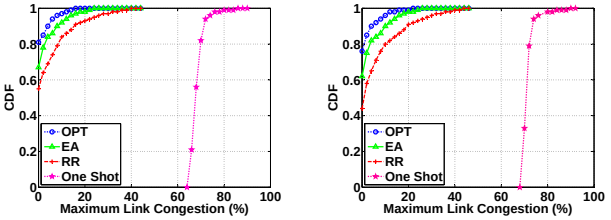
Introducing intermediate stages can reduce transient congestion [4, 7]. zUpdate [7] takes advantage of vacant capacity to find a congestion-free update plan. It first moves 80% of F_B onto path $\langle R_1, R_6, R_5 \rangle$ and keeps the remaining 20% in the place. Then F_A is moved from path $\langle R_1, R_2, R_5 \rangle$ to $\langle R_1, R_6, R_5 \rangle$. Finally the remaining 20% of F_B and all F_C are moved to their final paths. The whole process introduces 4 intermediate stages and solving 4 LPs, one for each stage. Yet a congestion-free update plan may not always exist especially when the number of flows is large. Thus SWAN [4] sets aside some capacity on each link to guarantee its existence. In that case, F_B with demand 1 cannot be completely satisfied through path $\langle R_1, R_2, R_5 \rangle$. It must be split onto different paths. Thus SWAN reduces link utilization and stresses the scarce resource of flow table entries.

In contrast, if we first change routing from Figure 1(a) to Figure 1(d), and then to Figure 1(b), congestion can be reduced significantly. Specifically, transitioning from Figure 1(a) to Figure 1(d) only causes the link capacity to exceed by 0.2, with two possible transient states shown in Figure 1(e) and Figure 1(f), and the transition from Figure 1(d) to Figure 1(b) is congestion-free. So the overall transient congestion is 0.2, and there is only one intermediate stage which requires just one LP. This update plan may be acceptable in some scenarios, because switches have buffers to accommodate bursty traffic, and some applications may not be mission-critical and tolerate packet loss.

2. AN OPTIMIZATION FRAMEWORK

Formulation We propose an optimization framework for the minimum congestion update problem (MCUP) both in data center networks (DCN) and wide area networks (WAN) which connects geographically distributed data centers. Generally speaking, the optimization program aims to determine routing for all n intermediate stages of the update, where n is given, such that the maximum link congestion during the transition is minimized. We model single-path routing for flows as constraints so as to reduce the number of entries required in flow tables.

Algorithm Design Given the scale of the integer program, we aim to develop efficient heuristics for MCUP. We first propose an approximation algorithm based on randomized rounding (RR). It only requires solving one LP and thus has lower complexity than prior work. We further propose an enhanced algorithm (EA) which improves upon the result of randomized rounding by greedily rerouting flows in each stage when possible.



(a) DCN scenario (b) WAN scenario

Figure 2: The link congestion in different scenarios.

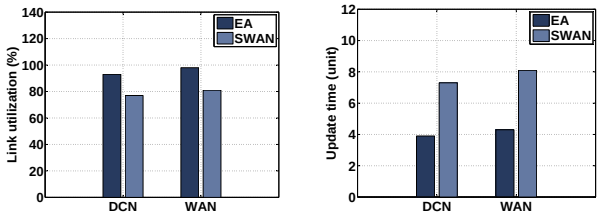


Figure 3: Link utilization Figure 4: Update time

Implementation We develop a prototype of our algorithms using Mininet [6] for performance evaluation. We use Floodlight 0.9 [1] as the controller, which runs on a PC with an Intel Core i5-2400 quad-core processor. Switches are emulated by Mininet and run Openflow v2.0. The switch communicates with the controller and the controller manages the switch via the Openflow protocol. We install forwarding rules via Floodlight's static flow pusher API. We adopt both fat-tree [2] and Microsoft's backbone topology [5] respectively to simulate the scenario of intra-datacenter and inter-datacenter. The link bandwidth is 10 Mb with 1 ms delay and port buffer size is 1 Mb. We use Iperf to generate flows, with average flow size of 5Mb. The source and destination of flows are chosen randomly.

Preliminary Results We study the maximum link congestion with One Shot (transition directly from initial stage to final stage), our algorithms—RR and EA, and OPT obtained using branch and bound method. Figure 2 shows the measured maximum link congestion for One Shot and for RR, EA and OPT when introducing 5 intermediate stages in DCN and WAN scenario respectively. Both EA and RR can effectively decrease congestion: EA and RR decrease link congestion by 67% and 60% respectively compared to One Shot. Furthermore, EA consistently outperforms RR by up to 12% on average, and provides near-optimal performance compared to OPT. Figure 3 shows that for link utilization, EA outperform SWAN by around 20% in both DCN and WAN scenarios. Finally, we look at update time which is defined as the total time units that the routing policies are reconfigured from the initial to final stage without traffic loss. We observe from Figure 4 that EA is almost 50% faster than SWAN.

3. SUMMARY AND FUTURE WORK

We studied the problem of minimizing transient congestion during network update. We proposed efficient heuristics to solve the optimization. Preliminary results show that our solutions lead to higher link utilization and lower update time. We plan to further our study with rigorous analyses of the hardness of the MCUP formulation and the approximation ratios of both algorithms in the near future.

4. REFERENCES

[1] Floodlight. `http://floodlight.openflowhub.org/`.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.

[3] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz. On the effect of forwarding table size on SDN network utilization. In *Proc. IEEE INFOCOM*, 2014.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. ACM SIGCOMM*, 2013.

[5] X. Jin, H. H. Liu, X. Wu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *Proc. ACM SIGCOMM*, 2014.

[6] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proc. ACM HotNets*, 2010.

[7] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz. zUpdate: Updating data center networks with zero loss. In *Proc. ACM SIGCOMM*, 2013.

[8] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proc. ACM SIGCOMM*, 2012.