Time-Aware Congestion-Free Routing Reconfiguration

Shih-Hao Tseng, Chiun Lin Lim, Ning Wu, and Ao Tang
School of Electrical and Computer Engineering, Cornell University, Ithaca, New York 14853, U.S.A.
Email: {st688, cl377, nw276}@cornell.edu, atang@ece.cornell.edu

***Abstract*—A general model is developed to study how network routing can be reconfigured quickly without incurring transient congestion. Assuming both initial and target configurations are congestion-free, it is known that transient congestion may still occur during the reconfiguration process if links contain a mix of traffic flows following old and new routing rules, resulting from variation of switch reaction time and propagation delay differences among paths. We consider these factors by explicitly incorporating timing uncertainty intervals into the model. The model leads to an optimization problem whose solution represents a fast (in terms of actual physical time) congestion-free routing reconfiguration. Our formulation naturally reduces to existing work of finding minimal number of algorithmic update steps when the timing uncertainty intervals are very large, meaning we have little prior knowledge about them. The optimization problem is shown to be a Mixed Integer Linear Program (MILP) with a polynomial-size constraint set, and is proved to be NP-hard. We then further introduce an approximation algorithm with performance guarantee to solve the problem efficiently. Several numerical examples are provided to illustrate our results. In particular, it is demonstrated that timing information can possibly accelerate the update process, even if more steps are involved.**

I. Introduction

Network routes are frequently reconfigured to accomplish tasks such as middlebox traversal constraint satisfaction, virtual machine live migration, and scheduled network maintenance [1]–[5]. There are three key challenges for route reconfiguration, namely *optimality*, *consistency* and *swiftness*.

An optimal update ensures the final routing configuration is the targeted optimal solution. Network operators derive the optimal solution from the new network conditions and traffic demands by solving the well-known multi-commodity flow problem [6]–[8]. In practice, we have various protocols attempting to achieve the *static* solution of the multi-commodity flow problem. Under a distributed setting, routes could be reconfigured with switches choosing different per-destination next-hops as in link-state routing protocols such as OSPF [9]. Alternatively, the routes could be predetermined and route reconfiguration could be done by ingress switches individually selecting different tunnels as in MPLS [10]. A drawback of these distributed methods is that the achieved configuration is not necessarily the desired one [11], [12]. However, having a centralized controller with a global view, as in a Software Defined Network (SDN), can guarantee optimality by directly establishing the optimal routing configuration.

With a centralized controller guaranteeing optimality, the concern moves on to the *transient* stages while getting to the optimal solution, and this is where consistency and swiftness requirements come in. A consistent update ensures certain network properties of interest, such as in-order delivery, loop-freedom or capacity constraint, are satisfied during all transient stages of the routing reconfiguration [13]–[16]. When implemented incorrectly, an inconsistent update could be a very costly exercise to the operator by causing severe service disruptions that would take days to fully recover [17]. On the other hand, swiftness refers to the ability to reconfigure the network from the initial state to the target state in the least amount of time possible. A swift update prevents the new routing setup from becoming obsolete due to fast changing network conditions. This becomes increasingly critical especially for data center networks where traffic dynamics fluctuates very fast [18].

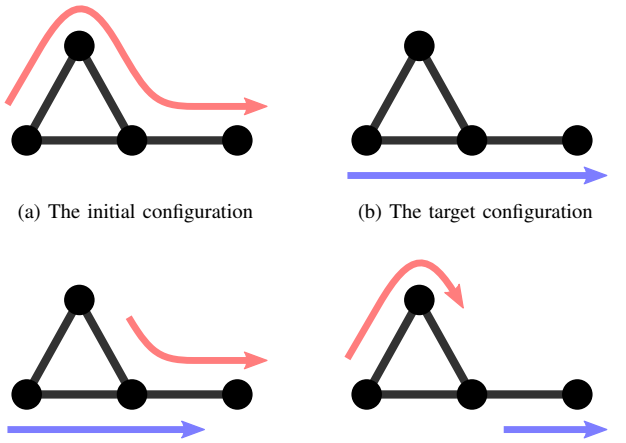
Recently, several methods have been developed to acquire timing information in the network [19]–[21], which enable us to present our approach to achieve fast congestion-free routing reconfiguration. Given an initial routing configuration and a target one, our goal is to produce a series of update steps to move from the initial to the target configuration as quickly as possible while congesting no link during any update step. The key problem is that congestion can still occur during the transient even if both old and new routing configurations are congestion-free, since traffic flows following the new configuration could enter the links containing some traffic flows keeping the old one. We capture this behavior by explicitly incorporating timing information such as propagation delay and time variability into updates.

Our work differs from prior works [15], [22], [23] by optimizing reconfiguration time instead of the number of algorithmic update steps. The key motivation behind the extension is that minimizing the number of update steps does not necessarily translate to a faster update (Example 3). Timing information is useful for network operators to achieve faster reconfiguration as it helps rule out the impossible scenarios which are still considered by worst-case analysis [15]. Our framework reduces to SWAN [15] when the uncertainty dominates the network and we have essentially no prior timing knowledge. zUpdate [22] is also a special case of our framework when we have perfect timing information (zero uncertainty) and the network has layered structure.

II. Background and Motivation

As discussed in Section I, even if initial and the target configurations both obey capacity constraints, congestion may still occur during transient stages. There are two main factors that can lead to transient congestion: propagation delay and timing uncertainty. In this section, we provide two examples

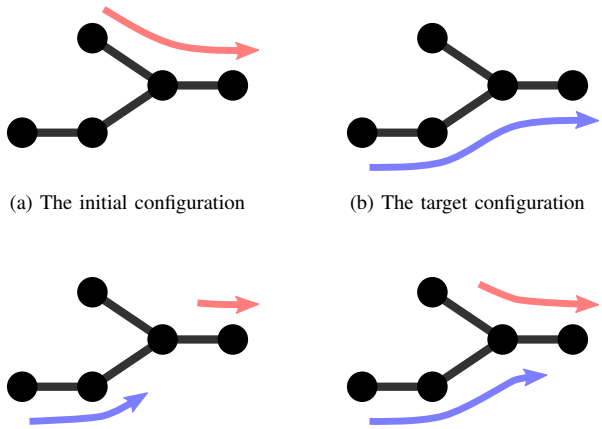
ISBN 978-3-901882-83-8



(a) The initial configuration (b) The target configuration

(c) Transient congestion (d) A congestion-free update exists when the bottom flow switches to the top path

Fig. 1. Differences in propagation delay can cause congestion



(a) The initial configuration (b) The target configuration

(c) A congestion-free update (d) Uncertainty causes congestion

Fig. 2. Timing uncertainty can cause congestion

(Example 1, 2) to intuitively illustrate how that can happen. Furthermore, we also demonstrate how timing information can help achieve faster reconfiguration (Example 3).

A. Causes of Transient Congestion

1) Propagation Delay: When switching traffics between different paths, the differences between propagation delays of the paths may incur transient congestion. We will illustrate this phenomenon with the following example.

Example 1. Consider the network as shown in Fig. 1. The flow along the top path (Fig. 1(a)) is directly rerouted to the bottom one (Fig. 1(b)). Assume the bottom path has a shorter delay, the new flow can arrive at the rightmost link before the previous flow on the longer path clears there (Fig. 1(c)). The shared link of the two paths may thus exceed its capacity and cause congestion.

Propagation delay can cause congestion as Example 1 reveals. However, it also grants better performance. We will discuss this opportunity in Section II-B with Fig. 1(d).

2) Timing Uncertainty: In practice, network operator cannot precisely specify when each path reconfiguration is executed. This can be due to imperfect synchronization among different switches. Some other reasons include the inevitable varying reaction time of different switches for an update instruction [23] as well as varying processing time that cannot be accurately predicted [24].

Although accurate prediction is practically hard, estimations still help order the upcoming events. The arrivals and the clearances of flows on a link can be depicted as intervals, and the uncertainty is reflected by the length of the interval, which converges to zero if the event timing is certain. The overlapping intervals indicate that the link may consist of flows in different configurations at the same time. In the extreme case, all the intervals intersect with each other because of uncertainty, and we call this scenario *order-oblivious*.

Example 2. Fig. 2 shows a part of a network. The operator shifts the flow along the top path (Fig. 2(a)) to somewhere else and routes another flow to the bottom path (Fig. 2(b)). If the two source switches perform the update without uncertainties, the reconfiguration is congestion-free (Fig. 2(c)). However, uncertainty of the source switch on the top may postpone the update and congest the rightmost link (Fig. 2(d)).

B. Benefits of Timing Information

Previous work considers only the order-oblivious worst-case scenario, which assumes the arrivals of the new flows are independent of the others [15]. However, better performance is possible as the timing information reveals the arriving order.

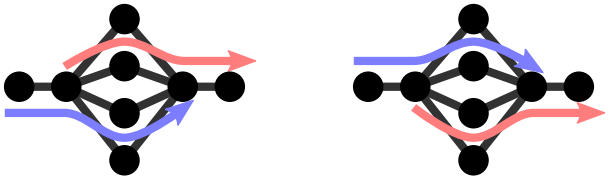
To illustrate this, reversed update in Fig. 1 is considered: moving the flow from the bottom path to the top one. The clearance of the old flow finishes earlier than the arrival of the new flow on the rightmost link (Fig. 1(d)), which suggests a congestion-free one-shot update. Nevertheless, order-oblivious scenario rejects this possibility if the shared link cannot accommodate the two flows simultaneously.

In addition to enlarging the set of feasible update sequences, timing information in general can help accelerate the reconfiguration process. We define the *required time* as the minimum time needed to ensure the change has been fully deployed to the network. For example, the required time for a flow variation along a path is the maximum time needed for the new flow to propagate through the path; the required time for the clearance of an update step is the maximum required time for the flows varied in the step (is zero if all the flows stay the same); the required time for an update sequence is the sum of the required time for the clearance of each step.

Adopting required time can expedite an update. Previous work finds the least step update sequence without timing information [15]. At each step, a long fixed period of time should be waited to avoid the interference between consecutive steps. Replacing the fixed time with the required time yields a more aggressive update. Moreover, an update can achieve shorter required time with more steps.

TABLE I
RELEVANT CONFIGURATIONS

Configurations	Top Traffic				Bottom Traffic			
Initial	0	2	0	0	0	0	2	0
Target	0	0	2	0	0	2	0	0
Intermediate 1	1	0	0	1	0	0	0	2
Intermediate 2	1	1	0	0	0	1	1	0
Intermediate 3	1	0	1	0	0	2	0	0



(a) The initial configuration (b) The target configuration

Fig. 3. Minimizing the number of update steps does not necessarily minimize the update time

Example 3. In Fig. 3, two traffics with the same rate 2 are swapped in a network without uncertainty. Each traffic has four candidate acyclic paths to flow through, and the capacities of the paths are $1, 2, 2, 4$ respectively from the top to the bottom. Each link causes propagation delay 1 unit time except for the bottom two 5-unit delay links. Table I lists the relevant configurations. The columns under each traffic are the flows on the paths. The leftmost column corresponds to the topmost path, and the rightmost column refers to the bottommost one.

Clearly, one-shot update is infeasible, and thus we need to stagger the update into multiple steps. Least step update involves two steps by using the bottom spare path (Intermediate 1). The required time for the update is $11 + 11 = 22$.

However, we can leverage the top path and shift half top flow through it first. The bottom flow is also split half to its target path (Intermediate 2). In the second step, we shift the flows to the target except for the unit flow along the top path (Intermediate 3). Clearing the flow on the top path serves as the last step of the three-step update, which has the required time $3 + 3 + 3 = 9$ faster than the least step solution.

III. FORMULATION

Section II discloses the potential of timing information to achieve a fast congestion-free reconfiguration. Similar to [15], we are interested in congestion-free update in upper bounded number of steps. **In contrast to finding the least-step solution as in [15], we seek the fastest solution.** In this section, we introduce the notations to model the centralized routing scheme and express our model as an optimization problem in Section III-B3.

A. Notations

A centralized controller controls a network consisting of a set V of switches and a set L of directed links. N users utilize the network, and each user n demands a traffic rate d^n from a source switch to a destination switch (or simply the source/destination). A set of acyclic paths P^n is predetermined and established for each user n to communicate between the source and the destination. The controller performs routing by specifying the split ratio among the paths in P^n for each user n at its source. Define $P = \bigcup_{n \in N} P^n$ as all acyclic paths.

TABLE II
DEFINITIONS OF THE MAJOR VARIABLES

For each link $l \in L$	
c_l	The capacity on the link l
f_l	The total flow on the link l
P_l	The set of paths passing through the link l
$P_l^{\mp}$	A P-partition (P_l^-, P_l^+)
$\mathcal{P}_l$	The collection of all valid P-partitions of P_l
For each user $n \in N$	
d^n	The demanded flow rate into the network for the user n
P^n	The set of acyclic paths for the user n
For each path $p \in P$	
x_p	The flow on the path p
z_p	The binary integer variable for the path p
$\left[w_p^{\min}, w_p^{\max}\right]$	The time interval for the updated flow to clear the path or subpath p
For each level $0 \le \pi \le \pi^{\max}$ (see Section IV)	
r_π	The level variable for the level π
$\overline{w_\pi^{\max}}$	The required time for the level π
Other variable	
u	The required time to finish an update step

For each link l in a path $p \in P$, q_{pl} denotes the subpath from the source to the switch prior to link l. For example, if $p = (v_1, l_1, v_2, l_2, v_3)$ connects the source v_1 and the destination v_3, $q_{pl_2} = (v_1, l_1, v_2)$. Define $P_l = \{p \in P : l \in p\}$ as the paths passing through link l, which is a subset of P.

Define x_p as the rate of the *flow* along a path p. f_l denotes the total flow injecting to link l. Due to the timing issues discussed in Section II, we assume the delays to pass through switches and links are interval-ranged. Those delays are accumulated along the paths and encountered by the flows. An interval-ranged delay is also assumed for an update instruction between being issued and becoming effective.

The interval $[w_p^{\min}, w_p^{\max}]$ specifies the time needed for a packet to go through a path p. The lower bound $w_p^{\min}$ is derived by adding all the lower bound of the delays for the source-switch instruction update and the on-path device transitions. Analogous computation gives the upper bound $w_p^{\max}$. Notice that $w_p^{\max}$ is the required time for the path p, which is introduced in Section II-B. We can define the interval $[w_q^{\min}, w_q^{\max}]$ for a subpath q in the similar way.

An update sequence generally consists of multiple steps, which are labeled by step numbers a in chronological order. The parentheses enclosing a is attached behind a variable to denote the value of the variable between step a and $a + 1$. For example, $x_p(a)$ is the flow along path p after the network applies update step a; $u(a)$ is the required time for the update to fully switch from step a to $a+1$ configuration. t_a refers to the time elapsed since the step a is applied to the network.

We may drop the subscript of path p or the step number a to refer to a vector, such as x and u.

B. Model

1) Objective: Given an upper bound b, our objective is to find the fastest update sequence in b steps to reconfigure the network from a given initial state to a given target state, both static feasible and congestion-free, while remaining

congestion-free during the whole transient stage. The initial and the target routing configurations are denoted by $x_p(0)$ and $x_p(b)$ for all $p \in P$, respectively. Without loss of generality, we assume there exists $p \in P$ such that $x_p(0) \neq x_p(b)$. Finding an update sequence in b steps involves deciding a series of flows $x_p(a)$ for all $p \in P$ and $a = 1, \ldots, b-1$.

For simplicity, we impose the assumption that the user traffic demands remain the same during the whole update, although our framework can easily deal with general cases. Extending the update mechanism in [15], we also assume the centralized controller waits the required time $u(a)$ between consecutive steps a and $a+1$ to ensure the full deployment of the new update $(a+1)$ before performing the next $(a+2)$.

2) Constraints: According to the update mechanism assumption in Section III-B1, each flow can only follow either step a or $a+1$ configuration at an arbitrary time t_a after each step a. Congestion-free property requires that all the possible combinations of the flows at each link do not exceed the capacity. The number of potential combinations is exponential to the number of flows [22], thus it is critical to identify the essential ones. Example 4 shows how timing information helps select the relevant constraints and avoid enumerating all the unnecessary combinations.

Example 4. Consider a link l shared by two paths $p_1, p_2 \in P$. For simplicity, we use $q_1 = q_{p_1 l}$ and $q_2 = q_{p_2 l}$ in this example.

Once updated from step a to $a+1$, the flow along p_i arrives at the link l within the time interval $\left[w_{q_i}^{\min}, w_{q_i}^{\max}\right]$ for $i = 1, 2$. When the time intervals overlap, such as $w_{q_1}^{\min} \leq w_{q_2}^{\min} \leq w_{q_1}^{\max}$, either flow can arrive before the other. That results in the order-oblivious scenario, and all possible combinations should be covered by the following four constraints:

$$\begin{aligned} x_{p_1}(a) + x_{p_2}(a) &\leq c_l, \quad & x_{p_1}(a+1) + x_{p_2}(a) &\leq c_l, \\ x_{p_1}(a) + x_{p_2}(a+1) &\leq c_l, \quad & x_{p_1}(a+1) + x_{p_2}(a+1) &\leq c_l, \end{aligned}$$

or we can represent them as

$$\max(x_{p_1}(a), x_{p_1}(a+1)) + \max(x_{p_2}(a), x_{p_2}(a+1)) \leq c_l.$$

When the two intervals are separated, assuming $w_{q_1}^{\max} < w_{q_2}^{\min}$ without loss of generality, $x_{p_1}(a+1)$ always arrives earlier than $x_{p_2}(a+1)$. Instead of considering all four combinations of flows, only three combinations are possible and they lead to the less strict constraint set:

$$\begin{aligned} x_{p_1}(a) &+ x_{p_2}(a) &&\leq c_l, \\ x_{p_1}(a+1) &+ \max(x_{p_2}(a), x_{p_2}(a+1)) &&\leq c_l. \end{aligned}$$

To write down the congestion-free constraints, we define a *P-partition* of P_l as a pair of subsets $P_l^{\mp} = (P_l^-, P_l^+)$ such that $P_l^- \cap P_l^+ = \emptyset$ and $P_l^- \cup P_l^+ = P_l$. With the P-partition notation, we can express each of the $2^{|P_l|}$ order-oblivious constraints at each link l in the following form:

$$f_l(P_l^{\mp}, a) = \sum_{p \in P_l^-} x_p(a) + \sum_{p \in P_l^+} x_p(a+1) \leq c_l$$

where $f_l(P_l^{\mp}, a)$ is the corresponding total flow on the link. In Example 4, $P_l = \{p_1, p_2\}$, and four possible P-partitions $P_l^{\mp} = (P_l^-, P_l^+)$ are $(\{p_1, p_2\}, \emptyset)$, $(\{p_1\}, \{p_2\})$, $(\{p_2\}, \{p_1\})$, and $(\emptyset, \{p_1, p_2\})$, corresponding to the four $f_l(P_l^{\mp}, a) \leq c_l$ constraints respectively.

We then characterize the P-partitions $P_l^{\mp}$ which lead to possible flow combinations $f_l(P_l^{\mp}, a)$. A P-partition is *valid* if there exists a time interval $[t_a, t_a + \epsilon]$ for some constant $\epsilon > 0$[1], such that all the flows in P_l^+ have its update propagated to link l while those in P_l^- still remain in the old configuration. We define the collection $\mathcal{P}_l$ as the set of all the valid P-partitions of P_l. As demonstrated in Example 4, we can decide the validity of a P-partition by scrutinizing the arrival intervals of the flows. In particular, there must exist $t_a \geq 0$ and $\epsilon > 0$ for a valid P-partition such that $t_a \geq w_{q_{pl}}^{\min}$ for every $p \in P_l^+$ and $t_a + \epsilon \leq w_{q_{pl}}^{\max}$, for every $p \in P_l^-$. Therefore, we have the following equation:

$$\max_{q_{pl}: p \in P_l^+} w_{q_{pl}}^{\min} < \min_{q_{pl}: p \in P_l^-} w_{q_{pl}}^{\max}. \tag{1}$$

Since $|P|$ is finite, both sides of the inequality are attained finite, which implies the existence of ϵ by setting it as the gap. It is clear that a P-partition satisfies the condition (1) if and only if it is valid.

3) Optimization Problem: Now we summarize the objective in Section III-B1 and the constraints in Section III-B2 into an optimization problem. Our objective can be expressed as an optimization problem of minimizing the objective function $\sum_{a=0}^{b-1} u(a)$, which is the required time for the resulted update sequence. At each step a, the required time can be written as

$$u(a) = \max\left\{w_p^{\max} : x_p(a+1) \neq x_p(a)\right\}.$$

By convention, we set the expression to zero if none of the flows is updated. We create the artificial binary integer variable $z_p(a)$ for each path p, which is one if and only if $x_p(a+1) \neq x_p(a)$, so that

$$u(a) = \max_{p \in P}\left\{w_p^{\max} z_p(a)\right\}. \tag{2}$$

We name the optimization problem Fast Congestion-free Reconfiguration problem in b steps (FCR(b), or simply FCR for an arbitrary b), and formulate it in a form of Mixed Integer Linear Programming (MILP):

$$\begin{aligned} \min \quad & \sum_{a=0}^{b-1} u(a) \\ \text{s.t.} \quad & f_l(P_l^{\mp}, a) \leq c_l && \forall l \in L, P_l^{\mp} \in \mathcal{P}_l \\ & && 0 \leq a \leq b-1 && (3) \\ & \sum_{p \in P^n} x_p(a) = d^n && \forall n \in N, 1 \leq a \leq b-1 && (4) \\ & x_p(a) \geq 0 && \forall p \in P, 1 \leq a \leq b-1 && (5) \\ & x_p(0), x_p(b) \text{ are given} && \forall p \in P && (6) \\ & u(a) \geq w_p^{\max} \cdot z_p(a) && \forall p \in P, 0 \leq a \leq b-1 && (7) \\ & z_p(a) \cdot \alpha_p \geq |x_p(a+1) - x_p(a)| && \forall p \in P, 0 \leq a \leq b-1 && (8) \\ & z_p(a) \in \{0, 1\} && \forall p \in P, 0 \leq a \leq b-1 && (9) \end{aligned}$$

[1]By requiring $\epsilon > 0$, we exclude the case that the network congests for 0 time.

where the constant α_p for path p is set as the bottleneck link capacity $\min_{l\in p} c_l$ so that $z_p(a)$ behaves as expected.

The constraints (3) - (6) are the feasibility constraints. x satisfying these constraints gives a feasible congestion-free update sequence. We introduce the constraints (7) - (9) to realize the relationship (2). The optimal value of FCR(b) is written as $\mathrm{OPT}_{\mathrm{FCR}}(b)$ for given step upper bound b. One important observation is that if FCR is feasible in b steps, it is also feasible for $b+1$ steps and $\mathrm{OPT}_{\mathrm{FCR}}(b+1) \leq \mathrm{OPT}_{\mathrm{FCR}}(b)$. Conversely, if the problem is not feasible in b steps, there exists no solution for fewer steps.

IV. Main Results

In Section III-B, we formulate the reconfiguration problem as an MILP problem FCR. In this section, we present a proof sketch of the NP-hardness of FCR, which motivates us to develop an approximation algorithm for FCR. In particular, we provide a polynomial time relaxation-rounding based approximation algorithm, which involves only the Linear Program (LP) solutions. We start the section with Theorem 1.

Theorem 1. FCR *is NP-hard.*

Theorem 1 can be proved by showing that 3-SAT, a well-know NP-complete problem, polynomial reduces to FCR. For each 3-SAT instance, there exist a network and its corresponding initial and target configurations such that the optimal FCR solution meets the lower bound if and only if the 3-SAT instance is satisfiable. We omit the details as they are not the main focus of this work.

Since FCR is NP-hard, we develop an approximation algorithm for FCR as the following. We first define the term *kernel*. A kernel $A(b)$ is an algorithm which gives a feasible solution to FCR(b) when it is feasible; and FCR is not feasible in b steps if $A(b)$ finds no solution to it. We write $\mathrm{SOL}_A(b)$ to denote the objective value of the feasible solution given by the kernel $A(b)$, and we set $\mathrm{SOL}_A(b) = \infty$ if the kernel $A(b)$ declares FCR(b) infeasible. We then propose the algorithm $\mathrm{ALG}[A](b)$ (Algorithm 1) with respect to a kernel $A(\cdot)$, which essentially picks the best solution from the feasible solutions generated by $A(1), A(2), \cdots, A(b)$.

Algorithm 1 Algorithm $\mathrm{ALG}[A](b)$

1: $\mathrm{SOL}_{\mathrm{ALG}[A]}(b) \leftarrow \min_{1\leq\hat{b}\leq b} \mathrm{SOL}_A(\hat{b})$.
2: **if** $\mathrm{SOL}_{\mathrm{ALG}[A]}(b) = \infty$ **then**
3: Output "no congestion-free solution in b steps."
4: **else**
5: Output the solution corresponding to the minimum $\mathrm{SOL}_A(\cdot)$.
6: **end if**

If the required time $u(a)$ for each step given by the kernel A is upper bounded by a constant $W^{\max}$ and the required time $w_p^{\max}$ for each path is lower bounded by a constant $W^{\min}$, we have the following theorem to ensure that $\mathrm{ALG}[A](b)$ is a $\frac{W^{\max}}{W^{\min}}$-approximation algorithm.

Theorem 2. *Let $W^{\max}$ and $W^{\min}$ be positive constants and u belong to the feasible solution to* $\mathrm{FCR}(b)$ *given by a kernel $A(b)$.* $\mathrm{ALG}[A](b)$ *is a $\frac{W^{\max}}{W^{\min}}$-approximation algorithm, if the following conditions hold:*

- $0 \leq u(a) \leq W^{\max}$ *for all* $0 \leq a \leq b-1$.
- $w_p^{\max} \geq W^{\min}$ *for all* $p \in P$.

Proof. If FCR(b) is infeasible, the kernel algorithm gives $\mathrm{SOL}_A(\hat{b}) = \infty$ for all $1 \leq \hat{b} \leq b$ and $\mathrm{ALG}[A](b)$ declares infeasibility of FCR(b). If FCR(b) is feasible, there exists $1 \leq b' \leq b$ such that $OPT_{FCR}(b') = OPT_{FCR}(b)$ and $u(a) > 0$ for all $0 \leq a \leq b'-1$. Since $u(a) \geq w_p^{\max} z_p(a)$ for all $p \in P$, we know $\exists p' \in P$ such that $z_{p'}(a) = 1$, and hence $u(a) \geq w_{p'}^{\max} \geq W^{\min}$. Denote this optimal u by u_{OPT}. Let the solution u corresponding to $\mathrm{SOL}_A(b)$ be u_b, we know

$$
\begin{aligned}
&\mathrm{SOL}_{\mathrm{ALG}[A]}(b) = \min_{1\leq\hat{b}\leq b} \mathrm{SOL}_A(\hat{b}) \\
&\leq \mathrm{SOL}_A(b') = \sum_{a=0}^{b'-1} u_{b'}(a) \\
&\leq \sum_{a=0}^{b'-1} W^{\max} \leq \sum_{a=0}^{b'-1} W^{\max} \frac{u_{\mathrm{OPT}}(a)}{W^{\min}} = \frac{W^{\max}}{W^{\min}} \sum_{a=0}^{b'-1} u_{\mathrm{OPT}}(a) \\
&= \frac{W^{\max}}{W^{\min}} OPT_{FCR}(b') = \frac{W^{\max}}{W^{\min}} OPT_{FCR}(b).
\end{aligned}
$$

□

There are generally more than one kernels satisfying the first condition in Theorem 2. For instance, we have the **kernel $R_{\mathrm{FCR}}(b)$**, which solves the linear relaxation of FCR(b), uprounds all the resulted non-zero $z_p(a)$ to 1 and sets $u(a) = \max_{p:z_p(a)=1} w_p^{\max} \leq \max_{p\in P} w_p^{\max}$. In that case, $W^{\max} = \max_{p\in P} w_p^{\max}$ and $\mathrm{ALG}[R_{\mathrm{FCR}}](b)$ is a $\frac{\max_{p\in P} w_p^{\max}}{\min_{p\in P} w_p^{\max}}$-approximation algorithm.

Unlike the kernel-independent approximation ratio, the complexity of $\mathrm{ALG}[A](b)$ depends on the complexity of its kernel. If the kernel $A(b)$ is a polynomial-time algorithm, $\mathrm{ALG}[A](b)$ also terminates in polynomial time. LP-based kernel, such as $R_{\mathrm{FCR}}(b)$, is indeed polynomial-time when the constraint set is also in polynomial size of its input variables. Among the constraints of FCR, we only need to find an polynomial-size expression of the constraint (3) to have a polynomial-time LP-based kernel. Equivalently, we need to consider all valid P-partitions $P_l^{\mp}$ and their corresponding total flows $f_l(P_l^{\mp}, a)$. We develop a polynomial time constraint set generation algorithm (Algorithm 2) and give a theorem (Theorem 3) to prove that the generated constraint set is equivalent to the one derived from all valid P-partitions.

Theorem 3. *A solution x satisfies the constraint (3) if and only if there exists s such that (x, s) is feasible for the constraints generated by the Algorithm 2.*

Theorem 3 is proved by showing that every valid P-partition can be expressed as $(\Sigma_y \cup \Sigma_c^-, \Sigma_c^+ \cup \Sigma_u)$ and each $(\Sigma_y \cup \Sigma_c^-, \Sigma_c^+ \cup \Sigma_u)$ is a valid P-partition, where Σ_y, Σ_c and Σ_u are obtained at some iteration of Algorithm 2 and Σ_c^-, Σ_c^+ is a partition of Σ_c. The proof is straightforward, and we omit the details here because of the space limitation.

Besides the size of the constraint set, the number of input variables also plays an important role in the time complexity

Algorithm 2 Constraint Set Generation

for each path $p \in P$ and $0 \le a \le b-1$ **do**
  Add a slack variable $s_p(a)$ and two slack constraints
$$s_p(a) \ge x_p(a), \qquad s_p(a) \ge x_p(a+1).$$
end for
for $l \in L$ and $a = 0$ **to** $b-1$ **do**
  $P_l \leftarrow \{p \in P : l \in p\}$
  Set the updated set $\Sigma_u \leftarrow \emptyset$.
  Set the current set $\Sigma_c \leftarrow \emptyset$.
  Set the yet-updated set $\Sigma_y \leftarrow P_l$.
  Collect $w^{\max}_{q_{pl}}$ and $w^{\min}_{q_{pl}}$ for all $p \in P_l$ to be a list.
  for w in the list from the smallest to the largest **do**
    while $w = w^{\min}_{q_{pl}}$ for some $p \in \Sigma_y$ **do**
      Remove p from Σ_y and add it to Σ_c.
    end while
    while $w = w^{\max}_{q_{pl}}$ for some $p \in \Sigma_c$ **do**
      Remove p from Σ_c and add it to Σ_u.
    end while
    Generate a constraint
$$\sum_{p\in\Sigma_y} x_p(a) + \sum_{p\in\Sigma_c} s_p(a) + \sum_{p\in\Sigma_u} x_p(a+1) \le c_l.$$
  end for
end for

of a kernel. $R_{\mathrm{FCR}}(b)$ introduces $b|P|$ more artificial variables $z_p(a)$. There exist algorithms introducing fewer variables while still giving a feasible solution to FCR(b). We now introduce the concept of *level* in the following paragraphs and show how it incorporates timing benefits with fewer additional variables.

The concept of level is motivated by the constraint (7). Notice that $u(a)$ is determined by the flow variations on the paths with the longest $w^{\max}_p$. Hence, we can partition P into several level sets and assign each path a level π_p according to the level set it belongs to. Each level π associates with a required time $\overline{w^{\max}_\pi}$ such that $u(a) > \overline{w^{\max}_{\pi-1}}$ while any flow along a level π path changes between step a and $a+1$. In other words, each path p in level $\pi_p = \pi$ satisfies $\overline{w^{\max}_{\pi-1}} < w^{\max}_p \le \overline{w^{\max}_\pi}$. We define the level of the paths with the longest required time to be $\pi^{\max}$ and level 0 has $\overline{w^{\max}_0} = 0$. Without loss of generality, we can set $\overline{w^{\max}_{\pi^{\max}}} = \max_{p\in P} w^{\max}_p$ and by definition

$$0 = \overline{w^{\max}_0} < \overline{w^{\max}_1} < \cdots < \overline{w^{\max}_{\pi^{\max}}} = \max_{p\in P} w^{\max}_p.$$

We create the binary level variable $r_\pi(a)$ to indicate whether the level π is the highest level involving flow change between step a and $a+1$. $r_\pi(a) = 1$ if and only if a flow along a level π path changes and all the flows along higher level paths remain the same between step a and $a+1$. We say a level variable r depicts a FCR solution x if $r_\pi(a) = 1$ implies $x_p(a) = x_p(a+1)$ for every path p with $\pi_p > \pi$. By definition, if the level variable r depicts a feasible solution x, we can construct a feasible solution (x, u, z) by setting $z_p(a) = \left(\sum_{i=\pi_p}^{\pi^{\max}} r_i(a)\right)$ and $u(a) = \sum_{i=0}^{\pi^{\max}} \overline{w^{\max}_i} r_i(a)$ for all $p \in P$ and $0 \le a \le b-1$. Hence, finding a feasible (x, u, z) to FCR is equivalent to finding a feasible x and a level variable r depicting x.

We can find a feasible x and its depicting level variable r by the transformed FCR problem with $\pi^{\max}$ levels in b steps (tFCR$\langle\pi^{\max}\rangle(b)$):

$$\begin{aligned}
\min \quad & \sum_{a=0}^{b-1} u(a) \\
\text{s.t.} \quad & x \in \mathcal{X} && (10)\\
& u(a) = \sum_{i=1}^{\pi^{\max}} \overline{w^{\max}_i} \cdot r_i(a) \quad \forall 0 \le a \le b-1 \\
& r_0(a) + \sum_{i=1}^{\pi^{\max}} r_i(a) = 1 \qquad \forall 0 \le a \le b-1 \\
& \left(\sum_{i=\pi_p}^{\pi^{\max}} r_i(a)\right) \cdot \alpha_p \ge |x_p(a+1) - x_p(a)| \\
& \qquad\qquad \forall p \in P, 0 \le a \le b-1 \\
& r_i(a) \in \{0,1\} \qquad\qquad \forall 0 \le i \le \pi^{\max}, \\
& \qquad\qquad 0 \le a \le b-1 && (11)
\end{aligned}$$

where the constraint (10) represents the constraints (3) - (6).

Consider the subproblem tFCRs$\langle\pi^{\max}\rangle(b)$ which assumes $u(a) > 0$ (or $r_0(a) = 0$) for all steps. We form tFCRs$\langle\pi^{\max}\rangle(b)$ by removing all $r_0(a)$ from tFCR$\langle\pi^{\max}\rangle(b)$. Then we propose the **kernel $R_{\mathbf{tFCRs}\langle\boldsymbol{\pi^{\max}}\rangle}(\boldsymbol{b})$**, which solves the linear relaxation of tFCRs$\langle\pi^{\max}\rangle(b)$ (relax the constraint (11)), uprounds the non-zero $r_i(a)$ with the highest level to 1 (the other level variables are set to 0) and sets $u(a)$ to its corresponding $\overline{w^{\max}_i}$ for each a. $R_{\mathrm{tFCRs}}\langle\pi^{\max}\rangle(b)$ solves a feasible r for every feasible x, and hence gives feasible u and z to FCR(b). Theorem 2 suggests that $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle\pi^{\max}\rangle}](b)$ is also a $\frac{\max_{p\in P} w^{\max}_p}{\min_{p\in P} w^{\max}_p}$-approximation algorithm. Nevertheless, we add only $b\pi^{\max}$ more variables in $R_{\mathrm{tFCRs}\langle\pi^{\max}\rangle}(b)$ instead of $b|P|$ in $R_{\mathrm{FCR}}(b)$. In practice, kernel $R_{\mathrm{tFCRs}\langle\pi^{\max}\rangle}(b)$ can even perform near-optimal (much better than $R_{\mathrm{FCR}}(b)$) as shown in Section V-A and V-B.

We know that $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 1\rangle}](b)$ gives the least-step solution. Under order-oblivious scenario, it gives the same update steps as the SWAN solution [15]. SWAN considers no timing information, so its solutions wait at least $\max_{p\in P} w^{\max}_p$ between steps, which equals to $\overline{w^{\max}_1}$. Thus $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 1\rangle}](b)$ performs at least the same as SWAN. Since a multilevel solution can be truncated to a single level solution, we have $\mathrm{SOL}_{ALG[R_{\mathrm{tFCRs}\langle 1\rangle}](b)} \ge \mathrm{SOL}_{ALG[R_{\mathrm{tFCRs}\langle \pi\rangle}](b)}$ if $\pi > 1$. Thus, by considering multilevel $\pi > 1$, $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle\pi\rangle}](b)$ outperforms SWAN.

V. Simulations

We have implemented Algorithm 1 and 2 with two different kernels mentioned in Section IV and the two state-of-the-art methods regarding transient congestion-free reconfiguration on ns-3.24 [25]. Together with the optimal solution, they are summarized below:

- $\mathrm{OPT}_{\mathrm{FCR}}$: The optimal FCR solution obtained by solving the MILP problem.
- $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}]$: Algorithm 1 with the LP-based kernel solving the linear-relaxed transformed FCR subproblem involving 3 levels (set $\pi^{\max}=3$, delete $r_0(a)$ and relax the constraint (11)).
- $\mathrm{ALG}[R_{\mathrm{FCR}}]$: Algorithm 1 with the LP-based kernel solving the linear-relaxed FCR problem (relax the constraint (9)).
- SWAN: The order-oblivious solution in [15] with the step waiting time equal to the maximum path required time.
- zUpdate: The switch-based routing method proposed in [22] for layered structured networks.

Those algorithms are installed on a controller which communicates with other OpenFlow switches via OpenFlow protocol 0.8.9 [26], which is the latest supported version in ns-3.24, with Multiprotocol Label Switching (MPLS) extension. Controller conducts tunnel-based routing via extended switches supporting Weighted Cost Multipath (WCMP). For the algorithms, CBC-2.9.0 [27] serves as the LP/MILP solver. All measurements are obtained from a 2.4GHz quad core laptop with 8 GB memory on Fedora 20.

We compare the algorithms with the step upper bound $b=10$. The initial and target configurations for the experiments in Section V-B and V-C result from user and traffic generation. Users are set by a Bernoulli process: each source-destination pair can be selected as a user with a specified probability. For each selected user, 2 acyclic paths are predetermined by Yen's k-shortest-path Algorithm [28] with $k=2$ to send traffic through. Each user uses UDP to send 1 kb packets at a constant data rate which distributes uniformly between 0 and 1 Mbps. We vary the data rate to form the initial and target configurations.

Each link capacity is set as $1/(1-\lambda)$ times the maximum traffic the link might carry under both configurations. As such, a scratch capacity rate λ for every link is guaranteed, and hence a congestion-free update sequence exists as shown in [15]. The scratch capacity rate results from the fact that the backbone links are in general underutilized [29]. We alter λ in each case to compare the algorithms under different scenarios.

Table III shows some attributes of the algorithms we will compare in this section. We first examine Example 3 in Section V-A to compare the performance of the algorithms. In Section V-B, we show how our algorithm helps a practical large scale inter-data center network achieve faster reconfiguration in reasonable time. Finally, we demonstrate how uncertainty can cause congestion significantly in a data center network with layered structure in Section V-C.

A. A Simple Example

We solve Example 3 by four applicable algorithms (zUpdate is excluded because it requires equal delay between layers), and the results are shown in Table IV.

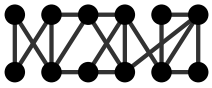
$\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ attains the 3-step optimal solution as given by $\mathrm{OPT}_{\mathrm{FCR}}(10)$. $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ not only introduces less variables but also achieves a better solution than $\mathrm{ALG}[R_{\mathrm{FCR}}](10)$. By comparing $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ with SWAN, we find that the update time can be shortened with the help of the timing information.

TABLE III
COMPARISON OF THE METHODS

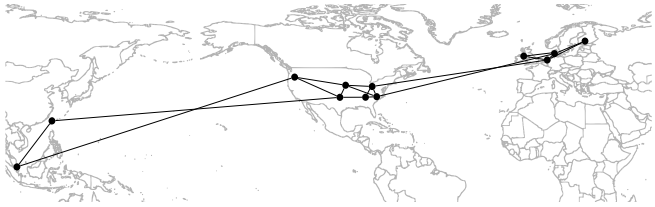
Method	$\mathrm{ALG}[R_{\mathrm{tFCRs}\langle\cdot\rangle}]$	SWAN	zUpdate
Applicable Network	arbitrary	arbitrary	layered structure
Update Objective	minimum time	minimum step	minimum step
Applicable Routing	tunnel-based	tunnel-based	switch-based
Uncertainty Tolerance	yes	yes	no

TABLE IV
PERFORMANCE COMPARISON

Method	Solution Steps	Update Time (unit)
$\mathrm{OPT}_{\mathrm{FCR}}(10)$	10	9
$\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$	3	9
$\mathrm{ALG}[R_{\mathrm{FCR}}](10)$	2	22
SWAN	2	22



(a) The B4 topology



(b) The geographical distribution of the Google Data Centers

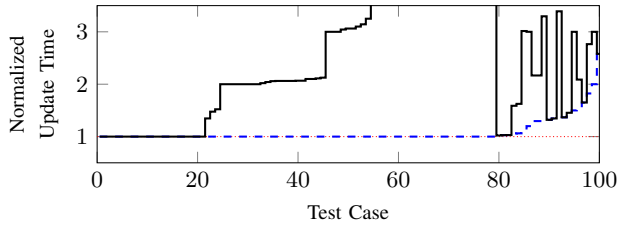
Fig. 4. The B4 topology of 12 data centers

By Theorem 2, the theoretical approximation ratio is $\frac{11}{3}$. In this case, $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ achieves the ratio 1 and $\mathrm{ALG}[R_{\mathrm{FCR}}](10)$ achieves $\frac{22}{9}$.

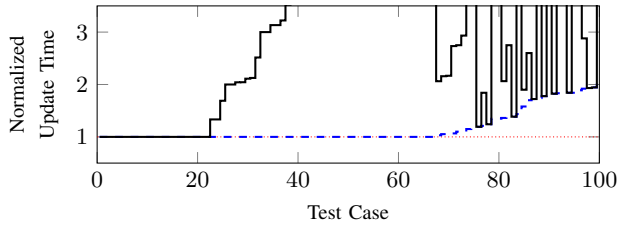
B. WAN (Inter-Datacenter Network)

Our method is also applicable for wide area networks (WANs), such as Google B4 [30]. Google implements B4 to connect their data centers [31]. We create a network consisting of 12 nodes representing those data centers and 19 interconnected links based on the topology described in [30] (Fig. 4(a)) with the link latency (ms) proportional to their actual geographical distance as shown in [31] (Fig. 4(b)). We assume the data centers perform packet switching within a millisecond, which contributes to the uncertainty intervals.

100 random traffic patterns are generated for both $\lambda=10\%$ and $\lambda=5\%$ by adding source-destination pair with 0.05 probability. We solve the patterns by $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$, SWAN and $\mathrm{OPT}_{\mathrm{FCR}}(10)$. We know $\mathrm{OPT}_{\mathrm{FCR}}(10)$ gives the shortest update time in 10 steps. Thus for each test case, we normalize the results of $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ and SWAN by $\mathrm{OPT}_{\mathrm{FCR}}(10)$ if they have solutions (Fig. 5). We sort the test cases by the normalized update time of $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$.



(a) Scratch capacity rate $\lambda = 10\%$



(b) Scratch capacity rate $\lambda = 5\%$

Fig. 5. The resulted update time normalized by the shortest update time. Dashed line: $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$; normal line: SWAN; dotted constant 1 line: $\mathrm{OPT}_{\mathrm{FCR}}(10)$

When $\lambda = 10\%$, $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ updates strictly faster than SWAN in 70 out of 100 cases. Also, we can find that $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ takes less than 2 times the shortest update time given by $\mathrm{OPT}_{\mathrm{FCR}}(10)$ in general (Fig. 5(a)).

Whilst $\lambda = 5\%$, there exists no guarantee that we can find a congestion-free update plan in 10 steps. We collect 100 solvable cases and 11 unsolvable ones. SWAN fails in all 11 unsolvable cases, while our method $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ and the optimal method $\mathrm{OPT}_{\mathrm{FCR}}(10)$ can still provide congestion-free reconfiguration update plans for 7 cases because of the timing information. In 69% of the solvable cases, $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ strictly outperforms SWAN (Fig. 5(b)). Again, we can observe that the normalized update time of $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$ is mostly less than 2, even though Theorem 2 promises $\frac{W^{\max}}{W^{\min}}$ only[2].

To verify the congestion-free property, we adjust the buffer size of every network card interface to be only two packets in ns-3 (one in progress and one arriving). For each solution, we monitor the packet drop event. No packet is dropped during the reconfiguration, which implies those methods are truly congestion-free.

This simple example shows how timing information enables us to expand the feasible solution set and update faster. In fact, SWAN considers only the order-oblivious case when the network is totally uncertain for the operator, which is just an extreme case of our framework.

C. Layered Structure (Intra-Datacenter Network)

For a layered network, zUpdate [22] searches for a switch-based least step congestion-free update sequence toward a target set of configurations described by constraints, which can be the target state. The flows from the same user do not interfere with each other, resulting from the assumptions of the layered structure and the absence of uncertainty. Hence, it is another special case of our framework with uncertainty issue eliminated. In practice, rule change will not take effect immediately and thus the deviation can cause congestion during the transient stage.

Fat-tree topology [32] is a layered network structure proposed for data center networks. We implement a simple fat-tree network with 1 ms delay links, as shown in Fig. 6(a), to verify the uncertainty effect. Each circle mark represents a switch and the users are located at the squares. We select source-destination pairs as users with probability 0.1. The scratch capacity rate is set to $\lambda = 17\%$ and we find congestion-free update plans in 5 steps.

We consider two timing uncertainty effects: rule-update processing delay and packet switching delay. When we update the rules of an user at a switch, we encounter a processing delay uniformly distributed over $[0, \delta_d]$ (ms); as a flow arrives at a switch $v \in V$, it gets delayed by a time uniformly distributed over $[0, \delta_v]$ (ms) before it leaves the output interface. We apply both $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ and zUpdate to solve for congestion-free update plans under $\delta_d = 0.5$ and $\delta_v = 10$. Our method is tunnel-based, while zUpdate reconfigures the network switch-by-switch. We assume further that once a new rule-update instruction is set to a switch, the switch discards the previous in-progress update and starts adopting the new rules.

The time domain simulations are done in MATLAB. We do not simulate the link congestion phenomenon, such as buffering or packet dropping, since those decisions are operator-dependent. Instead, we simply allow flows to exceed the link capacity and we define the *utilization* of a link as the total flow on the link divided by its capacity. When the utilization is greater than one, it implies congestion occurs in the network (not necessarily on the corresponding link).

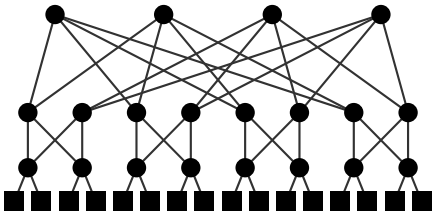
Both $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ and zUpdate are examined under three different network conditions: without uncertainty, low uncertainty and high uncertainty. We simulate the zUpdate solution without uncertainty, pick the most utilized link during the reconfiguration and show its utilization along the time under each scenario. The simulation results are shown in Fig. 6(b), 6(c) and 6(d). The left charts are the update results of $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$, while the right ones belong to zUpdate.

Both $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ and zUpdate are congestion-free without uncertainty (Fig. 6(b)). However, uncertainty may result in timing deviation and cause congestion for the zUpdate solution (Fig. 6(c)). The less precise control we can achieve, the more congested situation we will encounter. In all three uncertainty scenarios, we can still update without congestion by applying our algorithm $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$. It ensures congestion-free property during the whole reconfiguration by updating in a slower pace than zUpdate.

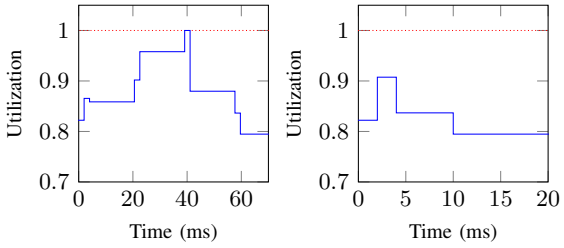
VI. Conclusion

We formulate a time-aware optimization model to find fast congestion-free routing reconfiguration plans. Our approach benefit from given timing information with any level of uncertainty. Several existing models become special cases of our formulation when we have perfect timing information or no timing information at all. This framework helps determine

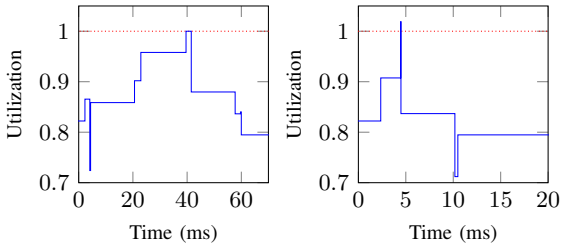
[2]The bound is actually tight for $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](10)$. The algorithm tends to choose smaller latency/capacity ratio instead of shorter latency, and hence we can construct an example showing that the bound is tight.



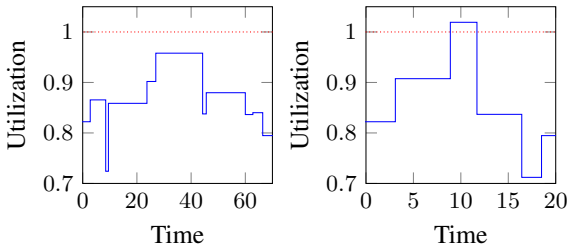
(a) The fat-tree topology



(b) Without uncertainty ($\delta_d = 0, \delta_v = 0$): Both $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ (left chart) and zUpdate (right chart) are congestion-free



(c) Low uncertainty ($\delta_d = 0.5, \delta_v = 0.5$): $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ (left chart) is congestion-free while zUpdate (left chart) congests



(d) High uncertainty ($\delta_d = 10, \delta_v = 0.5$): $\mathrm{ALG}[R_{\mathrm{tFCRs}\langle 3\rangle}](5)$ (left chart) remains congestion-free and zUpdate (right chart) endures a long congestion period

Fig. 6. The network topology and the timing charts of the busiest link utilization

less conservative update schedule. We further provide an efficient approximation algorithm to solve this new optimization problem, which is proven to be NP-hard, with performance guarantee. Extensive packet-level simulations confirm our predictions.

REFERENCES

[1] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 13–24, 2012.

[2] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 27–38, 2013.

[3] C. Clark *et al.*, "Live migration of virtual machines," in *Proc. USENIX NSDI*, 2005, pp. 273–286.

[4] A. Strunk, "Costs of virtual machine live migration: A survey," in *IEEE SERVICES*, 2012, pp. 323–329.

[5] A. Markopoulou *et al.*, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[6] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. Commun.*, vol. 25, no. 1, pp. 73–85, Jan 1977.

[7] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks*, vol. 3, no. 2, pp. 97–133, 1973.

[8] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Commun. Mag.*, vol. 40, no. 10, pp. 118–124, 2002.

[9] J. Moy, "RFC 2328: OSPF version 2," 1998.

[10] M. Meyer and J. Vasseur, "RFC 5712: MPLS traffic engineering soft preemption," 2010.

[11] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, vol. 2, 2000, pp. 519–528.

[12] A. Pathak *et al.*, "Latency inflation with MPLS-based traffic engineering," in *Proc. ACM IMC*, 2011, pp. 463–472.

[13] M. Reitblatt *et al.*, "Abstractions for network update," in *Proc. ACM SIGCOMM*, 2012, pp. 323–334.

[14] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. ACM SIGCOMM HotSDN Workshop*, 2013, pp. 49–54.

[15] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 15–26, 2013.

[16] L. Vanbever *et al.*, "Lossless migrations of link-state IGPs," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1842–1855, 2012.

[17] Summary of the Amazon EC2 and Amazon RDS service disruption in the US East region. [Online]. Available: http://aws.amazon.com/message/65648/

[18] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.

[19] N. L. Van Adrichem *et al.*, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *IEEE/IFIP NOMS*, 2014.

[20] M. Azizi, R. Benaini, and M. B. Mamoun, "Delay measurement in OpenFlow-enabled MPLS-TP network," *Modern Applied Science*, vol. 9, no. 3, pp. 90–101, 2015.

[21] C. Yu *et al.*, "Software-defined latency monitoring in data center networks," in *Passive and Active Measurement*. Springer, 2015, pp. 360–372.

[22] H. H. Liu *et al.*, "zUpdate: Updating data center networks with zero loss," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 411–422, 2013.

[23] X. Jin *et al.*, "Dynamic scheduling of network updates," in *Proc. ACM SIGCOMM*, 2014, pp. 539–550.

[24] C. L. Lim *et al.*, "Packet clustering introduced by routers: Modeling, analysis and experiments," in *Proc. IEEE CISS*, 2014.

[25] ns-3. [Online]. Available: https://www.nsnam.org/

[26] OpenFlow switch specification 0.8.9. [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v0.8.9.pdf

[27] CBC (COIN-OR branch and cut). [Online]. Available: https://projects.coin-or.org/Cbc

[28] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[29] A. Hassidim *et al.*, "Network utilization: The flow view," in *Proc. IEEE INFOCOM*, 2013, pp. 1429–1437.

[30] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 3–14, 2013.

[31] Google data center locations. [Online]. Available: http://www.google.com/about/datacenters/inside/locations/index.html

[32] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM CCR*, vol. 38, no. 4, pp. 63–74, 2008.