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...
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.

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.
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.

For each link $l \in L$, $c_l$ denotes the capacity on the link $l$. $f_l$ denotes the total flow on the link $l$. $P_l$ is the set of paths passing through the link $l$. $P^+_l$ is a P-partition ($P^+_l \subset P_l$). $P^\dagger_l$ is the collection of all valid P-partitions of $P^+_l$.

For each user $n \in N$, $d^n$ denotes the demanded flow rate into the network for the user $n$. $P^n$ is the set of acyclic paths for the user $n$.

For each path $p \in P$, $x_p$ denotes the flow on the path $p$. $z_p$ is the binary integer variable for the path $p$. $[w^\text{min}_p, w^\text{max}_p]$ is the time interval for the updated flow to clear the path or subpath $p$.

For each level $0 \leq \pi \leq \pi^\text{max}$ (see Section IV), $r_\pi$ denotes the level variable for the level $\pi$. $u^\text{max}_\pi$ is the required time for the level $\pi$.

Other variable $u$ is the required time to finish an update step.

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 $q_1 = q_{p_1}$ and $q_2 = q_{p_2}$ 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 $[w_{q_1}^\text{min}, w_{q_2}^\text{max}]$ for $i = 1, 2$. When the time intervals overlap, such as $w_{q_1}^\text{min} \leq w_{q_2}^\text{min} \leq w_{q_2}^\text{max}$, either flow can arrive before the other. That results in the order-oblivious situation, and all possible combinations should be covered by the following four constraints:

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

or we can represent them as

$$\max(x_{p_1}(a), x_{p_1}(a + 1)) + \max(x_{p_2}(a), x_{p_2}(a + 1)) \leq c_l.$$
where the constant \( \alpha_p \) for path \( p \) is set as the bottleneck link capacity \( \text{min}_{e \in g} q_e \) 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 \( \text{FCR}(b) \) is written as \( \text{OPT}_{\text{FCR}}(b) \) for given step upper bound \( b \). One important observation is that if \( \text{FCR} \) is feasible in \( b \) steps, it is also feasible for \( b + 1 \) steps and \( \text{OPT}_{\text{FCR}}(b + 1) \leq \text{OPT}_{\text{FCR}}(b) \). 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 \( \text{FCR} \). In this section, we present a proof sketch of the NP-hardness of \( \text{FCR} \), which motivates us to develop an approximation algorithm for \( \text{FCR} \). 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.** \( \text{FCR} \) is NP-hard.

Theorem 1 can be proved by showing that 3-SAT, a well-known NP-complete problem, polynomial reduces to \( \text{FCR} \). For each 3-SAT instance, there exist a network and its corresponding initial and target configurations such that the optimal \( \text{FCR} \) 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 \( \text{FCR} \) is NP-hard, we develop an approximation algorithm for \( \text{FCR} \) as the following. We first define the term kernel. A kernel \( A(b) \) is an algorithm which gives a feasible solution to \( \text{FCR}(b) \) when it is feasible; and \( \text{FCR} \) is not feasible in \( b \) steps if \( A(b) \) finds no solution to it. We write \( \text{SOL}_A(b) \) to denote the objective value of the feasible solution given by the kernel \( A(b) \), and we set \( \text{SOL}_A(b) = \infty \) if the kernel \( A(b) \) declares \( \text{FCR}(b) \) infeasible. We then propose the algorithm \( \text{ALG}[A](b) \) (Algorithm 1) with respect to a kernel \( A(\cdot) \), which essentially picks the best solution from the feasible solutions generated by \( A(1), A(2), \ldots, A(b) \).

**Algorithm 1** Algorithm \( \text{ALG}[A](b) \)

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

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

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

- \( 0 \leq u(a) \leq W_{\text{max}} \) for all \( 0 \leq a \leq b - 1 \).
- \( w^p_{\text{max}} \geq W_{\text{min}} \) for all \( p \in P \).

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

\[
\text{SOL}_{\text{ALG}[A]}(b) = \min_{1 \leq b \leq b} \text{SOL}_A(b) \leq \text{SOL}_A(b') = \sum_{a=0}^{b'-1} u_{a_{\text{OPT}}} \leq \sum_{a=0}^{b'-1} W_{\text{max}} u_{\text{OPT}}(a) = W_{\text{max}} \sum_{a=0}^{b'-1} u_{\text{OPT}}(a)
\]

This is generally more than one kernels satisfying the first condition in Theorem 2. For instance, we have the kernel \( R_{\text{FCR}}(b) \), which solves the linear relaxation of \( \text{FCR}(b) \), upronds all the result non-zero \( z_p(a) \) to 1 and sets \( u(a) = \max_{p \in P} z_p(a) = 1 \) \( w^p_{\text{max}} \leq \max_{p \in P} w^p_{\text{max}} \). In that case, \( W_{\text{max}} = \max_{p \in P} w^p_{\text{max}} \) and \( \text{ALG}[R_{\text{FCR}}](b) \) is a \( w^p_{\text{max}} \)-approximation algorithm.

Unlike the kernel-independent approximation ratio, the complexity of \( \text{ALG}[A](b) \) depends on the complexity of its kernel. If the kernel \( A(b) \) is a polynomial-time algorithm, \( \text{ALG}[A](b) \) also terminates in polynomial time. LP-based kernel, such as \( R_{\text{FCR}}(b) \), is indeed polynomial-time when the constraint set is also in polynomial size of its input variables. Among the constraints of \( \text{FCR} \), 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 \( P_T \) and their corresponding total flows \( f_1(P_T, a) \). We develop a polynomial time constraint set generation algorithm (Algorithm 2) and give a theorem (Theorem 3) to prove that the constrained 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 \( \Sigma_y \cup \Sigma_c \cup \Sigma_u \) and \( \Sigma_y \cup \Sigma_c \cup \Sigma_u \) is a valid P-partition, where \( \Sigma_y \), \( \Sigma_c \), and \( \Sigma_u \) are obtained at some iteration of Algorithm 2 and \( \Sigma_y \cup \Sigma_c \cup \Sigma_u \) is a partition of \( \Sigma_c \). 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.
of a kernel. $R_{\text{FCR}}(b)$ 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 $w_p^{ax}$. Hence, we can partition $P$ into several level sets and assign each path a level $\pi_p$ according to the level set it belongs to. Each level $\pi$ associates with a required time $w_{\pi}^{ax}$ such that $u(a) > w_{\pi-1}^{ax}$ while any flow along a level $\pi$ path changes between step $a$ and $a+1$. In other words, each path $p$ in level $\pi_p = \pi$ satisfies $w_{\pi}^{ax} < x_p^{ax} \leq w_{\pi}^{ax}$. We define the levels of the paths with the longest required time to be $\pi_{\text{max}}$ and level 0 has $w_0^{ax} = 0$. Without loss of generality, we can set $w_{\pi_{\text{max}}}^{ax} = \max_{p \in P} w_p^{ax}$ and by definition

$$0 = w_0^{ax} < w_1^{ax} < \cdots < w_{\pi_{\text{max}}}^{ax} = \max_{p \in P} w_p^{ax}.$$

We create the binary level variable $r_{\pi}(a)$ to indicate whether the level $\pi$ 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 $\pi$ 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 $z_p(a) = \sum_{i=\pi_p}^{\pi_{\text{max}}} r_i(a)$ and $u(a) = \sum_{i=0}^{\pi_{\text{max}}} w_i^{ax} r_i(a)$ for all $p \in P$ and $0 \leq a \leq b - 1$. 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 $\pi_{\text{max}}^{ax}$ levels in $b$ steps ($t_{\text{FCR}}(\pi_{\text{max}}^{ax})(b)$):

$$\begin{align*}
\min \quad & \sum_{a=0}^{b-1} u(a) \\
\text{s.t.} \quad & x \in \mathcal{X} \\
 & u(a) = \sum_{i=1}^{\pi_{\text{max}}} w_{\pi_i}^{ax} r_i(a) \quad \forall 0 \leq a \leq b - 1 \\
 & r_0(a) + \sum_{i=1}^{\pi_{\text{max}}} r_i(a) = 1 \quad \forall 0 \leq a \leq b - 1 \\
 & \left( \sum_{i=\pi}^{\pi_{\text{max}}} r_i(a) \right) \cdot \alpha_p \geq |x_p(a+1) - x_p(a)| \\
 & r_i(a) \in \{0, 1\} \quad \forall p \in P, 0 \leq a \leq b - 1 \\
 & 0 \leq a \leq b - 1
\end{align*}$$

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

Consider the subproblem $t_{\text{FCRs}}(\pi_{\text{max}}^{ax})(b)$ by removing all $r_0(a)$ from $t_{\text{FCR}}(\pi_{\text{max}}^{ax})(b)$. Then we propose the kernel $R_{t_{\text{FCRs}}(\pi_{\text{max}}^{ax})}$, which solves the linear relaxation of $t_{\text{FCRs}}(\pi_{\text{max}}^{ax})(b)$ (relax the constraint (11)), upgrades 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 $w_{\pi_i}^{ax}$ for each $a$. $R_{t_{\text{FCRs}}(\pi_{\text{max}}^{ax})}$ solves a feasible $r$ for every feasible $x$, and hence gives feasible $u$ and $z$ to FCR(b). Theorem 2 suggests that $\text{ALG}[R_{t_{\text{FCR}}(\pi_{\text{max}}^{ax})}]$ is also a $\min_{p \in P} \max_{a \in [0, b-1]} w_p^{ax}$-approximation algorithm. Nevertheless, we add only $ln_{\pi_{\text{max}}^{ax}}$ more variables in $R_{t_{\text{FCR}}(\pi_{\text{max}}^{ax})}$ instead of $b|P|$ in $R_{\text{FCR}}(b)$. In practice, kernel $R_{t_{\text{FCRs}}(\pi_{\text{max}}^{ax})}$ can even perform near-optimal (much better than $R_{\text{FCR}}(b)$ as shown in Section V-A and V-B.

We know that $\text{ALG}[R_{t_{\text{FCR}}(\pi_{\text{max}}^{ax})}]$ 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 $\max_{p \in P} w_p^{ax}$ between steps, which equals to $w_1^{ax}$. Thus $\text{ALG}[R_{t_{\text{FCR}}(\pi_{\text{max}}^{ax})}]$ performs at least as the same as SWAN. Since a multilevel solution can be truncated to a single level solution, we have $\text{SOL-ALG}[R_{t_{\text{FCR}}(\pi_{\text{max}}^{ax})}] \geq \text{SOL-ALG}[R_{t_{\text{FCR}}(\pi)}]$, if $\pi > 1$. Thus, by considering multilevel $\pi > 1$, $\text{ALG}[R_{t_{\text{FCR}}(\pi)}]$ 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:
• **OPT$_{FCR}$**: The optimal FCR solution obtained by solving the MILP problem.

• **ALG[\text{R}_i^{FCR}(3)]**: Algorithm 1 with the LP-based kernel solving the linear-relaxed transformed FCR subproblem involving 3 levels (set $n^{\text{max}} = 3$, delete $r_0(a)$ and relax the constraint (11)).

• **ALG[\text{R}_i^{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 $\lambda$ 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 $\lambda$ 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.

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

By Theorem 2, the theoretical approximation ratio is $\frac{11}{3}$. In this case, $\text{ALG}[\text{R}_i^{FCR}(3)](10)$ achieves the ratio 1 and $\text{ALG}[\text{R}_i^{FCR}](10)$ achieves $\frac{11}{3}$.

### 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 $\text{ALG}[\text{R}_i^{FCR}(3)](10)$, SWAN and OPT$_{FCR}(10)$. We know OPT$_{FCR}(10)$ gives the shortest update time in 10 steps. Thus for each test case, we normalize the results of $\text{ALG}[\text{R}_i^{FCR}(3)](10)$ and SWAN by OPT$_{FCR}(10)$ if they have solutions (Fig. 5). We sort the test cases by the normalized update time of $\text{ALG}[\text{R}_i^{FCR}(3)](10)$.
Again, we can observe that the normalized update time interferes with each other, resulting from the assumptions based least step congestion-free update sequence toward a congestion-free.

Theorem 2 promises this simple example shows how timing information enables us to expand the feasible solution set and update faster. In particular, 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 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 $[0, \delta_\tau]$ (ms) before it leaves the output interface. We apply both ALG$[R_{4FCRs3}](5)$ and zUpdate to solve for congestion-free update plans under $\delta_d = 0.5$ and $\delta_\tau = 10$. 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 ALG$[R_{4FCRs3}](5)$ 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 ALG$[R_{4FCRs3}](5)$, while the right ones belong to zUpdate. Both ALG$[R_{4FCRs3}](5)$ 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 ALG$[R_{4FCRs3}](5)$. 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 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.
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-level simulations confirm our predictions.

REFERENCES