Dynamic Control for Failure Recovery and Flow Reconfiguration in SDN

Stefano Paris, Georgios S. Paschos, and Jérémie Leguay
Mathematical and Algorithmic Sciences Lab
France Research Center - Huawei Technologies Co. Ltd.
Email: {name.surname}@huawei.com

Abstract—**By separating control and data planes, Software-Defined Networks (SDN) pave the way to a plethora of online routing mechanisms that react instantaneously to sudden traffic variation and network failures. In this paper, we introduce a dynamic routing system where these online mechanisms strive to solve an optimization instance that constantly evolves due to new arrivals and departures as well as network failures and repairs. We model the optimality gap, which is characteristic to the surcharge cost paid by network operators, of such a system as an auto-regressive stochastic process and we show that its evolving average does not diverge. We further propose a control policy that operates on top of the online routing solver to dynamically decide when to reconfigure the network. The policy minimizes the average surcharge cost and guarantees that the network reconfiguration rate does not exceed a threshold set by the operator. Numerical results show that our policy outperforms periodic schemes and approaches that reconfigure constantly the network after any system changes occur.**

Index Terms*—*Software Defined Networking, Online Reconfiguration, Routing Optimization, Lyapunov Stability.

I. Introduction

Software-Defined Networking (SDN) technologies have radically transformed the network architecture of data centers, network overlays, and carrier networks [1]. They provide programmable data planes that can be configured from a remote controller platform. This control and data planes separation creates an opportunity to implement more efficient routing processes than classical protocols, since the controller can take real-time decisions at a (logically) centralized place using an accurate view of the network.

Traditionally, routers collaboratively share topological information and establish end-to-end paths using traffic engineering protocols such as OSPF and MPLS with TE extensions [2]. The two different approaches used are: (i) hot standby mechanisms, which suffer from long recovering times, and (ii) IP-level mechanisms, which are reactive but do not offer fine-tuned control and may lead to suboptimal flow allocations [3]. Hence both approaches fall short of optimizing network utilization while meeting QoS requirements and guaranteeing resilience at the same time. Conversely, SDN architectures unleash the potential to reconfigure in real-time custom paths for each flow, so that network resources are optimized and performance requirements are met (in terms of QoS and resilience). This has already been showcased with the B4 controller [4] developed for inter-datacenter traffic, which achieves high network utilization.

By enabling a global and online routing optimization, SDN brings back to light online traffic engineering techniques, such those summarized in [2], with new algorithmic challenges when large networks have to be controlled. First, to set up appropriately the data plane, the SDN controller needs to solve an optimization problem of large size, often an extension of the Multi-Commodity Flow (MCF) problem with a great number of constraints and variables. To solve this problem, the SDN controller usually implements iterative methods. Then, contrary to existing routing solutions which allocate new flows on top of existing allocations, the SDN controller can reoptimize the flow allocation in real-time. However, as the network must be configured within tight time constraints and the instance to be optimized constantly evolves due to network changes (e.g. new arrivals and link failures), there might be a discrepancy between the solution computed by the SDN controller and the actual flow configuration in the network. Therefore, at each iteration of the computation, the SDN controller can reconfigure the network according to the flow allocation computed in the last iteration. However, network reconfiguration itself degrades the QoS of data transmission, since the system may need to tear down old routing paths and set up new end-to-end connections. On the other hand, leaving the network in the sub-optimal configuration may become more expensive, since new incoming demands and old ones that need to be rerouted after a failure, are quickly rearranged with sub-optimal routing decisions. Therefore, a key problem is to determine when to apply the computed network reconfigurations.

We investigate the evolution of such a dynamical routing system and model it with an auto-regressive stochastic process. Under this framework, we prove that the optimality gap for the routing cost does not diverge. Furthermore, we propose a control framework that operates on top of the iterative routing solver to limit the number of network reconfigurations and to reduce at the same time the flow allocation cost. The control policy applies the solution of the SDN solver to the switches only when the benefit is larger than a threshold, which is dynamically tuned over time. At any time instant such a dynamic threshold combines the gap between the current and optimal configurations and the number of reconfigurations applied so far. Therefore the control policy strives to balance the optimality gap and the reconfiguration rate.

Finally, we validate our system model and show the performance of our control framework on a realistic network scenario. Our results show that our control policy well approaches the performance that can be achieved by constantly and optimally re-configuring the network even in the presence of network failures. Additionally, our solution greatly decreases the number of system reconfigurations, thus increasing the network stability and reliability.

The paper is structured as follows. Sec. II discusses related work. Sec. III introduces the system model as well as the assumptions considered in our work, while Sec. IV formalizes the problem and provides closed-form formula to predict the system performance. Sec. V introduces the framework and the policy we propose to optimally control the system under reconfiguration rate constraints. Sec. VI illustrates numerical results in a realistic network scenario that validate our solution. Finally, concluding remarks are discussed in Sec. VII.

II. Related Work

The SDN controller can quickly optimize the routing configuration of the network, thus increasing the link utilization and the reactivity to failures. Google has already showcased in 2013 that they could use nearly 100% of the network capacity with their OpenFlow WAN controller [4]. Programmable data planes and centralized controllers for the network optimization represent the main factors for this evolution. Protocols like PCEP [5], Forces [6], and OpenFlow [7] have been recently proposed to make network devices programmable and controllable by a remote unit. At the same time, works like [1], [8] have proposed distributed architectures that keep a consistent view of the network in order to quickly compute new configurations when the network change.

The work [9] presents several techniques to solve routing problems in SDN infrastractures. To make such approaches scalable with the problem size, several approaches have been proposed using column generation [10] and rounding techniques. However, these approaches have been mainly designed for offline network planning. Therefore, they do not consider an evolving instance of the problem that captures more closely an online scenario where the network might be in a non-optimal configuration.

As described [11], the online version of the MCF problem, where new variables and constraints appear over the time without any prior knowledge, has been thoroughly analyzed both for throughput maximization and load balancing. The methodology has been extended to consider more general online packing problems [12] to derive algorithms with theoretical performance guarantees. Specifically, online algorithms that can compete with offline oracles that know in advance the sequence of future events have been proposed. However, they are mainly appropriate for admission control of new requests, since they cannot change previous decisions and reconfigure the flow allocation.

III. System Architecture

We consider an SDN controller with two main components: 1) one to quickly accept new demands and react to failures and 2) one to re-consider the flow allocation over time. We explain the two systems below.

A. Components Overview

• *Fast Connection Setup/Fast Recovery (FCS/FR)*. When connection requests arrive at ingress nodes or when a group of demands have to be rerouted after a failure, the SDN controller has to find a set of feasible paths satisfying multiple constraints (e.g., capacity, and QoS). Although this problem concerns one or few demands, it is still quite complicated to solve due to the numerous constraints relating to QoS and reliability. Additionally, the SDN controller needs to quickly compute a solution to route demands. Hence at this stage, the goal is not to optimize the network, but rather to find a quick approximate path allocation, which can even be precomputed.

• *Garbage Collection (GC) of network resources*. The sequence of sub-optimal network configurations obtained as a result of FCS/FR poses significant concerns on the evolution over time of the global objective function. Apart from arrivals or failures, the departures and repairs also affect the utility of an allocation. Suppose a demand is reallocated to an expensive backup path after a link failure on the primary path. After the link has been repaired, the demand can be routed on the cheaper primary path. Therefore, periodic or event-based reconfiguration of the overall network can improve the optimality gap. We call this mechanism *Garbage collection of network resources* since it mirrors the way a Java virtual machine collects garbages and reorganizes the memory.

Fig. 1 shows an example of an SDN controller which strives to minimize cost. In this example, for simplicity the FCS/FR uses a shortest path algorithm, while the GC uses a mincost MCF solver. The numbers on links indicate the link costs, while all link capacities are 1 Mbit/s. Both demands ($A-B$ and $A-D$) require 1 Mbit/s. After the failure of link $(B;D)$, FCS/FR reallocates demands $A-B$ and $A-D$ on paths $\{A,B\}$ and $\{A,C,E,F,D\}$, respectively (step 1). When the link $(B;D)$ has been repaired (step 2), the network operates in a sub-optimal state, since a cheaper flow allocation exists. Finally, the controller runs GC and decides to change the flow allocation (step 3), saving in this way 20% of the cost.

B. Solving Large Routing Problems

In this section we focus on the GC and explain how this mechanism can be implemented. To re-optimize the flow allocation, the controller has to solve an extension of the *mincost MCF* problem, which involves millions of variables and constraints on large scale networks. Due to the immense size of the problem instance (network graph and set of demands), we consider in the rest of this paper a Linear Programming (LP) approach, which iteratively improves the solution at each step until it converges to the optimal solution. The advantage of the LP approach is that by considering the path formulation of the problem[1]and using the *column generation* extension of

[1] In the path formulation, each optimization variable is the flow volume assigned to a path for a specific commodity.

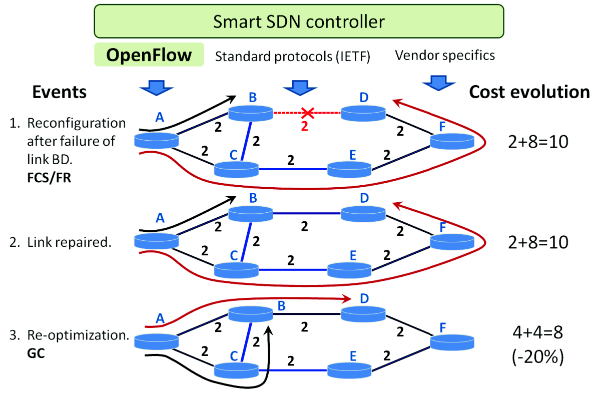


Fig. 1: Example of an online SDN routing optimization with two demands. Edges labels represent costs of links. All links have capacity of 1 Mb/s. Both demands (red and black arrows) require 1 Mb/s. The cost evolution is computed according to the allocation performed by FCS/FR and GC.

the simplex method [13], only a small number of variables and constraints are considered at each iteration.

Due to the arrivals and departures of network demands as well as network failures and repairs, the instance of the optimization problem under consideration changes over time. Since the convergence time of the approach is considerable, and comparable to the rate of network changes (traffic and failures), a number of research questions are raised with regards to the performance of this dynamical system. For example, it may be possible that the solver is unable to track adequately the rapid system changes, and the cost of the solution over time deviates significantly from the optimal one. In the rest of the paper we study the evolution of the objective function under repeated approximated solutions, investigating its divergence from the optimum cost.

IV. Problem Formulation

This section presents the model for the general class of online optimization routing algorithms we consider.

A. *Optimization Instance*

We begin by formulating an optimization problem motivated from datacenter interconnection operators. Datacenter operators route the interconnection traffic through leased infrastructure which may vary in cost per link. Thus, they are interested in optimizing the network traffic in order to minimize the total used cost.

We model the network infrastructure with an undirected graph $G = (\mathcal{N}, \mathcal{E})$, where the vertex set $\mathcal{N}$ represents the set of network devices and the edge set $\mathcal{L}$ models the set of links $e = \{i, j\}, i, j \in \mathcal{N}$ connecting network devices. Each link $e \in \mathcal{L}$ has a limited capacity b_e and a cost c_e, which refer the maximum amount of traffic that can be transmitted and the price (either a monetary price or operational cost) paid for using that link, respectively.

A unicast demand $k \in \mathcal{K}$ is identified by a source-destination pair $s_k, t_k \in \mathcal{N}$, and the amount of traffic d_k that has to be transmitted from s_k to t_k. The set $\mathcal{K}$ represents the active demands on this problem instance that need to be routed through the network. To this end, the controller solves an instance of the min-cost MCF problem in order to find the network flow that satisfies the set of active demands at the minimum cost. More specifically, the min-cost MCF problem can be formulated as the linear program (1)-(3), where real variables $(x_p)_{p\in\mathcal{P}}$ and $(y_e)_{e\in\mathcal{E}}$ represent the path and link utilization and take values in $[0, 1]$. In this model, P is the set of all network paths, while $\mathcal{P}_k \subseteq P$ represents the set of all paths which can be used to transmit the traffic of the demand $k \in \mathcal{K}$ from the source s_k to the destination t_k. In contrast, the set $\mathcal{P}_e \subseteq P$ contains all paths that use edge $e \in \mathcal{E}$.

$$C^{OPT} = \min_{(x_p),(y_e)} \sum_{e\in\mathcal{L}} c_e y_e \tag{1}$$

subject to

$$\sum_{p\in\mathcal{P}_k} x_p = 1 \qquad \forall k \in \mathcal{K} \tag{2}$$

$$\sum_{p\in\mathcal{P}_e : k \in P_k} d_k x_p \le b_e y_e \qquad \forall e \in \mathcal{L}. \tag{3}$$

The objective function (1) models the overall price paid by the operator for using the network links, the constraints (2) ensure that each demand $k \in \mathcal{K}$ is routed from its source to its destination over at least one path and the constraints (3) are the link capacity constraints.

B. *Online Version With Diminishing Returns*

In order to scale with the network size, we use the column generation technique to solve the LP problem (1)-(3). This technique starts from a feasible point and at each iteration t adds new variables (i.e., new paths) that can improve the objective function $C(t)$ until it converges to the optimum C^{OPT}. Let us denote the optimality gap with $Q(t) \triangleq C(t) - C^{OPT}(t) \ge 0$. The column generation algorithm shows diminishing returns in the reduction of the optimality gap, with a very steep improvement in the very first iterations. Supported by our validation, we model the improvement of the optimality gap at every iteration t with an exponential function of the type $(\frac{1}{1-\eta})^{-t}, t = 1, 2, \ldots$. This corresponds to the following recursive equation:

$$Q(t+1) = (1-\eta)Q(t), \tag{4}$$

where $\eta \in (0, 1)$ is a constant that relates the volume of the next improvement to current optimality gap values. Essentially, at each iteration of the solver the optimality gap is reduced by a fraction η. To validate our model, we evaluate the evolution over time of the optimality gap for our iterative algorithm in different traffic conditions using the GEANT network topology. Fig. 2 shows such evolution as a function of the number of iterations and two exponential functions that fit the 50 and 99 percentiles obtained over 500 simulations. Specifically, red and black solid lines represents 50 and 99 percentiles, respectively. It can be observed that the optimality gap $Q(t)$ obtained in numerical simulations is always lower and upper bounded by two exponential functions, namely 9^{-t} and 2^{-t}, which are depicted as green and blue dashed lines in the figure. Based on this observation, we approximate the convergence rate of our column generation solver to be exponential. Other solvers using full gradient and stochastic gradient methods have been shown to have an exponential convergence rate for strongly convex objective functions [14]. Nonetheless, we leave as future work the analysis of other solution techniques, with sublinear and linear convergence rates [15], [16].

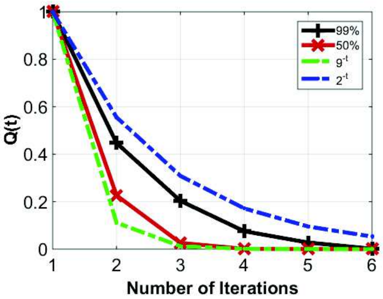


Fig. 2: Evolution of the optimality gap Q(t) in the GEANT scenario. The 50 and 99 percentiles are computed considering 500 simulations.

C. Modeling Network Events

In the above section we defined the MCF instance for a given set of demands $\mathcal{K}$ and a given network topology $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Next, we allow this instance to vary over time. In particular, at each iteration t we may have a potentially different set of demands $\mathcal{K}(t)$ and network topology $\mathcal{G}(t)$, due to demand arrivals/departures or network failures/repairs. In case of a new demand arriving, a "fast connection setup" algorithm is utilized to find a feasible path. We assume that such a path can always be found. This assumption is not restrictive in practice because (1) systems are often over-provisioned, (2) in the case of a network overload due to too many arrivals, a congestion controller may reject some demands to make the system feasible. In case a link failure occur, a "fast recovery mechanism" restores the dropped connections on alternative paths (e.g., in case of Dedicated Backup Path Protection schemes) or might drop demands if there is not enough capacity (e.g., in case of Shared Backup Path Protection schemes). In both cases, we model the link failure as an extra cost in the objective function, since the change that has occurred leads to a new min-cost instance for which the current flow allocation might not be optimal.

New events correspond to arrivals or departures of demands as well as failures or repairs of links. They all result in a "jump" in the optimality gap $Q(t)$. The reason an arrival "hurts" the optimality gap is related to the fact that the new arrival is badly configured by the "fast connection setup" algorithm, and hence it is not globally optimized with respect to existing demands. The departure of a demand and a network failure also hurts. In practice, the impact of a departure depends on the actual cost of the path the departed demand was using. A low cost path will result in a significant jump in the optimality gap, since this path is not used any more, and thus it is possible that with appropriate swapping we could redirect a demand using a high cost path to this cheap path. In contrast, a node or link failure reduces the network capacity and causes the drop of demands routed through the failed link/node, triggering a "fast recovery" mechanism that brings the system in a suboptimal state. Similarly, when a link or node is repaired, the demands can be rearranged to reduce the overall network cost. To simplify the considerations though, we begin by assuming that all events incur the same cost, which is denoted by e. More elaborate models are left for future work. We assume that the arrivals, departures, network failures, and repairs occur according to an i.i.d. stochastic process $A(t)$ with mean $E[A(t)] = E[A] = \lambda$, and finite variance $\text{Var}[A(t)] = \sigma_A^2$.

The optimality gap evolution can now be rewritten to include the addition of the cost due to the evolving system:

$$Q(t+1) = (1-\eta)Q(t) + eA(t). \tag{5}$$

This is a first order auto-regressive stochastic process with discrete non-Gaussian disturbance.

D. Performance Analysis

Considering the previous model, we study the performance of the routing system. Using the recursive equation above, we may determine the optimality gap at time t

$$Q(t) = (1-\eta)^t Q(0) + e\sum_{i=1}^{t}(1-\eta)^{i-1}A(t-i).$$

Taking expectation at time slot t we have

$$\begin{aligned} E[Q(t)] &= (1-\eta)^t Q(0) + e\lambda\sum_{i=1}^{t}(1-\eta)^{i-1} \\ &= (1-\eta)^t\left(Q(0) - \frac{e\lambda}{\eta}\right) + \frac{e\lambda}{\eta}. \end{aligned} \tag{6}$$

A first question we may pose is that of stability. Will the system diverge to infinite cost, or the cost is guaranteed to remain bounded? We use the formal definition of "strong stability" from Queueing Theory, whereby a stochastic process $Q(t)$ is stable if

$$\limsup_{T\to\infty}\frac{\sum_{t=0}^{T-1}E[Q(t)]}{T} < \infty.$$

Strong stability implies other weaker forms of stability like rate stability and mean rate stability [17]. By summing up, dividing by T, and applying limsup on (6), we get

$$\limsup_{T\to\infty}\frac{\sum_{t=0}^{T-1}E[Q(t)]}{T} = \frac{e\lambda}{\eta}.$$

This satisfies the stability criterion as long as $\eta > 0$. Conclusively, the average optimality gap remains finite irrespective of how high the arrival rate of events is, or how slow the exponential slope of our solver is.

Since stability is guaranteed, we turn our attention to the evaluation of the time average optimality gap incurred by the dynamics in our system, as described by (6). If we define

$\overline{Q} \triangleq \liminf_{t\to\infty} E[Q(t)]$ as the limiting behavior of $E[Q(t)]$, it follows that the limit always exists, and specifically

$$\overline{Q} = \frac{e\lambda}{\eta}. \tag{7}$$

We see that the dependence of average optimality gap on the initial condition $Q(0)$ washes away with time. The average optimality gap then can be improved either by 1) reducing the cost of event arrival e or 2) increasing η, which corresponds to a faster solver.

In a similar fashion we may also derive the variance $\mathrm{Var}[Q(t)] \triangleq E[Q^2(t)] - E[Q(t)]^2$. We give here directly the limit of the variance σ_Q^2:

$$\sigma_Q^2 = \lim_{t\to\infty} \mathrm{Var}[Q(t)] = \frac{e^2\sigma_A^2}{\eta(2-\eta)}, \tag{8}$$

where σ_A^2 is the variance of the arrival process.

V. Minimizing Network Changes

In the previous section, we assumed that at each iteration t the best solution found so far is applied to the network. By the modeling assumption, the new solution always reduces the operational cost. Since the new solution is different from the previous, it is implied that we need to reconfigure some of the flows. Reconfiguring a flow takes time and causes small disturbances in the performance, hence we want to avoid it if possible. For example, at a given iteration, we may decide not to apply the obtained solution from the solver and keep the flows unaffected. In this case, however the cost diverges from the optimum over time, thus increasing the surcharge cost paid by the operator. In this section we will present a control framework design around the SDN controller to decide dynamically when to reconfigure the network. Furthermore, we will introduce a control policy that provides good performance with a limited reconfiguration budget.

A. Control Framework for SDN Controllers

The goal of minimizing the number of flow reconfigurations clearly conflicts with the goal of minimizing the average optimality gap $\overline{Q}$. It is possible to apply the best found solution at a frequency less than one, which will incur less reconfigurations but it will increase $\overline{Q}$. Hereafter, we investigate this important tradeoff.

Instead of the number of reconfiguration points, we care more about the number of reconfigurations. To get to the gist of this problem, we need to accurately model the law behind the number of reconfigurations needed in order to obtain a certain gain from the network. In particular, we observe a diminishing returns law, namely the closer we are to the optimal, the more reconfigurations we need to impose to get an improvement of certain value. This results in the following: to improve the optimality gap by a certain fraction, the number of configurations needed are roughly the same, say h, irrespective of the value of the optimality gap Q(t). In our model, we therefore consider that the number of reconfigurations is roughly the same at each reconfiguration point, and we denote this with h.

At each iteration, our controller can decide whether to use a new solution, incurring h reconfigurations and yielding a multiplicative improvement on the cost $(1-\eta)$, or instead do not apply the solution leaving the cost unaffected but resulting into zero reconfigurations. Formally, let $I^\pi(t) = 1$ represent the decision of applying the current best solution to the system by means of h reconfigurations, and $I^\pi(t) = 0$ the decision not to apply it, where π denotes the algorithm we are using to decide $I^\pi(t)$. Figure 3 depicts such a dynamic controller, which decides at each iteration whether to apply the solution of the routing solver by metaphorically turning the switch controlled by $I^\pi(t)$. Clearly, the control decision $I^\pi(t)$ affects the evolution of the operational cost and the frequency of reconfigurations.

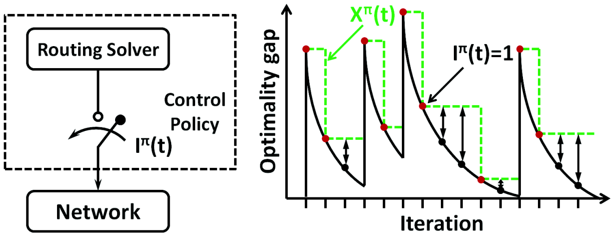


Fig. 3: Dynamic control of the routing solver. By selecting at each time slot the value of $I^\pi(t)$, the control policy decides whether to use the solution computed so far by the routing solver.

At this point we must note that the actual *computed* solution by the solver has the same error evolution $Q(t)$ as before, which is an uncontrollable process with time-average mean $\overline{Q}$, which is also the minimum possible average cost. On the other hand, the actual operational cost may be higher when we do not apply the best solution at each step. Let $X(t)$ be the operational cost, while $Q(t)$ still represents the theoretical cost that corresponds to the best current solution, which is not necessarily applied to the system.

As before, we have

$$Q(t+1) = (1-\eta)Q(t) + eA(t).$$

The evolution of the operational cost $X^\pi(t)$ is then

$$X^\pi(t+1) = (1-\eta)Q(t)I^\pi(t) + X^\pi(t)(1-I^\pi(t)) + eA(t), \tag{9}$$

We define the control policy to be a mapping from the state of the system $(X^\pi(t), Q(t))$ to the action set $\{0, 1\}$.

Based on the simple model explained above, the total number of reconfigurations under policy π by iteration t is simply

$$N_R^\pi(t) = h\sum_{\tau=1}^{t} I^\pi(\tau),$$

while the frequency of reconfigurations under policy π by iteration t is

$$R^{\pi}(t) = \frac{N_R^{\pi}(t)}{h} = \sum_{\tau=1}^{t} I^{\pi}(\tau).$$

We define the frequency of reconfigurations as

$$\overline{R}^{\pi} \triangleq \limsup_{t\to\infty} \frac{E[R^{\pi}(t)]}{t} = \limsup_{t\to\infty} \frac{E[\sum_{\tau=1}^{t} I^{\pi}(\tau)]}{t}$$

We are therefore interested in the following optimization

$$\min_{\pi} \limsup_{t\to\infty} \frac{\sum_{\tau=1}^{t} E[X^{\pi}(\tau)]}{t} \tag{10}$$

$$\text{s.t.} \quad \overline{R}^{\pi} \leq R_{\max} \tag{11}$$

where we pick a control policy with decisions $\{I^{\pi}(t)\}, t = 1, 2, \ldots$ in online fashion to minimize the average operational error $\limsup_{t\to\infty} \frac{\sum_{\tau=1}^{t} E[X^{\pi}(\tau)]}{t}$ while keeping the reconfiguration frequency less than $R_{\max}$.

B. Drift-plus-penalty Policy (DP)

We begin by reformulating the above problem. Instead of minimizing the time average of $X^{\pi}(t)$, we will attempt to minimize the time average of $X^{\pi}(t) - Q(t)$. Since $Q(t)$ is uncontrollable, the two are equivalent.

A fundamental tool of our proposition is a virtual queue, which is essentially a counter (which takes real values). Our goal is to use the stability of the virtual queue in order to ensure the satisfaction of constraint (11). We feed this queue with $I^{\pi}(t)$: in an iteration where the solution is applied to the system, we add one to the counter, while in an iteration where the solution is not applied, we do not. We also serve the queue with $R_{\max}$. Hence the queue evolves according to the following recursion:

$$V(t+1) = (V(t) - R_{\max})^{+} + I^{\pi}(t). \tag{12}$$

Suppose that we choose a control policy which stabilizes queue $V(t)$. Then it follows that the time average of arrivals is less than or equal to the time average of departures, i.e.,

$$\limsup_{t\to\infty} \frac{\sum_{\tau=1}^{t} E[I^{\pi}(t)]}{t} \leq R_{\max}, \tag{13}$$

which satisfies the constraint (11).

Let us first define the Lyapunov drift. Consider the quadratic function $L(x) = x^2$. We define the Lyapunov drift for the Markov chain $(V(t))$ as

$$\Delta(t) = E[V^2(t+1) - V^2(t) | X^{\pi}(t), Q(t), V(t)] \tag{14}$$

$$\leq (I^{\pi}(t))^2 + R_{\max}^2 - 2V(t)(R_{\max} - I^{\pi}(t)) \tag{15}$$

Next, consider the instantaneous penalty $\delta(t) = E[X^{\pi}(t+1) - Q(t+1) | X^{\pi}(t), Q(t), V(t)]$. Clearly the penalty depends on $I^{\pi}(t)$, see (9). For example, we may minimize it by choosing $I^{\pi}(t) = 1$. On the other hand, this adds a counter in the virtual queue. Essentially, a good control policy should try to achieve a good balance between the two.

For this purpose we define the instantaneous metric $\Delta(t) + K\delta(t)$, called *drift plus penalty*. At each slot we would like to minimize this metric. To achieve this our policy needs to strike a good balance between stabilizing the queue (hence choosing $I^{\pi}(t) = 0$ multiple times) and keeping the penalty small by choosing $I^{\pi}(t) = 1$ when appropriate.

We have

$$\begin{aligned}\Delta(t) + K\delta(t) \leq\ & (I^{\pi}(t))^2 + R_{\max}^2 - 2V(t)(R_{\max} - I^{\pi}(t)) + \\ & + K(1 - I^{\pi}(t))(X^{\pi}(t) - Q(t+1)) \\ & = I^{\pi}(t)\left[1 + 2V(t) - K(X^{\pi}(t) - Q(t+1))\right] + \\ & + R_{\max}^2 - 2V(t)R_{\max} + K(X^{\pi}(t) - Q(t+1)).\end{aligned}$$

To minimize the right hand side it suffices to solve the following optimization

$$\min_{x\in\{0,1\}} x\left[1 + 2V(t) - K(X^{\pi}(t) - Q(t+1))\right]. \tag{16}$$

To minimize the *drift plus penalty*, our policy uses a dynamic threshold to decide whether to reconfigure the network (i.e., to set $I^{\pi}(t) = 1$):

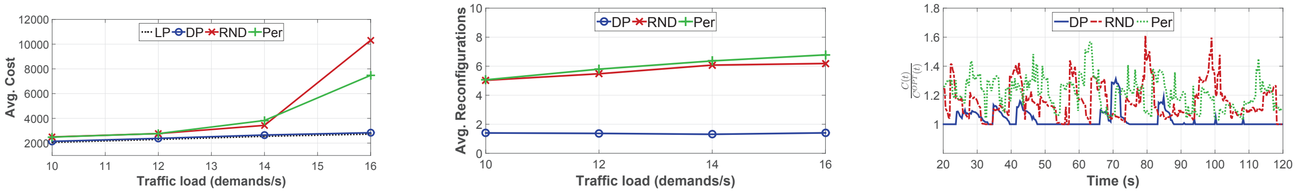
$$I(t) = \begin{cases} 1 & \text{if } V(t) < \frac{K(X^{\pi}(t) - Q(t+1)) - 1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

A first observation is that to implement this policy we only need to keep track of the virtual queue $V(t)$, the gap of the solution $Q(t) - Q(t+1)$ and the operational cost at the previous iteration $X^{\pi}(t)$; K is a constant we choose high enough to approximate well the optimal solution. Typical values for K are 1000. For static parameters (i.e., constant arrival rates of demands and failures) we may choose K arbitrarily large, and initialize the virtual queue with $V(0) = 2K$.

This analysis shows that a simple threshold mechanism minimizes the right hand side bound of the instantaneous drift-plus penalty. Prior work [17] shows that in specific scenarios it is possible to use such policies to make the drift negative in states where queues are large, and use it to solve stochastic convex optimizations like (10)-(11). In fact, this is not directly possible for our problem due to the memory our system has from (9). As a result, the drift-plus-penalty algorithm we derive in this section is not provably optimal, however as we show next by simulations, the performance evaluation of this algorithm is significantly good.

VI. Experimental Results

To evaluate the dynamics of a real online SDN routing optimization system, we used a scalable routing solver based on column generation to iteratively solve the linear program (Eq. (1)-(3)). The algorithm roughly works as follows: it maintains a feasible solution, allocating new demands or repairing damaged demands one by one using shortest paths routing on the residual graph. After this initialization phase, it iterates by adding and removing paths (i.e., x_p variables) to a restricted version of the problem until it converges to

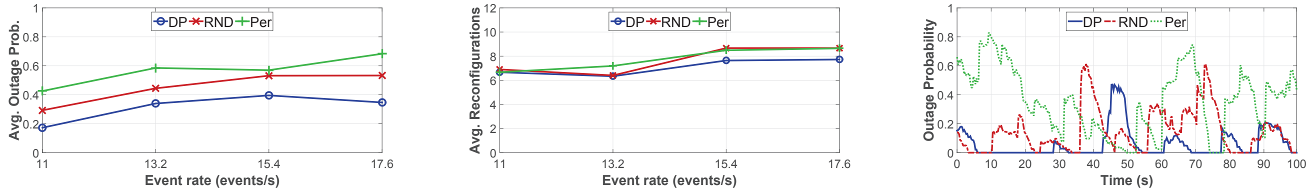


(a) Average cost

(b) Average routing changes

(c) Relative cost evolution, $\frac{C(t)}{C^{OPT}(t)}$

Fig. 4: Performance evaluation of the routing control policies over GEANT without system failures.



(a) Average outage probability

(b) Average routing changes

(c) Outage probability evolution

Fig. 5: Performance evaluation of the routing control policies over GEANT with system failures.

the optimal point. The algorithm uses the dual formulation of Eq. (1)-(3) to add only variables that can reduce the cost and to test the optimality of the current solution. Even if the algorithm can converge to the optimal point, it may stop before computing the optimum due to the limited amount of computational time that the SDN operator might have set.

On top of this iterative routing solver, we implemented the control framework and the *DP* control policy proposed in Section V (labeled '*DP*' in the figures). As the goal is to stay below a target reconfiguration rate while minimizing the routing cost, we compared the performance of our policy against random (labeled '*RND*') and periodic (labeled '*Per*') control policies that select reconfiguration points randomly with probability $\frac{1}{R_{max}}$ and periodically with rate $\frac{1}{R_{max}}$.

To evaluate our control policies in a realistic senario, we used GEANT [18], the high bandwidth pan-European research and education backbone, which is composed of 22 nodes and 36 high capacity 40G links. We generated random traffic demands and link failures as Poisson arrivals increasing the traffic load $\lambda_1 \in \{10, 12, 14, 16\}$ demands/s and the link failure rate $\lambda_2 \in \{1, 1.2, 1.4, 1.6\}$ failures/s. Therefore, in case of failures the network event rate is $\lambda \in \{11, 13.2, 15.4, 17.6\}$ events/s. All demands and failures have duration of 10s and 5s, respectively. We simulated $T = 100$s of traffic and set the limit of the computational time for one iteration of the CG algorithm to 0.3s. Demands arriving between two executions are buffered and served at the next time slot. For each of the two scenarios, we perform 10 independent measurements.

A. Performance without Failures

In this set of simulations, we varied only the traffic load and we compare the control policies against the optimal solution computed by the LP solver (curve labeled '*LP*'). Note that this value is a lower bound that can be obtained only if we have enough time to converge and we can reconfigure the network for each occurred event. To compare the operational routing cost obtained by the control policies with the optimal solution, we have considered the fractional version of the problem for computational reason. Without loss of generality, we omitted the rounding step required to obtain an integer solution from the routing solver. Fig. 4(a) illustrates the average cost of the flow allocation (i.e., $\frac{1}{T}\sum_{t=1}^{T} C(t)$). We can observe that the drift-plus-penalty policy (*DP*) well approaches the solution obtained by always reconfiguring the network (*LP*). In contrast, both the periodic (*Per*) and random (*RND*) policies have worse performance (from 1.5 to 4 times higher average cost), since they select wrong reconfiguration points with a small benefit for the system. This is even more evident from Figure 4(b), which shows the average number of reconfigured links per demand (*LP* is not depicted, since it reconfigures the network all the time). *Per* and *RND* impose more changes to the network and incur larger cost than *DP*. Therefore, periodically or randomly reconfiguring the network results in bad performance, since the controller cannot achieve the best trade off between optimality gap and number of reconfigurations.

Finally, Figure 4(c) illustrates the evolution of the relative cost over time (i.e., $\frac{C(t)}{C^{OPT}(t)}$) for one experiment with $\lambda_1 = 10$ demands/s. It can be observed that most of the time *DP* tracks the optimal cost (i.e., $\frac{C(t)}{C^{OPT}(t)} = 1$), which can only achieved by continuously reconfiguring the network. The gap from the optimal solution increases when the number of active demands in the system grows (i.e., when we have a burst). This is due to the size increase of the problem instance and the fact that the iterative algorithm does not have enough time to converge to the optimum.

B. Performance with Failures

In this scenario, we varied both the traffic load ($\lambda_1 \in \{10, 12, 14, 16\}$ demands/s) and the link failure rate ($\lambda_2 \in \{1, 1.2, 1.4, 1.6\}$ failures/s). In our simulation, we suspended any active demand whose bandwidth cannot be completely satisfied after an iteration of the CG algorithm due to a link failure (i.e., any demand that can be only partially rerouted after a link failure is blocked and reactivated only if it can be later satisfied). We compare the performance of the three control policies in terms of average outage probability of data connections[2] and average number of reconfigured links. Note that we omit the outage probability for the LP solver since it is always zero. Indeed, we assumed that the LP controller can instantaneously compute the optimum for the new problem.

Fig. 5(a) shows the average outage probability of data connections as a function of the rate of network events (demand arrivals plus link failures) occurring in the system. We can observe that *DP* outperforms simple control policies like *RND* and *Per*, thus reacting more efficiently to link failures in addition to demand arrivals/departures. Similarly to demands arrivals/departures, the failure causes a jump in the optimality gap, thus enabling the drift-plus-penalty control policy to apply a reconfiguration. In contrast, periodic and random policies need more time to recover from link failures, since they select reconfiguration points that do not necessarily provide a huge improvement in the optimality gap. Fig. 5(b) illustrates the average number of reconfigured links per demand. Differently from the scenario described above, link failures and repairs increase the overall number of reconfigurations since the solver is triggered by more events than before. Furthermore, a link failure requires to use longer paths, thus increasing the number of links that must be changed between consecutive reconfigurations.

Fig. 5(c) illustrates the evolution of the instantaneous outage probability for the three different control schemes when the traffic load is equal to $\lambda_1 = 10$ demands/s and the link failure rate is $\lambda_2 = 1$ failures/s. We can observe from the figure that *DP* reacts more quickly to link failures than *RND* and *Per*. The larger instantaneous outage probability of *DP* in some interval is simply due to the higher available capacity for *RND*. Indeed, if several demands are instantly suspended without their quick recovery, we might have in the near future higher residual capacity for future demands. However, in the long run such a myopic behavior leads to lower performance as illustrated in Fig. 5(a).

Finally, we underline that our control framework is orthogonal to typical backup approaches, which require the booking of additional resources (e.g., link disjoint backup paths) to handle failures in real time. In particular, our framework can effectively work with backup approaches in order to reduce the amount of resources dedicated to fast recovery. Indeed, our simulation confirms that our control policy can efficiently handle link failures with a limited reconfiguration budget.

VII. Conclusion and perspectives

Software-Defined Networking is foreseen as a means of using more efficiently network resources and dynamically adapting the routing configuration over time. In this context, we study the evolution over time of the optimality gap of a general class of iterative routing solvers, which compute a sequence of feasible solutions up to the optimum with diminishing returns. We show that the optimality gap does not diverge and we propose a control framework working on top of any routing solver to bound the network reconfiguration rate. Within this framework, we propose a control policy that achieve the best trade-off between reconfiguration rate and optimality gap. Simulations in realistic network conditions show that SDN controllers can effectively track the evolution of the system in terms of traffic evolution and network failures, thus opening a new era for the dynamic control of online routing solvers.

References

[1] Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617 – 1634, 2014.

[2] Ning Wang, Kin Ho, G. Pavlou, and M. Howarth. An overview of routing optimization for internet traffic engineering. *Communications Surveys Tutorials, IEEE*, 2008.

[3] C. Metz. Ip protection and restoration. *IEEE Internet Computing*, 4(2):97–102, 2000.

[4] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM*, 2013.

[5] A. Doria et al. Path Computation Element (PCE) Communication Protocol (PCEP), RFC 5440, IETF (March 2009).

[6] A. Doria et al. Forwarding and Control Element Separation (ForCES) Protocol Specification, RFC 5810, IETF (March 2010).

[7] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

[8] Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. A distributed and robust sdn control plane for transactional network updates. In *IEEE INFOCOM*, 2015.

[9] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic engineering in software defined networks. In *IEEE INFOCOM*, pages 2211–2219, 2013.

[10] K. Murakami and H. S. Kim. Optimal capacity and flow assignment for self-healing atm networks based on line and end-to-end restoration. *IEEE/ACM Transactions on Networking*, Apr 1998.

[11] Guy Even and Moti Medina. *Online multi-commodity flow with high demands.*, pages 16–29. Berlin: Springer, 2013.

[12] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *13th European Symposium on Algorithms*, 2005.

[13] T. Leventhal, G. Nemhauser, and L. Trotter. A column generation algorithm for optimal traffic assignment. *Transportation Science Journal*, 1973.

[14] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[15] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[16] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[17] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

[18] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Rev*, 36(1), 2006.

[2] Outage probability in a given iteration t is defined as the ratio of suspended connections w.r.t. the total active connections.