SPONGE: Software-Defined Traffic Engineering to Absorb Influx of Network Traffic

Benoît Henry, Shihabur Rahman Chowdhury, Abdelkader Lahmadi, Romain Azaïs, Jérôme François, Raouf Boutaba

SPONGE: Software-Defined Traffic Engineering to Absorb Influx of Network Traffic

Benoit Henry*, Shihabur Rahman Chowdhury†, Abdelkader Lahmadi‡, Romain Azais§,
Jérôme François§, and Raouf Boutaba†
*IMT Lille Douai benoit.henry@imt-lille-douai.fr
† University of Waterloo, Canada {sr2chowdhury|rboutaba}@uwaterloo.ca
‡ University of Lorraine, France abdelkader.lahmadi@loria.fr
§INRIA - Nancy Grand Est, France {jerome.francois|romain.azais}@inria.fr

***Abstract*—Existing shortest path-based routing in wide area networks or equal cost multi-path routing in data center networks do not consider the load on the links while taking routing decisions. As a consequence, an influx of network traffic stemming from events such as distributed link flooding attacks and data shuffle during large scale analytics can congest network links despite the network having sufficient capacity on alternate paths to absorb the traffic. This can have several negative consequences, service unavailability, delayed flow completion, packet losses, among others. In this regard, we propose SPONGE, a traffic engineering mechanism for handling sudden influx of network traffic. SPONGE models the network as a stochastic process, takes the switch queue occupancy and traffic rate as inputs, and leverages the multiple available paths in the network to route traffic in a way that minimizes the overall packet loss in the network. We demonstrate the practicality of SPONGE through an OpenFlow based implementation, where we periodically and pro-actively re-route network traffic to the routes computed by SPONGE. Mininet emulations using real network topologies show that SPONGE is capable of reducing packet drops by 20% on average even when the network is highly loaded because of an ongoing link flooding attack.**

I. Introduction

Traffic engineering is the process of assigning network flows to network paths while optimizing objectives such as minimizing the maximum link utilization and minimizing flow completion times [1]. It is essential for network operations in order to handle the dynamic nature of network traffic. Particularly, traffic engineering solutions should be capable of handling sudden influx of network traffic triggered by events such as a distributed link flooding attack, initiation of data shuffle phase for large-scale analytics, inter/intra-data center virtual machine migration or database replication, *etc*.. Contemporary traffic engineering solutions in wide-area networks (WANs) such as MPLS-TE or in data center (DC) networks such as Equal Cost Multipath do not consider the dynamically changing load on network links while assigning flows to the paths. Therefore, an influx of network traffic can congest links despite sufficient capacity to absorb the influx being present in the network.

Traffic engineering solutions focused on mitigating congestion has been extensively studied in the research literature. The general theme of different proposals in this area is to exploit the excess capacity present on alternate paths between endpoints to absorb any influx of network traffic instead of solely relying on shortest path based forwarding. However, most of the traffic engineering proposals are either tailored to work best for a particular network topology (*e.g.*, tree-like DC networks [2] or WANs [3]), or focuses on a particular event that causes influx in network traffic (*e.g.*, link flooding attacks [4], traffic bursts in DCs [5], link failures [6], *etc*.).

However, there is a lack of a general traffic engineering mechanism that is agnostic to the network topology, the traffic patterns and the objective function and that is not tailored to handle traffic influx caused by specific events. In this regard, we propose SPONGE, a traffic engineering mechanism for handling sudden influx of network traffic that can work with any network topology, does not assume any specific pattern in the traffic matrix, and supports pluggable objective functions. SPONGE is a general traffic engineering approach that models the network as a stochastic process of packet arrivals and departures on switch queues and can support multiple objective functions such as minimizing the number of dropped packets and minimizing the queue occupancy across the network. We also demonstrate the practicality of SPONGE through an OpenFlow based implementation, where we periodically and pro-actively re-route network traffic to the routes computed by SPONGE. Mininet emulations using real network topologies show that SPONGE is capable of reducing packet drops by 20% on average even when the network is highly loaded because of an ongoing link flooding attack.

The rest of this paper is organized as follows. We review the literature on traffic engineering for mitigating network congestion and contrast them with SPONGE in Section II. Then we present a stochastic model to capture network dynamics in Sections III followed by the proposed control algorithm for computing optimal forwarding rules in Section IV. Section V-A highlights numerical results using Matlab simulations and Section V-B presents our experiments using Mininet to evaluate SPONGE for WAN and DC use-cases. Section VI concludes this paper with some future research directions.

II. Related Works

There is a large body of research literature on traffic engineering mechanisms to mitigate congestion events. They could be grouped into two major categories with respect to the

underlying network topologies and into to other two categories with respect to their goals according to the type of the congestion events to mitigate. A first category of traffic engineering techniques are proposed for wide-area networks including the tunning of the weights of routing protocols such as OSPF where traffic flows are routed along shortest paths [7], equal-cost multi-path routing (ECMP) are proposed where each switch randomly routes the flows across the available equal-cost paths. The second category is proposed for DC networks including approaches based on also ECMP and spanning trees where all the traffic traverses a single or multiple trees [8], which avoids loops but still leaves many links unused. In [2], the authors proposed the system Hedera for dynamically scheduling flows in multi-stage switch topologies used in data centers. Hedera collects information and computes non-conflicting paths by resolving to multi-commodity flow problem with heuristics. For both of these categories, OpenFlow capabilities offered by SDN controllers have been leveraged to schedule traffic and select forwarding paths using optimization and prediction algorithms of traffic matrices. In the data center network, Benson et al [5] proposed MicroTE that configures routes dynamically by aggregating monitoring traffic demands in a fine-grained way and using short term predictability to perform multipath routing. For wide-area networks, the most prominent approaches in production settings are Google's B4 [3] and Microsoft's SWAN [9] that rely on centralized SDN controllers by splitting application flows across multiple paths, where B4 was able to obtain 20% increase in throughput compared to shortest path forwarding. More recently, segment routing based approaches in SDN-based WANs have been proposed [10] for traffic engineering by using shortest paths and few middlepoint nodes to load balance traffic among their paths. Regarding the type of events, a first category of techniques are used to handle classical congestion events including sudden traffic and link failures. In this category, we found the same techniques that have been detailed above and applied either for wide area networks or data centers including B4 and SWAN and also the work of Suchara et al [6] where traffic engineering and failure recovery are handled jointly. In [11], the authors proposed the Kulfi toolkit for the comparison between existing algorithms of traffic engineering and in particular those using semi-oblivious routing which combines a static set of paths while dynamically adjusting the distribution of flow over those paths. In their study, they evaluated the capabilities of these algorithm to handle link failures.

More recently, a second category of techniques has emerged to handle link flooding attacks (LFA). D. Gkounis *et al.* proposed an SDN-based traffic engineering scheme to expose attackers in Crossfire-like attacks [12]. The proposed scheme constantly reroutes traffic during attacks while tracking hosts participating in link floods. The hosts that consistently participate in attacks are considered as attackers. In [13], C. Liaskos *et al.* further investigated this idea with a relational algebra model of the detection process. The approach, however, may involve long delays before the full mitigation of the attack as traffic is rerouted several times to identify each new group of attacking bots. With CoDef [14], S. B. Lee *et al.* proposed a defense system to reroute and rate-limit the attack flows before they reach the targeted link. CoDef leverages a collaboration mechanism between Autonomous Systems (ASes) and therefore, requires a widespread deployment to be effective. FLoc [15] and PSP [16] are two bandwidth allocation mechanisms introduced to alleviate the effect of link flooding attacks. Whereas FLoc requires router support and provides fair allocation for flows within an AS, PSP leverages current Internet routers and only provides bandwidth isolation between ASes. Besides FLoc, several active queue management schemes [17], [18] have been proposed specifically to alleviate the effects of flooding attacks. By nature, these solutions require support in Internet routers. In [4], D. Gkounis *et al.* studied the role of traffic engineering (TE) algorithms in the mitigation of Crossfire-like attacks and found that both existing and new kind of TE algorithms can expose the attackers, although the latest TE algorithms are faster.

SPONGE shares with other traffic engineering approaches the goal of mitigating congestion situations including link saturation, link flooding attacks and traffic bursts. Like B4 [3] and SWAN [9] for wide-area networks and MicroTE [5] for DCs, it relies on a logically centralized SDN controller for re-routing traffic at each switch according to a computed set of paths. However, SPONGE is not optimized for a specific network topology and derives optimized routes for mitigating a congestion events from only ingress-egress traffic matrix of the network, *i.e.* regardless of the cause of the congestion. For example, knowing the attacker in case of the link flooding attack is not a prerequisite.

III. Stochastic Network model

A. Notations

1) Network State: We model the network as a graph $\mathcal{G} = (V, E)$, where V and E are the set of network nodes and edges, respectively. The network state, at time t is represented as a system of queues (one per network node): $X_t = (X_t^i)_{i \in V}$. For any $i \in V$, X_t^i is a vector containing the final destinations of the packets waiting at node i at time t. Therefore, $X_t^i(1)$, the first coordinate of the vector X_t^i, is the destination of the next packet to be forwarded by node i. X_t^i is a random vector with a random length, *i.e.*, a stochastic process with values in: $\mathcal{S} := \bigcup_{n \geq 0} V^n$. In the remainder of this paper, $|X_t^i|$ denotes the length of the random vector X_t^i, *i.e.*, $|X_t^i| = N \Leftrightarrow X_t^i \in V^N$, where N is the number of packets in a node's queue. We assume that all network nodes $i \in V$ can hold a maximum of c_i number of packets in its queue, *i.e.*, $|X^i| \leq c_i$.

2) Network Dynamics: The dynamics of the network, *i.e.*, packet arrivals and departures at network nodes, is represented as *local variations* δX_t^i of the queues X_t^i, which describes the inputs and outputs of node i at time t. More precisely, the occupancy changes of the queue X_t^i can be decomposed into two parts as follows:

$$\delta X_t^i := \left(\delta^+ X_t^i,\ \delta^- X_t^i\right).$$

The *positive local variation*, $\delta^+ X_t^i$, represents the inputs at node i at time t, while the *negative local variation*, $\delta^- X_t^i$, corresponds to the packets forwarded by node i at time t. If a node i is neither receiving nor forwarding packets at time t, the local variations are equal to $\emptyset$. More formally:

$$\delta^- X_t^i = \begin{cases} \emptyset & \text{if } i \text{ does not emit any packet at time } t \\ k & \text{if } i \text{ forwards a packet at time } t \text{ to dest. } k \end{cases}$$

$$\delta^+ X_t^i = \begin{cases} \emptyset & \text{if } i \text{ did not receive any packet} \\ & \text{at time } t \\ v \in \bigcup_{n\geq 1} V^n & \text{if } i \text{ received packets at time } t \\ & \text{with destinations } v = (v_1, v_2 \dots) \end{cases}$$

To describe the dynamics of the model, we introduce the following two operators on $\mathcal{S}$. First, we define the addition operator $\oplus$. Let $v = (v_1, \dots, v_{n_1})$ and $w = (w_1, \dots, w_{n_2})$ be two elements of $\mathcal{S}$. Then, the operator $\oplus$ is defined as follows:

$$v \oplus w = (v_1, \dots, v_{n_1}, w_1, \dots, w_{n_2}).$$

Note that $\oplus$ is a non-commutative, *i.e.*, $v \oplus w \neq w \oplus v$. Second, the truncation operator, τ, is defined for any positive integer j and $v = (v_1, \dots, v_n) \in \mathcal{S}$ as follows:

$$\tau_j v = (v_1, \dots, v_{n-j}) \text{ and } \tau^j v = (v_{i+1}, \dots, v_n)$$

We follow the convention that if $j \geq n$ then $\tau_j v = \tau^j v = \emptyset$. Note that we have the following important relationship between $\oplus$ and τ when $i < |v|$:

$$\tau^i(v \oplus w) = (\tau^i v) \oplus w, \text{and}$$
$$\tau_i(w \oplus v) = w \oplus (\tau_i v),$$

Given the initial state of a node i, X_0^i and δX_s^i for all $s < t$, we can compute the network state at time t as follows:

$$X_t^i = \tau^k \left(X_0^i \oplus_{s \leq t} \delta^+ X_s^i \right), \text{ where } k = |\{s \leq t \mid \delta^- X_t^i \neq \emptyset\}| \quad (1)$$

(1) computes the state of node i by successively applying the addition operator $\oplus$ (*i.e.*, adding packets to the queue), and the truncation operator τ (*i.e.*, removing packets from the queue for forwarding). Note that the above definition is valid only if the number of packets passing through the network nodes up to time t is finite, which is the case in real networks.

3) External Inputs to the Network: For each node $i \in V$, the external inputs are described by a sequence of random variables $(\mathcal{I}_k^i, \mathcal{D}_k^i)_{k\geq 0}$, where $\mathcal{I}_k^i$ is the arrival time of the kth packet at node i and D_k^i is the destination of that packet. This allows us to define the input function of node i as follows:

$$I_t^i = \begin{cases} \mathcal{D}_k^i, & \text{if } \exists k, t = \mathcal{I}_k^i, \\ \emptyset, & \text{else.} \end{cases}$$

This function takes the destination of the received packet as value when node i receives a packet, $\emptyset$ otherwise. This implicitly assumes that a node cannot receive more than one packet from outside of the network at a given time.

4) Processing Time: For each node $i \in V$, time required to process packets in its queue is represented by a the vector $(T_{k,v}^i)_{k\geq 1, v\in V}$, where $T_{k,v}^i$ corresponds to the time spent by node i to process and transmit its kth packet to the next hop v. For each next hop v, the processing time includes the latency incurred on the network link between i and v.

5) Routing Table: We model the routing table of a node $i \in V$ as a function $R : V^2 \to V$ such that, for any pair of nodes $(i, j) \in V^2$, $R(i, j)$ gives the next hop on the chosen route from i to j. Such function should be compatible with the network topology, *i.e.*, for any $i, j \in V$, $(i, R(i, j)) \in E$. The set of routing functions over all nodes is denoted by $\mathcal{R}$.

B. Stochastic model of Network Dynamics

We represent the dynamics of network nodes using what a node receives as input and the node's positive and negative local variations introduced in Section III-A2. We will derive these variations from the model inputs, *i.e.*, packets entering in the network (Section III-A3) and the time required to process them (Section III-A4). We first focus on the positive variation $\delta^+ X_t^i$ and the inputs of node i. At a given time t, node i has two sources of inputs. The first is from outside of the network, *i.e.*, new packet arrivals, modeled by the function I_t^i. Second, in-transit packets forwarded by i's neighbors to i. The packets sent by node j to node i at time t are given by:

$$\delta_t^- X_t^j \mathbb{1}_{\{R_t(j, \delta_t^- X_t^j) = i,\ \delta_t^- X_t^j \neq i\}}$$

Here, $\delta_t^- X_t^j$ is the destination of the packets. $\mathbb{1}_{\{R_t(j,\delta_t^- X_t^j)=i,\ \delta_t^- X_t^j \neq i\}}$ is an indicator function involving two conditions: (i) $R_t(j, \delta_t^- X_t^j) = i$, requires that node i be the next node on the path of the packet to its destination; and (ii) $\delta_t^- X_t^j \neq i$, requires i not to be the destination of the packet. Otherwise, the packet is not forwarded and does not appear in the transmission queue.

Now we can fully describe the evaluation of $\delta^+ X_t^i$ in terms of the negative variations of the other nodes as follows:

$$\delta_t^+ X_t^i = \tau_{|v_t^i| - (|X_t^i| - c_i)}(v_t^i), \text{where}$$
$$v_t^i := I_t^i \oplus \left(\bigoplus_{j\in V,\ j\neq i} \delta_t^- X_t^j \mathbb{1}_{\{R_t(j,\delta_t^- X_t^j)=i,\ \delta_t^- X_t^j \neq i\}} \right)$$

v_t^i represents new packets arriving at node i consisting of both new external inputs and in-transit packets forwarded by i's neighbors. The operator $\tau_{|v_t^i| - (|X_t^i| - c_i)}$ ensures that a node i does not exceed its queue capacity. For instance, if at time t, $|X_t^i| = c_i$, then $\tau_{|v_t^i| - (|X_t^i| - c_i)} = \tau_{|v_t^i|}$, Hence, all the packets received at this time beyond i's queue capacity are dropped.

Finally, we need to describe the dynamics of $\delta_t^- X_t^i$. To do so, we need to know when node i started processing its last packet. We denote by L_t^i this time and by N_t^i the number of packets forwarded by node i at time t. Then, we can obtain:

$$\delta_t^- X_{t-}^i = X_t^i(1) \mathbb{1}_{t = L_t^i + T^i_{N_t^i, X_t^i(1)}}$$

C. Model instanciations

The stochastic model capturing network dynamics presented in Section III-B is generic in terms of inputs and processing times, *i.e.*, the model does not assume any specific distribution of packet inter-arrival and processing time. In the following, we provide several example instantiations of this model to be used later for validation.

- **Poissonian inputs:** Poisson arrival of input packets at the edge nodes.
- **Discrete time model:** $\forall (i, v) \in V^2$ and $k \in \mathbb{N}$, $T_{k,v}^i = 1$.
- **General deterministic model:** $\forall (i, v) \in V^2$ and $k \in \mathbb{N}$, $T_{k,v}^i = b_{i,v} \in \mathbb{R}_+$.
- **Poissonian model:** $\forall i \in V$, $(T_{k,v}^i)_{k\geq 1, v\in V}$ is an i.i.d. (independent and identically distributed) sequence of exponential random variables.

- **Noisy random model:** $\forall i \in V$, $T^i_{k,v} = (b_i + \varepsilon^i_{k,v})$, where $(\varepsilon^i_{k,v})_{k\geq 1, v\in V}$ is an i.i.d sequence of Gaussian random variables with a null mean and variance σ_i.

IV. Control and routing optimization

Given a stochastic model of network dynamics, our next step is to mathematically solve the problem of dynamic network reconfiguration in a way to uniformly distribute the traffic across all network nodes. Mathematically formulating this problem requires the network-wide routing table R to be time-dependent. Hence, from this point, we assume that R is a function from $\mathbb{R}_+$ with value R_t in $\mathcal{R}$. This means that at each time instance t, network dynamics is guided by the routing table R_t.

A. Optimal states of the network

In order to decide on an optimal routing table $(R_t,\ t \in [0, T])$ for a time window T (possibly infinite), we first need to define the optimality criteria. To do so, we first devise a way to quantify how "healthy" the network is, *i.e.*, the quality of a network state. Inspired by statistical mechanics, we use a potential function $\mathcal{H}$, called the Hamiltonian of the system, to describe the quality of a network state. A network state is an element of $\mathcal{S}^{|V|}$ (corresponding to the state space of our system $(X_t,\ t \in \mathbb{R}_+)$) describing the state of each of the queues in the network. There can be a different choices of the Hamiltonian, each resulting in a different network behavior. Therefore, given an objective such as uniformly distributing traffic across all network nodes, it is important to chose a proper Hamiltonian that will result in such desired behavior.

For instance, a possible choice for the Hamiltonian, called *direct routing potential* is defined as follows:

$$\mathcal{H}(X_t) = \sum_{i\in V}\sum_{k=1}^{|X^i_t|} d\left(i, X^i_t(j)\right)$$

Here, d is the hop count between nodes in $\mathcal{G}$, and $X^i_t(j)$ stands for the jth coordinate of the vector X^i_t. The direct routing potential sums the cumulative distances of the packets from their destinations. According to this Hamiltonian, the closer packets are to their destinations, the healthier the network is.

Another possible choice is the *low load potential* defined as follows:

$$\mathcal{H}(X_t) = |V| \exp\left(-\sum_{i\in V}(|X^i_t| - c_i)^2\right),$$

With the low load potential, healthy states are those where the queues are far from reaching their capacities. However, in order for a potential function to ensure delivery of packets to their destinations, the function needs to take the distances of the packets from their destinations into account. Since the low load potential does not take this into account, packets traversing a network under the low load potential may not reach their destination. Therefore, the low load potential function alone is unsuitable for our purpose. Our intention is to bring the network to a "low energy state", which ensures successful delivery of packets as well as reduces load on the queues across the nodes. In this regard, we propose to combine the two aforementioned potential functions as follows:

$$\begin{aligned}\mathcal{H}(X_t) &= \alpha \sum_{i\in V}\sum_{k=1}^{|X^i_t|} d\left(i, X^i_t(j)\right) \\ &+ (1-\alpha)|V| \exp\left(-\sum_{i\in V}(|X^i_t| - c_i)^2\right) \quad (2)\end{aligned}$$

This potential function combines direct routing and low load potentials to lead packets to their destination and to mitigate the load on queues, respectively. The weight $\alpha \in [0, 1]$ balances the strategy between direct and low load routing.

B. Control problem

Suppose one is given a potential function which suits her purpose (in our case: load mitigation) and is given the dynamic routing table $t \to R_t$. The running cost of the strategy at time horizon T is given by

$$J(R) = \int_0^T \mathcal{H}(X_s)\ ds.$$

The dependence on R is implicit since it appears only in the dynamics of the queuing system. Solving the optimal control problem is finding a dynamic routing table R that minimizes the running cost J, at least on average. To be consistent, one also wants the optimal routing table R_t, at a given time t, to be computed from the previous states of the network $(X_s,\ s \leq t)$. Mathematically, this means the process $t \mapsto R_t$ is adapted with respect to the natural filtration $(\mathcal{F}_t,\ t \in \mathbb{R}_+)$ associated to the process $(X_t,\ s \in \mathbb{R}_+)$. The optimal cost function is given by

$$u(T) = \inf_{R_t\in\mathcal{F}_t} \mathbb{E}\left[\int_0^T \mathcal{H}(X_s)\ ds\right]. \quad (3)$$

In the following, we will devise a numerical method for computing such an optimal control through model simulation.

C. Optimization method

For practical considerations, we assume the routing tables to be piecewise linear over time, *i.e.*, the routing table does not change within a time step δt. This is a reasonable assumption since in practice traffic engineering is either performed periodically or is triggered by network events, and the routing table remains constant in between. In the sequel, we propose an algorithm to find a dynamic routing table that is constant in the time interval $[j\delta t, (j+1)\delta t)$ for all positive integer j. We employ a simulated-annealing like algorithm to compute routing tables that are *locally optimal in time*.

Simulated annealing [19] is a meta-heuristic algorithm for finding locally optimal solutions from a large search space of combinatorial optimization problems. To leverage this technique, we use the Gibbs measure [20], which quantifies how healthy a network state is. The Gibbs measure associated to the potential function $\mathcal{H}$ is a measure on $\mathcal{S}^N$ that assigns a weight $G_\beta(x)$ to each state $x \in \mathcal{S}^N$ such that

$$G_\beta(x) = \frac{1}{Z(\beta)} \exp\left(-\beta\, \mathcal{H}(x)\right)$$

Here, $Z(\beta)$ is a re-normalizing constant defined in a way such that G_β becomes a probability measure on $\mathcal{S}^N$. By construction, Gibbs measure gives an important weight to low energy states, *i.e.*, the healthy network states in our context. A particularly important feature is that G_β converges weakly, as β goes to 0, to a linear combination of Dirac masses localized in the optimal states of the network.

To sample according to the distribution G_β, we use the Metropolis-Hasting (MH) algorithm [21], a Monte-Carlo Markov chains technique. To this end, a usual choice for the transition function P of the Markov chain is:

$$P(x,y) = \min\left\{1,\ \frac{G_\beta(x)\pi(y,x)}{G_\beta(y)\pi(x,y)}\right\}$$

Here, π is the proposal kernel. As a consequence, in the context of finite state space, the condition $\pi(x,y) > 0$, for all states x, y, is sufficient to ensure the ergodicity of the Markov chain and the convergence of the algorithm to a sample of law G_β. The idea of simulated annealing is to combine the weak convergence of G_β with iterations of MH algorithm in order to sample, for low temperatures and many iterations of MH, states x that are close to optimality.

However, in our context, optimal states and optimal routing table do not live in the same space. Consequently, we choose π as a proposal kernel of the space of routing tables $\mathcal{R}$. The new state of the system is then given by the dynamical system described in Section III-B. This idea leads to Algorithm 1 for finding the optimal routing table in times $[j\delta t, (j+1)\delta t)$ for all j.

Data: Initial time t, time step δt, current state X_t and initial routing table R.
Result: Optimal routing table R to apply in the network for times $[t, t+\delta t]$.
for j from 1 to N **do**
  $\beta = \frac{c}{\log(j+1)}$;
  $\widetilde{R}$ distributed according to $\pi(R,\cdot)$;
  simulate $\widetilde{X}_{t+\delta t}$ starting from X_t with routing table $\widetilde{R}$;
  $\alpha = \min\left\{\frac{\pi(\widetilde{R},R)\mathbb{E}G_\beta(\widetilde{X}_{t+\delta t})}{\mathbb{E}G_\beta(X_{t+\delta t})\pi(R,\widetilde{R})}, 1\right\}$;
  if $\alpha < \mathcal{U}$ **then**
    replace R by $\widetilde{R}$;
    replace $X_{t+\delta t}$ by $\widetilde{X}_{t+\delta t}$;
  end
end

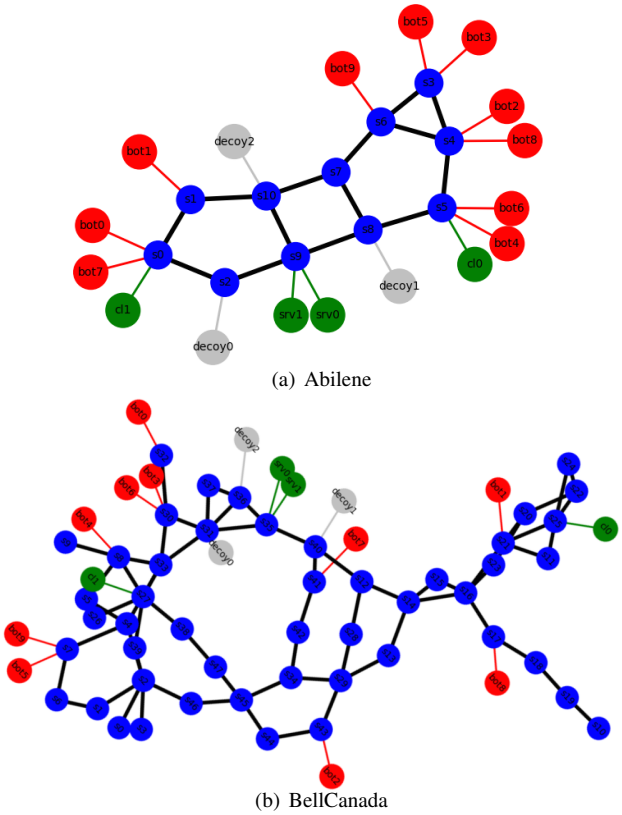
Algorithm 1: Simulated annealing for routing optimization

V. Evaluation

We evaluate SPONGE through model simulation in Matlab (Section V-A) and network emulation using Mininet [22] (Section V-B). Our evaluation is focused on showing the impact of model parameters as well as SPONGE's practical capability to absorb traffic influx in the network.

A. Results from Model Simulation

In this section, we present results from Matlab simulation of the model described in Section III-B. Our simulation is intended to demonstrate that instantiation of the general model from Section III-B with specific distributions of input arrivals and processing times work on realistic network topologies. In our model instantiations, we assume the sequence $(\mathcal{I}_k^i, \mathcal{D}_k^i)_{k\geq 1}$ to be i.i.d. or Markovian, and the sequence $(T_{k,v}^i)_{k\geq 1, v\in V}$ to be i.i.d. Random packet arrivals and random packet lengths are common assumptions in network queuing theory and largely adopted in the literature. Specifically, we use the Poissonian inputs model instantiation as described in Section III-C to compute local optimal routing table for mitigating a link flooding attack. For the processing times, we consider two models for our Matlab simulation: one with deterministic processing times and one with random processing times.



(a) Abilene

(b) BellCanada

Fig. 1. Topologies deployed on Mininet

We use the Bell Canada network topology (Figure 1(b)) from the *Internet zoo topology project* [23] for this simulation study. This topology is rather convenient for our simulations since it has a high path diversity. In order to mitigate heavy traffic in the simulated network, our optimization uses the potential from (2). Our main goal is to observe the effect of parameter α on the successful delivery rate of packets (hereafter referred as *arrival fraction*). To understand the effect of re-routing on the mean packet delivery time (*i.e.*, the time it takes for a packet to reach its destination after being sent from the source), the inputs are stopped at a fixed time but the network simulation continues until all packets have been absorbed or dropped. We particularly look at two metrics:

- *arrival fraction*: the ratio between the total number of inputs and the number of packets actually arrived at their destinations;

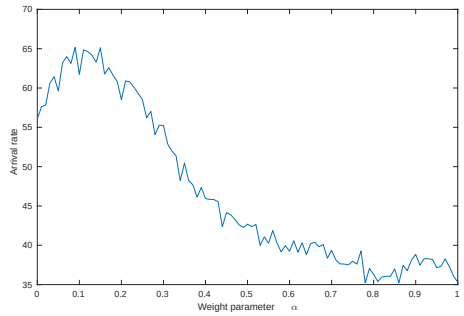


Fig. 2. Effect of parameter α on the arrival rate of packets (discrete model)

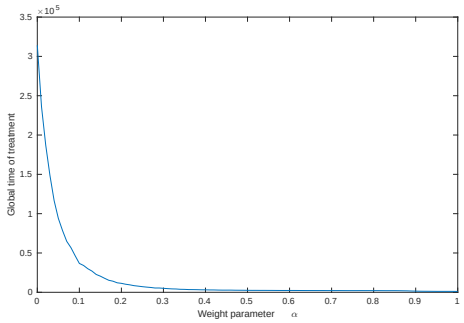


Fig. 3. Effect of parameter α on absorption time (discrete model)

- *absorption time*: the time needed for the network to absorb the whole traffic.

The first model we simulate is a *discrete model* where the processing times are constant and the inputs follow Poisson point processes. The destinations of the inputs are chosen uniformly at random in the network. Figure 2 shows the arrival fraction for different values of α in equation (2). We observe a 25% gain in arrival fraction for $\alpha \simeq 0.13$ compared to the direct routing strategy ($\alpha = 1$). Note that the arrival fraction is low for the purely low load strategy (*i.e.*, $\alpha = 0$) because the packets have no incentive to reach their destinations.

Another important aspect, the global absorption time of the network (*i.e.*, the time needed for all the packets to be absorbed), is shown in Figure 3. Figure 3 shows that absorption time increases exponentially as α goes towards zero. This is intuitive since the low load strategy is not meant to lead packets to their destinations. Rather, this strategy tries to avoid high load at the nodes. As a consequence, the only way to absorb a packet is to reach randomly its destination or to drop the packet. The latter is expected to be rare since this strategy avoids high load at the nodes. From a practical standpoint, the unit of time on the Y-axis of this figure has to be considered the same as the unit used for the processing times. For instance, if the processing times are random variables with values in milliseconds, so is the unit of time in this figure.

The second model considered in our simulation is a Poisson processing time model. The difference with the first model is that the processing times of the packets are random instead of constant. As shown in Figures 4 and 5, we first note the lack of regularity in the plot for the random model, because the optimization is made on the expectation. Although the plot of Figure 5 is less regular, one can see that we obtain substantial gain in the arrival fraction using our method. Furthermore, in

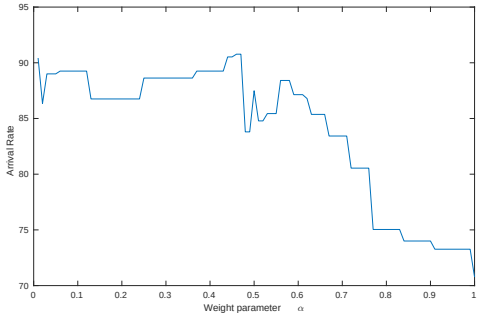


Fig. 4. Effect of parameter α on the arrival rate of packets (random model)

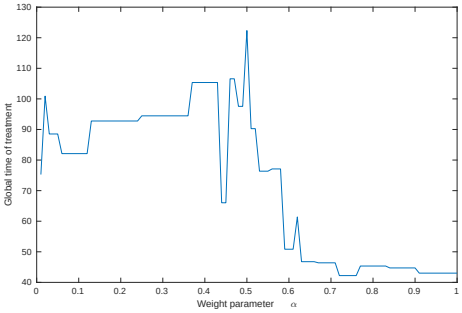


Fig. 5. Effect of parameter α on absorption time (random model)

practical situations the uncertainty in processing and transfer times in the network is often expected to have low variances.

There are two main consequences to using a Poisson processing time model. First, the numerical evaluation of the following equation in Algorithm 1 needs several simulations of the network to obtain an accurate numerical estimation through Monte Carlo methods.

$$\mathbb{E}\left[G_\beta(\widetilde{X}_{t+\delta t})\right] \tag{4}$$

This increases the computational cost of the method notably because of the large variance of the exponential distribution. This implies that a high number of trials is needed to obtain an accurate estimation. This was also true in the first model because of the randomness of the inputs but without negative consequences on the quality of the solution to the optimization problem. In order to keep the computational cost reasonable, one should either reduce the number of iterations of Algorithm 1 or the quality of the estimation of (4). The second consequence is that the obtained solution becomes very sensitive to deviation of a network's behavior from the mean behavior.

B. Experimental evaluation

We implement SPONGE as a traffic engineering application using Floodlight OpenFlow controller's REST API (http://www.projectfloodlight.org/floodlight/). The control application periodically and pro-actively pushes routes into the OpenFlow network to minimize packet drops. We demonstrate the effectiveness of these computed routes by evaluating SPONGE for the following use-cases: (i) to mitigate distributed link flooding attack similar to a Crossfire attack [24] in a WAN, and (ii) to handle in-cast congestion often caused by aggregate traffic patterns observed in data center networks. As a baseline, we use the default forwarding module of Floodlight controller. We emulate WAN and DC network topologies using

Mininet [22] and generate traffic using D-ITG tool [25]. We use packet loss between a pair of communicating hosts as the metric to evaluate the quality of routes produced by SPONGE.

1) Experimental setup:

a) Network Topology: We use Abilene (11 nodes) and BellCanada (48 nodes) network topologies from the Internet Topology Zoo [23]. We use the latency values from the topology and set the link bandwidths to 10Mbps. We deployed two legitimate clients and two legitimate servers, three decoy hosts and 10 bot hosts in both of these networks. This setting is similar to the one described for a Crossfire link flooding attack [24]. In a Crossfire attack, a collection of bots send traffic to a selected set of decoy servers at moderate rates. However, collectively they saturate certain links in the network and increase the drop rate for legitimate traffic. The resulting networks after deploying the hosts are shown in Fig. 1(a) and Fig. 1(b). The blue, red, gray, and green nodes represent the switches, the bots, the decoy servers, and the legitimate clients and servers, respectively. The decoy servers are placed close to the legitimate server nodes to increase the chances of congesting links on the paths leading to the servers. The bot nodes are placed randomly across the network. Their goal is to saturate the links connecting the legitimate clients and servers (cl0, cl1, srv0, and srv1).

For the data center network topology we generate a leaf-spine network with 4 spine switches and 8 leaf switches. We set the capacity of the links connecting leaf and spine switches to 40Mbps. Each spine switch has 4 hosts attached to it, each with a 40Mbps link. The leaf-spine network is not oversubscribed to ensure that the network is not a bottleneck.

b) Traffic Matrix: The bots and decoy servers act as senders and receivers, respectively. We emulate Crossfire attack [24], a distributed link flooding attack, by exchanging UDP traffic (512 byte sized packets at 600 packets/sec rate) between the bots and the decoy servers according to the traffic matrix in Table I. For this use-case, we experiment with different combinations of experiment duration, attack duration, and rerouting interval summarized in Table I. Note that we use only one configuration for Abilene topology, to experiment with multiple values of α in (2) while keeping the attack and experiment duration fixed.

For the data center use case, the hosts exchange UDP packets according to the traffic matrix presented in Table II. Note that, this traffic matrix presents an aggregation traffic pattern [2], where all the traffic is destined to hosts in one rack, *i.e.*, connected to one leaf switch. We set the packet size to 200 bytes (average packet size reported in a recent data center network measurement study [26]) and set the packet rate to 10000 packets/second. We set α to 0.5, change the routes every 1s, and let the experiment run for 20s.

2) Results:

a) Mitigation of Link Flooding Attack: Table III compares SPONGE against the baseline in terms of average packet drop over all host pairs for the configurations in Table I. For all configurations, SPONGE is able to reduce the average packet drop rate. This reduction ranges between 30 65% compared to

TABLE I
TRAFFIC MATRIX FOR LINK FLOODING MITIGATION USE-CASE

Sender	Receiver
Bot 0	Decoy 1
Bot 1	Decoy 0
Bot 7	Decoy 0
Bot 9	Decoy 2
Bot 5	Decoy 2
Bot 3	Decoy 1
Bot 2	Decoy 2
Bot 8	Decoy 2
Bot 6	Decoy 1
Bot 4	Decoy 1
Cl0	Srv 0
Cl1	Srv 1

TABLE II
TRAFFIC MATRIX FOR DATA CENTER USE-CASE

Sender	Receiver
h0	h12
h2	h12
h10	h13
h8	h13
h16	h14
h18	h14
h20	h15
h22	h15

TABLE III
AVERAGE PACKETS DROP (%)

Scenario	**Packet drop (%)**	
	Baseline	**SPONGE**
AB-1	31.10	16.24
BE-1	24.90	15.40
BE-2	24.90	17.53
BE-3	44.37	15.17
DC	55.21	33.34

TABLE IV
IMPACT OF α

α	Avg. Delay (s)
0.1	0.85
0.5	0.70
0.9	0.72

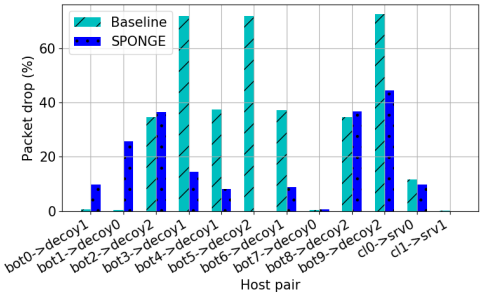


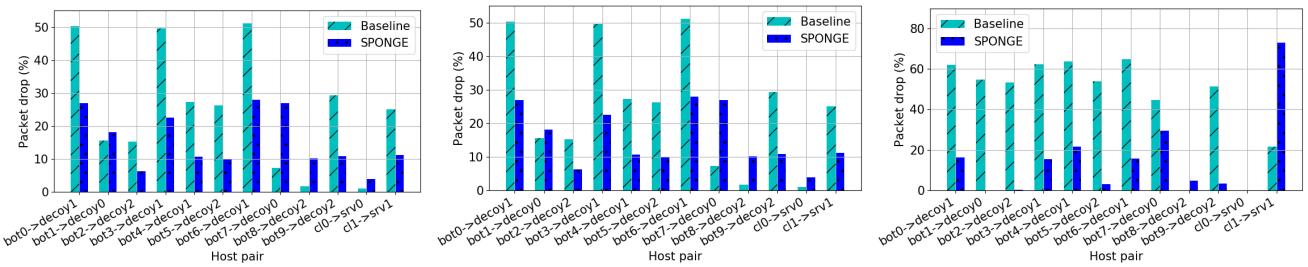
Fig. 6. Packet drops (AB-1 Scenario)

the baseline approach. This demonstrates SPONGEs capability to absorb the influx of traffic during link flooding attacks.

We take a closer look at the packet drop percentage between pairs of communicating hosts under different configurations by plotting the loss rate of individual communication pairs in Fig. V-B2a and Fig. 7. We have designed SPONGE to be agnostic of underlying traffic pattern, therefore, SPONGE cannot differentiate between legitimate and attack traffic. As a result, we can see that SPONGE reduces the overall packet drop by sometimes negatively impacting the legitimate traffic. However, at this expense, SPONGE reduces the overall packet drop across all host pairs by up to 65%.

Finally, we present the impact of α on the computed routes for Abilene topology in Table V. From the simulation results in Section V-A, it is expected that a lower value of α will result in longer paths, which will be reflected in the delay. Our emulation results also support this finding.

b) Data center use-case: For the data center use-case, the aggregation traffic pattern, when forwarded with Floodlights default forwarding mechanism, resulted in 55.21% average packet drop over all host pairs. With SPONGE, this drop rate reduced to 33.34%, which is 39.6% improvement over the



(a) BE-1 (b) BE-2 (c) BE-3

Fig. 7. Packets drop for host pairs in scenario BE-2.

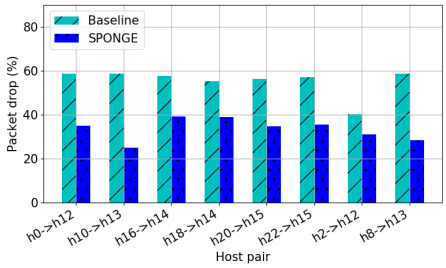


Fig. 8. Packets drop between host pairs in data center

baseline approach. Per host pair packet drop is presented in Fig. 8. SPONGE reduced the packet drop rate in all cases by exploiting the multiple paths available in the leaf-spine network. However, due to using a limited number of iterations in the Simulated Annealing algorithm, SPONGE did not fully exploit the many alternate paths available in the leaf spine topology, therefore, still causing packet drops.

VI. Conclusion and Future Work

In this paper, we presented SPONGE, a traffic engineering mechanism that captures the dynamic behavior of a network through a novel stochastic model and optimizes the routing policy based on the proposed model. SPONGE is topology and traffic agnostic, i.e., does not assume any specific topology or pattern in network traffic matrix. Matlab simulations of SPONGE show promising results in terms of being able to absorb influx of network traffic caused by events such as a DDoS attacks. We also evaluate the effectiveness of SPONGE using an OpenFlow based implementation. Our experimental results on Mininet covering two use-cases from wide-area and data center networks demonstrate that SPONGE is capable of reducing packet drops by more than 20% when there is an influx of network traffic. However, the current design of SPONGE does not distinguish between different classes of traffic, therefore, can penalize a legitimate traffic over a malicious one. Consideration for differential traffic classes is left as a future extension. The possibility of applying machine learning to automatically identify traffic as legitimate or malicious and treat them accordingly is also another interesting future research direction.

References

[1] R. P. Roess, E. S. Prassas, and W. R. McShane, *Traffic engineering*. Pearson/Prentice Hall, 2004.

[2] M. Al-Fares *et al.*, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. of USENIX NSDI*, 2010, pp. 19–19.

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

[4] D. Gkounis and ohers, "On the interplay of link-flooding attacks and traffic engineering," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 2, pp. 5–11, May 2016.

[5] T. Benson *et al.*, "Microte: Fine grained traffic engineering for data centers," in *Proc. of ACM CoNeXT*, 2011, pp. 8:1–8:12.

[6] M. Suchara *et al.*, "Network architecture for joint failure recovery and traffic engineering," in *Proc. of ACM SIGMETRICS*, 2011, pp. 97–108.

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

[8] H. T. Viet *et al.*, "Traffic engineering for multiple spanning tree protocol in large data centers," in *Proc. of ITC*, 2011, pp. 23–30.

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

[10] G. Trimponias *et al.*, "On traffic engineering with segment routing in SDN based wans," *CoRR*, vol. abs/1703.05907, 2017.

[11] P. Kumar *et al.*, "Kulfi: Robust traffic engineering using semi-oblivious routing," *CoRR*, vol. abs/1603.01203, 2016.

[12] D. Gkounis, V. Kotronis, and X. A. Dimitropoulos, "Towards defeating the Crossfire attack using SDN," *CoRR*, vol. abs/1412.2013, 2014.

[13] C. Liaskos, V. Kotronis, and X. Dimitropoulos, "A novel framework for modeling and mitigating distributed link flooding attacks," in *Proc. of IEEE ICC*, 2016, pp. 1–9.

[14] S. B. Lee, M. S. Kang, and V. D. Gligor, "Codef: Collaborative defense against large-scale link-flooding attacks," in *Proc. of ACM CoNeXT*, 2013, pp. 417–428.

[15] S. B. Lee and V. D. Gligor, "FLoc: Dependable link access for legitimate traffic in flooding attacks," in *Proc. of IEEE ICDCS*, 2010, pp. 327–338.

[16] J. C.-Y. Chou *et al.*, "Proactive surge protection: A defense mechanism for bandwidth-based attacks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 6, pp. 1711–1723, Dec. 2009.

[17] H. Bedi, S. Roy, and S. Shiva, "Mitigating congestion-based denial of service attacks with active queue management," in *Proc. of IEEE GLOBECOM*, 2013, pp. 1440–1445.

[18] C. Zhang *et al.*, "Flow level detection and filtering of low-rate DDoS," *Comput. Netw.*, vol. 56, no. 15, pp. 3417–3431, Oct. 2012.

[19] A. Das and B. Chakrabarti, *Quantum annealing and related optimization methods*. Springer Science & Business Media, 2005, vol. 679.

[20] H.-O. Georgii, *Gibbs measures and phase transitions*, 2nd ed., ser. De Gruyter Studies in Mathematics. Walter de Gruyter & Co., Berlin, 2011, vol. 9.

[21] C. P. Robert and G. Casella, *Monte Carlo statistical methods*, 2nd ed., ser. Springer Texts in Statistics. Springer-Verlag, New York, 2004.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. of ACM HotNets*, 2010, pp. 19:1–19:6.

[23] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.

[24] M. S. Kang, S. B. Lee, and V. D. Gligor, "The Crossfire attack," in *Proc. of IEEE S&P*, 2013, pp. 127–141.

[25] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[26] A. Roy *et al.*, "Inside the social network's (datacenter) network," in *Computer Comm. Review*, vol. 45, no. 4, 2015, pp. 123–137.