Computer Networks 110 (2016) 243–252

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Efficient routing for middlebox policy enforcement in software-defined networking

Xin Li[a,*], Haotian Wu[a], Don Gruenbacher[a], Caterina Scoglio[a], Tricha Anjali[b]

[a] Kansas State University, Manhattan, Kansas, USA
[b] International Institute of Information Technology, Bangalore, India

ARTICLE INFO

Article history:
Received 18 February 2016
Revised 10 September 2016
Accepted 4 October 2016
Available online 5 October 2016

Keywords:
Sequenced-middlebox policy
OpenFlow
Load balancing
Middlebox failures

ABSTRACT

Network applications require traffic to sequence through multiple types of middleboxes to enhance network functions, e.g., providing security and guaranteeing performance. Sequenced-middlebox policy routing on top of regular layer 2/3 flow routing is challenging to be flexibly managed by network administrators. In addition, various types of middlebox resources concurrently obtained by numerous applications complicate network-resource management. Furthermore, middlebox failures can lead to a lack of security and the malfunction of entire network. In this paper, we formulate a mixed-integer linear programming problem to achieve a network load-balancing objective in the context of sequenced-middlebox policy routing. Our global routing approach manages network resources efficiently by simplifying candidate-path selections, balancing the entire network and using the simulated annealing algorithm. Moreover, in case of middlebox failures, we design a fast recovery mechanism by exploiting the remaining link and middlebox resources locally. To the best of our knowledge, this is the first work to handle failures in sequenced-middlebox scenarios using OpenFlow. Finally, we implement proposed routing approaches on Mininet testbed and evaluate experiments' scalability, assessing the effectiveness of the approaches. Results of the optimization on a test topology include an increase up to 26.4% of the throughput with respect to a sequenced shortest-path routing.

1. Introduction

Today's network relies on middleboxes to guarantee critical network functions, e.g., security inspection and performance improvement. Network applications require traffic to sequence through multiple types of middleboxes to accomplish desired network functions. For example, Web traffic needs to go through a proxy and then a firewall [1]. To fulfill network functions, various types of middleboxes are utilized and each type might have $O(100)$ devices in a large network [2,3]. In traditional communication networks, traffic steering to meet the above goals is a critical problem [4], which might create false configurations. It is also difficult to dynamically update the routing policy. It is even more challenging to enable the stateful policy routing (sequenced-middlebox routing) within limited network resources (network link bandwidth, middlebox-processing capability, and switch high-speed searching memory). Software defined networking (SDN) can be used to solve these problems and reduce manual configurations.

The problem of routing under middlebox sequence constraints has recently gained remarkable attention due to the role played by many network devices called middleboxes (e.g., firewalls, VPN gateways, proxies, intrusion detection systems (IDS), WAN optimizers) on the network performance [5–9]. To enforce middlebox policy, a novel middlebox architecture was presented by Sekar et al. in [5]. In this paper, the authors designed a network-wide controller and a local coordinator to manage middlebox resources, resembling the architecture of SDN networks. As a matter of fact, a centralized SDN controller makes a network transparent and synchronous [6], and more efficient for network administrators to manage. Furthermore, Joseph et al. proposed a policy-aware switching layer to enforce middlebox policy and increase middleboxes utilization [7]. They also presented an off-path middlebox deployment. However, under this off-path deployment, flows are often required to travel on one link multiple times, increasing the probability of link overload. Fayazbakhsh et al. further modified legacy middleboxes to support FlowTags, which is used to differentiate flows with different policy requirements [8]. Alternatively, OpenFlow allows the identification of stateful policy flows using the available tuples in the packet header. Using OpenFlow, Qazi et al. elaborated the complexity of selecting middleboxes

* Corresponding author.
E-mail address: xinli1125@ksu.edu (X. Li).

http://dx.doi.org/10.1016/j.comnet.2016.10.002
1389-1286/© 2016 Elsevier B.V. All rights reserved.

and scheduling flows, and simplified the middlebox traffic steering problem by offline pruning some less promising routing paths [2]. The proposed offline calculation is time-consuming, and it was performed each time when failures occurred or policy changed. This aspect is problematic since networks should quickly respond towards middlebox/link overloads and failures. This is the most related work with our paper.

Another topic of research in the field of middlebox management concerns how to deal with link failures and middlebox failures [10–13]. Research indicated that middleboxes contribute to 43% of high-severity incidents [13,14]. Thus, it is critical to study middlebox failures. Existing solutions are either preventing middlebox failures effect beforehand [15] or reacting after middlebox failures, for example, reconstructing middlebox state after a failure [16]. However, today's network relies on sequenced types of middleboxes to provide network functions and different types of traffic go through different sequences, which are all beyond the scope of existing approaches. Restarting middleboxes is a common approach to deal with middlebox failures, but few articles considered the impact during the restarting period.

In this paper, we narrow the choices of candidate paths to accelerate routing path selection procedure for network scalability purposes. Then we formulate a flow-routing optimization problem with the goal of globally balancing the loads of the entire network, including balancing link loads and middlebox loads. We determine flow routing paths by solving this optimization problem, and then install the routing paths through the OpenFlow controller. We validate our approach's efficiency, compared with the classic flow-routing approach, by testing network performance on Mininet [17]. In addition, we employ a fast local rerouting approach to respond to transient disturbances occurring in the network. Finally, we conduct complementary experiments to validate the efficiency of our fast local rerouting approach and analyze experiment scalability. Our approaches are appropriate to tackle both long-term network changes and transient network disturbances.

The paper is organized as follows. We introduce the challenges and feasibility of a sequenced-middlebox policy routing in Section 2. In Section 3, we introduce a global load-balancing routing approach by selecting candidate paths and formulating a mixed-integer linear programming problem to allocate network resources. Solutions to the optimization problem are introduced in Section 4. We propose to reroute affected flows locally to deal with transient network disturbances and make network more resilient towards failures in Section 5. In Section 6, we test our approaches using Mininet and analyze the results of the experiments. Finally, conclusion and future work are discussed in the last section.

2. Problems and challenges

To guarantee security and performance of network applications, traffic must sequence through multiple types of middleboxes, which is called stateful middlebox policy routing. To fulfill such routing, one flow can go through a switch several times that is beyond the scope of regular layer 2/3 flow routing. Thus, we need to keep track of the state of a flow, and that's why the routing is stateful. In the following sections, we call this policy "middlebox policy" for short. Many issues arise when implementing the middlebox policy. One is the mapping between logical middlebox policy and physical device selection. Also, the physical selection must update as the logical policy changes, bringing the routing paths' reconfiguration. Another issue is the allocation of network resources, e.g., link bandwidth and middlebox-processing capability. Finally, a fast-recovery mechanism is necessary in case of failures, but it is also challenging to design an efficient and effective approach to realize it.

2.1. Policy mapping

Mapping the logical policy and physical routing path is difficult. For example, we have Web traffic which needs to go through a proxy middlebox, then a firewall middlebox. If we have 100 proxies and 100 firewalls in the network, we have 10,000 middlebox selection combinations [2]. In addition, for a given middlebox combination, there are various routing paths to choose from. Furthermore, different applications have distinct middlebox policies. The selection of middleboxes and routing paths is very critical to network performance. Moreover, the solution search space of the problem grows exponentially as the number of flows increases.

Realizing the physical routing can also be a problem. The biggest challenge is, since a flow may travel through a switch multiple times, the switch must be able to assign different actions to the same flow. Therefore, the switch should keep track of the state of a given flow.

2.2. Policy updating

Network applications require various types of middleboxes and distinct routing sequences. These policies are updated when application requirements change. To avoid the misconfiguration during policy update, centralized and programmable management is required in the network to solve this middlebox policy routing problem.

2.3. Network resources

Network resources include link bandwidth, middlebox-processing capability and switch-processing capability. Link bandwidth is the most studied network resource constraint in traffic engineering research. Switch-processing capability constraint is often automatically satisfied. However, in the middlebox policy problem, the switch-processing capability must be considered [2]. Furthermore, the constraints of limited middlebox-processing capability must be taken into consideration and, therefore, this problem becomes much more complex.

2.4. Fast response towards failures

Our global load-balancing optimization quickly responds to long-term network topology changes or failures. As a response to transient network changes, global flow management makes network convergence difficult from the view of network scalability [18]. We find a fast local rerouting mechanism that only considers the flows, which are affected by the failures. If one middlebox fails, we can restart it. When we assign an alternative rerouting path during the restarting period, we need to assure the alternative path has no effect on the normally working subnetwork. Middlebox centralized control and monitoring are required to deal with middlebox failures.

2.5. Software-defined networking technique

Flow routing along a sequence of middleboxes is complex. It includes traffic steering, network configuration, failures monitoring, policy update, network resilience and network convergence, etc. Software-defined networks (SDN) [19,20] are the techniques to solve such problems. The SDN architecture decouples the network control plane from the data plane: the network administrator specifies middlebox policy and manages the network from the centralized controller; the programmable datacenter makes flow routing more flexible and realizes faster network recovery. Thus, we can realize middlebox policy routing easily using SDN; specifically, we use OpenFlow protocol to enable the communication between the control plane and the data plane.

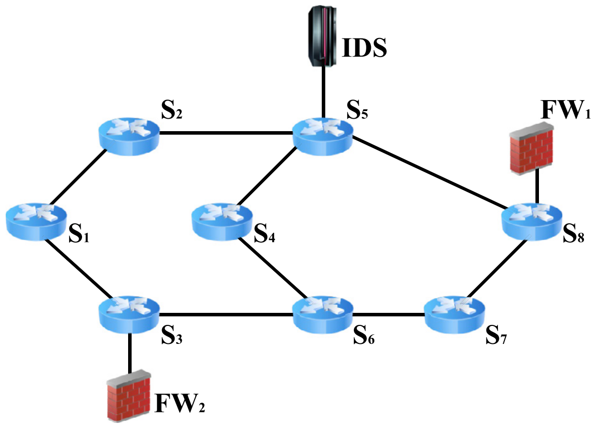


Fig. 1. Middlebox-by-middlebox shortest paths.

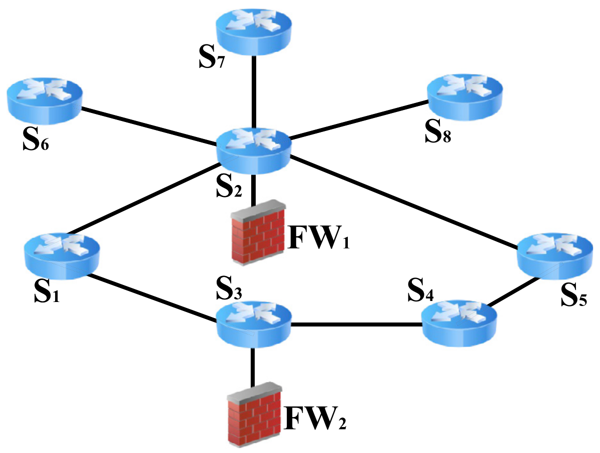


Fig. 2. Middlebox overloaded example.

Table 1
Routing path ($S_1 \rightarrow$ Firewall (FW) $\rightarrow$ IDS $\rightarrow S_6$).

Path\Step	1	2	3	4	5	6	7	8	9	10	11
a	S_1	S_2	S_5	S_8	FW_1	S_8	S_5	*IDS*	S_5	S_4	S_6
b	S_1	S_3	FW_2	S_3	S_6	S_4	S_5	*IDS*	S_5	S_4	S_6
c	S_1	S_3	FW_2	S_3	S_1	S_2	S_5	*IDS*	S_5	S_4	S_6

3. Global load-balancing routing

In this section, we first propose a selection approach of candidate routing paths, by removing the paths with a low probability to be selected. Our selection approach largely reduces choices of routing paths. Then we formulate a flow-routing problem to optimize network resource allocation in the sequenced-middlebox routing scenario. The objective of the optimization problem is to balance the loads of entire network, including link load balancing and middlebox load balancing. To maintain flow integrity and reduce the number of flow entries on each switch, we adopt a flow-level routing where one flow is assigned one path.

3.1. Candidate paths generation

We propose a set of routing paths called "middlebox-by-middlebox shortest routing paths" (m-by-m routing paths). When the middlebox policy is specified by network administrators, candidate paths are determined. M_i denotes the set of the i_{th} type middleboxes, e.g., firewalls, and $|M_i|$ denotes the number of middleboxes of the type *i*. A middlebox policy is "$Source \rightarrow M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_n \rightarrow Destination$." Each candidate path is determined by finding the shortest path to or from each middlebox: $Source \rightarrow M_1$, $M_1 \rightarrow M_2$, ..., and $M_n \rightarrow Destination$. An example is shown in Fig. 1. There is a directed demand from S_1 to S_6 with the logical policy "$S_1 \rightarrow$ firewall $\rightarrow$ intrusion detection system $\rightarrow S_6$." Step 1 is to find the shortest path from S_1 to FW_1 (or FW_2). Step 2 is determination of the shortest path from FW_1 (or FW_2) to *IDS*. Step 3 is to find the shortest path from *IDS* to S_6. Since there are two firewalls and one IDS, there will be at least two candidate paths for the demand "$S_1 \rightarrow S_6$" to route along. To be more general, if there are $|M_{FW}|$ firewalls and $|M_{IDS}|$ IDSs, and the flow must go through a firewall then an IDS, there will be at least $|M_{FW}| \times |M_{IDS}|$ possible paths for this flow to choose from.

We show three different m-by-m routing paths in Table 1. The difference between path *a* and path *b* is a different firewall selection; the difference between path *b* and path *c* is a different shortest-path selection among multiple shortest paths.

We propose the m-by-m routing paths as candidate paths for the following reasons:

- The m-by-m approach allows the simple generation of many candidate paths and one of them will be chosen by the centralized controller to achieve network load balancing. On the chosen path, we can easily record link and middlebox loads. Therefore, we are able to balance link and middlebox loads simultaneously.
- To avoid network congestion, the flows may be routed through a longer m-by-m routing path. However, within a certain step, the shortest path is always chosen to save network resources.
- Using this approach, flow-level routing can largely reduce the number of flow entries installed on each switch, compared with the flow-splitting routing.
- The m-by-m approach can reduce congestion on a middlebox. Let's consider a routing example, shown in Fig. 2. The middlebox policy is "Source $\rightarrow$ Firewall $\rightarrow$ Destination"; the directed pairwise demand list is {$S_1 \rightarrow S_5$: 10 Mbps, $S_6 \rightarrow S_5$: 10 Mbps, $S_7 \rightarrow S_5$: 10 Mbps, $S_8 \rightarrow S_5$: 10 Mbps}; and the capability of either firewall is 30 Mbps. FW_1 will be overloaded if all flows choose the sequenced, shortest source-destination path (SP routing approach). We use m-by-m routing paths instead, with which a middlebox of the same type (FW_2) can help reduce congestion on the highly used middlebox FW_1 (because FW_1 is connected with a high-betweenness switch S_2).

3.2. Mixed-integer linear programming problem

In the middlebox policy routing problem, we need to achieve network load balancing not only on the links but also on the middleboxes. In this section, we formulate a mixed-integer linear programming (MILP) problem to describe the network [21] and accomplish our goal, given the set of candidate paths. We call this global load-balancing routing approach (global LB approach).

3.2.1. Notations

Traffic matrix D_t represents demands of traffic type *t*, $t \in T$. Each traffic type has a different policy requirement. For example, traffic matrix D_1 represents HTTP traffic demands, which need to route through a firewall middlebox then an IDS middlebox; traffic matrix D_2 represents all other traffic demands, which need to route through a firewall middlebox only. Traffic demands are directed pairwise demands. P_d denotes the set of candidate paths for a given pairwise demand *d*. Given a network topology, *E* denotes the set of links and *M* denotes the set of middleboxes. Vertical bars denote the cardinality of a set, for example, $|D_t|$ denotes the number of demand pairs of traffic type *t*.

Constants:

n_{tdpe}:	the number of times link e occurs in path p of demand pair d of traffic type t.
δ_{tdpm}:	1 if middlebox m belongs to path p of demand pair d of traffic type t; 0, otherwise.
h_{td}:	volume of demand pair d of traffic type t.
c_e:	capacity of link e.
c_m:	capacity of middlebox m.

Variables:

x_{tdp}:	flow allocated to path p of demand pair d of traffic matrix t.
u_{tdp}:	binary variable associated with x_{tdp}.
θ_e:	utilization of link e.
θ_m:	utilization of middlebox m.
θ:	maximum utilization of links and middleboxes.

3.2.2. Formulations

The demand satisfaction constraints are shown in Eq. (1). From the candidate paths, if we choose path p to route the flow, the corresponding binary variable u_{tdp} equals 1; otherwise 0. The demand volume can be routed on only one path out of all the candidate paths, shown in Eq. (2). Here, Eqs. (3) and (4) represent link-capacity constraints and middlebox-capacity constraints, respectively. We use the variable θ to constrain link and middlebox utilization. Both link and middlebox utilizations should be no greater than 1. The goal is to minimize the maximum link or middlebox utilization, therefore achieving network load balancing. We call θ the network utilization in the following sections.

P1:
minimize θ
subject to

$$x_{tdp} = h_{td}u_{tdp}, t \in T, d \in D_t, p \in P_d. \tag{1}$$

$$\sum_p u_{tdp} = 1, t \in T, d \in D_t. \tag{2}$$

$$\sum_t \sum_d \sum_p n_{tdpe}x_{tdp} \le \theta_e c_e, e \in E. \tag{3}$$

$$\sum_t \sum_d \sum_p \delta_{tdpm}x_{tdp} \le \theta_m c_m, m \in M. \tag{4}$$

$$\theta_e \le \theta \le 1, e \in E. \tag{5}$$

$$\theta_m \le \theta \le 1, m \in M. \tag{6}$$

We substitute x_{tdp} by $h_{td}u_{tdp}$ using Eq. (1), which largely reduces the number of variables and simplifies $P1$. Also we can directly use variable θ, so the variables θ_e and θ_m are omitted. $P2$ is the same problem derived from $P1$.

P2:
minimize θ
subject to

$$\sum_p u_{tdp} = 1, t \in T, d \in D_t. \tag{7}$$

$$\sum_t \sum_d h_{td} \sum_p n_{tdpe}u_{tdp} \le \theta c_e, e \in E. \tag{8}$$

$$\sum_t \sum_d h_{td} \sum_p \delta_{tdpm}u_{tdp} \le \theta c_m, m \in M. \tag{9}$$

3.2.3. Effectiveness and extension

Nowadays, communication network demands are increasing dramatically so network resources are limited. Our LB routing approach improves network performance by balancing the entire network loads. Our approach relies on the accuracy of estimated traffic demands, which can be guaranteed using the approaches in [22–24]. By minimizing the maximum utilization, the approach is effective to balance the entire network resources well.

Other network features (cost, delay, congestion, etc.) can also be formulation objectives. We can slightly modify the load-balancing formulation to meet the new requirements. For example, we formulate a problem $P3$ to minimize network cost in a simplified case of a single traffic type.

- Problem formulation: y_e denotes load of link e. y_m denotes load of middlebox m. ξ_e represents unit cost of link e. ξ_m represents unit cost of middlebox m.
  P3:
  minimize $F = \sum_e \xi_e y_e + \sum_m \xi_m y_m$
  subject to

$$x_{dp} = h_d u_{dp}, d \in D, p \in P_d. \tag{10}$$

$$\sum_p u_{dp} = 1, d \in D. \tag{11}$$

$$\sum_d \sum_p n_{dpe}x_{dp} = y_e, e \in E. \tag{12}$$

$$y_e \le c_e, e \in E. \tag{13}$$

$$\sum_d \sum_p \delta_{dpm}x_{dp} = y_m, m \in M. \tag{14}$$

$$y_m \le c_m, m \in M. \tag{15}$$

3.3. Complexity of our approach

$P2$ is an MILP problem and cannot be solved in polynomial time. It's challenging for the controller to make routing decisions in a short time once the network topology or policy is updated. We will illustrate how to solve this problem in Section 4.

For the complexity of management, we have an estimation of the number of flow entries. The upper bounds of the number of flow entries on each switch in the simplified case of a single traffic type: $\frac{|D|(|Len|+1)\varnothing}{\#switch}$. $|Len|$ denotes the number of distinct types of middleboxes in the middlebox policy sequence of that single traffic type. $\varnothing$ represents the diameter of network. $\#switch$ indicates the number of switches in the network.

4. Solutions of the global load-balancing routing

We propose how to solve the load-balancing optimization problem $P2$ in this section. Branch-and-bound algorithm (BBA) [21] is able to find the optimal solution of this problem. However, in the optimization problem, there are $\prod_{t\in T}(\prod_i |M_i|)^{|D_t|}$ possible combinations of variables. Though BBA is an optimized algorithm, the running time grows exponentially with the number of variables. Therefore, the problem cannot be solved by BBA in a larger network. We omit the detailed discussion of BBA for simplification. We use the simulated annealing algorithm (SAN) [21] instead. After adjusting SAN's parameters, it shows a good convergence compared with optimal solutions. We introduce our parameter's selection in SAN, shown in Algorithm 1. SAN is a general optimization technique for solving combinatorial optimization problems, based on randomization techniques [25]. SAN is a heuristic algorithm and gives us an acceptably good solution; and, more importantly, it is much faster than search-based algorithms.

4.1. Realization of the SAN algorithm

Our stopping criterion is either the outer loop counter reaches K (in our case $K = 1000$), or θ_{SAN} haven't updated for 10 outer loops. Initial temperature T^0 represents the ability of jumping out of a local minimum of the algorithm. We reduce the temperature every L inner loops. $L = 200$. Since in our algorithm, the running time of computing $F(x)$ is $O((|E| + |M|)X)$, where X is the number of binary variables, the worst-case overall running time of SAN is $O(KL(|E| + |M|)X)$. This is a polynomial-time heuristic algorithm.

Algorithm 1 Simulated Annealing (SAN) Algorithm.

```
Input: A feasible solution $x$, $T \leftarrow T^0$ and $L$
Output: $\theta_{SAN}$
 1: $x^{best} \leftarrow x$, $\theta_{SAN} \leftarrow F(x^{best})$
 2: while $stopping\_criterion$ not true do
 3:     $l \leftarrow 0$
 4:     while $l < L$ do
 5:         $z \leftarrow random\_neighbor(N(x))$
 6:         $\Delta\theta \leftarrow F(z) - F(x)$
 7:         if $\Delta\theta \leq 0$ then
 8:             $x \leftarrow z$
 9:             if $F(x) < \theta_{SAN}$ then
10:                 $\theta_{SAN} \leftarrow F(x)$, $x^{best} \leftarrow x$
11:         else if $random(0, 1) < e^{-\Delta\theta/T}$ then
12:             $x \leftarrow z$
13:         $l \leftarrow l + 1$
14:     end
15:     $reduce\_temperature(T)$
16: end
```

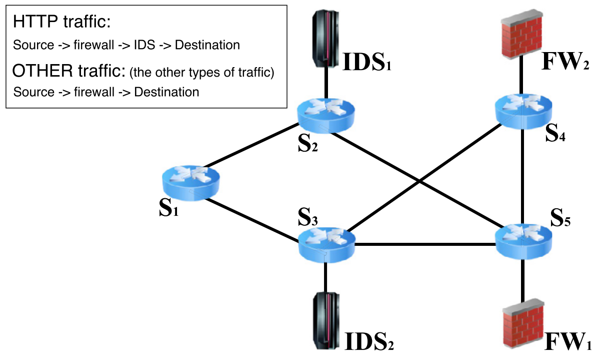

HTTP traffic:
Source -> firewall -> IDS -> Destination
OTHER traffic: (the other types of traffic)
Source -> firewall -> Destination
IDS₁
FW₂
S₂
S₄
S₁
S₃
S₅
IDS₂
FW₁

Fig. 3. SAN and BBA test network.

4.2. Algorithm test and comparison

We test the running time of these two algorithms as the number of binary variables increases. Our test topology is shown in Fig. 3. Test results are listed in Table 2. There are two types of traffic: one is HTTP traffic, which needs to route through a firewall then an IDS; the second is OTHER traffic, which only needs to route through a firewall. The number of binary variables u_{tdp}s is related to the number of demand pairs and the number of candidate paths of each demand pair. In the matrix of HTTP traffic, there are five demand pairs and each demand pair has four candidate paths; while in the matrix of OTHER traffic, there are two demand pairs and each demand pair has two candidate paths. The number of binary variables is "$5 \times 4 + 2 \times 2 = 24$." We get a group of test cases by increasing the number of demand pairs. We keep the total traffic volume in the network identical for all eight test cases. We have two observations from the results:

- The ratio grows exponentially as the number of variables increases. The ratio is defined as the BBA algorithm's actual running time divided by the SAN algorithm's actual running time.
- The SAN algorithm can always achieve a near-optimal solution. Specifically, the SAN algorithm can achieve the optimal solution θ in all eight test cases.

5. Fast local rerouting

When there are long-term changes, recalculating routing paths of the entire network is unavoidable but should be done quickly. When there is a transient disturbance (temporary failure) on the network, we can centrally recalculate the MILP problem according to the working subnetwork (global LB approach). However, this process takes more time and forces the unaffected flows to reroute. Consequently, it increases the delay, packet loss, and use of network resources. Therefore, the entire network recalculation is not a good way to handle the transient disturbance. We have already evaluated the efficiency of solving the MILP problem when dealing with long-term network changes in last section. In this section, we design a local rerouting strategy to deal with network transient disturbance. With regard to the transient disturbance, a redundant backup path is used until the disturbance has ended.

5.1. Middlebox failures

Network failures may happen on links, switches, or middleboxes. Link and switch failures are mostly studied; therefore, we only consider failures on middleboxes. We call the flows that are affected by the failures "affected flows," while flows not affected by the failures are called "unaffected flows." Middlebox failures belong to the transient disturbance, since one of the approaches of handling middlebox failures is to restart it [14]. We propose a fast-recovery mechanism to handle middlebox failures by rerouting the affected flows during the restarting period. More importantly, this mechanism is realized locally, and therefore does not disturb the unaffected flows.

5.2. Backup middlebox selection

Let's look at the example in Table 3 which lists all the candidate paths. These paths are saved before failures by solving the MILP, and thus we can find alternative paths immediately when there are failures on the middleboxes. For example, path 1 is chosen for routing a directed demand from S_1 to S_2 when no failures. If FW_1 fails, path 3 and 4 can be the alternative paths. The challenge is to quickly check whether the alternative path has enough resources to accommodate the affected flows.

5.3. Rerouting strategy analysis

We next consider a network scenario in which one middlebox fails. Flows processed by this middlebox lose some functionality and need to be routed through another middlebox with equivalent functionality during the restarting period. We work on finding a

Table 2
Test cases and results.

Results\Test case	1	2	3	4	5	6	7	8
Binary variables	24	26	28	30	32	34	42	46
Combinations of variables	4096	8192	16,384	32,768	65,536	131,072	2,097,152	8,388,608
Running time ratio (BBA / SAN)	88	157	876	974	1534	2109	12,884	23,891
θ_{BBA}^{best}	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85
θ_{SAN}	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85

Table 3
Alternative paths ($S_1 \rightarrow FW \rightarrow IDS \rightarrow S_2$ with the topology in Fig. 3).

	Virtual path	Physical path
Path 1	$S_1 \rightarrow FW_1 \rightarrow IDS_1 \rightarrow S_2$	$S_1 \rightarrow S_2 \rightarrow S_5 \rightarrow FW_1 \rightarrow S_5 \rightarrow S_2 \rightarrow IDS_1 \rightarrow S_2$
Path 2	$S_1 \rightarrow FW_1 \rightarrow IDS_2 \rightarrow S_2$	$S_1 \rightarrow S_2 \rightarrow S_5 \rightarrow FW_1 \rightarrow S_5 \rightarrow S_3 \rightarrow IDS_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_2$
Path 3	$S_1 \rightarrow FW_2 \rightarrow IDS_1 \rightarrow S_2$	$S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow FW_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_2 \rightarrow IDS_1 \rightarrow S_2$
Path 4	$S_1 \rightarrow FW_2 \rightarrow IDS_2 \rightarrow S_2$	$S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow FW_2 \rightarrow S_4 \rightarrow S_3 \rightarrow IDS_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_2$

rerouting mechanism to achieve higher speed and better performances towards failures without focusing on the routing path update process. The process is referred to as the network convergence process, during which the network responds to the failures and the network performance decreases. Routing paths will come back to the original states when the failed middlebox resumes work after restarting. To not influence the unaffected flows, we have two approaches discussed below.

5.3.1. Flow modification

One of the approaches is to assign lower priority to the affected flows than the unaffected flows. A similar approach is also discussed in [26]. That is, the switch routes the flows based on priority. The affected flows can be routed only when all unaffected flows with higher priority are delivered. This approach works well to route the unaffected flows first; however, it does not take limited network resources into consideration. This may lead to packets from the affected flows being dropped due to queuing-buffer overflow.

5.3.2. Flow accommodation

We propose a local fast-recovery mechanism without modifying the flows. The switch considers the affected and unaffected flows equally. The affected flows compete with the unaffected flows for the link bandwidth and middlebox-processing capability. Our approach is to seek an alternative path with enough bandwidth and a backup middlebox with enough processing capability to accommodate the affected flows. In this way, there will be no failure effect on the unaffected flows. Once there are several available paths (including the backup middlebox), we will choose the one which makes the entire network balanced as much as possible. We measure the network balance feature using θ and expect to choose the path with the smallest θ.

5.4. Traffic demand fluctuation

Our fast local rerouting approach is also applicable to the case where there are minor fluctuations of some flows. For instance, one flow with a minor increase might overwhelm the middleboxes and links on its routing path. We can seek for an alternative path which can accommodate this flow. It can be solved by the local rerouting approach as middlebox-failure scenarios.

6. Implementation and evaluation

In this section, we first implement our global LB approach using OpenFlow on Mininet testbed, and evaluate its effectiveness, compared with SP routing. Then we test our local rerouting approach when there are middlebox failures. We use the topology shown in Fig. 4 where the number of middleboxes is comparable to the number of nodes [5,27].

6.1. Evaluation of global LB routing

The m-by-m approach provides several choices of paths to balance network link and middlebox utilization. Each flow is assigned

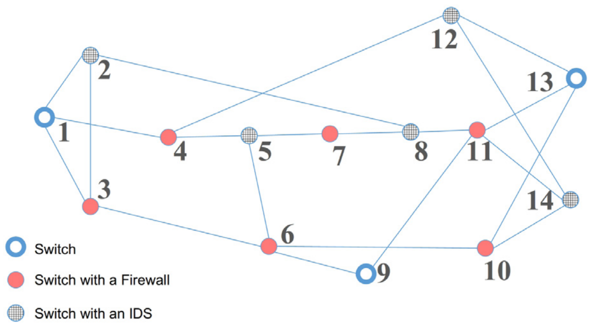


Fig. 4. Test topology.

to only one candidate path according to the LB optimization problem, and all candidate m-by-m shortest routing paths are evaluated and stored at the same time. Flow entries on the switches are installed by the centralized POX controller. 'Iperf' is used to generate traffic at constant rates and measure network performances. Normalized throughput is defined as the ratio of received packets over sent packets, that are given by 'Iperf'. End-to-end latency is measured by sending ICMP packets in addition to the regular traffic.

6.1.1. Experiment setup

In our test, there are two types of traffic: HTTP and OTHER traffic, as described in Section 4.2. Here we use traffic destined to port 5001 to denote 'HTTP' traffic, and port 5002 to denote 'OTHER' traffic. We test our approach with homogeneous and heterogeneous traffic matrices. Homogeneous traffic matrix is a demand matrix where all directed pairwise traffic demands are identical, and the data rate of each pair ranges from 1.5 Mbps to 3.6 Mbps. We have 182 directed pairwise traffic demands in total. Heterogeneous traffic matrix means all but one outgoing traffic from a source node have identical data rate, and the exceptional one has a much higher data rate than others. The exceptional node is chosen at random for each source node. For both homogeneous and heterogeneous matrix, it is the case that half of the traffic are HTTP traffic, and the other half are OTHER traffic. Link capacity is 115.0 Mbps and middlebox capacity is 93.0 Mbps. For end-to-end latency measurement, during each 10 second trial of regular traffic, default size ICMP packets with an interval of 50 ms are injected between seconds 8 and 9, so that the network has an opportunity to stabilize. The average of fifty independent trials is used for each total traffic volume.

6.1.2. Observations from experiment results

In Fig. 5, our global LB routing achieves almost the highest normalized throughput, when the total traffic volume equals 546.0 Mbps. At this point, network utilization is 1. When the total traffic volume exceeds 546.0 Mbps, network resources are no longer sufficient to accommodate all flows and the normalized throughput decreases. Our LB approach shows an increase up to 26.4% on the throughput, when compared with the SP approach, discussed in Section 3.1. Throughput varies little between homogeneous traffic and heterogeneous traffic in both approaches. We also measure overall packet losses. Since the results of normalized

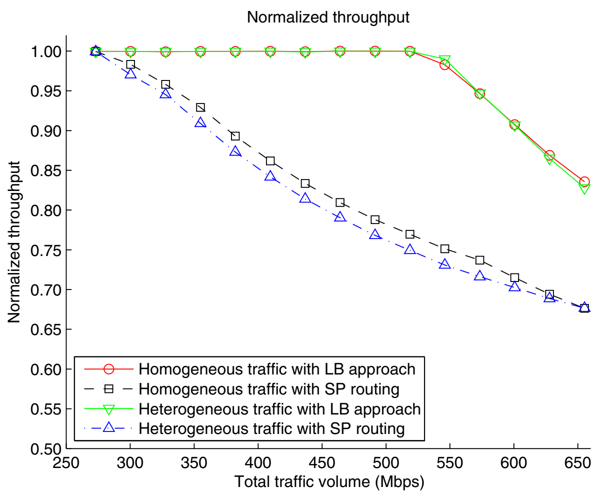


Fig. 5. Evaluation of global LB routing.

throughput and normalized packet losses are complementary, we don't show the results of overall packet loss.

In Fig. 6, the end-to-end latency and loss rate from node 1 to node 14 are shown as total traffic increases. Node 1 to node 14 is a representative node pair and the shortest path between them is the diameter of the test topology. When total traffic volume is between 273.0 Mbps and 546.0 Mbps, network resources are relatively sufficient for each demand pair and the LB approach achieves much lower end-to-end latency and loss rate than the SP routing. When the total traffic volume is greater than 546.0 Mbps, there is a bottleneck link on all possible paths from node 1 to node 14, i.e. both LB approach and SP routing are running on a congested network. Though the network is congested, the LB approach tries to balance the traffic so each flow is on a "less congested" path, while SP routing leads to "very congested" paths. This is the reason why the LB approach achieves lower end-to-end loss rate than SP routing. As expected, the latency for any paths which are not shortest, including those of the LB approach, is greater than SP routing when the entire network becomes fully congested (the total traffic exceeds 546.0 Mbps), as shown in Fig. 6.

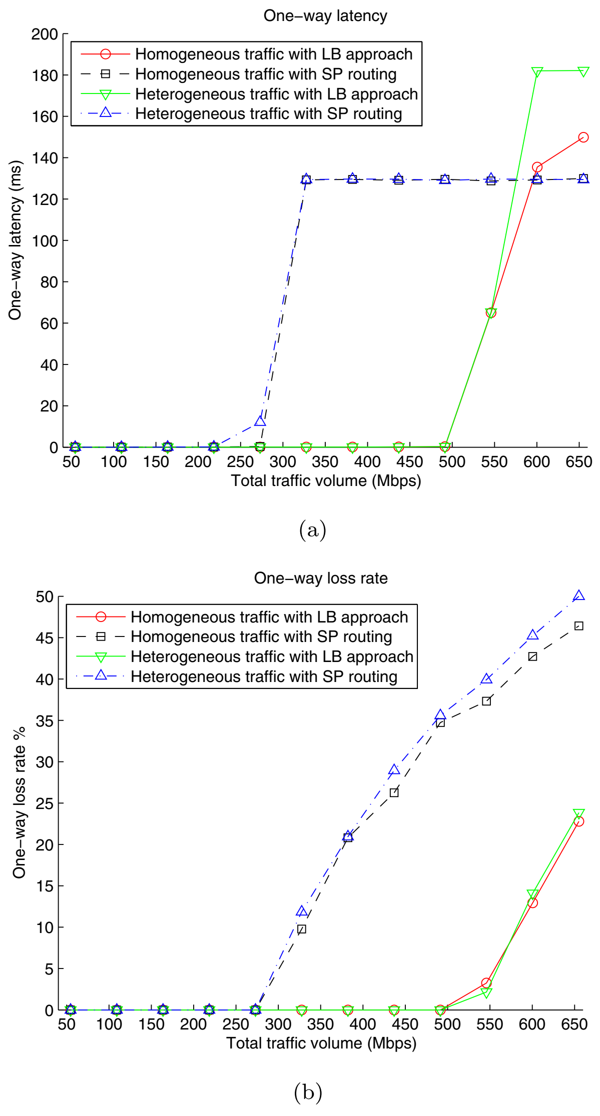


Fig. 6. End-to-end latency and loss rate.

6.2. Evaluation of fast local rerouting

Our local rerouting quickly responds to middlebox failures and has no impact on unaffected flows. When there is a failure on a middlebox, all flows routed through this middlebox need to be rerouted to another middlebox with the same functionality, introduced in Section 5.3.2. SDN flexibly allows this rerouting by installing new flow entries. There are eleven middleboxes (six firewalls and five IDSs), and our implementation includes all failure possibilities: a single failure on each middlebox, respectively.

6.2.1. Experiment setup

We measure our fast local rerouting approach, and find how many flows are reallocated with a new path and how much normalized throughput is increased when failures occur. Based on our experiment settings, firewalls act as the network bottleneck and can be considered as critical resources. IDSs are considered as non-critical resources since IDS resources are relatively sufficient. We also consider three other scenarios for comparisons: global rerouting flows, using the global LB approach proposed in Section 3, when failure occurs; dropping affected flows when failure occurs; and no middlebox failures. We test four scenarios with the homogeneous traffic matrix.

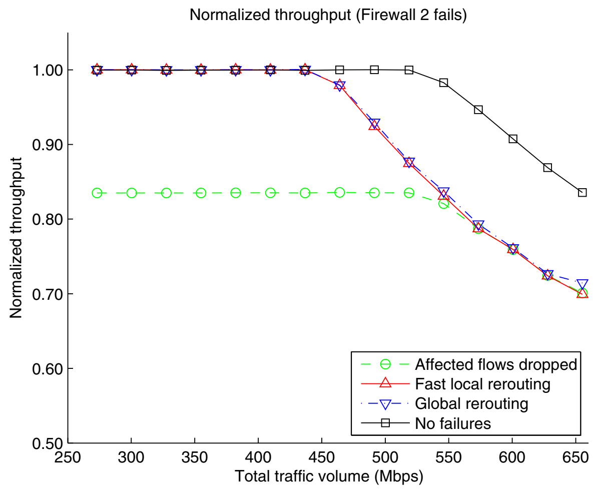
6.2.2. Major observations from experiment results

When a device with critical resources fails, e.g., FW2, our fast local rerouting requires a maximum of 21.1% flow reallocations of those required in the global rerouting approach. When a device with non-critical resources fails, e.g., IDS2, our fast local rerouting requires a maximum of 9.1% flow reallocations.

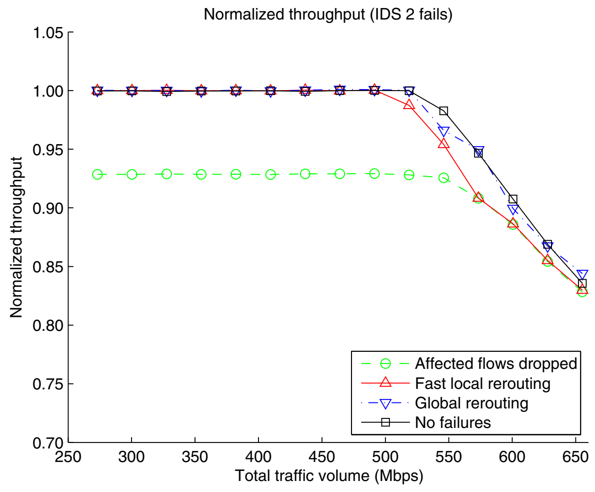
Moreover, our local rerouting approach achieves good network throughput, compared with the global rerouting approach. In Fig. 7, two subfigures represent the normalized throughput with a failure on FW2 and IDS2, as examples, respectively. When the failure occurs on the critical-resource device FW2, our fast local rerouting achieves almost the same normalized throughput. Results of our fast local rerouting approach reside between the global rerouting approach and the affected-flows-dropped approach, as expected. Our local rerouting increase the throughput up to 16.8%, compared with the affected-flows-dropped approach.

6.2.3. Further analysis on experiment results

We test all possible failures occurred on each middlebox. We obtain a group of results with the same trend on critical resource devices, namely, firewalls; and the other group of results with the same trend on non-critical resource devices, namely IDSs. Thus, we



(a)



(b)

Fig. 7. Failures on different middleboxes. Firewall 2 is connected to Switch 4, and IDS 2 is connected to Switch 5. Firewall 2 provides critical resources while IDS 2 does not.

select one failed firewall and one failed IDS, respectively, as examples in Fig. 7.

When the total traffic volume is greater than or equal to 546.0 Mbps, the local rerouting approach and the global LB approach cannot find better routing paths, since no resource is available to allow rerouting. When there are failures on IDSs with non-critical resources and total traffic volume is larger than 546.0 Mbps, global rerouting routing and fast local rerouting can still find alternative paths with IDS resources, shown in Fig. 7(b). When network resources are scarce (the total traffic volume is 655.2 Mbps), normalized throughput of any curve converges. As the global rerouting approach achieves near-optimal results, there is a shift on the normalized throughput, that is, global rerouting approach dealing with failures achieves a little bit higher throughput than the one with no failures.

Failures on middleboxes can also overwhelm the links. Let's examine the HTTP traffic. When a firewall fails, each affected flow arrives at one of the working firewalls and selects one IDS. Thus, there are $(6-1)\times 5 = 25$ working candidate paths and each affected flow chooses the best one. Near-optimal routing paths can always be found. However, an IDS's failure is different from a firewall's, as the IDS is the last-type middlebox in the policy sequence. When an IDS fails, each affected flow arrives at one working IDS and then routes along the shortest path to its destination. There are less alternative paths $(5-1=4)$; thus, some links can be more easily overloaded.

6.3. *Network topology's scalability*

We discuss network scalability to verify the practicability of our approaches. First, we narrowed the choices of candidate paths to accelerate routing path selection procedure. Then we demonstrated the upper bounds of the number of flow entries in Section 3.3. The number of flow entries on each switch scales linearly as the number of distinct types of middleboxes in a middlebox policy sequence increases. It also linearly depends on the number of demand pairs. The number of flow entries on each switch is within the switch-processing capability since $\varnothing$ and *#switch* increase simultaneously and are canceled out. Furthermore, the SAN algorithm solved that NP-complete problem efficiently. The algorithm converges fast because of its polynomial time complexity, shown in Section 4.1. In addition, our design of fast local rerouting indicated only affected flows need to be rerouted during the restarting period of the failed middlebox. This mechanism is realized locally which does not disturb the working subnetwork. This is a great feature that supports network topology's scalability.

The possible limitation of scaling the network resides in the controller. The limitation includes the difficulty of synchronized communication between the controller and all switches, controller's limited processing capability, etc. Distributed controller techniques can help reduce the burden on the controller [28]. The problem on how to exploit the benefits of centralized SDN controller and scale the controller's processing capability remains an open question in SDN researches, which is beyond the scope of our paper.

7. Conclusion

The middlebox policy makes the network routing problem more difficult. To solve this problem, we formulated an MILP optimization problem to allocate limited network resources and then solved the problem with the simulated annealing algorithm. The solution of the optimization problem indicates one path out of the candidate paths is assigned to each flow for balancing network loads. The global load-balancing routing not only balances network loads well but also keeps the number of flow entries on each switch within the range of its processing capability. Furthermore, we proposed fast local rerouting to tackle middlebox failures. The rerouting has no effect on the working subnetwork and can respond to middlebox failures quickly. With all these features implemented, we evaluated the efficiency and effectiveness of our approaches. Finally, our experiments on Mininet further validated our approach and attested the feasibility of applying our approaches to real networks. On our test topology, our LB approach shows an increase up to 26.4% on the throughput, when compared with the SP approach; and the LB approach also indicates expected end-to-end latency. Our fast local rerouting approach achieves similar results with the global rerouting approach: both approaches increase the throughput up to 16.8%, compared with the affected-flows-dropped approach. Our future work will include experiments on a real network testbed in addition to Mininet, and the design and evaluation of flow aggregation approaches.

Acknowledgment

This work was supported by the Electrical Power Affiliates Program (EPAP) at Kansas State University.

References

[1] S.K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, J.C. Mogul, Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags, in: Proc. USENIX NSDI, 2014.

[2] Z.A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, Simple-fying middlebox policy enforcement using sdn, in: ACM SIGCOMM Computer Communication Review, 43, ACM, 2013, pp. 27–38.

[3] V. Sekar, N. Egi, S. Ratnasamy, M.K. Reiter, G. Shi, Design and implementation of a consolidated middlebox architecture, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, 2012. 24–24.

[4] H. Kim, N. Feamster, Improving network management with software defined networking, Commun. Mag. IEEE 51 (2) (2013) 114–119.

[5] V. Sekar, S. Ratnasamy, M.K. Reiter, N. Egi, G. Shi, The middlebox manifesto: enabling innovation in middlebox deployment, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, ACM, 2011, p. 21.

[6] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, et al., Steering: a software-defined networking for inline service chaining, in: Network Protocols (ICNP), 2013 21st IEEE International Conference, IEEE, 2013, pp. 1–10.

[7] D.A. Joseph, A. Tavakoli, I. Stoica, A policy-aware switching layer for data centers, ACM SIGCOMM Comput. Commun. Rev. 38 (4) (2008) 51–62.

[8] S.K. Fayazbakhsh, V. Sekar, M. Yu, J.C. Mogul, Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ACM, 2013, pp. 19–24.

[9] P. Wang, J. Lan, X. Zhang, Y. Hu, S. Chen, Dynamic function composition for network service chain: model and optimization, Comput. Netw. 92 (2015) 408–418.

[10] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, ACM SIGMOBILE Mob. Comput. Commun. Rev. 5 (4) (2001) 11–25.

[11] H. Yang, L. Cheng, J. Yuan, J. Zhang, Y. Zhao, Y. Lee, Multipath protection for data center services in openflow-based software defined elastic optical networks, Opt. Fiber Technol. 23 (2015) 108–115.

[12] W. Cui, I. Stoica, R.H. Katz, Backup path allocation based on a correlated link failure probability model in overlay networks, in: Network Protocols, 2002. Proceedings. 10th IEEE International Conference, IEEE, 2002, pp. 236–245.

[13] R. Potharaju, N. Jain, Demystifying the dark side of the middle: a field study of middlebox failures in datacenters, in: Proceedings of the 2013 Conference on Internet Measurement Conference, ACM, 2013, pp. 9–22.

[14] B. Carpenter, S. Brim, Middleboxes: taxonomy and issues, Technical Report, 2002.

[15] S. Rajagopalan, D. Williams, H. Jamjoom, Pico replication: a high availability framework for middleboxes, in: Proceedings of the 4th Annual Symposium on Cloud Computing, ACM, 2013, p. 1.

[16] J. Sherry, P.X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, et al., Rollback-recovery for middleboxes, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, ACM, 2015, pp. 227–240.

[17] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 2010, p. 19.

[18] M. Caesar, M. Casado, T. Koponen, J. Rexford, S. Shenker, Dynamic route recomputation considered harmful, ACM SIGCOMM Comput. Commun. Rev. 40 (2) (2010) 66–71.

[19] H. Farhady, H. Lee, A. Nakao, Software-defined networking: a survey, Comput. Netw. 81 (2015) 79–95.

[20] A.S. Da Silva, P. Smith, A. Mauthe, A. Schaeffer-Filho, Resilience support in software-defined networking: a survey, Comput. Netw. 92 (2015) 189–207.

[21] M. Pióro, D. Medhi, Routing, Flow, and Capacity Design in Communication and Computer Networks, Elsevier, 2004.

[22] A. Nucci, A. Sridharan, N. Taft, The problem of synthetically generating ip traffic matrices: initial recommendations, ACM SIGCOMM Comput. Commun. Rev. 35 (3) (2005) 19–32.

[23] M. Roughan, M. Thorup, Y. Zhang, Traffic engineering with estimated traffic matrices, in: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, ACM, 2003, pp. 248–258.

[24] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, F. True, Deriving traffic demands for operational ip networks: methodology and experience, in: ACM SIGCOMM Computer Communication Review, 30, ACM, 2000, pp. 257–270.

[25] P.J. Van Laarhoven, E.H. Aarts, Simulated Annealing: Theory and Applications, 37, Springer Science & Business Media, 1987.

[26] D. Li, Y. Shang, C. Chen, Software defined green data center network with exclusive routing, in: INFOCOM, 2014 Proceedings IEEE, IEEE, 2014, pp. 1743–1751.

[27] N. Jain, R. Potharaju, Middlebox reliability, 2012, (US Patent App. 13/536,782).

[28] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in sdn-openflow networks, Comput. Netw. 71 (2014) 1–30.

Xin Li received her B.S. degree in electronics and information engineering from Beihang University, China, in 2012. She is currently pursuing the PhD degree in electrical and computer engineering at Kansas State Unversity. Her research interests include software defined networking, security applications for smart grids and network optimization.

Haotian Wu received his B.S. degree in electronics and information engineering at Beihang University, China, in 2012. He is currently pursuing the PhD degree in electrical and computer engineering at Kansas State Unversity. He is interested in programmable applications with SDN, network security and complex networks.

Don Gruenbacher received the PhD degree in electrical engineering from Kansas State University. He is an associate professor and the department head in the Department of Electrical and Computer Engineering at Kansas State University. He is interested in software defined networking, secure applications for smart grids, and intrusion detection.

Caterina Scoglio is a professor in the Department of Electrical and Computer Engineering at Kansas State University since 2005. Her main research interests include software defined networking, the modeling and analysis of complex networks, and applications in epidemic spreading and power grids. She received the Dr. Eng. degree from the "Sapienza" Rome University, Italy, in 1987. Before joining Kansas State University, she worked at the Fondazione Ugo Bordoni from 1987 to 2000, and at the Georgia Institute of Technology from 2000 to 2005.

Tricha Anjali received her Integrated M.Tech. (EE) from Indian Institute of Technology, Mumbai in 1998, and Ph.D. from Georgia Institute of Technology in Atlanta in 2004. From 2004 to 2010, she was an assistant professor in the Department of Electrical and Computer Engineering at Illinois Institute of Technology. She was promoted to associate professor in the same department in 2010. She is currently an associate professor at International Institute of Information Technology, India. Her research interests are in the area of wireless mesh networks, multipath routing, heterogeneous networks and network optimization.