Proactive Rerouting in Network Overlays

Reuven Cohen   Yuval Dagan   Gabi Nakibly
Dept. of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel

***Abstract*—Virtual overlay network technology provides important benefits to large data centers and to service providers. These benefits include traffic isolation and ease of service provisioning. When the underlying network supports traffic engineering, tunneling brings another important benefit: the ability to control the exact route of all packets without handling each independent flow. This paper addresses the problem of rerouting when the core network supports traffic engineering. We introduce the novel concept of proactive (time-driven) rerouting, which we distinguish from the well-known concept of reactive (event-driven) rerouting. One important advantage of proactive rerouting is reducing the communication between the core network controller and the edge network controller. Another advantage is that new flows do not have to wait before they are admitted into a rerouted tunnel. Unlike a reactive rerouting algorithm that knows which tunnel has to be rerouted, a proactive rerouting algorithm does not receive as an input the identity of a specific tunnel. Thus, its main goal is to predict which tunnel to reroute in order to increase the probability that future flows will be accommodated. Our main contribution is the development of a proactive rerouting algorithm that performs very well, sometimes even better than the reactive algorithms.**

I. Introduction

Software defined networking (SDN) and network virtualization use tunneling as a means of communication. Tunneling is used in the building of virtual overlay networks, which are known to better support virtual machine (VM) provisioning, to enable scalability and to improve automation. Virtual overlay network technology provides benefits to large data centers, the most important of which is traffic isolation for multi-tenancy. Another important benefit is ease of VM provisioning, because a VM can be migrated to a new subnet without changing its IP address or other network-dependent attributes. However, when the underlying network is based on a virtual circuit technology, such as MPLS, tunneling brings yet another important benefit: the ability to control the exact routes of the data packet, a process also known as traffic engineering, *without handling each independent flow*.

In a network overlay, also known as "overlay SDN," a packet is encapsulated inside another packet. The encapsulated packet is then forwarded along the route determined by the encapsulating header, until it reaches the end of tunnel, where it is de-encapsulated. Many different tunneling protocols are used today, including MPLS, VXLAN, NVGRE, STT and NVO3. The idea is that tunnels are built along routes with available bandwidth and flows are routed over these tunnels while taking advantage of the network resources reserved for their tunnels. The combination of SDN with MPLS has been identified by [2] as the best way to bring SDN into the carrier networks.

This research was partially funded by the Office of the Chief Scientist of the Israel Ministry of Economy under the Neptune generic research project. Neptune is the Israeli consortium for network programming.

ISBN 978-3-903176-08-9 © 2018 IFIP

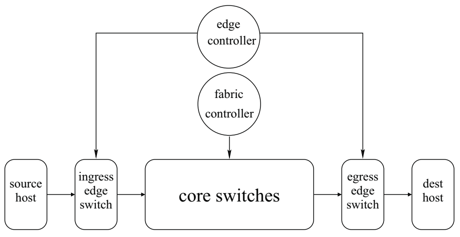


Fig. 1.   The considered network architecture, as proposed by [2]

Figure 1 shows the three components of the considered network as described in [2]: hosts, edge switches, and the core fabric. The core and the edge are controlled by separate controllers: the edge controller handles the interface between the operator and the network, whereas the core controller is responsible for building tunnels and allocating resources to them.

The core network controller is responsible for creating new tunnels, each of which may contain thousands of flows at any given time. Once a tunnel is ready, the admission of new flows into it is the responsibility of the edge controller. These two controllers sometimes need to interact, as they do, for example, when the edge controller wants to admit a new flow but there is no tunnel with enough available bandwidth. The interaction between the two controllers can be a bottleneck in the network. We therefore seek to minimize it by requesting that the core controller maintain the tunnels without receiving explicit requests from the edge controller. This is the rationale behind the proactive algorithm presented later on.

Consider a pair of nodes (s, d). Let the tunnel between them be $t(s, d)$. A default tunnel is usually established over the shortest path with sufficient resources between s and d (i.e., the shortest path after ignoring the links without sufficient resources), but it can be established over any other path as well. When $t(s, d)$ runs out of resources, the edge controller cannot admit new flows into it. This controller communicates with the core network controller and requests that additional resources be allocated to this tunnel. If additional resources cannot be allocated, the only way to admit additional flows is either by building a new tunnel or by moving the tunnel to a new path, a process also known as *rerouting.* The advantage to rerouting of tunnels is that all the already admitted flows are rerouted together with the tunnel, with no additional per-flow overhead [7].

Rerouting is preferable over creating additional tunnels for several reasons:

1) Every tunnel requires expensive forwarding entries in the network switches.
2) Every tunnel needs to be protected against failures [6].
3) With many tunnels between every pair of nodes, bandwidth utilization decreases due to a lower multiplexing ratio (a single 10Gb/s tunnel can be better utilized than 10 1Gb/s tunnels).

This paper addresses the problem of tunnel management and rerouting when the core network controller can determine the exact route over which every tunnel is established. Rerouting schemes have been extensively studied in the context of virtual circuit technologies, such as ATM, WDM, and MPLS. While most works on rerouting have focused on restoration following a link or node failure [7], [6], [19], [22], there is also extensive work on rerouting for throughput maximization[10], [27], [16], [25], [3], [14], [26]. The main difference between these works and the present work is that they all assume exact knowledge of the flows introduced into the network and of the flows that cannot be accommodated. Thus, they are all *reactive*. In contrast, the SDN core controller is not directly aware of the flows introduced into the network. Thus, it does not know about specific routing failure events. Such a controller can therefore use only proactive rerouting algorithms, which require no exact knowledge about how current flows are routed.

An important advantage of proactive rerouting is that *it requires no communication between the two controllers.* Another important advantage is that new flows do not have to wait for rerouting before they are admitted into a tunnel, because it alleviates congestion in hot-spots before the appearance of new flows.

Our key contribution is the distinction between reactive (event-driven) and proactive (time-driven) reroutings. With reactive rerouting, a tunnel may be rerouted by the core network controller only after the edge controller fails to admit a new flow into the network due to lack of bandwidth in the existing (default) tunnel between the corresponding end nodes. With proactive rerouting, the core controller is periodically invoked in order to replace the tunnels in the network such that the likelihood that the edge controller will fail to admit future data flows is minimized. We present algorithms for each model, in order to find the one that yields the best trade-off between the number of reroutings and the core network throughput.

The rest of the paper is organized as follows. Section III describes an algorithm for proactive rerouting. Section IV describes several algorithms for reactive rerouting. Section V presents simulation results for the various algorithms. Finally, Section VI concludes the paper.

II. Related Work

This paper is largely motivated by recent works that show the potential of combining SDN with MPLS. In [2], the authors suggest that a network fabric should be included as an architectural building block within SDN. They also identify the key properties for these fabrics: separation of forwarding and separation of control. They suggest that this separation would require an "edge" version of OpenFlow, which is much more general than the legacy OpenFlow, and a "core" version of OpenFlow, which resembles a slightly expanded version of MPLS label-based forwarding.

The benefit from using an overlay SDN is also discussed in [12]. The authors indicate that SDN has been successfully applied to data centers and campus networks but it has had little impact in the fixed wireline and mobile Telecom domain. They propose using "vertical forwarding" (tunneling) for extending SDN so that it can tackle the challenges of the Telecom domain. They also claim that tunneling enables flow-based policy enforcement, mobility and security.

Tunnel rerouting has been explored mainly in the context of virtual circuit technologies such as ATM [5], [7], MPLS [6] and WDM [21], [18]. It is usually used either when the original tunnel fails or does not have sufficient bandwidth for new flows. In [20], a "fast reroute" scheme is proposed in the context of MPLS. The main idea is to build a predefined bypass for each switch or link along the tunnel. When a switch learns that its upstream link or upstream neighbor on a given tunnel has failed, this switch can immediately forward the traffic along the pre-established bypass. The main advantage of this scheme is its very fast reaction to failures, thereby minimizing packet loss. However, this scheme is expensive from a management perspective, because it requires many bypasses for each tunnel, and from a bandwidth perspective, because bandwidth must be reserved in advance for each bypass.

In [6], the bandwidth cost of fast reroute was compared to the bandwidth cost of other rerouting schemes. This paper presents a comprehensive study of restorable throughput maximization in MPLS networks. One of its conclusions is that if the goal is to maximize revenue, fast reroute (referred to as "local recovery" in [6]) should be the recovery scheme of choice.

In [4], the authors study four rerouting algorithms to determine how the characteristics of the underlying network topology might affect their performance. They found that when the average node degree is small, most common practices for route placements, such as the shortest path algorithm, yield good performance in terms of the blocking ratio and that there is probably little advantage to rerouting. But when the average node degree increases, so does the number of available paths, and rerouting tends to improve the performance.

A recent line of research deals with rerouting of connections in elastic optical networks to alleviate bandwidth fragmentation [28], [24], [29]. Elastic optical networks allocate spectrum based on contiguous subcarrier slots with bandwidth. In such networks, dynamic setup and tear-down of connections can create bandwidth fragmentation, namely, non-contiguous slots that are not aligned along the routing paths and therefore cannot be used by new connections. In such networks, rerouting is needed to decrease the level of fragmentation and reduce the blocking probability of new requests. In [28], for example, a proactive rerouting scheme is proposed based on the state of the network links. However, rerouting in elastic optical networks have different constraints and objective function compared to the SDN case we deal with. In particular, in SDN there is no bandwidth fragmentation.

Our paper focuses on tunnel rerouting due to lack of bandwidth over the original path. Therefore, the main theoretical problem is to find an alternative path with sufficient bandwidth. Sometimes rerouting a single tunnel will not suffice, and more tunnels need to be rerouted together. In such a case, the rerouting problem is computationally equivalent to the well-known NP-hard unsplittable multicommodity flow problem [9], [8], [11], [13], [15].

III. Proactive Rerouting

In proactive rerouting, the core controller is periodically invoked in order to reroute tunnels in a way that gives the edge controller more flexibility in accommodating future flows. The core network controller has no idea about future demands, not even their statistical distribution. We present the Network State Algorithm, which associates a "cost" with every link and takes the sum of the costs of all links as the network cost. The algorithm seeks to find a tunnel and a new path for this tunnel such that the network cost after rerouting the tunnel to the new path is minimized. The cost of every link reflects the load and the available bandwidth on it.

Let $G = (V, E)$ be a directed graph representing the considered core network. There is a default tunnel $t(s, d)$ that should accommodate the flows from s to d. Let F be a set of traffic flows to be admitted into the network one at a time, in an online fashion. When a flow is introduced, its termination time is unknown and future flows are also unknown.

A high level description of the algorithm is as follows. Note that each link has a cost attribute and a different weight attribute:

1) For every tunnel $t(s, d)$
   (a) Remove the tunnel from the network.
   (b) Assign a weight to every link. This weight is used only for finding a new shortest path for t. The weight of each link e is the cost of this link if t is routed over e minus the cost of the link if t is not routed over e.
   (c) Run a shortest path algorithm between s and d on the graph with the weights determined in the previous step.
2) Choose for rerouting the tunnel whose new route imposes a minimum total network cost.

We now explain how the cost of each link is determined. For every link e, define

- $\text{cap}(e)$ as the link capacity (in Mb/s, say);
- $\text{used}(e)$ as the capacity used by tunnels that are routed over e (also in Mb/s);
- $\text{load}(e)$ as $\text{cap}(e)/\text{used}(e)$.

The cost of link e depends on $\text{load}(e)$ using the following relationship:

$$c(e) = \frac{1}{\theta}\left(\exp(\theta \cdot \text{load}(e)) - 1\right), \qquad (1)$$

where $\theta > 0$ is a free parameter that determines how quickly the cost increases with the load. Specifically, when θ is close to 0, the relationship is linear, and when θ increases, the cost growth is much faster, as depicted in Figure 2. The (-1) in Eq. 1 makes no algorithmic difference, because adding a constant to

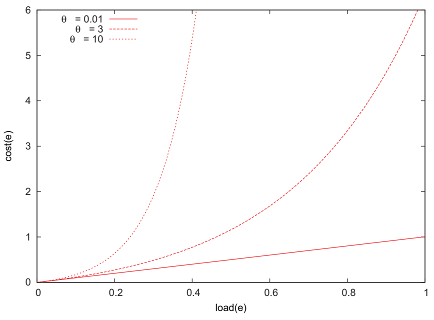


Fig. 2. The correlation between the cost and the load as a function of θ

```
Function RerouteOne()
    for every tunnel $t \in T$ do
        for every link $e \in E$ do
            if $e$ can accommodate $t$ then
                $c(e)_{+t} \leftarrow$ the cost of $e$ if $t$ is rerouted to a path that contains $e$
                $c(e)_{-t} \leftarrow$ the cost of $e$ if $t$ is rerouted to a path that does not contain $e$
                $w_t(e) \leftarrow c(e)_{+t} - c(e)_{-t}$
            else
                $w_t(e) \leftarrow \infty$
            end
        end
        $p_t \leftarrow$ the shortest path under the weight $w_t$
        $r_t \leftarrow$ the difference between weight of $p_t$ and the weight of the old path of $t$, under the weight function $w_t$
    end
    Reroute the tunnel $t^*$ into $p_{t^*}$, where $t^*$ minimizes $r_t$ over all tunnels $t$

Function RerouteMany()
    for $i = 1, \ldots, num\_reroutes$ do
        RerouteOne()
    end
```

Fig. 3. A formal description of the Network State Algorithm (NSA)

the cost does not change the identity of the selected reroute. It just guarantees that the cost equals zero if the link is unused.

Recall that the cost of the network is $\sum_{e \in E} \text{c}(e)$. Therefore, if θ is close to zero, the network cost equals the sum of the loads of all links in the network, whereas if θ is very big ($\theta > 1000$), the cost of the network is approximately equal to the cost of the most congested link.

Figure 3 gives a formal description of the algorithm. In this figure, $w_t(e)$ is the weight of link e considered by the shortest path in Step 1(c) of the informal description.

The cost function c gets a load level $\ell \in [0, 1]$ and returns the cost associated with this load. This can be expressed formally as follows:

$$c(\ell) = \frac{1}{\theta}\left(\exp(\theta\ell) - 1\right).$$

In the following discussion we use c interchangeably as a function that receives a link and as a function that receives a load.

The derivative $c'(\ell)$ indicates how costly it is to increase the load of a link whose current load is ℓ. Increasing the load from ℓ to $\ell + b$ increases the cost by

$$c(\ell + b) - c(\ell) = \int_{x=\ell}^{\ell+b} c'(\ell) du.$$

A calculation shows that $c'(\ell) = \exp(\theta \ell)$.

When θ is very small, the derivative is constant for any value of ℓ. Thus, the cost of increasing the load of a link from ℓ to $\ell+b$ is not affected by the link's current load ℓ. Therefore, the rerouting algorithm ignores the current loads on the links.

When θ is big, $c'(\ell)$ is much larger for big values of ℓ than for small values. Thus, it is much more costly to increase the load of an already congested link than it is to increase the load of a non-congested link. Generally, increasing the value of θ makes it more costly to increase the load of already congested links than of non-congested links.

Given a tunnel $t(s, d)$ whose bandwidth demand is demand(t), we now analyze the weight $w_t(e)$ assigned to e in order to find a shortest path from s to d when t is to be rerouted. We use the notations from the formal definition of the algorithm (Figure 3).

First, assume that θ is very small, namely, $c(\ell) \approx \ell$. For a link e, let load$(e)_{-t}$ be the load on e if t is rerouted to a path that does not contain e, and load$(e)_{+t}$ be the load on e if we reroute t to a path that contains e. It holds that

$$\begin{aligned} w_t(e) = c(e)_{+t} - c(e)_{-t} &= c(\text{load}(e)_{+t}) - c(\text{load}(e)_{-t}) \\ &\approx \text{load}(e)_{+t} - \text{load}(e)_{-t} = \frac{\text{demand}(t)}{\text{cap}(e)}. \end{aligned}$$

Thus, the shortest path for rerouting t is the path p from s to d for which

$$\sum_{e \in p} \frac{1}{\text{cap}(e)}$$

is minimized.

Next, assume that θ is very big; thus, for most values of $\ell_1 > l_2$ it holds that $c(\ell_1) >> c(\ell_2)$. The weight assigned to any link e is $c(e)_{+t} - c(e)_{-t} \approx c(e)_{+t}$. Thus, the shortest path for rerouting t is the path p from s to d for which

$$\sum_{e \in p} c(e)_{+t} \approx \max_{e \in p} c(e)_{+t} = \max_{e \in p} c\left(\text{load}(e)_{+t}\right)$$

is minimized.

The shortest path is the one that minimizes

$$\max_{e \in p} \text{load}(e)_{+t}.$$

Increasing the value of θ results in choosing longer rerouting paths, because a large number of low-load links can be added to the path without significantly affecting the cost. A simple example for this is given in Figure 4. In this figure, the numbers on the links indicate their capacities. Assume that the network has one active tunnel, from v_1 to v_3, whose bandwidth demand is 1. Assume that this tunnel is currently established over the path $v_1 \rightarrow v_3$. The controller is invoked for performing one reroute. The controller can actually leave the tunnel on its current route or move it to $v_1 \rightarrow v_2 \rightarrow v_3$. For $x = 3$, simple mathematical analysis reveals that if $\theta < 2.887$, the Network State Algorithm will leave the tunnel on its current path, and if $\theta > 2.887$ the algorithm will move the tunnel to the longer path $v_1 \rightarrow v_2 \rightarrow v_3$. For $x = 3.5$, the θ threshold decreases to ≈ 1.159, and for $x > 4$ the threshold is 0, namely, the algorithm always chooses the long path.

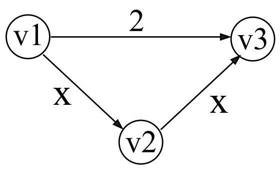


Fig. 4. An example of a network. The numbers indicate the capacity of each link.

IV. Reactive Rerouting

The purpose of this section is two-fold. First, we classify reactive rerouting into several schemes, depending on how much freedom is given to the controller during the rerouting process. Second, we present efficient and simple algorithm for each scheme. These algorithms are later used for comparing between the performance of reactive and proactive rerouting.

When the edge controller is unable to admit a new flow f into the network due to lack of bandwidth in the appropriate tunnel, it contacts the core network controller and requests to increase the bandwidth of the tunnel. If there is not enough spare bandwidth that can be added to the tunnel, the core network controller can invoke a rerouting algorithm. We will study the following rerouting schemes:

Scheme-A: The flow is rejected (rerouting is not performed). This scheme will be used as a benchmark.

Scheme-B: The default tunnel $t(s, d)$ of the considered flow, f, is rerouted to a new path that has sufficient bandwidth for all the existing flows that use this tunnel as well as for the new flow f.

Scheme-C: Any single tunnel from T can be rerouted in order to admit the new flow f, not necessarily $t(s, d)$.

Scheme-D: Up to N tunnels from T can be rerouted in order to admit the new flow.

Scheme-C is a generalization of Scheme-B, because for a given network state, every solution found by Scheme-B can also be found by Scheme-C. However, Scheme-C may reroute tunnels that are not related to the new flow, nor, in particular, to the same customer. Thus, not every network operator will prefer Scheme-C over Scheme-B. In the same way, Scheme-D is a generalization of Scheme-C.

A. Shortest-Path Algorithms

This subsection presents efficient (polynomial time) online algorithms for Scheme-B and Scheme-C, referred to as SP(B) and SP(C). For a given new flow, these algorithms are optimal in the sense that if a reroute exists, they will find it. For Scheme-D, an efficient algorithm that accommodates the new flow, if possible, while minimizing the number of rerouted tunnels, is unlikely to exist because this problem is NP-hard. This can be easily shown using a reduction to the well-known NP-hard unsplittable multicommodity flow problem [9], [8], [11], [13], [15]. Thus, the shortest path version for Scheme-D, referred to as SP(D), is only heuristic.

In all the following algorithms, let f' be a new flow that cannot be admitted into its default tunnel t'.

SP(B) Consider $G(V, E)$ while excluding the bandwidth used by all the tunnels except t' (none of this bandwidth can be used for the to-be-rerouted tunnel t'). From this graph, also excluded are the links whose available bandwidth is less than what is required to accommodate all the current flows of t' and the new flow f'. On the residual graph, if s and d are connected, the shortest path is chosen for t'. If s and d are not connected, f' is rejected because t' cannot be rerouted by Scheme-B.

SP(C) For each $t \in T$, where T is the set of tunnels, consider $G(V, E)$ while excluding the bandwidth used by all the tunnels except t. Try to accommodate the new flow f' along tunnel t'. If this is possible, try to accommodate t over the shortest path while considering only links whose residual bandwidth is sufficiently large. If f' cannot be accommodated or if t cannot be accommodated after f' is accommodated, repeat the procedure with another $t \in T$. If the procedure fails for all ts, reject f'.

SP(D) A similar procedure to $SP(C)$ is used. However, if a single reroute is insufficient, the tunnel whose rerouting maximizes the minimum available bandwidth over t' is chosen for rerouting. This can be stated formally as follows: for every $e \in E$, let avail(e) be the available bandwidth in link e. The tunnel chosen for rerouting is either t' or any other tunnel whose rerouting maximizes $\min_{e \in \text{path}(t')}$ avail(e). This procedure is repeated until flow f' can be admitted into t', but no more than N times. If f' cannot be admitted after N reroutings then no tunnel is actually rerouted. The pseudocode of this algorithm is as follows:

```
Run algorithm SP(C)
for $i = 1, \ldots, N$ do
    if $f'$ can be admitted into $t'$ then
        stop
    end
    find the tunnel $t \in T$ whose rerouting maximizes the
      minimum available bandwidth along $t'$, and reroute
      it (the actual rerouting will be performed only if we
      succeed in admitting $f'$)
end
No rerouting is performed and flow $f'$ cannot be
  admitted into tunnel $t'$
```

B. A Linear Program Algorithm for Scheme-D

Since SP(D) does not guarantee an optimal solution to Scheme-D, we present here another algorithm for this scheme. This is an integer linear program algorithm, referred to as LP(D), whose main purpose is to serve as a benchmark for SP(D).

Let $F' \subset F$ be the set of flows admitted so far into the network. Recall that f' is a new flow that cannot be admitted into its default tunnel t'. The following LP parameters are defined:

- y_{te} – indicates whether tunnel t is currently routed over link e, $\forall t \in T$.
- b_t – denotes the bandwidth routed on tunnel t, while $b_{t'}$ already includes the demand of f'.

The following LP variables are defined:

- y'_{te} – indicates whether tunnel t will be routed over link e after the current iteration, $\forall t \in T$
- r_t – indicates whether tunnel t is rerouted.

The target function is to minimize the number of rerouted tunnels, namely, to minimize $\sum_t r_t$, subject to several sets of constraints. The first set of constraints ensures flow conservation. The second set ensures that no link e carries more than its capacity cap(e). The third set ensures that the new path of each tunnel is identical to its old path unless this tunnel is rerouted. The fourth set ensures that at most N tunnels are rerouted, and the fifth set ensures that each flow is rerouted in only one path.

(1) $\sum_{e=(u,v)} y'_{te} - \sum_{e=(v,u)} y'_{te} = \begin{cases} -1 & v = s \\ 1 & v = d \\ 0 & \text{else} \end{cases}$
$\forall v \in V, \forall t \in T,$ where s and d are the tunnel source and destination.

(2) $\sum_t b_t \cdot y'_{te} \leq \text{cap}(e) \quad \forall e \in E$

(3) $y'_{te} + r_t \geq y_{te} \quad \forall$ tunnel $t \in T$

(4) $\sum_t r_t \leq N$

(5) $\sum_{e=(v,u)} y'_{te} = 1 \quad \forall t \in T, v = s$ (the source of t)

(6) $y'_{te} \in \{0, 1\} \quad \forall e \in E \ \forall t \in T$
$r_t \in \{0, 1\} \quad \forall t \in T$

V. Simulation Study

In Section V-A we evaluate the performance of the various reactive algorithms and in Section V-B we compare them to the proactive algorithm. Two criteria are relevant for this comparison:

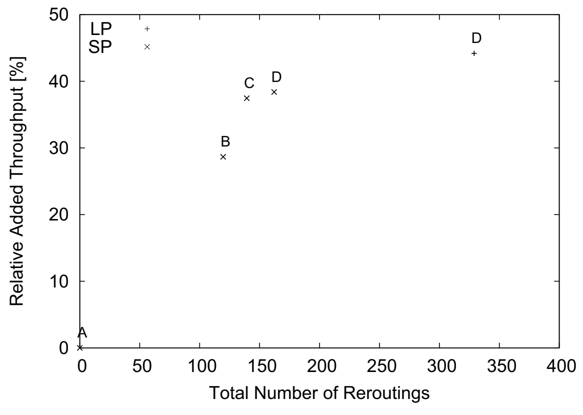
1) Relative added throughput, namely, how much greater the percent of throughput that each algorithm admits into the network compared to the case where no rerouting is allowed (Scheme-A). Formally, if a rerouting algorithm admits B Mb/s while only B' Mb/s is admitted without rerouting, then the relative added throughput for this algorithm is $(B - B')/B$. The relative added throughput is indicated in the y-axis of all the graphs presented in this section.
2) Management burden, defined as the total number of rerouting events. This number is indicated in the x-axis of all the graphs presented in this section.

Scheme-A gets no further mention because for this scheme the value of the y-axis and the value of the x-axis are always 0, by definition.

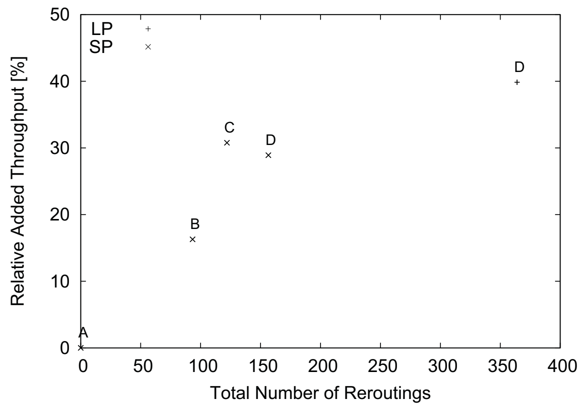
We expect to see some cases where the increase in relative added throughput is due to a substantial increase in the number of rerouting events. Thus, to get a good understanding on how each algorithm really performs with respect to the tradeoff between throughput and number of rerouting events, we always consider both criteria together.

A. Reactive Rerouting Simulations

We first study the performance of the reactive algorithms. Since SP(B) and SP(C) are optimal online algorithms for their schemes (when flows are received and considered one by



(a) Offered load = 0.5



(b) Offered load = 1

Fig. 5. The performance of the reactive algorithms SP(B), SP(C), SP(D) and LP(D) for small networks

one) while SP(D) is not necessarily optimal for its scheme, the LP(D) algorithm for Scheme-D is also simulated as a benchmark but, due to its computational complexity, this is done only for small-sized networks. Later, the results for larger networks will also be presented, but without LP(D).

Figure 5 shows the simulation results for the small scale networks. In this figure, 5 artificially generated topologies are considered, each with 15 nodes and 30 links. These topologies are generated using the BRITE simulator [1], which captures two important characteristics of network topologies: incremental growth and preferential connectivity of a new node to well-connected existing nodes. These characteristics yield a power law distribution to the degrees of the nodes. For each topology, 5 sequences of flows are generated, each with 1,000 requests. Each request is either for initiating a new flow or removing an existing flow. When a new flow is introduced, the algorithm is executed and the flow is either rejected or admitted following 0 or more rerouting events. The flow sequences are generated using the gravity model [17].

Figure 5 shows the simulation results for two levels of offered load: 0.5 and 1. The offered load of a network at a given time τ is defined as the sum of the offered loads of all the flows introduced to the network before τ (and that were accepted or rejected) whose termination time is after τ, divided by the total capacity of the network links. The offered load of a flow is the bandwidth demand of the flow multiplied by the number of links on the shortest path (while assigning a weight of 1 to every link) between the flow's source and destination. The flow sequences are generated for each simulation such that the network load remains roughly constant, and this yields one point on our graph, after it is averaged with additional executions using the same set of parameters but with a different seed.

It is evident from both graphs of Figure 5 that tunnel rerouting *significantly* increases the accommodated bandwidth. As expected, the schemes that accommodate more bandwidth impose a greater rerouting burden. In absolute numbers, more bandwidth is accommodated in heavily loaded networks. But the relative added throughput decreases when the load on the network increases because, in a heavily loaded network, there are fewer options for rerouting to gain additional throughput. This behavior is particularly noticeable for Scheme-B, because in this scheme only one tunnel can be rerouted.

For Scheme-D, the LP algorithm performs significantly more reroutings compared to SP(D), with only moderate increase in the admitted throughput. This is due to the fact that LP(D) is completely flexible in choosing the tunnels to reroute. In contrast, SP(D) is limited in the selection of a new path to rerouted tunnels.

To compare the performance of the various schemes, SP(B), SP(C) and SP(D), we define the "benefit-cost ratio" of an algorithm as the fraction obtained by dividing the relative added throughput by the number of reroutings, y/x. Using this ratio, SP(C) is the best algorithm, and SP(D) is better than LP(D).

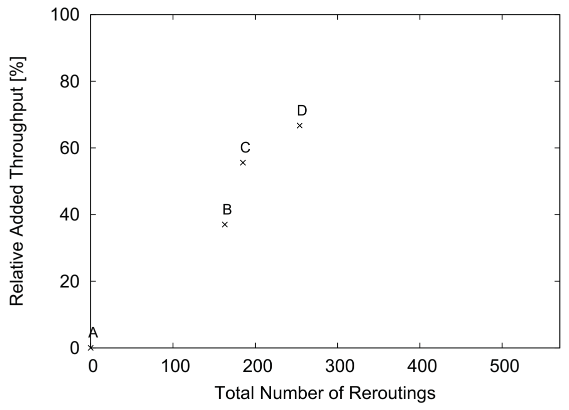
Figure 6 depicts the results for large networks. The networks simulated in Figure 6(a), (b) and (c) are synthetic networks built using BRITE with 40 nodes and 80 links, 40 nodes and 160 links, and 80 nodes and 320 links respectively. The network simulated in Figure 6(d) is a RocketFuel inferred topology[23] with 138 nodes and 730 links. LP(D) is not executed due to its exponential running time. For all these graphs, the offered load ratio is 0.5.

It is evident that the bandwidth accommodated by the algorithms, as well as the rerouting overhead, grow with the network size: as the network grows, there are more tunnel rerouting options. It is interesting to note that the various algorithms in the RocketFuel topology perform worse than they do for the synthetic topologies, despite the RocketFuel topology having more node links. But another property of this topology is that it has a few nodes with very large degree. These nodes create hotspots for which rerouting is less effective.

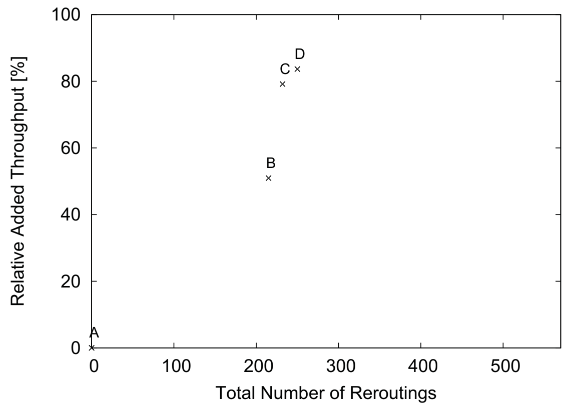
The results in Figure 6 strengthen our earlier finding that SP(C) yields a better benefit-cost ratio than SP(B) and SP(D). In other words, it results in a better tradeoff between the relative added throughput and the cost of rerouting.

B. Proactive Rerouting Simulations

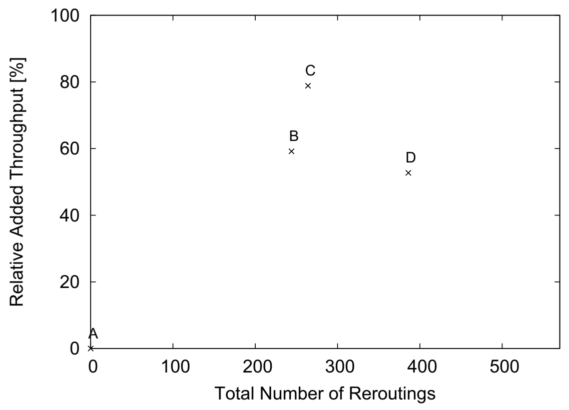
This subsection compares the proactive approach and the reactive approach, by comparing the results of the three SP algorithms – SP(B), SP(C) and SP(D) – to those of the Network



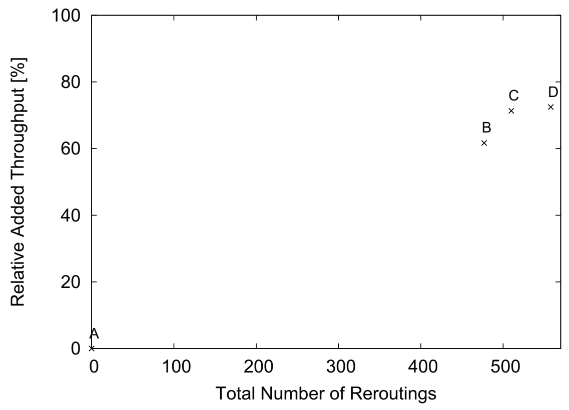
(a) $|V| = 40, |E|/|V| = 2$



(b) $|V| = 40, |E|/|V| = 4$



(c) $|V| = 80, |E|/|V| = 4$



(d) RocketFuel Topology, $|V| = 138, |E|/|V| = 5.3$

Fig. 6. The performance of the reactive algorithms SP(B), SP(C) and SP(D) for large networks

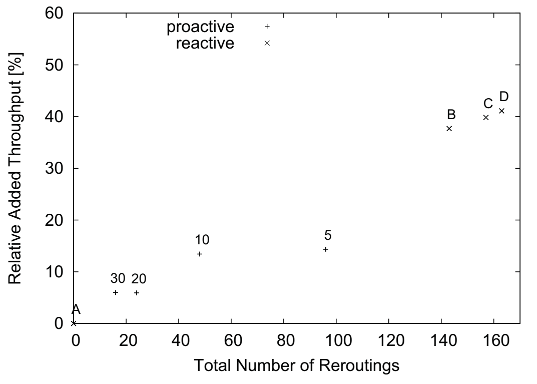
State Algorithm. Consider first Figure 7. The figure shows 4 graphs, for the same networks considered in the discussion of Figure 6. For each network type, the reactive SP algorithm is simulated for schemes B, C and D. Also simulated is the proactive Network State Algorithm for different execution periods: 5, 10, 20 and 40. The *execution period* is the average number of flows arriving between two consecutive invocations of the algorithm. Unless otherwise specified, for all these graphs the offered load is 0.25 and $\theta = 10$.

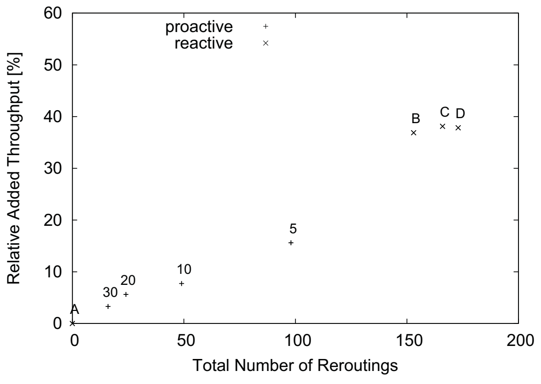
One cannot expect the proactive algorithm to admit as many flows as the reactive algorithms for a very simple reason: reactive algorithms are invoked only when the admission of a new flow fails, and the algorithm knows exactly which flow should be admitted over which tunnel. Thus, a reactive algorithm focuses on a specific tunnel and tries to increase the bandwidth available for it. In contrast, the proactive algorithm has no information about the flows that enter the network, and it is invoked regardless of whether past flows could or could not be accommodated.

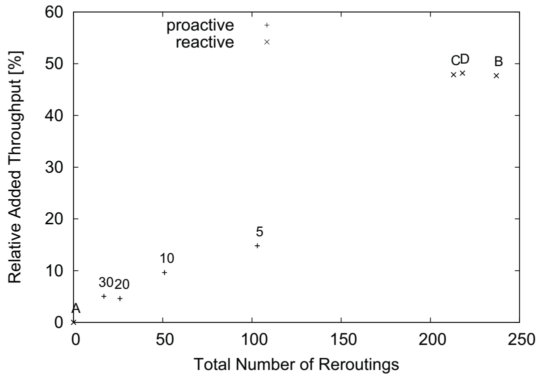
For each graph in Figure 7, consider first the four points that represent different execution periods for the Network State Algorithm (NSA). From these points we learn that determining the length of this interval is the most significant performance parameter. For example, in Figure 7(a) we see that when NSA is invoked every 5 time units it is able to admit into the network 15% more bandwidth while performing 95 reroutings. However, when it is invoked every 10 time units, it is able to admit the same percentage of extra bandwidth, but with an overhead of only 48 reroutings. This indicates that for this simulation instance NSA(10) performs much better than NSA(5). However, this is not always the case: in Figure 7(d) NSA(5) also performs many more reroutings than NSA(10), but it succeeds in accommodating much more bandwidth. This suggests that on small networks it is better to invoke NSA not very often, whereas in big networks NSA should be invoked more frequently.

When the NSA execution period is well chosen, the performance of NSA is surprisingly good compared to the performance of the best reactive algorithm. For example, consider Figure 7(d). In this figure, among all the reactive algorithms, SP(B) has the best benefit-cost ratio: $50/390 = 0.13$. However, for NSA(5) and NSA(10) the benefit-cost ratios are much better: $35/195 = 0.18$ and $21/100 = 0.2$ respectively! In Figure 7(c), the benefit-cost ratio of the best reactive algorithm, SP(C), is $48/220 = 0.22$, whereas the ratio of the best proactive algorithm, NSA(10), is $10/50 = 0.2$. We can conclude from this analysis that although the NSA algorithm cannot accommodate as much extra bandwidth as the reactive algorithms, it is competitive in terms of its benefit-cost ratio.

Figure 8 presents simulation results for the Network State Algorithm in two different Rocketfuel topology networks. In each network, the input traffic load is 0.25 and 0.5. The first


(a) $|V| = 40, |E|/|V| = 2$


(b) $|V| = 40, |E|/|V| = 4$


(c) $|V| = 80, |E|/|V| = 4$

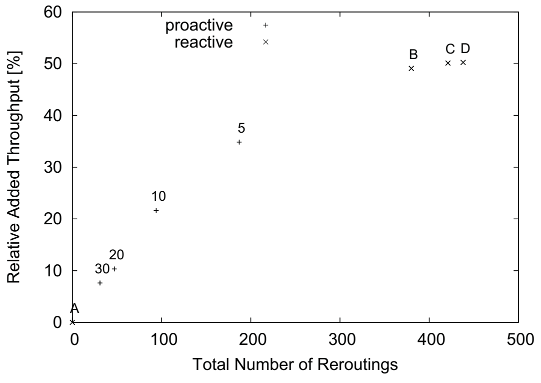
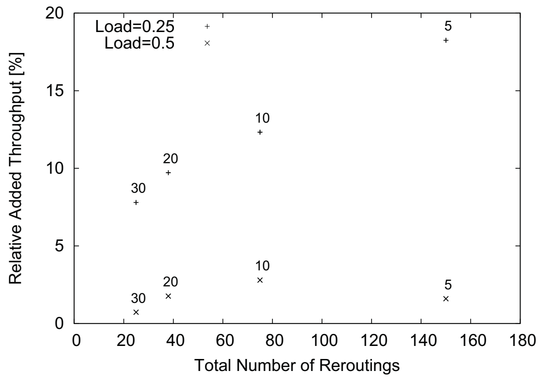

(d) RocketFuel Topology, $|V| = 138, |E|/|V| = 5.3$

Fig. 7. The performance of the reactive algorithms vs. the performance of the proactive algorithm for various networks


(a) RocketFuel Topology $|V| = 80, |E|/|V| = 1.8$

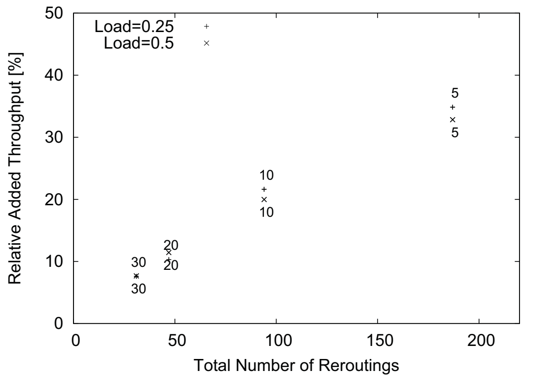

(b) RocketFuel Topology $|V| = 138, |E|/|V| = 5.3$

Fig. 8. The performance of the Network State Algorithm for different offered loads

thing to note is that the offered load does not affect the number of reroutings. But this is expected because, in contrast to the reactive algorithms, the proactive algorithms are invoked periodically, regardless of the input traffic load.

When the benefit-cost ratio of the various algorithms is considered, it is evident that the performance of NSA for all execution periods is better on light loads. This is consistent with the results shown earlier for the reactive algorithms, and can be attributed again to the fact that a rerouting algorithm has more flexibility when the load is lighter. The simulation shown in Figure 8(a) resulted in much greater performance differences among the various algorithms as compared to the simulation shown in Figure 8(b). This is due to the much smaller link degree in the former, which makes effective rerouting more difficult. Figure 8(a) also shows that increasing the load has greater negative impact in this simulation.

Finally, Figure 9 shows the impact of θ on the simulation results for different execution periods. Each simulation is performed on 100 randomly created networks, using the same BRITE generator, each with 100 nodes whose average degree is 6. The x-axis denotes the execution period of NSA, namely, the average number of flows arriving to the network between

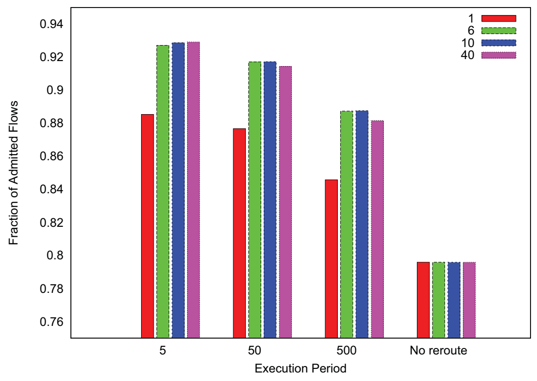


Fig. 9. The impact of θ on the performance of the Network State Algorithm

two invocations of the algorithm. The y-axis denotes the fraction of admitted flows. For each value of execution period there are 4 bars, denoting different θ values: 1, 6, 10 and 40. The leftmost bar for each period is $\theta = 1$. The y-axis denotes the total throughput. It is evident that θ has a significant impact on the performance. Moreover, large values of θ show significant improvement compared to small values. This can be explained by the fact that the initial routing of each tunnel is performed over the shortest path while ignoring the load on the various links. When rerouting is necessary, for high values of θ, the maximum load is kept low, and the network has room for new incoming flows. We can also see in this graph that **our proactive rerouting algorithm is very efficient even if it is invoked relatively rarely:** when the execution period is 500, namely, 500 new flows are introduced between two consecutive invocations of the algorithm, the percentage of rejected flows is reduced by 50%, from 0.2 (with no reroute) to only 0.1.

VI. Conclusions

This paper addressed the problem of tunnel rerouting in network overlays when the core network supports traffic engineering. For the first time, we introduced the concept of proactive (time-driven) rerouting, which we distinguish from the well-known concept of reactive (event-driven) rerouting. The main motivation behind proactive rerouting is to reduce the communication between the core network controller and the edge network controller, and to expedite the admission of new flows into the network.

We presented efficient algorithms for reactive rerouting, and then a novel Network State Algorithm for proactive rerouting. This algorithm associates a value with every network state, which indicates how well current tunnels take advantage of the available bandwidth resources. The algorithm tries to move tunnels such that the network state is maximally improved. Using simulations we show that although the Network State Algorithm cannot accommodate as many flows as the reactive algorithms, it performs surprisingly well with respect to the tradeoff between cost and value.

References

[1] I. M. A. Medina, A. Lakhina and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of MASCOTS*, 2001.

[2] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A retrospective on evolving SDN. In *HotSDN'12, Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[3] M. C. Chan and Y.-J. Lin. Behaviors and effectiveness of rerouting: A study. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 1, pages 218–223. IEEE, 2005.

[4] M. C. Chan and Y.-J. Lin. Behaviors and effectiveness of rerouting: a study. In *IEEE International Conference on Communications (ICC)*, volume 1, 2005.

[5] R. Cohen. Smooth intentional rerouting and its applications in ATM networks. In *Infocom'94*, June 1994.

[6] R. Cohen and G. Nakibly. Maximizing restorable throughput in MPLS networks. *IEEE/ACM Transactions on Networking*, 18(2), April 2010.

[7] R. Cohen and A. Segall. Connection management in ATM networks. In *Infocom'94*, June 1994.

[8] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.

[9] J. A. et al. Online load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44(3):486–504, 1997.

[10] V. Friesen, J. J. Harms, and J. Wong. Resource management with virtual paths in atm networks. *IEEE network*, 10(5):10–20, 1996.

[11] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2), 2007.

[12] G. Hampel, M. Steiner, and T. Bu. Applying software-defined networking to the telecom domain. In *Infocom workshop*, 2013.

[13] J. T. Havill and W. Mao. Greedy online algorithms for routing permanent virtual circuits. *Networks*, 34:136–153, 1999.

[14] A. E. Helvaci, C. Cetinkaya, and M. B. Yildirim. Using rerouting to improve aggregate based resource allocation. 2008.

[15] S. G. Kollopoulus and C. Stein. Improved approximation algorithms for the unsplittable flow problems. In *Proceedings of FOCS*, pages 426–435, 1997.

[16] M. Koubàa and M. Gagnaire. Lightpath rerouting strategies in wdm all-optical networks under scheduled and random traffic. *Journal of Optical Communications and Networking*, 2(10):859–871, 2010.

[17] J. P. Kowalski and B. Warfield. Modelling traffic demand between nodes in a telecommunications network. In *in ATNAC95*, 1995.

[18] M. Liu, M. Tornatore, and B. Mukherjee. Survivable traffic grooming in elastic optical networksshared protection. *Journal of lightwave technology*, 31(6):903–909, 2013.

[19] G. Mohan and C. S. R. Murthy. A time optimal wavelength rerouting algorithm for dynamic traffic in wdm networks. *J. Lightwave Technol.*, 17(3), Mar 1999.

[20] P. Pan et al. Fast reroute extensions to RSVP-TE for LSP tunnels. IETF RFC 4090, May 2005.

[21] C. Rozic and G. Sasaki. Optical protection cost of ip fast reroute on a fully connected ip network over a wdm ring. In *National Fiber Optic Engineers Conference*, page JWA3. Optical Society of America, 2011.

[22] D. A. Schupke and R. Prinz. Performance of path protection and rerouting for WDM networks subject to dual failures. In *Optical Fiber Communication Conference*, 2003.

[23] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with RocketFuel. In *Proceedings of the ACM SIGCOMM*, August 2002.

[24] T. Takagi, H. Hasegawa, K.-i. Sato, Y. Sone, A. Hirano, and M. Jinno. Disruption minimized spectrum defragmentation in elastic optical path networks that adopt distance adaptive modulation. In *European Conference and Exposition on Optical Communications*, pages Mo–2. Optical Society of America, 2011.

[25] S.-W. Wang and C.-Y. Wen. Lightpath-level active rerouting algorithms in all-optical wdm networks with alternate routing and traffic grooming. In *Information Networking (ICOIN), 2012 International Conference on*, pages 42–46. IEEE, 2012.

[26] A. Wason and R. Kaler. Rerouting technique with dynamic traffic in wdm optical networks. *Optical Fiber Technology*, 16(1):50–54, 2010.

[27] E. W. Wong, A. K. Chan, and T.-S. P. Yum. Analysis of rerouting in circuit-switched networks. *IEEE/ACM Transactions on Networking (TON)*, 8(3):419–427, 2000.

[28] M. Zhang, W. Shi, L. Gong, W. Lu, and Z. Zhu. Bandwidth defragmentation in dynamic elastic optical networks with minimum traffic disruptions. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3894–3898. IEEE, 2013.

[29] M. Zhang, C. You, H. Jiang, and Z. Zhu. Dynamic and adaptive bandwidth defragmentation in spectrum-sliced elastic optical networks with time-varying traffic. *Journal of Lightwave Technology*, 32(5):1014–1023, 2014.