Sweet Little Lies: Fake Topologies for Flexible Routing

Stefano Vissicchio *, Laurent Vanbever †, Jennifer Rexford †

* Université catholique de Louvain † Princeton University

* stefano.vissicchio@uclouvain.be † {vanbever,jrex}@cs.princeton.edu

ABSTRACT

Link-state routing protocols (*e.g.*, OSPF and IS-IS) are widely used because they are scalable, robust, and based on simple abstractions. Unfortunately, these protocols are also relatively inflexible, since they direct all traffic over shortest paths. In contrast, Software Defined Networking (SDN) offers fine-grained control over routing, at the expense of controller overhead, failover latency, and deployment challenges.

We argue that future networks can achieve the benefits of both approaches through *central control* over the *distributed route computation*. The key idea, which we call *Fibbing*, is to have the controller trick the routers into seeing a fake topology that is carefully constructed to achieve the desired Forwarding Information Base (FIB). Given an acyclic forwarding graph for each destination, the controller computes an augmented topology with fake nodes (and destinations to announce there) and fake links (and link weights). The controller injects these "lies" into the link-state routing protocol, and the routers simply compute the paths accordingly. The controller can also select an augmented topology that triggers the use of specific backup paths when real links and routers fail. To reduce router load, our Fibbing algorithms compute augmented topologies of minimal size. Our preliminary evaluation on realistic ISP topologies shows that Fibbing works well in practice.

Categories and Subject Descriptors: Network Architecture and Design; Network Management

General Terms: Algorithms; Management; Theory

Keywords: Fibbing; link-state routing; hybrid SDN

*S. Vissicchio is a postdoctoral researcher of F.R.S.-FNRS. This work has been partially supported by the ARC grant 13/18-054 from Communauté française de Belgique.

Hotnets '14, October 27–28, 2014, Los Angeles, CA, USA.

http://dx.doi.org/10.1145/2670518.2673868

1. INTRODUCTION

"A lie well told is immortal" – Mark Twain
"Tell me lies, tell me sweet little lies" – Fleetwood Mac

Network operators need flexible intra-domain routing to perform fine-grained Traffic Engineering (TE), provision backup paths, and steer traffic through middleboxes. To do so, they cannot rely on distributed routing protocols which mandate all traffic to flow along the shortest paths. Instead, they typically use dedicated TE mechanisms, prominently MPLS RSVP-TE [1, 2]. Unfortunately, these mechanisms come with their own set of limitations [3]. Among others, the lack of coordination between routers can lead to long convergence time. They also introduce control-plane and data-plane overhead due to signaling and encapsulation, respectively.

As a result, some large cloud providers, like Google [4] and Microsoft [5], are shifting to SDN, to have a central controller micromanage the forwarding rules in their switches. However, the move away from distributed protocols and destination-based forwarding also comes at a cost [6]. Primarily, the controller must scale to compute and install rules for all switches, and respond quickly to topology changes. The switches must also support rules that match on many header fields, using expensive memory. Also, most networks have a huge installed base of routers and management tools (and human operators!) that do not support SDN protocols (*e.g.*, OpenFlow [7]), leading to deployment hurdles. As a comparison, link-state routing protocols are scalable, robust, and based on simple abstractions. They maintain a relatively small amount of state, respond locally to failures, and split the computational work over all of the routers. In addition, the emphasis on destination-based forwarding enables routers to support many more rules (because they match on many fewer header fields) than today's OpenFlow switches.

In this paper, we argue that many networks do not need to move all the way to SDN. Indeed, we present a technique, which we call *Fibbing*, that realizes flexible routing using *only* traditional link-state protocols (like OSPF and IS-IS). Fibbing combines the advantages of SDN, *i.e.*, flexibility and centralized coordination, with

the ones of distributed routing, using *existing* routers and protocols. Rather than installing low-level rules in the FIB, the controller coaxes the routers into computing the right paths—by presenting them with a carefully constructed topology. The routers simply run the "tried and true" link-state routing protocol to compute the forwarding decisions, not knowing that the topology they see includes fake nodes (with fake announcements of destination address blocks) and fake links (with fake weights). That is, the controller treats the routing protocol as a function from routing messages to forwarding paths (*e.g.*, using Dijkstra's algorithm), and "inverts" the function. Given the target FIB entries on the routers (*i.e.*, the output) and the routing protocol (*i.e.*, the function), the controller automatically computes the routing messages (*i.e.*, the input).

Fibbing differs from previous approaches that use routing protocols to program routers, most notably RCP [8, 9]. In the absence of APIs like OpenFlow, RCP uses BGP as a "poor man's" SDN protocol for the controller to install rules in the routers. As such, RCP must install a rule for each destination prefix on each router, and update these entries quickly in response to failures. In contrast, Fibbing leverages the routing protocol implementation on the routers. Doing so, it can adapt the forwarding behavior of many routers at once, while allowing them to converge on their own. That is, while the controller computes the routing *input* centrally, the routing *output* is still computed in a distributed fashion.

Faking the routing protocol is not a panacea. By relying on today's link-state protocols, Fibbing is limited to destination-based forwarding over loop-free paths. Nonetheless, it is powerful enough to support sophisticated network-management tasks, including fine-grained *a*) traffic steering; *b*) traffic engineering; *c*) load balancing; and *d*) fast failover. We believe that these enhanced capabilities are more than sufficient for many networks. Moreover, by offering a central controller with a high-level interface for network operators, Fibbing can be a step in the incremental deployment of SDN, for networks that ultimately need even greater flexibility.

We make the following contributions:

Fibbing: Faking the link-state protocol is a simple and powerful way to make routing much more flexible (§2).

Expressiveness: Given a directed acyclic graph (DAG) for each destination prefix, Fibbing can *always* compute a corresponding augmented topology (§3).

Efficiency: To limit router overhead, Fibbing can compute a *minimum* augmented topology by solving an Integer Linear Program (ILP) (§4).

Experiments: Simulations of Fibbing on realistic topologies show that small augmented topologies are able to express multiple deviations from shortest paths (§5).

2. LYING TO LINK-STATE ROUTING

In this section, we describe how a Fibbing controller creates fake nodes, links, and destinations. We then present three examples that show how Fibbing performs traffic steering, backup routing, and load balancing. Those examples highlight how Fibbing overcomes intrinsic inflexibilities of traditional protocols (which hold even when weights are modified, *e.g.*, using [10]) to implement simple traffic engineering and middleboxing requirements in practical, realistic cases. We discuss potential problems created by failures in §4

2.1 Fibbing By Injecting Fake LSAs

In a link-state routing protocol, routers (i) flood link-state advertisements (LSAs) to construct a shared view of the topology, (ii) compute shortest paths (as a sum of configurable link weights) for each destination prefix, and (iii) install forwarding entries that direct packets to the next hops in their paths. While routers may have vendor-specific configuration interfaces, they all follow the same protocol with standard formats for the LSAs.

Fibbing leverages the standard protocol to remotely control the behavior of the routers. A Fibbing controller forms protocol adjacencies with one or more routers, and inject well-crafted LSAs to (i) announce additional nodes and links, which we call *fake nodes* and *fake links*, respectively; (ii) set the *fake weights*, *i.e.*, weights of fake links; and (iii) announce the reachability of destination prefixes (*fake destinations*) from fake nodes. We globally refer to a topology enriched with fake nodes, links, and destinations as an *augmented topology*. To avoid changing the configuration of the real routers, the augmented topology includes all of the real nodes, links, and destinations, as well as *additional* fake information.

The Fibbing controller can introduce LSAs that appear to come from fake nodes and links. Many routing protocols, including OSPF, allow the router originating a message to differ from the router described in the message. In fact, this feature can be a security vulnerability, since a compromised router could intentionally introduce LSAs on behalf of other routers to pollute their routing tables [11]. In practice, we want the data traffic to flow through the real routers, not the fake nodes forged by the controller. In OSPF, a Fibbing controller can direct traffic to a real router r instead of a fake node f by specifying an IP address of f as the "forwarding address" [12] in the LSAs propagated by f.

2.2 Examples of Good Fibbing

The Fibbing controller presents an augmented topology to the routers to modify the shortest paths they compute. We now show simple examples where small "lies" can attract and detour traffic on a per-destination basis, for better traffic steering, backup routes, and load balancing. Our examples draw on the network in Fig-

ure 1 where a source s connected to A exchanges data with two destinations d_1 and d_2 connected to F and G, respectively. All of the links have the same capacity.

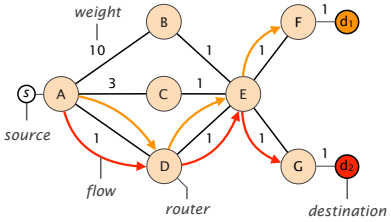
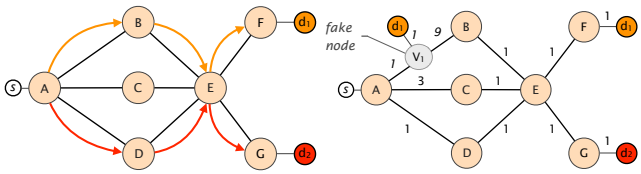


Figure 1: "Fishy" network, with shortest-path routing.

2.2.1 *Traffic Steering Through Middleboxes*

Per-destination traffic steering can be used to direct traffic selectively through middleboxes [13]. As an illustration, assume that a DDoS attack on d_1 congests the path (A, D, E, F). To address this issue, the operator wants to redirect traffic to d_1 via B, where (say) a DPI box is located, without impacting the traffic destined to d_2 (Figure 2a). However, this simple requirement is *impossible* to achieve in a traditional link-state protocol. Indeed, shortest-path routing forces all packets from A to both d_1 and d_2 to follow the same path to node E.



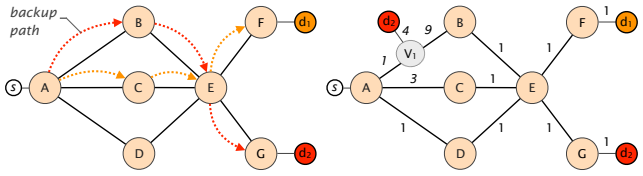
(a) Desired flows (b) Augmented IGP topology

Figure 2: Fibbing can *steer traffic* destined to d_1 away from the shortest path (via D) to B.

Figure 2b shows how Fibbing achieves the desired forwarding behavior. A fake node v_1 is connected to A and B with a weight of 1 to A and of 9 to B. Also, v_1 advertises that it can reach d_1 directly. Since the shortest path to v_1 is shorter than the one to F, A starts to use v_1 for d_1. Since v_1 is a fake node, packets actually flow through B. Since A is the only router to change its forwarding path due to the introduction of v_1, we can enforce the requirement in Figure 2a.

2.2.2 *Provisioning of Efficient Backup Paths*

Fibbing can provision backup paths, to prevent network congestion after link and node failures. As an illustration, assume that the path (A, D, E) has significantly more bandwidth than paths (A, B, E) and (A, C, E). Upon the failure of link (A, D), link (D, E), or node D, the operator would like to split the traffic directed to d_1 and d_2 over the two remaining disjoint paths, to increase network utilization (see Figure 3a).



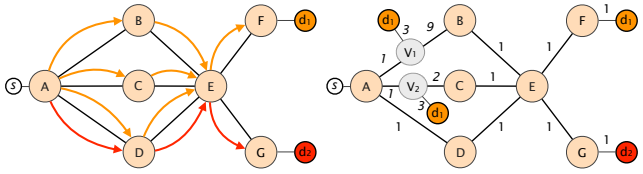
(a) Desired flows (b) Augmented IGP topology

Figure 3: Fibbing can *provision backup paths* on a per-destination basis. Upon the failure of link (A, D) or (D, E), traffic destined to d_1 traverses node B, whereas traffic destined to d_2 traverses node C.

Again, while this simple requirement would be *impossible* under conventional link-state routing, it is easily done using Fibbing. Figure 3b shows the corresponding augmented topology. As before, a single fake node v_1 is added, this time advertising d_2, with a weight of 4. This prevents A from using the virtual node to reach d_2 unless a failure occurs along the path (A, D, E). Note that v_1 does not advertise d_1 as the traffic would already follow the desired path (A, C, E) after the failure.

2.2.3 *Load Balancing Over Multiple Paths*

Fibbing can also be used to load-balance traffic over multiple (non-shortest) paths to maximize throughput, or minimize response time. For example, suppose source s sends a huge amount of traffic to d_1 in Figure 1. The operator might like to use all available paths between the two hosts, as shown in Figure 4a. Using all available paths between s and d_1 is possible under conventional link-state routing (*e.g.*, by re-weighting links (A, B) and (B, E) to 1). However, this would force the traffic from s to d_2 to also spread over all of the available paths. This might be undesirable, *e.g.*, if some of the paths cannot handle the extra load or have a propagation delay that is too high for that traffic. More generally, it is *impossible* to route the traffic for d_1 and d_2 on different links under conventional link-state routing.



(a) Desired flows (b) Augmented IGP topology

Figure 4: Fibbing can *load-balance traffic* over multiple paths. In this case, the traffic from s to d_1 spreads over three paths, while leaving traffic from s to d_2 untouched.

Figure 4b shows the augmented topology which realizes exactly the requirements in Figure 4a. Two fake nodes (v_1 and v_2) are required this time. Both fake nodes announce d_1 with a cost of 3 and are mapped to

link (A, B) and (A, C), respectively. After their introduction, A ends up with three equal-cost paths of cost 4 to reach d_1 and splits the traffic over all of them.

3. FIBBING IS EXPRESSIVE

The Fibbing controller takes, as input, a set of forwarding Directed Acyclic Graphs (DAGs), each corresponding to a subgraph of the real topology and representing the desired forwarding paths to a destination (*i.e.*, the sink of the DAG). The Fibbing controller can learn the real topology by listening to the LSAs over the IGP adjacencies that it forms with real routers (like in [14]). We assume that all real IGP links have weights that are greater than 1; if this is not the case, we can always double all link weights to enforce this condition without changing any forwarding path.

We proved that an augmented topology realizing any input forwarding DAGs can always be computed.

THEOREM 1. *Any set of DAGs, each representing forwarding paths to a different destination, can be enforced by an augmented topology.*

The proof of Theorem 1 (not fully reported for space limitation) is based on a constructive procedure. Beyond proving Theorem 1, this procedure is also used by the Fibbing controller to translate input DAGs into fake topologies. The procedure consists of two steps: i) translating the input DAGs into constraints on the shortest paths in the augmented topology, and ii) computing a compliant augmented topology.

3.1 Control-Plane Paths and Requirements

To describe our constructive procedure, we first introduce a concise model and some notation. We denote the total cost of the shortest path between nodes a and b in the real topology as $cost(a, b)$. Also, we distinguish between *forwarding paths*, *i.e.*, sequences of real routers traversed by user traffic, and *control-plane paths*, *i.e.*, shortest paths in a (possibly augmented) IGP topology. Observe that, in an augmented topology, forwarding and control-plane paths do not necessarily coincide. As an illustration, in Figure 2b, the forwarding and control-plane paths from A to d_1 are $(A\ B\ E\ F\ d_1)$ and $(A\ v_1\ d_1)$, respectively. These inconsistencies map to *shortest-path violations*, that is, deviations of traffic from the original shortest paths in the real topology.

We model (the need for) shortest-path violations with *control-plane requirements* containing sequences of real routers and special symbols \$ that represent (the need to traverse) a fake node. A control-plane requirement R is *matched* by a set of control-plane paths $P_1, \ldots, P_k$ if the next-hop n of each physical node u in any of the paths P_i is either i) n, if n is the next-hop of u also in R, or ii) any fake node, if the next-hop of u in R is \$. Referring again to Figure 2, $R = (A\ \$\ B\ E\ F\ d_1)$ is the control-plane requirement associated with Figure 2a. Following the definition of control-plane requirement matching, it is easy to check that R is matched by the shortest paths in the augmented topology of Figure 2b. Note that control-plane requirements naturally map to traffic steering needs. Moreover, multiple requirements between the same source-destination pair express load balancing specifications. Finally, for backup path provisioning cases, control-plane requirements can be tagged with the corresponding failure scenarios.

3.2 Mapping Input DAGs into Requirements

In the first step of our constructive proof of Theorem 1, we translate input DAGs into control-plane requirements satisfying the following condition. We rely on this condition to show (in §3.3) an efficient algorithm that computes the needed topology augmentation.

CONDITION 1. *Let $\mathcal{R}$ be a set of control-plane requirements. For any non-empty sequence of consecutive real nodes $F = (v_0 \ldots v_k)$, and any requirement in $\mathcal{R}$,*

1a) F is the shortest path from v_0 to v_k, and

1b) for every node $t_l \neq v_k$ at the end of any sequence of consecutive real nodes in any requirement in $\mathcal{R}$, $cost(v_0, v_k) < cost(v_0, t_l) - 1$.

The following lemma ensures that the requirement extraction can be performed for any Fibbing input.

LEMMA 1. *Any forwarding DAG can be translated to control-plane requirements satisfying Condition 1.*

To show the feasibility of the translation, a general procedure can be applied. It consists in mapping any forwarding path $(v_0\ v_1 \ldots v_k)$ in the considered DAG to a control-plane requirement $R = (v_0\ \$\ v_1\ \$ \ldots v_k)$. This requirement R matches i) Condition 1a trivially, and ii) Condition 1b because $cost(v_i, v_i) = 0 < cost(v_i\ t_l) - 1$ for any pair of nodes v_i and $t_l \neq v_i$ (in any topology where link weights are greater than 1). With this general procedure, we would end up with one shortest-path violation for each router and for each destination in the input DAGs. However, the number of shortest-path violations generated by a given set of forwarding DAGs can be optimized, for instance, by reusing sub-paths of IGP shortest paths which are compliant with Condition 1.

3.3 Computing the Augmented Topology

The second step of our procedure consists in translating control-plane requirements into augmented topologies. For requirements compliant with Condition 1, this is always possible. Indeed, the following lemma holds, completing the proof of Theorem 1.

LEMMA 2. *Control-plane requirements complying with Condition 1 can be always matched by shortest paths in an appropriately augmented topology.*

We proved Lemma 2 with a simple, efficient algorithm which we call LADDER (for Leaf ADDER). LADDER takes a real topology and a set $\mathcal{R}$ of control-plane requirements as input, and outputs an augmented topology. For each destination d in $\mathcal{R}$, the algorithm adds one fake node f_d announcing d, and connects f_d to all the nodes t_i appearing before a \$ in any requirement in $\mathcal{R}$. The weight of any link (t_i, f_d) is asymmetric: it is set to 1 from t_i to f_d and to ∞ in the opposite direction. This way, for each sequence $(v_0 \dots v_k)$ of consecutive real nodes in any requirement to d, the shortest path from v_0 to d becomes $(v_0 \dots v_k\ f_d)$. Indeed, Condition 1a and Condition 1b respectively guarantee that this path is shorter than i) old paths in the real topology, and ii) new paths traversing fake nodes. When stitched together, those new shortest paths match the original control-plane requirement, as shown in Figure 2b for requirement $(A\ \$\ B\ E\ F\ d_1)$. Note that the time complexity of LADDER is linear with the size of the input $\mathcal{R}$, since a single pass on each requirement is needed for LADDER to generate the appropriate fake nodes.

4. FIBBING IN A SCALABLE FASHION

In this section, we show how Fibbing can scale by computing augmented topologies of minimal size, and precomputing responses to failures in the real topology.

4.1 Computing Minimal Fake Topologies

Limiting the size of fake topologies has multiple benefits. Primarily, it limits the control-plane overhead, and does not degrade router performance (*e.g.*, for shortest path computation). It also enables Fibbing to influence many forwarding paths by tweaking a few fake weights (or fake destinations). This is especially useful when a lot of forwarding paths need to be updated, *e.g.*, to change network policies or in the case of failures.

Strategically positioning fake nodes to influence many forwarding paths is the key to minimizing the size of the augmented topology. For example, Figures 2b and 3b show how a fake node v_1 tunes two different forwarding paths. Since those paths pertain to two distinct destinations (d_1 and d_2), only one node v_1 (announcing both destinations with the weights as in the figures) can be used. Beyond using the same fake node to tweak paths to distinct destinations, other optimizations are possible, like using a single fake node for multiple forwarding paths terminating at the same destination.

We modeled the problem of minimizing the size of an augmented topology as an Integer Linear Program (ILP). Given an augmented topology (with fake weights initially set to ∞) and a set of control-plane requirements, the ILP i) formalizes each requirement as a set of constraints on fake weights, so that a path matching the requirement would result as the new shortest one in the augmented topology, and ii) defines the objective function (to minimize) as the sum of fake links with weight different from ∞. Until an augmented topology matching all the requirements is found, the Fibbing controller can, then, iteratively i) compute augmented topologies with an increasing number of fake nodes, which are connected to real nodes immediately before or after \$s in the requirements, and ii) solve the corresponding ILP to minimize the number of used fake nodes and edges. This approach, that we evaluate in §5, represents an alternative to using polynomial-time algorithms like LADDER. Being exponential, the ILP approach trades algorithm efficiency for fake topology minimization. Finding an efficient algorithm for computing topologies of limited size (*e.g.*, within a certain approximation bound with respect to the minimal one) is an interesting algorithmic problem which we plan to study in future work.

4.2 Precomputing Responses to Failure

Failures are a major challenge for logically-centralized approaches, since they require the controller to promptly update the affected paths, *e.g.*, to avoid permanent loops and blackholes. Fibbing is no exception. For example, consider again Figure 2b. If link (A, B) fails, the presence of the fake node v_1 creates a blackhole, since the forwarding path corresponding to the control-plane path traversing v_1 does not exist anymore. Similarly, a failure of link (B, E) leads to a forwarding loop.

Fibbing can quickly repair forwarding paths upon failures. It can pre-compute post-convergence topologies (*e.g.*, for the most likely or most critical combinations of failures like single link failures and shared-risk link groups), and store deltas with respect to the fake topology currently installed. For instance, in Figure 2b, the controller can pre-compute that it should remove v_1 upon the failure of link (B, E) or of nodes B or E. Deltas can also be computed on-the-fly upon failures (if potential short-lived disruptions are tolerated), by running efficient algorithms like LADDER.

By relying on link-state routing mechanisms, Fibbing can promptly detects failures by listening to flooded link-state messages. Moreover, it can inject new fake topology information *during routing convergence*, avoiding the cost of having the IGP converge twice (once for the real failure, and once for the fake topology changes). Also, Fibbing can leverage years of work on minimizing IGP convergence time on commercial routers [15] and can be combined with IGP fast rerouting techniques like Loop-Free Alternate [16] to further improve failure recovery time. Finally, we expect that Fibbing can repair many forwarding paths with changes on a few strategically-positioned fake nodes and links (as also suggested by our evaluation, see §5).

5. PRELIMINARY EVALUATION

We implemented the ILP-based approach described

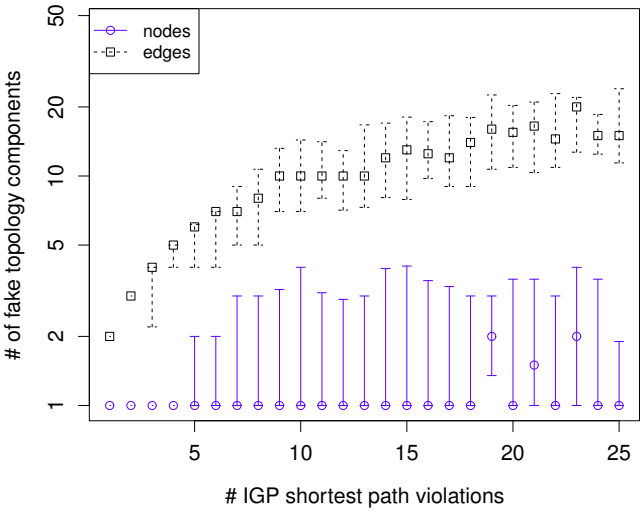


Figure 5: Small fake topologies can be used to enforce multiple deviations from shortest-path routing.

in §3. Our implementation relies on about 1,200 lines of Python code that extract the ILP formulation from the input (forwarding DAGs and the original IGP graph), and runs Gurobi [17] to solve it. We used the implementation to simulate arbitrary traffic steering in all the Rocketfuel topologies [18]. Our experiments consisted of simulating random variations of the shortest paths to 1 to 5 randomly-selected destinations. Those variations consisted in modifying the next-hop of some randomly-extracted nodes, while guaranteeing the absence of loops. This way, we generated 15 forwarding DAGs for each number of shortest-path violations between 1 and 25. We ran all our experiments on a machine with 2.5 GHz processor and 4 GB of memory.

Fake topology size. Figure 5 shows the distribution of the number of fake nodes and fake edges needed to enforce an increasing number of shortest-path violations. In particular, circles and squares represent the median number of needed fake nodes and edges, respectively, with solid and dashed error bars identifying the 5-th and the 95-th percentiles. Results reported in the figure are aggregated over all the Rocketfuel topologies, since they do not significantly vary by topology.

Quite intuitively, the number of fake components tends to increase when a higher number of IGP shortest paths need to be violated. The size of fake topologies is not strictly increasing. This suggests that some combinations of shortest-path violations require more Fibbing than others, *i.e.*, because fake nodes and edges cannot be reused to enforce multiple shortest-path violations. Figure 5 also indicates that the size of computed fake topologies grows slowly with an increasing number of shortest-path violations. The variability across different random selections of the same number of violations is also small. Indeed, up to 4 nodes (with a median of 1 or 2) and less than 20 edges (including the ones relative to fake destinations—like (v_1, d_1) in Figure 2b) are sufficient in the vast majority of our simulations. Actually, the maximum values for fake nodes and edges are 5 and 26, respectively, which is still quite limited with respect to the size of the corresponding real topology (AS6461, which has 141 nodes and 748 edges).

Computation time. Our ILP approach is fast. In the 99-th percentile of our experiments, the computation time ranged from about 0.4 seconds to slightly more than 10 minutes, with the 95-th percentile being around 2.5 minutes, which we consider already acceptable for long-term path optimizations and what-if scenarios. We plan to improve our prototype, with code and algorithm optimizations, in future work.

6. CONCLUSIONS AND PERSPECTIVES

This paper introduces Fibbing, which centrally controls the distributed computation of forwarding paths. We show its expressiveness, and its capability to scale and quickly react to failures. Simulations on realistic topologies suggest the practicality of Fibbing to exert SDN-like control of IGP networks. For these reasons, a hybrid SDN architecture based on Fibbing can be considered as a valid alternative for the long-term design of many networks. As argued in [6], it also provides operators with incentives to (partially) transition to SDN. Among others, we believe Fibbing opens the following new perspectives.

Fibbing is a unified interface to heterogeneous networks consisting of any devices (potentially, including SDN ones) which run an IGP daemon. Indeed, our work can be seen as a first step towards a SDN controller managing devices with different capabilities.

Fibbing is beneficial in partial SDN deployments which include a limited number of pure SDN devices (*e.g.*, OpenFlow switches). Indeed, its ability to fine-tune forwarding paths on a per-destination basis enables to steer traffic to SDN devices, *e.g.*, to implement advanced network functions like NFV on them. Also, Fibbing is much more powerful than using traditional devices as underlay layer between SDN ones as in [19] as Fibbing: (i) can engineer *all* forwarding paths, even the ones that do not traverse SDN devices; (ii) can fine-tune traffic to SDN devices, *e.g.*, balancing their load.

Fibbing enables SDN in the core and supports new SDN deployment patterns. Existing SDN techniques (*e.g.*, [20]) are based on deploying SDN functionalities only at the network borders, using software switches. Fibbing can complement those techniques by enabling SDN functionalities inside the network on traditional equipments (to optimize internal forwarding paths) or even replace them completely, by deploying SDN functions in the core.

7. REFERENCES

[1] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, 2001.

[2] A. Farrel, J.-P. Vasseur, and J. Ash, "A Path Computation Element (PCE)-Based Architecture," RFC 4655, 2006.

[3] A. Pathak, M. Zhang, Y. C. Hu, R. Mahajan, and D. A. Maltz, "Latency inflation with MPLS-based traffic engineering," in *Internet Measurement Conference*, 2011, pp. 463–472.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN," in *ACM SIGCOMM*, 2013.

[5] C. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *ACM SIGCOMM*, 2013.

[6] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, Apr. 2014.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[8] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *NSDI*, 2005.

[9] J. van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg *et al.*, "Dynamic connectivity management with an intelligent route service control point," in *INM*, 2006.

[10] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *INFOCOM*, 2000.

[11] G. Nakibly, E. Menahem, A. Waizel, and Y. Elovici, "Owning the Routing Table. Part II," Black Hat, 2013.

[12] J. Moy, "OSPF Version 2," RFC 2328, 1998.

[13] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *ACM SIGCOMM*, 2013.

[14] A. Shaikh and A. Greenberg, "OSPF Monitoring: Architecture, Design and Deployment Experience," in *NSDI*, 2004.

[15] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 33–44, 2005.

[16] C. Filsfils, P. Francois, M. Shand, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer, "LFA applicability in SP networks," RFC 6571, 2012.

[17] "Gurobi Solver," http://www.gurobi.com/.

[18] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *ACM SIGCOMM*, 2002.

[19] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in *USENIX ATC*, 2014.

[20] T. Koponen, K. Amidon, P. Balland, M. Casado, et al., "Network virtualization in multi-tenant datacenters," in *NSDI*, 2014.