**Central Control Over Distributed Routing**

**分布式路由器的集中控制**

ABSTRACT

Centralizing routing decisions offers tremendous flexibility, but sacrifices the robustness of distributed protocols. In this paper, we present Fibbing, an architecture that achieves both flexibility and robustness through central control over distributed routing. Fibbing introduces fake nodes and links into an underlying link-state routing protocol, so that routers compute their own forwarding tables based on the augmented topology. Fibbing is expressive, and readily supports flexible load balancing, traffic engineering, and backup routes. Based on high-level forwarding requirements, the Fibbing controller computes a compact augmented topology and injects the fake components through standard routing-protocol messages. Fibbing works with any un- modified routers speaking OSPF. Our experiments also show that it can scale to large networks with many forwarding requirements, introduces minimal overhead, and quickly reacts to network and controller failures.

摘要

集中路由决策提供了巨大的灵活性，但牺牲了分布式协议的稳健性。 在本文中，我们介绍Fibbing，一种通过集中控制分布式路由实现灵活性和健壮性的架构。 Fibbing将假节点和链接引入到链路状态路由协议，因此路由器可以根据扩展的拓扑结构计算自己的转发表。 Fibbing具有表现力，并且很容易支持灵活的负载平衡，流量工程和备份路由。 基于高级转发需求，Fibbing控制器计算紧凑的扩展拓扑，并通过标准路由协议消息注入假组件。 Fibbing可与任何运行OSPF协议的未修改的路由器配合使用。 我们的实验还表明，它可以扩展到具有许多转发需求的大型网络，引入最少的开销，并快速响应网络和控制器故障。

**1. INTRODUCTION**

Consider a large IP network with hundreds of devices, including the components shown in Fig. 1a. A set of IP addresses (D1) see a sudden surge of traffic, from multiple entry points (A, D, and E), that congests a part of the network. As a network operator, you suspect a denial-of-service attack (DoS), but cannot know for sure without inspecting the traffic as it could also be a flash crowd. Your goal is therefore to: (i) isolate the flows destined to these IP addresses, (ii) direct them to a scrubber connected between B and C, in order to “clean” them if needed, and (iii) reduce congestion by load-balancing the traffic on unused links, like (B,E).

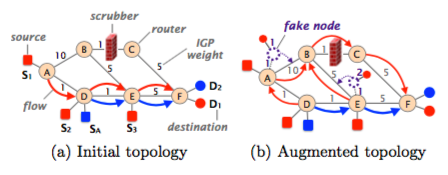


Figure 1: Fibbing can steer the initial forward- ing paths (see (a)) for D1 through a scrubber by adding fake nodes and links (see (b)).

1.引言

考虑一个包含数百个设备的大型IP网络，其中包括图1a所示的组件。一组IP地址（D1）可以看到来自多个入口点（A，D和E）的突然激增的流量，这些流量点占用了网络的一部分。 作为网络运营商，您怀疑是拒绝服务攻击（DoS），但无法确定未经检查的流量，因为它可能也是一个突然的大流。 因此，您的目标是：（i）隔离去往这些IP地址的流量，（ii）将它们导向连接在B和C之间的洗涤器，以便在需要时“清洁”它们，以及（iii）减少拥塞通过均衡负载到未使用的接上，如（B，E）。

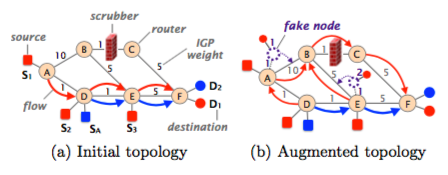


图1：Fibbing可以通过添加假节点和链接（参见（b））来控制D1通过洗涤器的初始转发路径（参见（a））。

Performing this routine task is very difficult in traditional networks. First, since the middlebox and the destinations are not adjacent to each other, the configuration of multiple devices needs to change. Also, since intra-domain routing is typically based on shortest path algorithms, modifying the routing configuration is likely to impact many other flows not involved in the attack. In Fig. 1a, any attempt to reroute flows to D1 would also reroute flows to D2 since they home to the same router. Advertising D1 from the middlebox would attract the right traffic, but would not necessarily alleviate the congestion, because all D1 traffic would traverse (and congest) path (A, D, E, B), leaving (A, B) unused. Well-known Traffic-Engineering (TE) protocols (e.g., MPLS RSVP-TE [1]) could help. Unfortunately, since D1 traffic enters the network from multiple points, many tunnels (three, on A, D, and E, in our tiny example) would need to be configured and signaled. This increases both control-plane and data-plane overhead.

在传统网络中执行这个例行任务是非常困难的。首先，由于途中路由器和目的地不相邻，所以需要改变多个设备的配置。而且，由于域内路由通常基于最短路径算法，因此修改路由配置可能会影响许多其他未涉及到攻击的流。在图1a中，任何将流重新路由到D1的尝试也会将流重新路由到D2，因为它们归属于相同的路由器。声明D1来自中间路由器将吸引正确的流量，但不一定会缓解拥塞，因为所有的D1流量都会穿过（引发拥挤）路径（A，D，E，B），而（A，B）未被使用。众所周知的流量工程（TE）协议​​（例如，MPLS RSVP-TE [1]）可能有所帮助。不幸的是，由于D1流量从多个点进入网络，因此需要配置和发送很多隧道（三个，在我们的示例中为A，D和E）。这增加了控制平面和数据平面的开销。

Software Defined Networking (SDN) could easily solve the problem as it enables centralized and direct control of the forwarding behavior. However, moving away from distributed routing protocols comes at a cost. Indeed, IGPs like OSPF and IS-IS are scalable (support networks with hundreds of nodes), robust, and quickly react to failures. Building a SDN controller with com- parable scalability and reliability is challenging. It must compute and install forwarding rules for all the switches, and respond quickly to topology changes. Even the simple task of updating the switch rule tables can then become a major bottleneck for a central controller man- aging hundreds of thousands of rules in hundreds of switches. In contrast, distributed routing protocols naturally parallelize this work. For reliability and scalability, a SDN controller should also be replicated and geographically distributed, leading to additional challenges in managing controller state. Finally, the deployment of SDN as a whole is a major hurdle as many networks have a huge installed base of devices, management tools, and human operators that are not familiar with the technology. As a result, existing SDN deployments are limited in scope, e.g., new deployments of private backbones [8, 9] and software deployments at the network edge [10].

软件定义网络（SDN）可以轻松解决问题，因为它可以集中和直接控制转发行为。但是，远离分布式路由协议需要付出代价。事实上，像OSPF和IS-IS这样的IGP是可扩展的（支持具有数百个节点的网络），强大且快速响应故障。构建具有可扩展性和可靠性的SDN控制器具有挑战性。它必须为所有交换机计算和安装转发规则，并快速响应拓扑变化。即使是更新交换机规则表的简单任务，也可能成为中央控制器管理数百台交换机的数十万条规则的主要瓶颈。相比之下，分布式路由协议自然的并行完成了这项工作。为了实现可靠性和可扩展性，SDN控制器也应该进行复制并在地理上分散，从而在管理控制器的状态时产生额外的挑战。最后，作为一个整体部署SDN是一个主要障碍，因为许多网络拥有庞大的设备，和不熟悉该技术和管理工具的操作人员。因此，现有的SDN部署范围受到限制，例如，专用骨干网的新部署[8,9]以及网络边缘的软件部署[10]。

This paper introduces Fibbing, a technique that offers direct control over the routers’ forwarding information base (FIB) by manipulating the input of a distributed routing protocol. Fibbing relies on traditional link-state protocols such as OSPF [5] and IS-IS [6], where routers compute shortest paths over a synchronized view of the topology. Fibbing controls routers by carefully lying to them, removing the need to configure them. It coaxes the routers into computing the target forwarding entries by presenting them with a carefully constructed augmented topology that includes fake nodes (providing fake announcements of destination address blocks) and fake links (with fake weights). In essence, Fibbing inverts the routing function: given the forwarding entries (i.e., the desired output) and the routing protocol (i.e., the function), Fibbing automatically computes the routing messages to send to the routers (i.e., the input).

本文介绍了Fibbing技术，该技术通过操纵分布式路由协议的输入来直接控制路由器的转发信息库（FIB）。 Fibbing依赖于传统的链路状态协议，如OSPF [5]和IS-IS [6]，其中路由器通过拓扑的同步视图计算最短路径。 Fibbing通过小心地欺骗路由器来控制路由器，而不需要配置路由器。 它通过向他们提供包含假节点（提供目的地址块的假通告）和假链接（具有假权重）的精心构造的增强拓扑来哄骗路由器计算目标转发条目。 本质上，Fibbing反转路由功能：给定转发条目（即所需输出）和路由协议（即功能），Fibbing自动计算发送给路由器（即输入）的路由消息。

Fibbing can solve the problem in Fig. 1a adding two fake nodes (Fig. 1b), connected to A and E with the depicted weights. Both fake nodes advertise that they can reach D1 directly. Based on the augmented topology, D starts to use A to reach D1, as the new cost (3) is lower than the original one (6). A and E also select different paths. Since the fake nodes do not really exist, packets forwarded by A or Eactually flow through B. Routers B and C do not change their forwarding decisions.

Fibbing可以解决图1a中问题：添加两个具有描述的权重的假节点（图1b）连接到A和E. 两个假节点都声称他们可以直接到达D1。 基于扩展的拓扑，D开始使用A来达到D1，所以新的成本（3）低于原始的（6）。 A和E也选择不同的路径。 由于假节点并不是真的存在，因此A或E转发的数据包将流过B.路由器B和C不会更改其转发决策。

Table 1 gives an overview of how Fibbing improves flexibility and manageability by adopting a SDN-like approach while keeping the advantages of distributed protocols (e.g., robustness and fast FIB modifications).

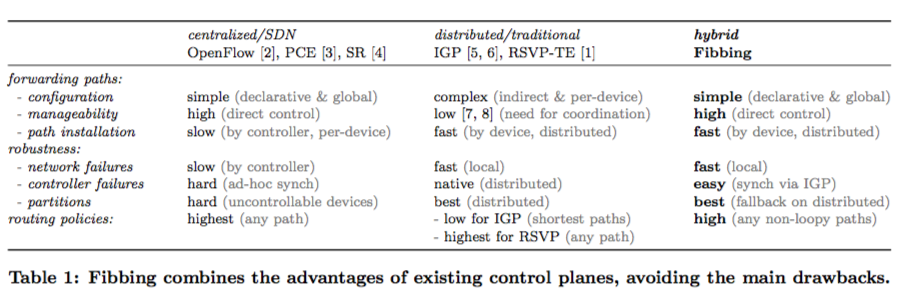


表1概述了Fibbing如何通过采用类似于SDN的方法，同时保持分布式协议的优势（例如稳健性和快速FIB修改）来提高灵活性和可管理性。

Fibbing is expressive. Fibbing can steer flows along any set of per-destination loop-free paths. In other words, it can exert full control at a per-destination granularity. For this reason, Fibbing readily supports advanced forwarding applications such as: (a) traffic engineering, (b) load balancing, (c) fast failover, and (d) traffic steering through middleboxes. By relying on destination-based routing protocols, Fibbing does not support finer-grained routing and forwarding policies such as matching on port numbers. Though, those policies can easily be supported via middleboxes.

**Fibbing具有表现力。**Fibbing可以沿任何一组每个目标无回路路径引导流量。换句话说，它可以在每个目标粒度上发挥完全控制权。 因此，Fibbing可以轻松地支持高级转发应用程序，例如：（a）流量工程，（b）负载平衡，（c）快速故障转移，以及（d）通过中间件进行流量转向。通过依赖基于目标的路由协议，Fibbing不支持更细粒度的路由和转发策略，例如匹配端口号。 但是，这些政策可以通过中间件轻松支持。

Fibbing scales and is robust to failures. Lying to routers is powerful but challenging. Indeed, Fibbing must be fast in computing augmented topologies to avoid loops and blackholes upon network failures. At the same time, Fibbing must compute small augmented topologies since routers have limited resources. Finally, Fibbing must be reliable and gracefully handle controller failures. We address all three challenges.

**Fibbing的规模，能有效应对失败。**欺骗路由器是强大的，但具有挑战性。 事实上，Fibbing在计算增强拓扑时必须快速，以避免网络故障时的循环和黑洞。与此同时，由于路由器资源有限，Fibbing必须计算尽可能少的拓扑增加。 最后，Fibbing必须可靠且优雅地处理控制器故障。我们解决了所有三个挑战。

**Fibbing differs from previous approaches that rely on routing protocols to program routers.** Prior approaches like the Routing Control Platform [11] rely on BGP as a “poor man’s” SDN protocol to install a forwarding rule for each destination prefix on each router. In contrast, Fibbing leverages the routing- protocol implementation on the routers. Doing so, Fibbing can adapt the forwarding behavior of many routers at once, while allowing them to compute forwarding- table entries and converge on their own. That is, while the controller computes the routing input centrally, the routing output is still computed in a distributed fashion.

**Fibbing不同于以前依靠路由协议编程路由器的方法。** 像Routing Control Platform [11]等先前的方法依靠BGP,像“穷人”一样的SDN协议来为每个路由器上的每个目的地前缀安装转发规则。 相反，Fibbing利用路由器上的路由协议实现。 这样做，Fibbing可以一次调整许多路由器的转发行为，同时允许他们计算转发表条目并自行收敛。 也就是说，当控制器集中计算路由输入时，路由输出仍然以分布式方式计算。

**Fibbing works on existing routers.** We implemented a fully-functional Fibbing prototype and used it to program real routers (both Cisco and Juniper). Based on an augmented topology, these routers can install hundreds of thousands of forwarding entries with an average installation time of less than 1ms per entry. This offers much greater scale and faster convergence than is possible with state-of-the-art SDN switches [12, 13], without requiring the deployment of new equipment and per-device actions from the controller. This also means that Fibbing can implement recent SDN proposals, like Google’s B4 [8] and Microsoft’s SWAN [9]—on top of existing networks.

**Fibbing适用于现有的路由器。**我们实现了一个全功能的Fibbing原型，并用它来编程真正的路由器（思科和瞻博网络）。基于扩展拓扑结构，这些路由器可以安装数十万个转发条目，平均安装时间小于每个条目1ms。 与现有技术的SDN交换机相比，这提供了更大的规模和更快的收敛速度[12,13]，而无需从控制器部署新设备和每个设备调整。 这也意味着Fibbing可以实施最近的SDN提案，如Google的B4 [8]和微软的SWAN [9] ——位于现有网络之上。

Our earlier work showed that Fibbing can enforce any set of forwarding DAGs [14]. This paper goes further by describing the complete design, implementation, and evaluation of a Fibbing controller managing intra- domain routing. Rather than focusing on specific use cases (like traffic engineering), we describe its support for different higher-level approaches (e.g., [8, 9]). We make the following contributions:

**Abstraction:** We show how to express and realize high-level forwarding requirements by manipulating a distributed link-state routing protocol (§2).

**Algorithms:** We propose new, efficient algorithms to compute compact augmented topologies (§3).

**Implementation:** We describe a complete Fibbing implementation which works with unmodified Cisco and Juniper routers (§4).

**Evaluation:** We show that our Fibbing controller quickly generates small augmented topologies, inducing minimal load on routers (§5).

我们之前的工作表明Fibbing可以强制执行任何一组转发DAGs [14]。 本文进一步描述了管理域内路由的Fibbing控制器的完整设计，实现和评估。我们不是专注于特定用例（如流量工程），而是描述了对不同高层次方法的支持（例如[8,9]）。我们做出以下贡献：

**摘要：**我们展示了如何通过操纵分布式链路状态路由协议来表达和实现高层次的转发需求（§2）。

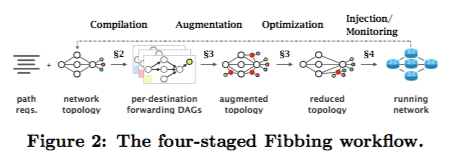
**算法：**我们提出了新的高效算法来计算紧凑增强拓扑（§3）。

**实施：**我们描述了一个完整的Fibbing实施方案，可与未修改的思科和瞻博网络路由器（§4）配合使用。

**评估：**我们展示了我们的Fibbing控制器能够快速生成小型增强拓扑，从而在路由器上减少负载（第5节）。

2. FLEXIBLE FIBBING

Fibbing workflow proceeds in four consecutive stages based on two inputs: the desired forwarding graphs (one Directed Acyclic Graph, or DAG, per destination) and the IGP topology (Fig. 2). The forwarding DAGs can either be provided directly or expressed indirectly, at a high-level, using a simple path-based language. In the latter case, the Compilation (§2) stage starts by com- piling the requirements into concrete forwarding DAGs. Then, the Topology Augmentation (§3) stage computes an augmented topology that achieves these for- warding DAGs. While computing an augmented topology is fast, the resulting topology can be large. As such, the job of the next Topology Optimization (§3) stage is to reduce the augmented topology while preserving the forwarding paths. Finally, the Injection & Monitoring (§4) stage turns fake information into actual “lies” that the controller injects into the network.



2.灵活的FIBBING

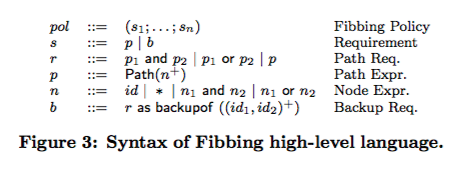
Fibbing工作流基于两个输入连续进行四个阶段：期望的转发图（一个有向无环图或每个目的地的DAG）和IGP拓扑（图2）。转发DAG可以直接提供，也可以使用简单的基于路径的高级语言间接进行表述。在后一种情况下，编译（§2）阶段首先将需求编译为具体的转发DAG。 然后，拓扑增强（§3）阶段计算一个扩展的拓扑，实现这些转发DAG。 虽然计算扩展拓扑的速度很快，但所得到的拓扑结构可能较大。因此，下一个阶段，拓扑优化（§3）的任务是在保留转发路径的同时减少扩展拓扑。 最后，注入和监控（§4）阶段将虚假信息转化为控制器注入网络的实际“谎言”。

In this section, we present the high-level language and compilation process (§2.1), and show that Fibbing can express a wide range of forwarding behaviors (§2.2).

在本节中，我们将介绍高级语言和编译过程（§2.1），并说明Fibbing可以表达广泛的转发行为（§2.2）。

2.1 Fibbing high-level language

Fibbing language (Fig. 3) provides operators with a succinct and easy way to specify their forwarding requirements. A Fibbing policy is a collection of requirements, naturally expressed as regular expressions on paths. Each requirement is either a path requirement which specifies how traffic should flow to a given destination, or a backup requirement which specifies how traffic should flow if the IGP topology changes. Each path requirement is recursively defined as a composition of path requirements through logical AND and OR. Operators can express load-balancing requirements using a conjunction of n requirements. Similarly, they can ensure that traffic to a specific destination will take one of n paths (e.g., containing a firewall) using disjunction. Path requirements are composed of a sequence of node requirements. A node requirement is either a node (router) identifier or the wildcard \*, representing any sequence of nodes. Like path requirements, node requirements can be combined using logical AND and OR. Whenever no path requirement is stated, the original IGP paths should be used. This way, operators do not have to express all the unmodified paths, only deviations from the IGP shortest paths.



2.1 Fibbing高级语言

Fibbing语言（图3）为操作员提供了一个简洁而简单的方式来指定他们的转发需求。 Fibbing策略是一组需求的集合，自然表示为路径上的正则表达式。每个需求要么是指定流量应该流向给定目标的路径要求，要么是指定IGP拓扑更改时流量应该如何流动的备份要求。每个路径需求通过逻辑AND和OR递归地定义为路径需求的组合。操作者可以使用n个要求的联合来表达负载平衡要求。同样，他们使用disjunction确保到达特定目的地的流量将采用n个路径之一（例如，包含防火墙）。路径需求由一系列节点需求组成。节点要求是节点（路由器）标识符或通配符\*，表示任何节点序列。像路径要求一样，节点要求可以使用逻辑“与”和“或”来组合而成。只要没有路径请求，都应该使用原始的IGP路径。这样，运营商不必表达所有未修改的路径，只需要偏离IGP最短路径。

The following example illustrates the main features of the language. It states that traffic between E and D1 should be load-balanced on two paths, traffic between A and D2 should cross B or C and that traffic from F to D3 should be rerouted via G if the link (F,H) fails.



以下示例说明了该语言的主要功能。 它指出E和D1之间的流量应该在两条路径上进行负载平衡，A和D2之间的流量应该跨越B或C，并且如果链路（F，H）发生故障，从F到D3的流量应该通过G重新路由。



Fibbing policies are compiled into per-destination forwarding DAGs by finding convenient network paths (if any). Compilation works in two consecutive stages. First, the compiler expands any requirement with wildcards into paths. This step can be computationally expensive as, in general, a network can have a number of paths exponential in the number of nodes. While this is unlikely, especially for networks designed according to best practices, we bounded the number of paths that can be expanded out of a single requirement. We only expand again if no solution is found with the current set of paths. Once all requirements are expanded, the compiler groups them by destination and computes the Disjunctive Normal Form (DNF) of each requirement. To finally produce a forwarding DAG, the compiler iterates over the disjunction of path requirements and checks whether the resulting graph is loop-free.

通过查找方便的网络路径（如果有的话），Fibbing策略被编译成按目的地转发DAG。 编辑工作连续两个阶段。 首先，编译器将通配符的任何要求扩展为路径。 这个步骤可能在计算上是昂贵的，因为一般来说，网络可以有多个节点数目指数的路径。 虽然这种可能性不大，特别是对于根据最佳实践设计的网络，我们限制了可以从单个需求扩展出来的路径数量。 如果在当前一组路径中找不到解决方案，我们才会再次展开。 一旦所有需求都被扩展，编译器按目的地对它们进行分组并计算每个Disjunctive Normal Form（DNF）。 为了最终生成转发DAG，编译器遍历路径需求的分离并检查结果图是否是无循环的。

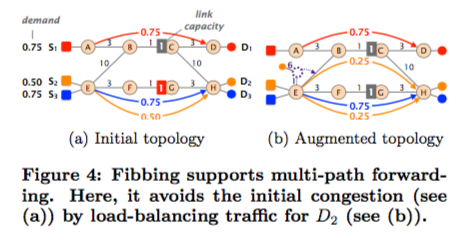
2.2 Fibbing expressiveness

Beyond steering traffic along a given path (§1), we now show that Fibbing can also: (i) balance load over multiple paths and; (ii) provision backup paths.

Fibbing can forward any flow on any set of paths. Fibbing can load-balance traffic over multiple paths to maximize throughput, minimize response time, or in- crease reliability. For example, consider the network in Fig. 4a where three sources S1, S2, and S3 send traffic to three corresponding destinations. Demands and link capacities are such that link (F,G) is congested. One way to alleviate congestion is to split traffic destined to D2 over the top (via (B,C)) and bottom (via (F,G)) paths. Load-balancing traffic coming from E on multiple paths is possible under conventional link-state routing (e.g., by re-weighting links (F,G) to 15. However, this would force the traffic from S2 and S3 to spread over both paths, creating congestion. More generally, it is impossible to route the traffic destined to D2 and D3 on different links under conventional link-state routing. This simple requirement can easily be expressed as:

((S2 ,E,B,C,H,D2) and (S2 ,E,F ,G,H,D2))

Fig. 4b shows the augmented topology which realizes this requirement. A fake node announcing D1 (with a weight of 6) is inserted between E and B. After introducing this node, E has two shortest paths (of cost 7) to reach D2 and, hence, splits D2 traffic over B and F . In this example, Fibbing enables maximum network efficiency as each link is used to its full capacity.



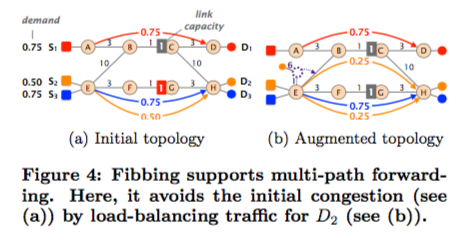
2.2Fibbing的表现

除了沿着给定路径（§1）转向流量之外，我们现在显示Fibbing还可以：（i）平衡多条路径上的负载; （ii）提供备份路径。

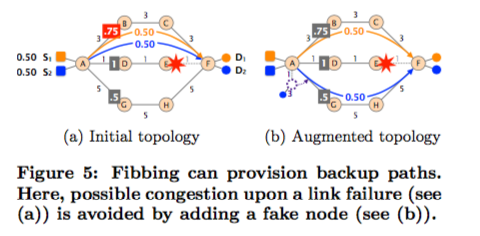
Fibbing可以转发任何路径上的任何流。 Fibbing可以对多个路径上的流量进行负载平衡，以最大化吞吐量，最小化响应时间或提高可靠性。例如，考虑图4a中的网络，其中三个源S1，S2和S3将流量发送到三个相应的目的地。由链路（F，G）的链路容量和需求可知产生了拥塞。缓解拥塞的一种方法是将流向D2的流量分流给顶部（通过（B，C））和底部（通过（F，G））路径。在传统的链路状态路由下（例如，通过将链路（F，G）重新加权到15），可以在多个路径对来自E的流量进行负载平衡。然而，这将迫使来自S2和S3的业务遍布两条路径，造成拥塞，更普遍的是，在传统的链路状态路由下，不可能在不同链路上发送到D2和D3的流量，这个简单的要求可以很容易地表示为：

（（S2，E，B，C，H，D2）和（S2，E，F，G，H，D2））

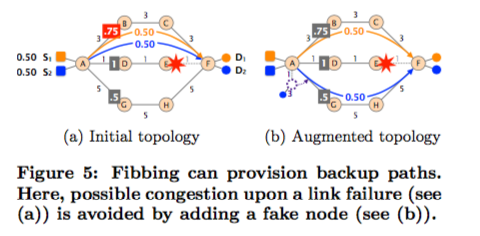
图4b显示了实现这一要求的增强拓扑。在E和B之间插入一个假节点D1（权重为6）。引入该节点后，E有两条到达D2的最短路径（成本为7），因此将D2流量分散到B和F.在这个例子中，Fibbing可以在每个链路达到其全部容量时实现最大的网络效率。



Fibbing can provision backup paths for any flow. Fibbing can provision backup paths to prevent congestion or increased delays after link and node failures. As an illustration, consider the network in Figure 5a. The failure of link (E , F ) leads to congestion since traffic flows for D1 and D2 are both rerouted to the same path via link (A, B). To prevent congestion, traffic des- tined to D1 and D2 should be split over the two remaining disjoint paths but only upon failure on the path (A, D, E, F ). This is another example of a requirement that is impossible to achieve with link-state routing, and would require significant control-plane overhead in MPLS. In contrast, it is easily done with Fibbing.



Fibbing可以为任何流提供备份路径。 Fibbing可以提供备份路径以防止链路和节点故障后的拥塞和延迟。 作为例子，考虑图5a中的网络。 链路（E，F）的故障导致拥塞，因为D1和D2的流都通过链路（A，B）重新路由到相同的路径。 为了防止拥塞，D1和D2的流量应该分散在剩下的两条不相交的路径上，但只有在路径（A，D，E，F）发生故障时。 这是使用链路状态路由不可能实现的另一个要求的示例，并且在控制平面引起显著的MPLS开销。 相反，使用Fibbing很容易完成。



Backup paths can be specified in our language as:

(A ,B ,∗ ,D1 ) and (A ,G ,∗ ,D2 ) asbackupof((A,D) or (D,E) or (E,F))

Fig. 5b shows the corresponding augmented topology, which has a single fake node advertising D2. Weights are set to prevent A from using the fake node to reach D2 unless a failure occurs along the path (A, D, E, F ). While successful for this example, Fibbing cannot satisfy all possible requirements for backup paths (§3.3).

备份路径可以用我们的语言指定为：

(A ,B ,∗ ,D1 ) and (A ,G ,∗ ,D2 ) asbackupof((A,D) or (D,E) or (E,F))

图5b显示了相应的增强拓扑结构，其具有广告D2的单个假节点。 权重设置为防止A使用假节点到达D2，除非路径（A，D，E，F）发生故障。 虽然此示例成功，但Fibbing无法满足备份路径的所有可能要求（第3.3节）。

3. AUGMENTING TOPOLOGY

In this section, we detail the augmentation problem (§3.1), and we show how the Fibbing controller quickly computes small augmented topologies from a set of for- warding DAGs. We rely on a divide-and-conquer approach based on three consecutive steps

**1）Topology initialization (§3.2):** We modify the initial weights in the link-state protocol (if necessary), to guarantee that any set of forwarding DAGs can be enforced by Fibbing. If needed, this operation has to be done only once, when Fibbing is first deployed.

**2) Per-destination augmentation (§3.3):** Starting from an initialized topology, we compute a suitable augmentation, individually for every destination of an in- put forwarding DAG. We designed two algorithms for this step, achieving different trade-offs between computation time and augmentation size. The fastest one, Simple, can compute augmented topologies within milliseconds, and works by injecting a dedicated fake node for every router that changes its next-hop. The relatively slower one, Merger, reduces the augmentation size by re-using the same fake nodes to program multiple routers. Simple and Merger are suited for different goals. The speed of Simple is useful for quick failure re- action. In contrast, Merger can be run in background to progressively re-optimize the augmented topology. We evaluate the trade-offs achieved by each algorithm in §5.

**3) Optimization across destinations (§3.4):** We merge the augmentations obtained in the per-destination augmentation step to further reduce the number of fake nodes and edges. Namely, whenever safe, we replace multiple fake nodes announcing different destinations with a single fake node which either announces all the destinations or creates a new path (a shortcut) between routers in the augmented topology.

3.增强拓扑

在本节中，我们详细介绍了增强问题（§3.1），并展示Fibbing控制器如何快速计算来自一组正在运行的DAG的小增强拓扑。 我们依靠三个连续步骤的分而治之的方法。

**1）拓扑初始化（§3.2）：**我们修改链路状态协议中的初始权重（如果需要的话），以保证任何一组转发DAG可以由Fibbing强制执行。 如果需要，此操作只能在第一次部署Fibbing时进行一次。

**2）按目的地增加（§3.3）：**从初始化的拓扑开始，我们为输入转发DAG的每个目标单独计算一个合适的增强拓扑。 我们为这一步设计了两种算法，实现了计算时间和增量大小之间的不同折衷。 最快的一种Simple可以在毫秒内计算增强拓扑，并且为每个路由器注入专用假节点来改变其下一跳。 相对较慢的方法，Merger，通过重复使用相同的假节点来编程多个路由器来减小拓扑增加。 Simple和Merger适合不同的目标。 Simple的速度对于快速故障反应很有用。 相反，Merger可以在后台运行以逐步重新优化扩展拓扑。 我们评估第5节中每种算法并取得折衷。

**3)跨目标优化(§3.4)：**我们合并了在每个目标扩展步骤中获得的拓扑增强，以进一步减少假节点和假边的数量。也就是说，只要安全，我们就用一个假节点替换多个假节点来宣告不同的目的地，这个假节点要么宣告所有的目的地，要么在增广拓扑的路由器之间创建一条新的路径(快捷方式)。

3.1 The Topology Augmentation Problem

We start the description of the topology augmentation algorithms by precisely defining the basic concepts on which they rely and the problem that they solve.

**Fake nodes scoping.** A Fibbing controller can generate both locally-scoped lies (targeted to a single router) and globally-scoped lies (targeted to all routers). Locally- scoped lies are useful as they enable local actions on one router without creating side effects on other routers. Globally-scoped lies affect the entire network. Hence, if carefully computed, they can reduce the size of the augmented topology. All of our previous examples used globally-scoped lies. We detail how to implement both kinds of lies in the current OSPF in §4.

**Fake edges to forwarding next-hop mapping function.** Fibbing can modify the routing path computed by any IGP router r. In particular, it can augment the IGP topology so that r’s shortest path is no longer the one in the original topology but includes some fake sub-paths. Throughout the paper, we assume that a fake edge in the shortest path from any router r to any destination d corresponds to the ability to force the next-hop of r for d to be any of its neighbors. In the example in Fig. 1, for instance, the fake edge between the A and its adjacent fake node translates into A forwarding traffic to B. We discuss in §4 how to achieve this ability in the current OSPF protocol, as well as in future IGPs.

Topology augmentation problem. Since we assume an arbitrary mapping between fake edges and forwarding next-hops, the topology augmentation problem is defined as follows: Given an initial topology G and a set of forwarding DAGs, compute an augmented topology G′ ⊃ G such that for each path (u,v,...,d) in the forwarding DAG for d, the next-hop of u in one of its shortest paths for d in G′ is either v or a fake node.

3.1拓扑增强解决的问题

我们从精确定义拓扑增强算法所依赖的基本概念和它们所解决的问题开始描述拓扑增强算法。

**假节点范围**。Fibbing控制器既可以生成局部范围的谎言(针对单个路由器)，也可以生成全局范围的谎言(针对所有路由器)。本地范围的谎言是有用的，因为它们在一个路由器上启用本地操作，而不会对其他路由器产生副作用。全局范围的谎言影响整个网络。因此，如果仔细计算，它们可以减小增广拓扑的大小。我们前面的所有示例都使用了全局范围的谎言。我们在第4节中详细介绍了如何在当前的OSPF中实现这两种谎言。

**伪边转发下一跳映射功能。**Fibbing可以修改由任何IGP路由器r计算的路由路径。特别地，它可以扩展IGP拓扑，使r的最短路径不再是原来拓扑中的最短路径，而是包含一些假子路径。在整篇文章中，我们假设从任意路由器r到任意目的地d的最短路径中的伪边有能力迫使r的下一跳成为它的任何邻居。在图中1的例子中，A与其相邻伪节点之间的伪边缘转化为A转发到B的流量。我们在第4节中讨论了如何在当前的OSPF协议中以及在未来的IGP中实现这种能力。

**拓扑扩展解决的问题。**由于我们假设伪边和转发下一跳之间的任意映射，拓扑增强问题定义如下：给定初始拓扑G和一组转发数据集，计算一个扩展拓扑G‘⊃G，使得对于转发数据集d中的每条路径(u，v，…，d)，u在其最短路径之一的下一跳是v或伪节点。

3.2 Topology Initialization

In the topology initialization, we scale the link weights of the original IGP topology G to guarantee arbitrary per-destination control through Fibbing and help re- duce the size of topology augmentations. In particular, we proportionally increase link weights (multiplying them by a constant factor) if they are too low in G. Moreover, we set very high announcement cost for any destination, at least equal to the length of the longest path in G times the maximum link weight.

**Topology initialization enables full Fibbing expressivity.** Indeed, it makes the IGP topology Fibbing compliant, which provably avoids cases in which a forwarding DAG cannot be implemented by Fibbing (see [15]). We say that a topology is Fibbing compliant if for every destination d, the cost of the shortest path from every router (including the ones announcing d) to d exceeds 2. In Fibbing compliant topologies, for any router r and destination d the controller can al- ways compute a fake path P such that (i) P is shorter than the original shortest path from r to d; and (ii) P is longer than the original shortest path from any other router v ̸= r to d. As proved by the following theorem, this implies the ability of Fibbing to forward flows for the same destination on any set of loop-free paths.

3.2拓扑初始化

在拓扑初始化中，通过对原IGP拓扑G的链路权值进行缩放，保证了任意的目的地控制，减少了拓扑增量的大小。特别是，如果G中的链路权重太低，我们会按比例增加链路权重(乘以常量因子)。此外，我们还为任何目的地设置了非常高的通告成本，至少等于G中最长路径的长度乘以最大链路权重。

**拓扑初始化使充分发挥作用。**事实上，它使IGP拓扑兼容Fibbing，这可以证明避免了转发DAG无法通过Fibbing实现的情况(见[15])。如果对于每个目的地d，从每个路由器(包括通知d的路由器)到d的最短路径的成本超过2，那么我们就说拓扑是兼容的。在Fibbing兼容拓扑中，对于任意路由器r和目的地d，控制器总是能够计算假路径P，使得(I)P比从r到d的原始最短路径短；和(Ii)P比从任何其他路由器(v不等于r)到d的原始最短路径长。正如下面的定理所证明的，这表明Fibbing有能力在任意一组无环路径上向同一目的地传送流。

**Theorem 1.** Any set of per-destination forwarding DAGs can always be enforced by augmenting a Fibbing- compliant topology even only with globally-scoped lies.

**Proof.** We prove the statement by showing a simple topology augmentation procedure. Let G be the initial topology. For every forwarding DAG with destination d, we add for each node r in the network a fake node fr announcing d. This generates a new fake path (r,fr,d) in the augmented topology. We set the total cost of this newly added fake path to 2. Since G is Fibbing compliant, then the cost of the shortest path from r to d in G is greater than 2. Hence, the shortest path of every node r in the augmented topology will be (r, fr , d). The forwarding DAG is then implemented by mapping the fake link on the right physical link.

Note that Theorem 1 applies to destinations in the augmented topology. Those destinations do not need to match the destination prefixes announced in the original IGP. Hence, Fibbing allows to control flows for de-aggregation of the original IGP destinations (up to IP address granularity) or even non-overlapping prefixes.

**定理1：**任何一组按目的地转发的DAG都可以通过增加符合谎言的拓扑来强制执行，即使只使用全局范围的谎言。

**证明：**我们给出了一个简单的拓扑扩充过程，证明了这一结论。设G是初始拓扑。对于每个具有目的地d的转发DAG，我们为网络中的每个节点r添加一个假节点fr指向d。这将在增广拓扑中生成一个新的伪路径(r，fr，d)。我们将这个新添加的伪路径的总成本设置为2。既然G是Fibbing兼容的，那么从r到d的最短路径在G中的代价大于2。因此，增广拓扑中每个节点r的最短路径将是(r，fr，d)。然后，通过在正确的物理链路上映射假链路来实现转发DAG。

注意，定理1适用于增广拓扑中的目的地。这些目的地不需要匹配在原始IGP中宣布的目标前缀。因此，FLEBING允许控制流，以解聚合最初的igp目的地(最高IP地址粒度)，甚至是不重叠的前缀。

**Topology initialization is non-intrusive.** Since it is based on the adaptation of a few configurable parameters, our initialization procedure can be applied to any link-state routing configuration, and preserves the original forwarding paths. It can be carried out in a running network, using known lossless reconfiguration techniques [16]. Moreover, it is strictly needed no more than once in the network lifetime. Indeed, since Fibbing compliance does not depend on the routing requirements or the presence of specific links, any topology remains Fibbing compliant independently of new requirements or the failures of nodes or links. Finally, topologies growing in size can be easily kept Fibbing compliant by ensuring that the new destinations are announced with high costs and the new links have weights consistent with pre-existing ones.

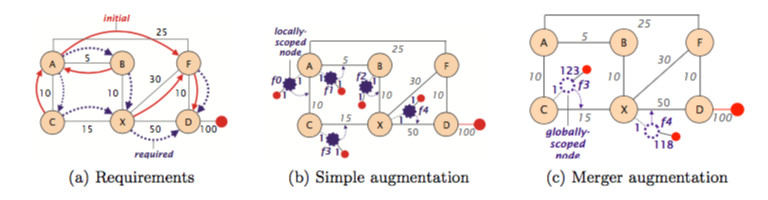
**拓扑初始化是非侵入性的。**由于它是基于一些可配置参数的自适应，我们的初始化过程可以应用于任何链路状态的路由配置，并保留原来的转发路径。它可以在运行的网络中使用已知的无损重构技术来实现[16]。而且，在网络生命周期中只需要一次。事实上，由于FIBBING不依赖于路由要求或特定链路的存在，因此包含FIBBING的任何拓扑都可以独立处理新的要求或节点或链路的故障。最后，通过确保新的目的地以较高的成本公布，并且新的链接的权重与先前存在的链接一致，可以很容易地保持拓扑结构在规模上的兼容性。

3.3 Per-destination augmentation

We now describe Simple and Merger. We use Figure 6 to illustrate the difference of the two algorithms.

3.3每个目标的扩展

我们现在描述Simple和Merger。我们使用图6来说明这两种算法的区别。



3.3.1 Simple

Simple relies solely on locally-scoped lies to avoid having to compute any fake path cost. For every destination d and corresponding forwarding DAG D, the algorithm adds fake nodes to each router whose next-hops in the original topology differ from those in D. Precisely, for every router r that changes its next-hop for d, Simple adds a fake node fr,d and a fake link (r,fr,d). Node fr,d announces d to r with a locally-scoped lie. We set the total cost of path (r, fr,d , d) to 2. Since the topology is Fibbing compliant, r is ensured to change its shortest path. Also, since the lie is locally-scoped, other routers are not affected by it.

Figure 6b shows the output of Simple for the example of Figure 6a. Nodes A, B, C and X are required to change their respective next-hops. Moreover, A needs to load-balance on B and C. Thus, Simple creates five locally-scoped nodes (two connected to A), all providing fake paths to the destination with a cost of 2.

3.3.1Simple

Simple完全依赖于局部范围的谎言，以避免计算任何假路径的成本。对于每个目的地d和相应的转发DAG，D，该算法向原拓扑中下一跳与D不同的每个路由器添加假节点，精确地说，对于每个d改变其下一跳路由器为r，SIMPLE添加一个假节点Frd和假链路(r，fr，d)。节点Frd用本地范围的谎言声明d到r。我们将路径的总成本(r，fr,d，d)设置为2。由于拓扑结构是Fibbing兼容的，因此可以确保r改变的是其最短路径。此外，由于谎言是本地范围的，其他路由器不受其影响。

图6b显示了图6a示例的Simple输出。节点A、B、C和X被要求改变它们各自的下一跳.。此外，A需要平衡B和C的负载。因此，Simple创建了5个本地范围内的节点(两个连接到A)，所有这些节点都以2的代价提供了到达目的地的假路径。

3.3.2 Merger

To reduce the number of fake nodes, Merger relies on globally-scoped lies that can change the forwarding behavior of multiple routers at once. When applied to Figure 6a, Merger creates only two fakes nodes (see Fig. 6c) to change the next-hops of A,B,C and X, instead of the five used by Simple. The added fake nodes create load-balancing on A (cost 134) via B and C, as required.

Merger performs the topology augmentation for any destination d in two phases. First, it adds an excessive number of fake nodes, and computes the lower and up- per bounds for their respective cost. Second, it merges fake nodes whenever possible, based on the value of the computed bounds. We now provide an intuitive description of those two phases. Additional details about them and Merger correctness proofs are reported in [15].

3.3.2 Merger

为了减少假节点的数量，合并依赖于全局范围的谎言，这些谎言可以一次改变多个路由器的转发行为。当应用到图6a时，Merge只创建两个伪节点(参见图6c)改变A、B、C和X的下一跳，而不是简单的使用五个。根据需要，添加的假节点通过B和C在A(成本134)上创建负载平衡。

Merger分两个阶段对任何目标d执行拓扑扩展。首先，它增加了过多的假节点，并计算了它们各自的成本的下界和上界。第二，只要有可能，它就根据计算出的边界值合并假节点。我们现在提供这两个阶段的直观描述。关于它们的更多细节和合并正确性证明见[15]。

**Step 1.** Fake bounds computation. Merger starts by adding fake nodes to every router r that is required to change the next-hop (according to the input forwarding DAG) for d. In this case, one new fake node fr,d is connected to r for every new r’s next-hop in the input forwarding DAGs. However, to enable merging of globally- visible fake nodes, Merger calculates lower and upper bounds for every newly-added fake path (r, fr,d, d). As the initial positioning of fake nodes is as in Simple, we illustrate bound computations referring to Figure 6b.

The upper bound ub(fr,d) represents the maximum value of the fake path cost that changes r’s shortest path to (r, fr,d , d). It is easy to compute by statically considering the non-augmented topology G. Indeed, it is equal to dist(r, d, G) − 1, where dist(u, v, G) is the cost of the shortest path from u to v in G.

The lower bound lb(fr,d) represents the minimal value of the fake path cost that does not change the shortest path of any real node different from r. To compute it, we divide nodes in two sets, depending on whether the input forwarding DAG prescribes to change their respective next-hops or not.

For the next-hop preserving nodes whose shortest path does not traverse r in the input topology, we impose that their original shortest path is not modified by fr,d. For example, when computing lb(f0) in Figure 6b, we ensure that f0 does not change the shortest path of F, by constraining lb(f0) + 25 > dist(F, d, G) = 110, that is, lb(f0) = 86. More generally, for every fake node fr,d and every next-hop preserving neighbor n of r, we impose that lb(fr,d) > dist(n, d, G) − dist(n, r, G).

For next-hop changing nodes (connected to other fake nodes), the final value of their shortest paths is not known in advance, but is determined by the augmen- tation itself. That is, the lower bound of any fake node generally depends on the lower bound of other fake nodes. For example, in Figure 6b, f4 changes the short- est path of X only if lb(f4) < dist(X, C, G) + lb(f3). To avoid that real nodes pass through fake nodes not directly connected to them, Merger runs a lower bound propagation procedure. This procedure takes as input the lower bounds initialized with values from next-hop preserving nodes. It then fixes one lower bound at the time, following a specific order and adjusting the others to be consistent with the fixed one. This order guar- antees that each lower bound must be considered only once. Sometimes, lower bounds cannot be made consis- tent. Indeed, Theorem 1 does not provide guarantees if fake nodes are connected only to next-hop changing nodes. We solve these cases by using locally-visible lie.

**第一步。**假边界计算。Merger首先在每个路由器r中添加假节点，以更改d的下一跳(根据输入转发DAG)。在这种情况下，一个新的假节点fr,d连接到r，每一个新的r的下一跳输入为转发DAG。但是，为了能够合并全局可见的伪节点，Merge计算每个新添加的伪路径(r，fr,d，d)的下界和上界。由于伪节点的初始位置与图6b中的一样简单，因此我们将举例说明边界计算。

上界ub(fr,d)表示将r的最短路径更改为(r，fr,d，d)的伪路径成本的最大值。静态地考虑非增广拓扑G是很容易计算的。实际上，它等于dist(r，d，G)−1，其中dist(u，v，G)是G中从u到v的最短路径的代价。

下界lb(fr,d)表示不改变与r不同的任何真实节点的最短路径的伪路径代价的最小值。为了计算它，我们将节点分成两组，这取决于输入转发DAG是否规定改变它们各自的下一跳。

对于最短路径不经过输入拓扑中的r的下一跳节点，我们强加它们的原始最短路径不会被fr,d修改。例如，当计算图6b中的lb（f0）时，通过约束lb（f0）+25> dist（F，d，G）= 110，我们确保f0不会改变F的最短路径，即lb （f0）= 86。更一般地，对于每个假节点fr,d和每个下一跳保存的r的邻居n，我们规定：lb（fr，d）> dist（n，d，G） r，G）。

对于下一跳转换节点（连接到其他假节点），其最短路径的最终值并不是事先知道的，而是由增量本身决定的。也就是说，任何假节点的下界通常取决于其他假节点的下界。例如，在图6b中，只有当lb（f4）<dist（X，C，G）+ lb（f3）时，f4才会改变X的最短路径。为了避免真正的节点通过不直接连接到它们的假节点，Merger运行下限传播过程。该过程将输入的下界用来自下一跳保存节点的值初始化。然后，它会根据特定的顺序修正其中一个下限，并调整其他顺序以使其与固定顺序一致。这个命令保证每个下限只能被考虑一次。有时候，下界不能一致。事实上，如果假节点仅连接到下一跳改变节点，定理1不提供保证。我们通过使用本地可见的谎言来解决这些情况。

**Step 2.** Fake nodes merging. In this step, Merger tries to merge fake nodes together. More precisely, it iterates over every simple path from a source to d in the input forwarding DAG. For each of those paths, it merges pairs of fake nodes whenever safe.

To assess when it is safe to merge a fake nodes f′ into f′′, Merger sequentially performs three checks. We illustrate these checks by considering the required path (A, B, X, D) and the merge of f1 into f2 in Figure 6b.

First, Merger assesses whether the IGP shortest paths are compliant with the considered source-sink path in the DAG. In our example, it verifies that the shortest path from A and B (i.e., (A,B)) is a sub-path of the required (A, B, X, D). If A had predecessors in the required path not connected to a fake node, this check would have been repeated for those predecessors as well.

Second, Merger checks the possibility to use f′′ as part of the shortest path of the real node connected to f′ without changing the current next-hops of any node. To this end, the algorithm assess the existence of feasible post-merging bounds for f′′. More precisely, it re-computes the modified lower bound of f′′ as the minimum value (if any) that forces the new shortest path of the real node connected to f′ and its next-hop pre- serving predecessors in the required path through f′′, without affecting nodes previously not crossing f′′. In our example, the lower bound of f2 is modified to exclude the constraint of not changing A’s next-hop, hence it is decreased to lb(f2) = 81 (it was greater before, for A’s next-hop to be f1). The upper bound of f′′ is also modified to ensure that the real node connected to f′ and its next-hop preserving predecessors use the fake path via f′′. In our example, ub(f2) is modified to 129, as the cost of the original shortest path from A to the destination is 135 and those between A and B is 5.

Third, Merger simulates the merge to assess whether all bounds can be consistently adjusted network-wide, given that the merging f ′ into f ′′ would change the bounds of f′′ and remove one fake node. To this end, we re-run the lower bound propagation procedure, devoting special attention to fake nodes used for load-balancing. In our example, for instance, we constrain the lower bound of f0 to be equal to the cost of path (A, B, f2, d), meant to be used by A after the merging.

If all the three checks pass, then Merger actually perform the merge, by removing f′ and updating the bounds of all other fake nodes (including f′′) according to the values computed during the last check.

**步骤2。**假节点合并。在这一步中，Merger尝试合并假节点。更确切地说，它在输入转发DAG中迭代从源到d的每条简单路径。对于这些路径中的每一条，只要安全，它就会合并成对的假节点。为了评估何时将伪造的节点f'合并到f'是安全的，Merger依次执行三次检查。我们通过考虑图6b中所需的路径（A，B，X，D）以及将f1合并到f2来说明这些检查。

首先，合并评估IGP最短路径是否符合DAG中所考虑的源宿路径。在我们的例子中，它验证了A和B（即（A，B））的最短路径是所需（A，B，X，D）的子路径。如果A在所需路径中没有连接到假节点的替代者，那么对于那些替代者也会重复该检查。

其次，合并检查使用f“作为连接到f'的真实节点的最短路径的一部分的可能性，而不改变任何节点的当前下一跳。为此，该算法评估f'的可行后合并边界的存在性。更准确地说，它将f“的修改下界重新计算为最小值（如果有的话），该最小值迫使连接到f'的真实节点的新的最短路径及其下一跳预处理所需路径中的前辈通过f''，而不会影响先前不跨越f''的节点。在我们的例子中，f2的下界被修改以排除不改变A的下一跳的约束，因此它减少到lb（f2）= 81（之前更大，对于A的下一跳是f1）。 f'的上界也被修改，以确保连接到f'的真实节点和其保留下一跳的前辈通过f“使用假路径。在我们的例子中，ub（f2）被修改为129，因为从A到目的地的原始最短路径的成本是135，而A和B之间的原始最短路径的成本是5。

第三，Merger模拟合并以评估是否所有边界可以在网络范围内一致地调整，因为将f'合并到f“会改变f”的边界并移除一个假节点。为此，我们重新运行下界传播过程，特别关注用于负载平衡的假节点。例如，在我们的例子中，我们约束f0的下界等于路径（A，B，f2，d）的开销，这意味着在合并之后由A使用。

如果所有三个检查都通过，则Merger实际上执行合并，方法是删除f'并根据上次检查期间计算的值更新所有其他虚假节点（包括f“）的边界。

3.3.3 Dealing with backup requirements

Backup requirements can be specified by providing additional sets of (tagged) forwarding DAGs to our algorithms. Let G′ be an augmented topology computed to accommodate primary requirements. To deal with backup ones, slight variants of Simple and Merger can be run after the computation of G′. The main modification of Simple consists in setting the cost of the fake paths added for backup requirements to 3 instead of 2. In contrast, backup requirements are supported in Merger by imposing that lower bounds are always greater than the cost of the shortest path in G′.

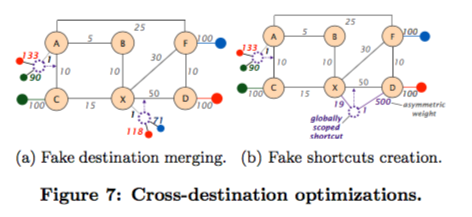
Contrary to primary requirements (see Theorem 1), Fibbing may not enforce backup requirements, even in a Fibbing-compliant topology (see [15]). Indeed, if the cost of the original shortest path of a node r is equal to a fake one (used for a primary requirement) in G′, then backup paths different from the original shortest path cannot be implemented on r. In those cases, we notify the operator on the impossibility to implement the given backup paths.

3.3.3处理备份需求

备份需求可以通过向我们的算法提供额外的（标记的）转发DAG集来指定。设G'为扩展拓扑结构，以适应主要需求。为了处理备份数据，可以在计算G'之后运行Simple和Merger的轻微变体。 Simple的主要修改包括将为备份需求添加的虚拟路径成本设置为3而不是2.相比之下，Merger中支持备份需求的方式是强制下限总是大于G中最短路径的成本“。

与主要要求（见定理1）相反，Fibbing可能无法强制执行备份要求，即使在符合Fibbing的拓扑中（请参见[15]）。事实上，如果节点r的原始最短路径的成本等于G'中的假最短路径（用于主要需求），那么不同于原始最短路径的备份路径不能在r上实现。在这些情况下，我们通知操作员无法实施给定的备份路径。

3.4 Cross-Destination Optimization



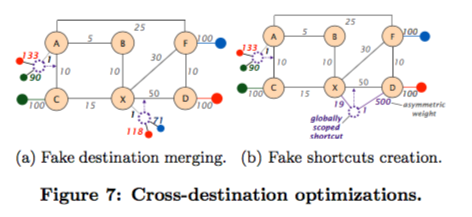
Fake nodes computed on a per-destination basis may be redundant. We reduce such redundancy in two ways, namely, by (i) merging fake nodes connected to the same real node, and (ii) replacing fake destination announcements with fake paths connecting real nodes.

**Cross-destination merging.** After per-destination augmentations, two fake nodes f1 and f2 announcing different destinations d1 and d2 can be connected to the same node r and used to force traffic to the same real link (r, n). Those fake nodes can always be merged. In- deed, we can replace f1 and f2 with a new fake node f′ such that (i) cost(r, f ′ ) = min{cost(r, f1 ), cost(r, f2 )}, and (ii) f′ announces both d1 and d2, with cost(f′, di) = cost(r,fi,di)−cost(x,f′) for i = 1,2. For example, assume that in the network in Figure 6 additional destinations are attached to C and F as in Figure 7a. The result of the cross-destination merging is shown in Figure 7a, where both A and X have a single fake neighbor announcing multiple destinations (rather than multiple fake neighbors each announcing a single destination). This reduces the number of fake nodes from 4 to 2.

**Creating shortcuts.** One of the most appealing features (unmatched by competitor solutions) of Fibbing is that a single lie can change the paths for multiple destinations. To this end, we need however to replace fake destination announcements with fake paths connecting real nodes together, i.e., fake shortcuts.

Currently, we use fake shortcuts only if a real link is never traversed in different directions. Consider, for example, the link between X and D in Figure 7a. It is traversed from X to D for the destinations attached to D and F, and never from D to X. In those cases, we try to transform X’s fake neighbor into a fake shortcut, as in Figure 7b. Let u and v be the two real nodes at the endpoints of the shortcut. First, we check if a shortcut cost c exists such that all the shortest paths1 are kept the same with and without the shortcut. If this condition is met, given a fake node f used to enforce the subpath (u,...,v) in the input forwarding DAGs, all fake destinations announced by f are replaced by a fake shortcut (u, f, v) and the cost of (u, f, v) is set to c. Figure 7b illustrates that we found such a value c = 20 for X in our example. Also, we use asymmetric weights to prevent the fake shortcut from being traversed in the opposite direction. To this end, the cost of path (v, f, u) is set to a very high value, e.g., by setting a high weight of the directed link (v,f). In Figure 7b, we indeed set the weight of the link from D to the fake node in the shortcut to 500.

3.4交叉目标优化



以每个目的地为基础计算的假节点可能是多余的。我们通过两种方式来减少这种冗余，即（i）合并连接到相同真实节点的虚假节点，以及（ii）用连接真实节点的假路径代替虚假目的地公告。

**跨目标合并。**在每个目的地增加之后，两个假的节点f1和f2通告不同的目的地d1和d2可以连接到相同的节点r并用于强制通信到相同的实际链路（r，n）。那些假节点总是可以合并。（i）成本（r，f'）= min {cost（r，f1），cost（r，f2）}和（ii）成本）f'指向d1和d2，其中成本（f'，di）=成本（r，fi，di） - 成本（x，f'），对于i = 1,2。例如，假设在图6的网络中，附加的目的地被连接到C和F，如图7a所示。交叉目的地合并的结果如图7a所示，其中A和X都有一个伪邻居指向多个目的地（而不是多个假邻居，每个通告一个目的地）。这将虚假节点的数量从4个减少到2个。

**创建快捷方式。** Fibbing最吸引人的特点之一（竞争者解决方案无法匹敌）是单一谎言可以改变多个目的地的路径。为此，我们需要用虚假路径替换假目的地公告，假道路径将真实节点连接在一起，即假快捷键。

目前，我们只有在真正的链接从未在不同方向上穿过时才使用假快捷键。例如，考虑图7a中X和D之间的关系。对于连接到D和F的目的地，它从X到D被遍历，并且从不从D到X.在这些情况下，我们试图将X的假邻居转换成假快捷方式，如图7b所示。让u和v成为快捷方式终点处的两个真正的节点。首先，我们检查是否存在一个快捷方式成本c，使所有最短路径在有快捷路径和没有快捷路径保持相同。如果满足这个条件，给定一个伪节点f用于强制输入转发DAG中的子路径（u，...，v），则由f宣告的所有虚假目的地将为虚假快捷方式（u，f，v）并且（u，f，v）的成本被设置为c。图7b说明我们在例子中发现X的值为c = 20。此外，我们使用非对称权重来防止伪造的快捷方式在相反的方向上移动。为此，例如通过设定有向链路（v，f）的高权重，路径（v，f，u）的成本被设置为非常高的值。在图7b中，我们确实将快捷方式中从D到假节点的链接权重设置为500。

4. IMPLEMENTATION

We built a complete prototype of Fibbing in Python (algorithmic part) and C (interaction with OSPF) by extending Quagga [17]. Fibbing code base spans over 2300 (resp. 400) lines of Python (resp. C) code. It is available at http://www.fibbing.net. In this section, we present our prototype (§4.1), describe how Fibbing works with current OSPF routers (§4.2), and propose two small modifications to link-state protocols that would make Fibbing even more efficient (§4.3). Finally, we describe how to ensure controller reliability (§4.4).

4.实施

通过扩展Quagga [17]，我们在Python（算法部分）和C（与OSPF的交互）中构建了一个完整的Fibbing原型。Fibbing代码库覆盖Python（C语言）代码的2300行（400行）。 它可以在http://www.fibbing.net上找到。 在本节中，我们将介绍我们的原型（第4.1节），描述Fibbing如何与当前的OSPF路由器（第4.2节）协同工作，并提议对链接状态协议进行两项小修改，使Fibbing更加高效（第4.3节）。 最后，我们描述如何确保控制器的可靠性（§4.4）。

4.1 Fibbing Controller

Our prototype consists of three main components:

Fake topology generator applies (i) the compilation algorithms (§2) to turn forwarding requirements into forwarding DAGs and (ii) the augmentation algorithms (§3) to convert the forwarding DAGs into fake nodes and links. The topology generator uses a JSON interface to register for update events produced by the event manager. Upon network updates, the generator automatically recomputes the augmented topology either using Simple to ensure fast convergence, or Merger to reduce the size of the topology. To ensure fast convergence and a small augmented topology, the genera- tor also pre-computes augmentations with Merger, e.g., those needed for any single link failure, and stores them in a deltas database.

Link-state translator interacts with the routers by establishing routing adjacencies to inject lies and track topology changes. Thanks to the flooding mechanism used by link-state protocols, a single adjacency is sufficient to send and receive all routing messages to/from all routers. Though, maintaining several adjacencies is useful for reliability. In this case, the translator simply injects the lies via all adjacencies. While this slightly increase the flooding load, doing so does not impact the routers memory as each message has a unique identifier and routers only maintain one copy per ID in memory.

Event manager maintains an update-to-date view of the network topology by (i) parsing the routing messages collected by the translator and (ii) constructing a network graph for the topology generator. The event manager checks whether each new event (e.g., a node failure or addition) affects any of the forwarding requirements. If so, it first checks the deltas database for a pre-computed lie, and otherwise notifies the topology generator to request a new augmented topology.

4.1Fibbing控制器

我们的原型由三个主要组件组成：

**假拓扑生成器**（i）编译算法（§2）将转发需求转化为转发DAG，和（ii）扩展算法（§3）将转发DAG转换为假节点和链接。拓扑生成器使用JSON接口来注册由事件管理器生成的更新事件。在网络更新时，生成器会自动重新计算扩展拓扑，可以使用Simple来确保快速收敛，也可以使用Merger来缩小拓扑的大小。为了确保快速收敛和小的拓扑增加，该生成器还可以预先计算合并的增量（例如，任何单链路故障所需的增量），并将它们存储在delta数据库中。

**链路状态转换器**通过建立路由邻接关系来注入谎言并跟踪拓扑变化，从而与路由器进行交互。由于链路状态协议使用泛洪机制，单个邻接就足以发送和接收所有路由器发来的所有路由消息。尽管如此，维护几个邻接点对于可靠性非常有用。在这种情况下，翻译者只是通过所有邻接来注入谎言。虽然这会稍微增加泛洪负载，但这样做不会影响路由器内存，因为每条消息都有唯一的标识符，并且路由器仅在内存中为每个ID保留一个副本。

**事件管理器通过**（i）解析由翻译器收集的路由消息和（ii）为拓扑生成器构建网络图来维护网络拓扑的更新视图。事件管理器检查每个新事件（例如，节点故障或附加）是否影响任何转发需求。如果是这样，它首先检查deltas数据库中的预先计算的谎言，否则通知拓扑生成器请求新的扩充拓扑。

4.2 Fibbing with Unmodified OSPF

Our Fibbing prototype works with unmodified OSPF- speaking routers (tested on Cisco and Juniper). To creates lies, our prototype leverages the Forwarding Ad- dress (FA) [18] field of OSPF messages. Suppose the controller wants routers to think that destination d is directly attached to the router with IP address y. Then, the controller injects the route for d with a forwarding address of y and the desired cost for the fake edge from y to d. Router y ignores the message, and all other routers compute the cost of the route as the sum of their cost to y plus the cost in the message.

**Locally-scoped lies in OSPF.** To support locally- scoped lies, we reserve a set of IP addresses to be used as FAs. All those addresses are propagated network-wide in OSPF. However, every router is configured not to in- stall routes to all those addresses. Consequently, only the directly-connected router can reach any of those ad- dresses, and accept routes specifying that address as FA. Both allocation and configuration of FA-associated IP addresses can be done just once in the network lifetime.

**Globally-scoped lies in OSPF, and limitations.** OSPF readily supports globally-scoped lies by simply propagating OSPF messages with the FA set to an IP address announced in OSPF. However, some subtle constraints hold in an unmodified OSPF network due to how FAs are resolved on the router. Prominently, (i) OSPF fake nodes are actually fake routes to the specified FA, hence both their positioning and the path to be used for reaching them are constrained; and (ii) OSPF routers discard any OSPF message whose FA is one of its own IP address and computes its shortest path ac- cording to the topology without the fake node. The combination of these two constraints limit the power of globally-scoped lies in OSPF, making them insufficient to implement all possible forwarding DAG. An example of those cases is reported in [15].

**Overcoming OSPF limitations.** To support the full expressiveness of Fibbing, our prototype controller uses an OSPF-compliant implementation of Merger with a combination of locally and globally-scoped lies. This implementation uses globally-scoped lies whenever possible, and falls back to locally-scoped lies for any requirements that cannot be met that way.

4.2使用未修改的OSPF进行Fibbing

我们的Fibbing原型与未经修改的OSPF路由器（在思科和瞻博网络上测试）协同工作。为了创造谎言，我们的原型利用了OSPF消息的转发地址（FA）[18]字段。假设控制器希望路由器认为目的地d直接连接到IP地址为y的路由器。然后，控制器将转发地址为y的d的路由以及从y到d的假边所需的成本注入到d中。路由器y忽略该消息，并且所有其他路由器将路由的成本计算为它们对y的成本加上消息成本的总和。

**本地范围在OSPF。**为了支持本地范围的谎言，我们保留一组IP地址用作FA。所有这些地址都在OSPF网络范围内传播。但是，每个路由器都配置为不将路由安装到所有这些地址。因此，只有直接连接的路由器才能到达这些地址中的任何一个，并接受将该地址指定为FA的路由。 FA关联的IP地址的分配和配置都只能在网络生命周期内完成一次。

**全局范围在于OSPF和局限性。**OSPF通过简单地传播OSPF消息来支持全局范围的谎言，其中FA设置为OSPF中公布的IP地址。但是，由于FA在路由器上的解析方式，未修改的OSPF网络中存在一些微妙的约束条件。显而易见的是，（i）OSPF假节点实际上是指向指定FA的假路由，因此它们的定位和用于达到它们的路径都受到限制；（ii）OSPF路由器丢弃FA为自己IP地址之一的任何OSPF消息，并按照没有假节点的拓扑结构计算其最短路径。这两个约束的组合限制了OSPF中全局范围的谎言的能力，使得它们不足以实现所有可能的转发DAG。这些案例的一个例子在[15]中报道。

**克服OSPF限制。**为了支持Fibbing的全面表现力，我们的原型控制器使用兼容OSPF的Merger实现，结合了本地和全局范围的谎言。这个实现尽可能使用全局范围的谎言，并回退到本地范围的谎言，以满足任何无法在全局谎言满足的需求。

4.3 Proposed Protocol Enhancements

Other link-state protocols, like IS-IS [6], do not sup- port forwarding addresses, and even the OSPF implementation of lies has limitations. However, minor protocol extensions can enable more flexible Fibbing in future routers. Fully-fledged Fibbing needs for the routing protocol to support two functions: (i) the creation of adjacencies (with fake nodes) on the basis of a received message; and (ii) a third-party next-hop mechanism which allows to specify in a route the forwarding next-hop to be used if that route is selected. Support for these functions can be added to protocol specifications (without impacting current functionalities), and can be deployed through router software updates.

Preliminary discussions with router vendors confirm that these changes are reasonable and could be integrated into current protocol implementations. More- over, backwards compatibility can easily be achieved as legacy routers would simply ignore any Fibbing-specific protocol features. Our algorithms can be modified to account for the fact that the shortest path of a legacy router is never changed by any fake node.

4.3对现有协议的建议

其他链路状态协议，如IS-IS [6]，不支持转发地址，甚至连谎言的OSPF实施都有局限性。但是，次要协议扩展可以在未来的路由器中实现更灵的Fibbing。对于路由协议，完全成熟的Fibbing需求支持两种功能：（i）基于收到的消息创建邻接点（带有假节点）;和（ii）第三方下一跳机制，其允许在路由中指定如果选择该路由将使用的转发下一跳。支持这些功能可以添加到协议规范中（不影响当前功能），并且可以通过路由器软件更新进行部署。

与路由器供应商的初步讨论确认这些更改是合理的，并可以集成到当前的协议实现中。此外，向后兼容性很容易实现，因为传统路由器会忽略任何Fibbing特定的协议功能。我们的算法可以进行修改，以解决传统路由器的最短路径永远不会被任何假节点更改的事实。

4.4 Controller Replication

As any network component, a Fibbing controller can fail at any time. Reliability can be ensured by running multiple copies the Fibbing controller in parallel and connecting them to different places in the network.

No state needs to be synchronized between the replicas besides the input forwarding requirements (mostly static anyway). Indeed, Fibbing algorithms are deterministic, hence all replicas will always compute exactly the same augmented topology. The only dynamic state maintained by a Fibbing replica is the network graph. This state, however, is implicitly synchronized through the shared topology offered by the underlying IGP: the link-state flooding mechanism keeps the network graph up to date and eventually consistent across all replicas.

The determinism of our algorithms enables all replicas to inject (the same) lies at the same time. However, this would increase the amount of flooded information.

To limit control-plane overhead, our implementation relies on a primary-backup architecture with a single active replica and an inexpensive election process. A pre-defined range of router IDs is reserved for controller replicas. The replica with the lowest router ID across all running ones is the active replica. It is the only one injecting lies, while the others only compute the topology augmentation. When the controller is booted or when an active replica fails, running replicas receive IGP messages on the current network topology. Based on those messages, every replica independently infers the possible presence of other replicas; it also checks whether it is the new active replica by comparing its router ID with the one of the other running replicas.

4.4控制器复制

作为任何网络组件，Fibbing控制器可能随时失败。通过并行运行Fibbing控制器的多个副本并将它们连接到网络中的不同位置，可以确保可靠性。

除了输入转发要求（主要是静态的）之外，任何状态都不需要在副本之间进行同步。事实上，Fibbing算法是具有确定性的，因此所有副本将始终计算完全相同的增强拓扑。由Fibbing复制品维护的唯一动态状态是网络图。但是，此状态通过基础IGP提供的共享拓扑隐式同步：链路状态泛洪机制使网络图保持最新，并最终在所有副本中保持一致。

我们算法的确定性使得所有副本可以同时注入（相同）。但是，这会增加洪水信息的数量。

为了限制控制平面的开销，我们的实现依赖于具有单个活动副本和廉价选举过程的主备份体系结构。预定义的路由器ID范围保留给控制器副本。在所有正在运行的路由器ID中具有最低路由器ID的副本是活动副本。它是唯一注入谎言的人，而其他人只计算拓扑增强。当控制器启动或活动副本失败时，运行副本将在当前网络拓扑上接收IGP消息。基于这些消息，每个副本独立推断可能存在其他副本;它还通过比较其路由器ID与其他正在运行的副本之一来检查它是否是新的活动副本。

5. EVALUATION

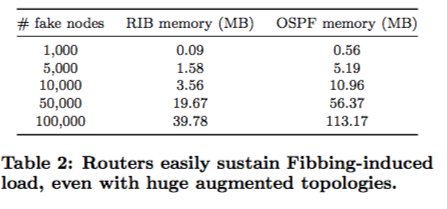
We now evaluate Fibbing along three axis. First, we show that existing routers are perfectly capable of handling the extra load induced by Fibbing (§5.1). We then demonstrate the efficiency of Fibbing’s augmentation algorithms in terms of speed and size of the topology (§5.2). Observe that Fibbing behaves as a plain IGP at the network level. Hence, given its negligible impact on single routers and the efficiency of our controller, current ISP networks can be seen as the best large-scale evaluation for Fibbing. We therefore complete the evaluation by illustrating how Fibbing can be used in a realistic case (§5.3).

5.评估

我们现在从三个方向评估Fibbing。 首先，我们证明现有路由器完全能够处理由Fibbing引起的额外负载（第5.1节）。 然后，我们以拓扑的速度和大小（§5.2）来展示Fibbing增强算法的效率。注意到Fibbing在网络级别上表现为普通的IGP。因此，鉴于其对单路由器的影响可以忽略不计，而且我们的控制器的效率很高，目前的ISP网络可以看作是对Fibbing的最佳大规模评估。 因此，我们通过说明如何在实际情况下使用Fibbing来完成评估（第5.3节）。

5.1 Router measurements

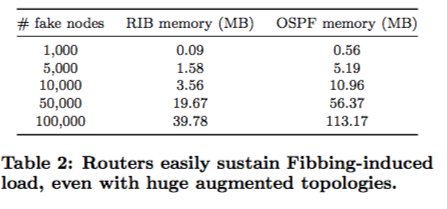
By increasing the size of the link-state routing topology, Fibbing could increase the CPU and memory over- head on the routers, or slow down protocol convergence. Our experiments demonstrate that the impact on load and convergence time is negligible. All our measurements were performed using OSPF on a recent Cisco ASR9K running IOS XR v5.2.2 equipped with 12GB of DRAM assigned to the routing engine, as well as on a (7- year-old) Juniper M120 running JunOS v9.2, equipped with 2GB of DRAM. Both routers are representative of typical edge devices (i.e., aggregation routers) found in commercial networks. We draw the same conclusions on both router platforms, and focus on measurements collected on the Cisco device in the following.



Fibbing induces very little CPU and memory overhead on routers. We first measured the memory increase caused by a growing number of fake nodes (Table 2). Two processes are impacted by the presence of fake nodes: (i) the RIB process, which maintains information about all the routes known to each destination, and (ii) the OSPF process which maintains the entire OSPF topology. Even with a huge number of fake nodes (100,000), the total overhead on both processes was only 154MB—a small fraction of the total memory available. We collected the CPU utilization on the router every minute immediately after we started injecting fake nodes. The utilization was systematically low, at most 4%. This is easily explained as Fibbing relies on OSPF Type-5 LSAs which do not cause the routers to recompute their shortest paths to each other.

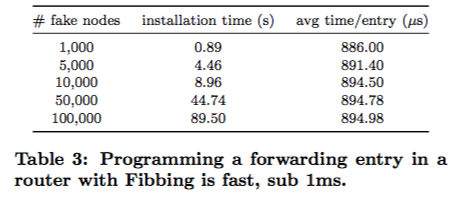
5.1路由器测量

通过增加链路状态路由拓扑的大小，Fibbing可以增加路由器的CPU和内存，或者减慢协议的收敛速度。我们的实验表明，对负载和收敛时间的影响可以忽略不计。我们所有的测量都是在运行IOS XR v5.2.2的最新Cisco ASR9K上运行的，配备12GB DRAM分配给路由引擎，以及运行JunOS v9.2的（7岁）瞻博网络M120上运行。与2GB的DRAM。两个路由器都是商业网络中典型的边缘设备（即聚合路由器）的代表。我们在两个路由器平台上得出了相同的结论，并重点关注在以下Cisco设备上收集的测量结果。

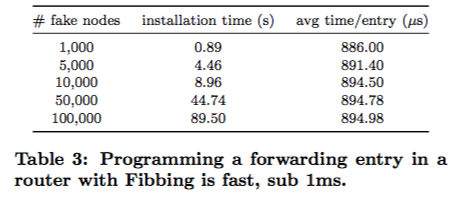


Fibbing在路由器上引起很少的CPU和内存开销。我们首先测量了由越来越多的假节点引起的内存增加（表2）。两个进程受到假节点存在的影响：（i）RIB进程，它维护关于每个目的地已知的所有路由的信息;（ii）维护整个OSPF拓扑的OSPF进程。即使有大量假节点（100,000个），两个进程的总开销也只有154MB，这是可用内存总量的一小部分。我们在开始注入假节点后立即每分钟收集一次路由器上的CPU利用率。利用率较低，最多4％。这很容易解释，因为Fibbing依赖于OSPF Type-5 LSA，它们不会导致路由器重新计算彼此的最短路径。

Fibbing can quickly program forwarding entries. In a second experiment, we measured how long it took for a router to install a growing number of forwarding entries (Table 3). We injected a growing number of fake nodes, one per destination, and measured the total installation time, by tracking the time at which the router updated the last entry in its FIB. The time to process and install one entry was constant (around 900μs), independent of the number of entries. This result is several orders of magnitude better than any OpenFlow switches currently on the market [12, 13]. Since installation of forwarding entries is distributed, routers can install their entries in parallel, meaning Fibbing can pro- gram thousands of network-wide entries within 1 second.



Fibbing可以快速编程转发条目。在第二个实验中，我们测量了路由器安装越来越多的转发条目需要多长时间（表3）。 我们通过追踪路由器更新其FIB中最后一个条目的时间，为每个目的地注入了越来越多的假节点，并测量了总安装时间。 处理和安装一个条目的时间是固定的（大约900μs），与条目数量无关。 这个结果比目前市场上的任何OpenFlow交换机好几个数量级[12,13]。 由于转发条目的安装是分布式的，因此路由器可以并行安装条目，这意味着Fibbing可以在1秒内编辑数千个网络范围的条目。



**Fibbing does not have any impact on routing protocol convergence time.** Finally, we compared the total time for routers to converge with and without fake nodes. We failed a link and measured the time for the last FIB entry to be updated considering two cases: (i) no lie was injected and (ii) one lie per destination was injected. Similar to the previous experiments, we repeated the measurements for a growing number of destinations and lies, between 100 and 100,000. In all our experiments, the presence of lies did not have any visible impact. The total convergence times with or without lies were systematically within 4ms, with the router being even faster to converge in the presence of lies in some cases.

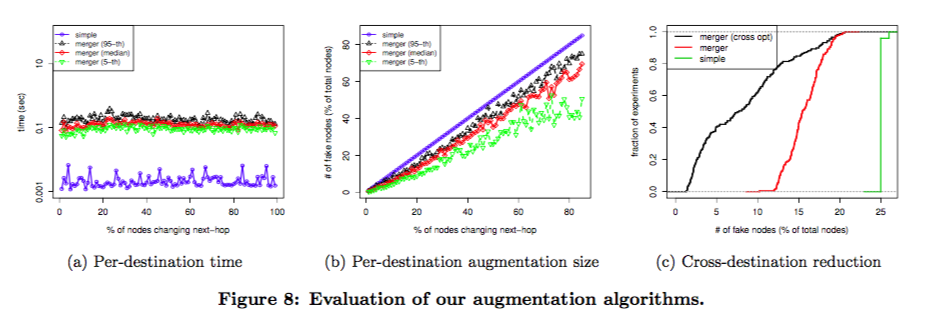
Fibbing对路由协议收敛时间没有任何影响。最后，我们比较了有和没有假节点的路由器汇聚的总时间。考虑到两种情况，我们没有链接并且测量了最后FIB条目的更新时间：（i）没有撒谎，并且（ii）每个目的地被注入一个谎言。 与之前的实验类似，我们重复测量数量在100到100000之间的越来越多的目的地和谎言。 在我们所有的实验中，谎言的存在并没有任何明显的影响。 有或没有谎言的总收敛时间系统地在4ms以内，在某些情况下，路由器甚至可以在存在谎言时更快地收敛。

5.2 Topology Augmentation Evaluation

We now evaluate Simple and Merger (§ 3) according to: (i) the time they take to compute an augmented topology for a given requirement, and (ii) the size of the resulting augmented topology. Results are depicted in Fig. 8. Our evaluation is based on simulation performed on realistic ISP topologies [19], whose sizes range from 80 nodes to over 300. On these topologies, we generated forwarding requirements by randomly changing the next-hop of randomly selected nodes. Destinations of requirement DAGs were also randomly generated.

Fibbing augments network topologies within ms. Fig. 8a shows the time (on the y-axis) taken by Simple and Merger for an increasing number of nodes that must change their next-hop (on the x-axis). The plot refers to simulations we ran on the biggest Rocketfuel topology (AS1239). The time taken by Simple to compute the per-destination augmentation varies in the order of milliseconds, ranging from 0.5 ms to 8 ms. While Merger took more time (as expected), its performance is still one order of magnitude lower than the second. For both algorithms, the computation time does not vary much with the number of nodes changing their next-hops.

Merger and cross-optimization effectively reduce the size of the augmented topology. Fig. 8b plots the fake topology size (on the y-axis) when the number of nodes that have to change their next-hop increases (on the x-axis) for a single destination on all topologies. The plot shows that Merger reduces the number of fake nodes by about 25% in the average case and al- most 50% in the best case. Fake topology reduction is further corroborated by our cross-destination optimization procedures (see §3.4). Fig. 8c shows a cumulative distribution function (CDF) of the topology augmentation size computed by Simple, Merger, and Merger with cross-destination optimization. The figure refers to simulations with a number of destinations varying between 1 and 100 with 26% of the nodes changing their nexthop. In more than 90% of our simulations, cross-destination optimization achieves a reduction of the augmented topology. Depending on the experiment, such a reduction is up to about 10% with respect to Merger without cross-destination optimization, and 20% with respect to Simple.



5.2拓扑增强评估

我们现在根据以下方式评估简单和合并（第3节）：（i）计算给定要求的扩展拓扑所用的时间，以及（ii）所得到的增强拓扑的大小。结果如图8所示。我们的评估是基于在现实ISP拓扑结构[19]上进行的仿真，其大小范围从80个节点到300多个。在这些拓扑中，我们通过随机更改随机下一跳选定的节点。需求DAG的目的地也是随机产生的。

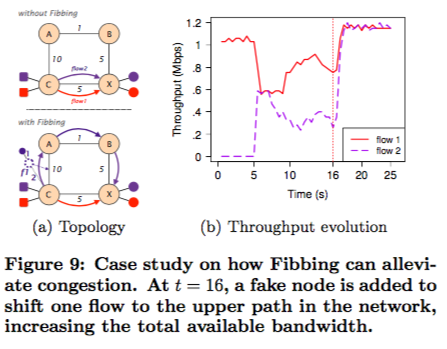
Fibbing在ms内扩展了网络拓扑。图8a显示了Simple和Merger对越来越多的必须改变其下一跳（在x轴上）的节点的时间（在y轴上）。该图是指我们在最大**节点度**拓扑（AS1239）上进行扥模拟。 Simple计算每个目的地增加所用的时间按毫秒级别变化，范围从0.5毫秒到8毫秒。虽然合并需要更多时间（如预期），但其表现仍低于第二个数量级。对于这两种算法，计算时间不会随着节点数量的变化而改变其下一跳。

合并和交叉优化有效地减小了扩展拓扑的大小。图8b绘制了当所有拓扑上的单个目的地必须改变其下一跳的节点的数量（在x轴上）时，虚拟拓扑大小（在y轴上）。该图表明，合并在平均情况下将虚假节点数量减少了大约25％，在最佳情况下减少了大约50％。我们的交叉目标优化程序进一步证实了伪拓扑的减少（见§3.4）。图8c示出了通过Simple，Merger和Merger通过交叉目的地优化计算的拓扑增大尺寸的累积分布函数（CDF）。该图指的是具有1到100之间的多个目的地的模拟，其中26％的节点改变它们的下跳。在90％以上的仿真中，交叉目标优化实现了增强拓扑的减少。取决于实验，这种减少对于没有交叉目的地优化的Merger而言高达约10％，而对于Simple则为20％。

5.3 Case Study

We now show the practicality of Fibbing by improving the performance of a real network consisting of four routers (Cisco 3700 running IOS v12.4(3)) connected in a square with link of 1 Mbps capacity (see Fig. 9a). In this network, we introduce two sources (bottom left) that send traffic to two destinations (bottom right) using iperf. The first source is introduced at time t = 0, the second one at time t = 5. OSPF weights are configured such that all traffic flows along link (C,X).

Such a network suffers from two inherent inefficiencies: (i) the upper path is never used and (ii) the two flows systematically traverse the same path, competing for bandwidth, no matter what the link weights are. Fig. 9b plots the throughput of each flow. Immediately after the introduction of the second flow at t = 5s, the two flows start competing for the available bandwidth. To improve network efficiency, the Fibbing controller injects a fake node f1 connected to C and announces one destination at time t = 16. A few ms after the injection, we see that the throughput of both flows double as each of them now traverses a different path.



5.3案例研究

我们现在通过改进由4个路由器（Cisco 3700运行IOS v12.4（3））组成的真实网络的性能来展示Fibbing的实用性，该路由器以1 Mbps容量的链路连接在一个正方形上（参见图9a）。在这个网络中，我们引入了两个源（左下角），使用iperf将流量发送到两个目的地（右下角）。第一个源在时间t = 0时引入，第二个源在时间t = 5时引入。OSPF权重配置为使得所有业务沿着链路（C，X）流动。

这样的网络存在两个固有的低效率：（i）从不使用上层路径，以及（ii）两个流有系统地遍历相同的路径，争夺带宽，而不管链路权重如何。图9b绘出了每个流量的吞吐量。紧接在t = 5s引入第二个流程之后，两个流程开始竞争可用带宽。为了提高网络效率，Fibbing控制器注入连接到C的伪节点f1，并在时间t = 16宣布一个目的地。在注入后几毫秒，我们看到两个流的吞吐量翻倍，因为它们中的每一个现在遍历不同的路径。

6. REACTION TO FAILURES

We now analyze Fibbing’s reaction to different kinds of failures. We distinguish between network (affecting real router or router-to-router links) and controller (shutting down replicas or replica-to-router links) failures. Also, we separately deal with failures inducing network partitions and non-partitioning ones.

Fibbing quickly reacts to non-partitioning failures. Upon network failures, forwarded flows fall in one of the following three cases. First, some flows are not impacted by the failure as their pre-failure forwarding path is not disrupted. Second, flows for which no input requirements have been specified require only the IGP to establish a new path, but no action from the Fibbing controller. Reaction to failures is extremely fast in this case (sub-second even in large networks), thanks to fast convergence [20] and local fast re-route [21] features commonly supported by current IGP implementations. Third, the remaining flows are forwarded on paths mod- ified by the Fibbing controller. They require reaction from the controller, both to remove possible blackholes or loops due to previously injected lies [14], and to avoid requirement violations due to new IGP paths. Theoretically, the total failure reaction time is equal to the sum of the notification time (for the controller to be notified of the failure), the processing time (for the controller to compute the new topology augmentation) and the IGP convergence time (for all routers to install the new lies). Our evaluation (§5) shows that the processing time is negligible, especially for the Simple algorithm. Moreover, the notification time is bounded by the IGP convergence time, as flooding is faster than re-convergence. Thus, in the worst case, the total re- action time is twice the IGP convergence time, that is, still below 2 seconds [20]. Also, in the average case, the notification time is smaller than IGP convergence, because the controller is notified about the failure be- fore all other routers complete convergence; hence, the controller injects new lies during the IGP convergence, and the total failure reaction time is slightly higher than IGP convergence without Fibbing.

In addition, if one or more controller replicas fail but others are running, we have no impact on forwarded flows, unless the failed replica is the active one and some of its injected lies expire before the new active replica is elected. Even in the latter case, the new active replica is quickly elected, in a time which is approximately equal to the detection and flooding of the failure event by the IGP. The short election time makes it unlikely that lies expire before the new active replica is elected, and limits the period with possible disruptions.

6.对失败的反应

我们现在分析Fibbing对各种失败的反应。我们区分网络（影响真正的路由器或路由器到路由器链路）和控制器（关闭复制品或复制品到路由器链路）失败。另外，我们分别处理导致网络分区和非分区的故障。

  Fibbing快速响应非分区故障。在网络故障时，转发的流量属于以下三种情况之一。首先，一些流量不受故障影响，因为它们的故障前转发路径不会中断。其次，没有指定输入要求的流量只需要IGP建立新的路径，但不需要Fibbing控制器的动作。由于当前IGP实施通常支持快速收敛和本地快速重新路由功能，这种情况下对故障的反应非常快（即使在大型网络中也是亚秒级）。第三，剩余流量在由Fibbing控制器修改的路径上转发。它们需要来自控制器的反应，以消除由于先前注入的谎言而导致的可能的黑洞或环路[14]，并且避免由于新的IGP路径导致的需求违反。理论上，总的故障反应时间等于通知时间（用于要通知故障的控制器），处理时间（用于控制器计算新拓扑增加）和IGP收敛时间（对于所有路由器来安装新的谎言）。我们的评估（§5）显示处理时间可以忽略不计，特别是对于Simple算法。此外，通知时间受IGP收敛时间限制，因为洪泛比重新收敛快。因此，在最坏的情况下，总的反应时间是IGP收敛时间的两倍，即仍然低于2秒[20]。而且，在平均情况下，通知时间小于IGP收敛，因为控制器会在所有其他路由器完成收敛之前收到故障通知;因此，控制器在IGP收敛期间注入新的谎言，并且总失败反应时间略高于没有Fibbing的IGP收敛。

此外，如果一个或多个控制器副本发生故障，但其他控制器副本发生故障，但其他副本正在运行，则对转发的流不会造成影响，除非故障副本是活动副本，并且其中一些注入的谎言在选举新的活动副本之前过期。即使在后一种情况下，新的活动副本也很快被选中，这个时间大约等于IGP对故障事件的检测和泛滥。选举时间短使得谎言不太可能在选举新的活动副本之前到期，并且会限制可能发生中断的时间段。

**Fibbing can implement both fail-open and fail- close semantics to deal with partitions.** Even if unlikely, catastrophic events like a simultaneous failure of all the controller replicas or network partitions may happen. As for any centralized solution, a major risk in those cases is to leave the network uncontrolled. This happens, for example, if some routers are not reach- able by a controller replica after a network partition. With respect to pure SDN solutions, Fibbing has the additional possibility to delegate control to the underlying IGP. This way, Fibbing can implement both the fail-open or fail-close semantics, on a per-destination basis. For non-critical (optimization) requirements like traffic engineering ones, the corresponding destinations can be injected in the IGP, so that connectivity can be preserved as long as the partition leaves at least one source-destination path. For stringent requirements like security ones (e.g., firewall traversal), Fibbing can implement fail-close semantics by not announcing the corresponding destinations in the IGP. As such, the corresponding flows stop to be forwarded in the absence of the controller. To quickly reach this configuration, we can set a low validity time of the injected lies, making them rapidly expire if not refreshed. This then comes at the cost of additional control-plane overhead.

Fibbing可以实现失效打开和失效关闭语义来处理分区。即使不太可能发生灾难性事件，例如所有控制器副本或网络分区同时发生故障。至于任何集中式解决方案，这些情况下的主要风险是使网络不受控制。例如，如果某些路由器在网络分区后无法通过控制器副本访问，则会发生这种情况。就纯SDN解决方案而言，Fibbing具有将控制委托给基础IGP的额外可能性。这样，Fibbing就可以在每个目的地的基础上实现失效打开或失效关闭语义。对于非关键（优化）要求（如流量工程），相应的目标可以注入到IGP中，以便只要分区至少保留一个源—目标路径，就可以保持连接。对于严格的安全要求（例如防火墙遍历），Fibbing可以通过不公布IGP中的相应目的地来实现故障关闭语义。如此，相应的流程在没有控制器的情况下停止转发。为了快速达到这个配置，我们可以设置一个低注入时间的有效时间，使它们在不刷新的情况下快速过期。这是以额外的控制平面开销为代价的。

**We confirmed Fibbing resilience.** We consider again the topology in Figure 9a, and we connect two controller replicas respectively to routers A and B. The active replica is initially the one connected to A. We assume a strict policy on the red flow forcing it to cross the link (C,X). We then configure a fail-close semantics to it, and a fail-open to the other flow. Starting from a state in which both replicas and all links are up, we successively fail (i) the active replica at time t = 5; (ii) link (A,B) at t = 12; and (iii) link (B,X) at t = 20. Finally, we reestablish both failed links, one at the time (at t = 36 and t = 48).

**我们证实了Fibbing弹性。** 我们再次考虑图9a中的拓扑，并且我们将两个控制器副本分别连接到路由器A和B。主动副本最初是连接到A的副本。我们假设严格的红色流策略强制它通过链路（C ，X）。 然后，我们为它配置失效关闭语义，并为其他流程失效打开。 从副本和所有链接都处于启动状态开始，我们连续失败（i）时间t = 5时的活动副本; （ii）在t = 12时链接（A，B）; 和（iii）在t = 20时链接（B，X）。最后，我们重新建立两个失效链接（在t = 36和t = 48）。

The results of this experiment, collected via iperf, are reported in Figure 10. Concretely, the failure of the active replica has no impact on the forwarded flows. Indeed, the initially passive replica (connected to B) quickly detects the failure of the other replica, and start refreshing the injected lies by the failed controller. When (A, B) fails, the active replica needs to remove the fake node f1: Since the physical path (C, A, B, X) is not available anymore, this fake node is creating a loop be- tween C and A for the violet flow. Upon failure detection, the controller then sends the LSA to remove f1, re-establishing the connectivity for the disrupted flow in approximately 1s. Note that this time can be lowered by relying on fast failure detection mechanisms (like BFD). When (B,X) also fails, we create a partition that makes it impossible for the running replica to interact with routers A, C, and X. After about 1s, the injected lies disappear, because they are not refreshed anymore by any controller. Consistent with the con- figured failure semantics, the red flow is blackholed (to avoid the IGP routing it over policy-violating paths) while the violet flow keeps using the IGP shortest path. Finally, re-adding the failed links allows the running replica to re-take control of the network: it re-builds a (safe) path for the red flow upon (B,X) restoration, and re-optimizes the distribution of both flow over the available paths when (A, B) is restored.

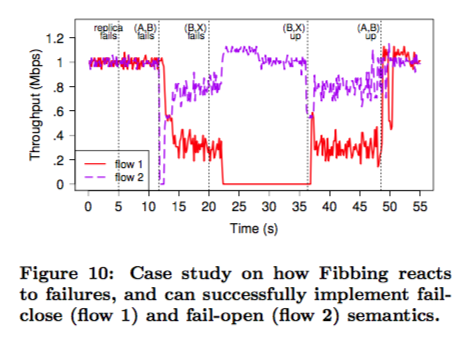


图10中报告了通过iperf收集的此实验的结果。具体而言，活动副本的故障对转发的流程没有影响。事实上，最初的被动副本（连接到B）可以快速检测到其他副本的故障，并通过失败的控制器开始刷新注入的谎言。当（A，B）失败时，活动副本需要删除假节点f1：由于物理路径（C，A，B，X）不再可用，此假节点正在创建C和A之间的循环为violet流。在检测到故障后，控制器将发送LSA以删除f1，并以约1秒的时间重新建立中断流的连接。请注意，这个时间可以通过依靠快速故障检测机制（如BFD）来降低。当（B，X）也失败时，我们创建一个分区，使正在运行的副本无法与路由器A，C和X进行交互。大约1秒后，注入的谎言消失，因为它们不再被任何控制器刷新。与配置的故障语义一致，红色流将被黑掉（以避免IGP路由到违反策略的路径上），而紫色流则继续使用IGP最短路径。最后，重新添加失败的链接允许正在运行的副本重新控制网络：它在（B，X）恢复时为红色流重新构建（安全）路径，并重新优化两者的分布，当（A，B）恢复时。

7. FREQUENTLY ASKED QUESTIONS

We now provide answers to high-level concerns of- ten raised against Fibbing. Since empirical analyses are hardly applicable to those concerns (e.g., debuggability), we describe qualitative considerations.

7.常见问题

我们现在提供针对Fibbing的高频关注问题的答案。 由于经验分析几乎不适用于这些问题（例如可调试性），因此我们描述了定性考虑因素。

**Is Fibbing a long-term solution?**Yes. We believe Fibbing is here to stay. In the short run, Fibbing of- fers programmability and is easy to deploy, at very lit- tle cost. A network that ultimately needs even greater flexibility could deploy finer-grained SDN functional- ity at the edge, and solutions like Fibbing in the core, as advocated by major industry [22] and academic ac- tors [10, 23]. By combining the best of centralized and distributed routing, Fibbing fits the needs of the net- work core (flexibility, robustness, low overhead) better than current forwarding paradigms.

**Fibbing是一个长期的解决方案吗？**是的。我们相信Fibbing会留在这里。在短期内，Fibbing改进了可编程性，并且易于部署，成本非常低。 最终需要更大灵活性的网络可以在边缘部署更细粒度的SDN功能，以及核心中的Fibbing等解决方案，正如主要行业[22]和学术人员[10,23]所倡导的那样。通过结合最佳的集中式和分布式路由，Fibbing能够比当前的转发范例更好地满足网络核心（灵活性，健壮性，低开销）的需求。

**Does Fibbing make networks harder to debug?** No. Fibbing relies on “tried and true” protocols. This has several implications. First, Fibbing routing matches the current mental model of operators, a major advantage with respect to other SDN proposals. Moreover, Fibbing is compatible with any existing management, monitoring, and debugging tools. Finally, the Fibbing controller can expose a higher-level interface for debugging, including a mapping between the injected lies and their usage (matched requirements and how).

**Fibbing是否会使网络难以调试？** 不。Fibbing依赖于“经过验证和真实”的协议。 这有几个含义。 首先，Fibbing路由匹配运营商的当前智能模型，这是相对其他SDN提议的主要优势。 此外，Fibbing与任何现有的管理，监控和调试工具兼容。 最后，Fibbing控制器可以公开更高级别的接口进行调试，包括注入谎言与其使用情况（匹配的需求和方式）之间的映射。

**Does Fibbing sum the complexities of centralized and distributed approaches?** No. Fibbing uses the underlying IGP in a very simple way. The IGP output is easy to predict and provides the controller with a powerful API to program routers. As a result, the design of the Fibbing controller is significantly simpler than for existing SDN controllers (e.g., [24, 25, 26, 27]) since heavy tasks such as path computation and topology maintenance are offloaded to the routers. Even basic primitives for controller replication and replica consistency are mainly delegated to current distributed routing protocols (see §6).

**Fibbing是否兼具集中式和分布式方法的复杂性？** 不会. Fibbing以非常简单的方式使用底层的IGP。 IGP输出很容易预测，并为控制器提供了一个强大的API来编程路由器。 因此，Fibbing控制器的设计比现有的SDN控制器（例如[24,25,26,27]）简单得多，因为诸如路径计算和拓扑维护之类的繁重任务被卸载到路由器。 甚至控制器复制和副本一致性的基本原语也主要委托给当前的分布式路由协议（请参阅第6节）。

**Does Fibbing impact security?** No. The lies introduced by the Fibbing controller can easily be authenticated, e.g., using MD5-based authentication [28, 29].

**Fibbing是否会影响安全性？** 不会。由Fibbing控制器引入的谎言可以很容易地进行身份验证，例如，使用基于MD5的身份验证[28,29]。

**Since Fibbing can only program loop-free paths, can it support middleboxes chaining?** Partially. Forwarding loops can be encountered when steering traffic through a chain of middleboxes (e.g., [30] and [31]). These requirements can be satisfied in Fibbing with local support from routers to break the loops. For in- stance, a router could match on the input interface in addition to the destination IP address using policy- based routing, a feature widely available on existing routers [32, 33] and provisioned centrally using BGP flowspec [34, 35]. Alternatively, middlebox traffic steering could be implemented through SDN functionality at the network edge, while still using Fibbing in the core.

**由于Fibbing只能编程无回路的路径，它可以支持中间件链吗？**部分。在通过一系列中间盒（例如[30]和[31]）转向流量时，可能会遇到转发循环。这些要求可以在Fibbing中通过路由器的本地支持来解决循环。 例如，除了目标IP地址之外，路由器还可以使用基于策略的路由在输入接口上进行匹配，这是现有路由器广泛使用的一项功能[32,33]，并使用BGP flowspec [34,35]集中进行配置。 另外，中间件流量转向可以通过网络边缘的SDN功能实现，同时仍然在核心中使用Fibbing。

8. RELATED WORK

Fibbing contributes to the larger debate about centralized and distributed control over routing by identifying a new point in the design space.

8.相关工作

     通发现别设计领域的新点，Fibbing为集中式和分布式路由控制问题做出了较大的贡献。

**Centralized configuration of distributed routing protocols:** A centralized management system can perform traffic engineering by optimizing the link weights in link-state routing protocols [36, 37]. Fibbing is more general, since it can implement any forwarding paths by injecting fake nodes and links into the link-state routing topology. The extra flexibility enables even better load balancing, as well as a wider range of functionality.

**分布式路由协议的集中式配置：**集中管理系统可以通过优化链路状态路由协议中的链路权重来执行流量工程[36,37]。 Fibbing更通用，因为它可以通过将假节点和链接注入链路状态路由拓扑来实现任何转发路径。额外的灵活性可实现更好的负载平衡，以及更广泛的功能。

**Centralized control using existing routing protocols as a control channel:** RCP [11] is a logically- centralized platform that uses BGP to install forwarding entries into routers. RCP must install forwarding entries one-by-one, on each device. In contrast, Fibbing can adapt the forwarding behavior of many routers at once, with little input (e.g., one fake node), and let them compute their own forwarding entries.

**使用现有路由协议作为控制通道的集中控制：**RCP [11]是一个逻辑集中的平台，它使用BGP将转发条目安装到路由器中。 RCP必须在每台设备上逐个安装转发条目。相比之下，Fibbing可以一次适应多个路由器的转发行为，只需很少的输入（例如，一个虚拟节点），并让他们计算自己的转发条目。

**Centralized control over the routing/forwarding tables:** In SDN, a central controller installs packet- processing rules directly in the switches, possibly reacting to the reception of specific packets. While more flexible (e.g., enabling stateful control logic) than Fibbing, SDN requires updating the switch-level rules one- by-one, and forgoes the scalability and reliability benefits of distributed routing. Recently, the IETF developed I2RS [38] which offers a new management interface for centralized updates the routing information bases (RIBs) in the routers. Still, I2RS must push RIB entries individually to each router.

**对路由/转发表的集中控制：**在SDN中，中央控制器直接在交换机中安装分组处理规则，可能会对特定分组作出更优的反应。虽然比Fibbing更灵活（例如，启用有状态控制逻辑），但SDN需要逐个更新交换机级规则，并放弃分布式路由的可扩展性和可靠性优势。最近，IETF开发了I2RS [38]，它提供了一个新的管理接口，用于集中更新路由器中的路由信息库（RIB）。不过，I2RS必须将RIB条目单独推送到每个路由器。

The Fibbing language for expressing requirements is similar in spirit to Merlin [39], but the mechanism for satisfying the requirements (i.e., fake nodes/links) is entirely different. Our main contributions are the Fibbing techniques and algorithms, not the language.

用于表达需求的Fibbing语言在精神上类似于Merlin [39]，但满足需求的机制（即假节点/链接）完全不同。 我们的主要贡献是Fibbing技术和算法，而不是语言。

For the networks which require the extra flexibility provided by OpenFlow, Fibbing helps during the transition by providing access to the FIBs of legacy routers to any SDN controller [40]. This contrasts to techniques like Panopticon [41], where programmability is only available in the SDN-enabled parts of the network.

对于需要OpenFlow提供额外灵活性的网络，Fibbing通过为任何SDN控制器提供对传统路由器的FIB的访问来帮助转换[40]。 这与Panopticon [41]等技术形成了鲜明对比，其可编程性仅适用于网络中支持SDN的部分。

9. CONCLUSIONS

The advent of SDN makes it clear that network operators want their networks to be more programmable and easier to manage centrally. In this paper, we show how Fibbing can achieve those objectives, by centrally and automatically controlling forwarding without forgoing the benefits of distributed routing protocols. Fibbing is expressive, scalable, and works with existing routers. In future work, we plan to look at extensions of IGP proto- cols (e.g., for source-destination routing [42] or network service header awareness) to enable finer-grained control via Fibbing. Abstractly, Fibbing shows how centralized and distributed approaches can be profitably combined. We believe that new research can further explore this direction, for example, investigating an alternative division of tasks between centralized and distributed network components.

SDN的出现清楚地表明，网络管理者希望他们的网络更具可编程性，更易于集中管理。 在本文中，我们展示了Fibbing如何实现这些目标，通过集中和自动控制转发而不必放弃分布式路由协议的好处。 Fibbing具有表现力，可扩展性，可与现有路由器配合使用。在未来的工作中，我们计划研究IGP协议的扩展（例如，源 - 目的地路由[42]或网络服务头部感知），以通过Fibbing实现更细粒度的控制。抽象地说，Fibbing展示了集中式和分布式方法如何可以进行有益的组合。我们相信新的研究可以进一步探索这个方向，例如，研究集中式和分布式网络组件之间的其他任务分工。