PEERING: Virtualizing BGP at the Edge for Research

Brandon Schlinker[†] Todd Arnold[‡] Italo Cunha[♮‡] Ethan Katz-Bassett[‡]

[†] University of Southern California [‡] Columbia University [♮] Universidade Federal de Minas Gerais

ABSTRACT

Internet routing research has long been hindered by obstacles to executing the wide class of experiments necessary to characterize problems and opportunities, and evaluate candidate solutions. Prior works proposed a platform that would provide experiments with control of an Internet-connected AS. However, because BGP does not natively support multiplexing or the requisite security policies for building such a platform, prior works were ultimately unable to realize this vision.

We present Peering, a community platform that provides multiple parallel experiments with control and visibility equivalent to directly operating a production AS. Peering is built atop vBGP, our design for virtualizing the data and control planes of a BGP edge router while simultaneously enforcing security policies to prevent experiments from disrupting the Internet and each other. With Peering, experiments operate in an environment qualitatively similar to that of a cloud provider, and can exchange routes and traffic with hundreds of neighboring networks and the broader Internet at locations around the world. To date, Peering's rich connectivity and flexibility have enabled it to support over 40 experiments and 15 publications in key research areas such as security, traffic engineering, and routing policies.

CCS CONCEPTS

• **Networks → Network architectures**; **Programmable networks**; *Network security*; *Network manageability*; *Public Internet*.

KEYWORDS

Research platform, virtualization, BGP, experimentation, routing

ACM Reference Format:

Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. 2019. PEERING: Virtualizing BGP at the Edge for Research. In *CoNEXT '19: International Conference On Emerging Networking Experiments And Technologies, December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3359989.3365414

1 INTRODUCTION

Recent changes to the Internet's topology are renewing interest in the routing ecosystem, as large content and cloud providers build out private intercontinental backbones and interconnect directly with thousands of peer networks [26, 29, 55, 81, 85, 104]. This change is spurring innovation at Internet eXchange Points (IXPs) [12, 43, 44]. The growth in connectivity is seen as an opportunity to improve performance, but the improvements come at a cost: increased complexity of network configuration and traffic engineering, accompanied by increased security risks [87]. Due to the shortcomings of the Border Gateway Protocol (BGP), the protocol responsible for inter-Autonomous System (AS) communication, content and cloud providers build sophisticated, customized controllers and measurement systems to handle the additional complexity [80, 81, 100, 104].

Researchers and network operators are well aware of BGP's limitations and their impact on performance [44, 47, 60, 81, 101], availability [54, 66, 104], and security [69, 87], but progress towards overcoming these challenges is slow. A significant barrier to exploring solutions is that BGP does not lend itself well to supporting experimentation. Emulation and simulation cannot accurately model the Internet due to the lack of transparency in BGP and the proprietary nature of routing policies [28, 53, 54, 59, 86]. Existing tools that provide visibility into the current state of BGP [42, 73, 76] or perform measurements [27, 62, 67, 72] cannot interact with the routing ecosystem, so they can only provide limited insight into the current policies and connectivity of an AS [13, 49].

To gain better insight into how solutions will perform, experiments need to interact with and affect the Internet's routing ecosystem. Interacting with the actual routing ecosystem would require researchers to *take control* of a *real* production AS and its resources: connectivity, policies, and traffic. Few network administrators are willing to allow experimentation on a production network due to the potential wide-ranging, negative effects [75], and few researchers have the resources required to deploy a network with a footprint similar to that of a large cloud or content provider.

Resource multiplexing and virtualization have repeatedly come to the rescue in similar scenarios to provide researchers with access to necessary resources: EmuLab, CloudLab, and XSEDE provide access to compute resources [2, 3, 10], PlanetLab shares machines around the world [8], and FlowVisor enables multiplexing layer 2 networks [84]. No equivalent for BGP exists.

Multiplexing control of a BGP router's interactions with other networks introduces control and security challenges. A BGP router applies policy and makes routing decisions locally, routes all traffic to a destination via a single "best" route, and only informs other routers of (at most) that single option which limits visibility of available connectivity. For an experiment to change a policy or decision would traditionally require manually modifying the router's configuration; granting that ability is equivalent to giving root access to experiments, which is untenable from a security perspective.

Transit Portal [96] and our earlier workshop paper [82] shared our vision of multiplexing the connectivity, traffic, and policies of an AS, with the workshop paper further proposing the creation of a community AS and platform to support Internet routing research experiments. However, neither work had fully developed or implemented the technologies required to multiplex an AS. As a

CoNEXT '19, December 9–12, 2019, Orlando, FL, USA

ACM ISBN 978-1-4503-6998-5/19/12…$15.00
https://doi.org/10.1145/3359989.3365414

result, both systems had limited fidelity and flexibility. For instance, the designs proposed made use of non-standard mechanisms, such as out of band signaling for route selection, that are incompatible with standard routers, BGP implementations, and tooling used in production networks. In addition, limitations in their approaches to multiplexing made it impossible to support a variety of experiments, such as those requiring fine-grained control over which BGP announcements and traffic are sent to which neighbors of the AS.

Further, operating a community AS and platform that enables turn-key Internet routing research presents a number of operational challenges. Infrastructure and tooling must be developed to model and actualize the configurations required to support experiments, maintain safety, and manage interconnections—all of which are key to enabling the platform to safely scale. The platform must also be able to safely evolve over time as researchers identify new capabilities required to execute experiments. The prototype designs and implementations of prior work [82, 96] required considerable manual intervention to support each experiment and were unable to reliably support or scale to the needs of a community platform (§2.3).

This paper presents vBGP, a framework for virtualizing the data and control planes of a BGP router (§3). Akin to a hypervisor multiplexing resources across VMs, vBGP virtualizes a router's data and control plane interactions with other networks, delegating them to multiple experiments (§3.2). It provides control and visibility equivalent to if each experiment had its own (non-virtual) router with a BGP session to each neighbor, and provides safety by interposing between experiments and the Internet on both planes (§3.3).

vBGP is the first approach to delegate control of a BGP router to experiments running as BGP routers themselves, including the ability for parallel experiments to specify routing decisions at a *per-packet* level. We use a novel combination of IP and layer 2 manipulation and intradomain BGP advertisements to expose data and control plane interfaces (§3.2). Because the mechanisms are protocol compliant, they are fully compatible with existing routers and BGP implementations: experiments that run on vBGP are directly transferable to native networks, and vice versa.

We use vBGP as the foundation of Peering (§4), a globally distributed AS open to the research community with routers, or Points of Presence (PoPs), at thirteen locations (§4.2). Each Peering PoP connects with at least one AS on the Internet, a subset of PoPs are connected to tens or hundreds of other ASes via IXPs, and a subset are interconnected via a backbone network (§§ 4.3 and 4.4). Peering provides experiments with turn-key access to a global AS (§4.6) with connectivity qualitatively similar to that of cloud or content providers and is capable of supporting any exchange of routes and traffic that an experiment could perform with dedicated control over the Peering infrastructure. Peering employs strict security policies on both the data and control planes to prevent experiments from disrupting the Internet (§4.7). We employed a principled approach to development, testing, deployment, and configuration management that eases operation and supports extensibility (§5). Our current software stack can be deployed at even the largest IXPs for the foreseeable future on off-the-shelf servers (§6). Peering has been used to support 15 publications to date [13, 15, 18–20, 38, 57, 69, 79, 81, 83, 87–90] (§7).

2 GOALS AND CHALLENGES

2.1 Goals

Our overarching goal is the design, implementation, and deployment of a platform for routing experimentation that can delegate control of a real AS to researchers, allowing them to exchange routes and traffic with real networks on the Internet. We decompose this high-level goal into the following subgoals:

Maintain safety. An experiment should be prevented from disrupting other experiments, the platform, and, critically, the broader Internet. BGP allows for disruptive behaviors including prefix hijacks, route leaks, interception attacks, blackholes, routing oscillations, and spoofed traffic. It is a challenge even for experts to design BGP configurations that operate as intended [35, 39, 61], so the platform must prevent even well-intentioned experiments from causing problems.

Allow parallel experiments. To support long-running studies, iteratively-refined experiments, and the synchronized demand before conference submission deadlines, the platform should support parallel experiments while isolating them. It should do so without compromising the degree of control given to experiments, and without requiring coordination between experiments or administrators.

Support a wide range of experiments. Since we cannot anticipate the full range of experiments researchers may want to run, our goal is a flexible platform that provides researchers with the same control over the data and control planes as they would have operating their own network (subject to the safety requirements). The platform should be able to support the following (and more):

- Supporting experiments that use existing routers, to allow fidelity and transitioning of experiments from the platform to non-multiplexed environments.
- Allowing experiments to host services (*e.g.,* an HTTP or DNS server) which are accessible from the Internet.
- Supporting settings qualitatively similar to content or cloud providers, with PoPs at geographically diverse locations, including IXPs with many interconnections, and a backbone that interconnects PoPs with data centers, since this setting is increasingly important to academia and industry [26, 29, 43, 44, 55, 81, 104]. This complex setting will also suffice for a range of experiments not specific to cloud providers.

2.2 Native Delegation with BGP and IP

Achieving our high-level goal requires developing an approach for multiplexing and delegating control of an AS to experiments. In developing our approach, we consider that (1) the interface between the platform and the Internet must be BGP and IP, since those are the protocols used by every AS, and (2) to flexibly support a wide range of experiments, experiments should be able to perform any (safe) action that they could do with direct control of an AS using standard protocols, and should be able to use standard routing implementations.

As such, we posit that *the interface between experiments and the platform should also be BGP and IP:* an experiment should get visibility by receiving BGP announcements and IP traffic, and it

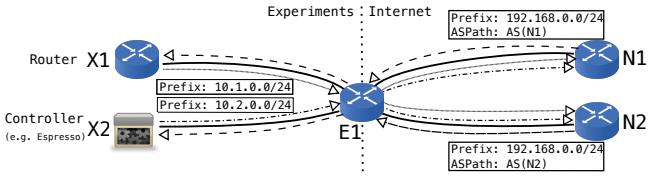


Figure 1: Basic scenario for what our platform should support: two parallel experiments (X1, X2) competing to use the resources of a single BGP edge router (E1). E1 connects to two neighbor routers (N1, N2). Both X1 and X2 announce prefixes, while N1 and N2 announce a path for the same prefix and E1 selects N1's path.

should control its announcements and route its outgoing traffic just as it would with direct control of the router.

However, the design of BGP presents a number of challenges to using it as an interface—in particular, BGP does not natively support multiplexing or delegating control. Understanding how BGP's design complicates multiplexing and delegation requires understanding the basic design of BGP, how BGP makes routing decisions, how those decisions impact IP forwarding, and the interactions between BGP speakers. In this section, we discuss these topics in detail. In Section 2.3 we discuss why, despite these challenges, native delegation is a superior option to other approaches, such as custom protocols and out-of-band interfaces.

2.2.1 BGP basics. BGP conveys reachability information to other BGP speakers while providing network operators with the ability to control route selection and propagation via static policies [102]. Each BGP router makes decisions locally based on information received from its neighbors and defined policy. By design, BGP's route selection process ensures the router selects a single "best" route for each prefix, regardless of the number of routes available. Once the router selects the single route, it may inform its BGP neighbors of the single path it selected according to its export policies. From this point forward, the router forwards all packets destined for the prefix along the single path.

2.2.2 Why delegating with native BGP and IP is challenging. In this section, we examine why delegating control without requiring a separate interface is challenging. Consider the scenario illustrated in Figure 1, in which a single edge router (E1), has two neighbors (N1 and N2). The envisioned platform should be able to support delegating visibility and control to two experiments (X1 and X2). Further, our design should be able to accommodate various types of experiments without having to customize an interface for each. For instance, in our example scenario, X1 is an experiment using a standard software router and making BGP announcements to uncover backup routes [13], and X2 is evaluating the benefits of a more sophisticated routing control system, such as Espresso [104], and thus requires flexible per-packet forwarding.

Challenges in controlling announcements. In our example, each experiment is assigned a prefix to announce, 10.1.0.0/24 for X1 and 10.2.0.0/24 for X2. If each experiment had direct control of E1, it would be able to define policies to control what it announced to each neighbor on a per-prefix basis. For instance, experiment X1 could manipulate the AS-path to perform prepending [25] or BGP poisoning [21] for announcements forwarded to N1, and perform a different set of manipulations (or none at all) for announcements forwarded to N2. Likewise, X1 could decide to only announce a route to a subset of neighbors (*e.g.,* just N1).

However, standard BGP advertises *at most* a single path for each destination to neighbors, and thus X1 can only advertise a single route to E1. By default, controlling announcement propagation or modifying announcement attributes would require configuration changes at E1, which does not meet our goal of providing experiments with dynamic control.

Challenges in controlling packet forwarding. In our example, both neighbors announce a route to the same destination (192.168.0.0/24). Again, if each experiment had direct control of E1, it could define policies to control which route is used. For instance, experiment X2 could choose to send a subset of its traffic via the route provided by N1, and the rest via N2.

However, per BGP's default behavior, E1 will select a route (in this case, the route through N1), forward only this route to experiments, and route all traffic via the chosen route. There is no native mechanism in BGP that can be used to allow earlier hops (such as X1 or X2) to signal how they want E1 to route traffic to the destination.

The ADD-PATH option [98] solves part of this problem, but is not a complete solution. While standard BGP advertises *at most* a single path to neighbors, ADD-PATH allows a BGP speaker to advertise multiple routes. However, ADD-PATH does not provide a method for experiments to override E1's local decision to forward all traffic destined for the Internet via N1, and thus while ADD-PATH extends visibility, it does not delegate control. This is because ADD-PATH is primarily intended for scenarios where there is value in learning multiple routes, such as when an aggregator (a route reflector or route server) collects each route from a different router to pass on as a collection. In such a scenario, the aggregator is on the control plane but not the data path, and a route is selected by forwarding traffic on a distinct data path. This scheme does not work for our scenario, when E1 must be on the data path for both routes. It is not feasible to deploy a distinct router for each neighbor, especially at IXPs with hundreds of neighbors.

2.3 Alternative Approaches to Delegation

The previous section proposed using native BGP and IP to multiplex and delegate control to experiments and explored the associated challenges. In this section, we explore alternative approaches and explain why we do not choose to pursue them.

By far, the simplest approach to delegate a router would be to give experiments direct access to the router's configuration interface. This approach, however, makes it impossible to guarantee safety and significantly complicates execution of parallel experiments. Having administrators configure routers on behalf of experiments addresses some of these concerns, but does not scale to shared environments and makes it impossible to run experiments that require dynamic control of routes or traffic.

Providing *indirect* control of the router via a separate protocol can solve some of the problems in simpler solutions. For instance, the platform could require that experiments communicate their routing decisions using OpenFlow; prior work provides a foundation for multiplexing control of forwarding decisions with OpenFlow [84]. However, BGP routing engines (*e.g.,* BIRD, Quagga, and hardware routers) expect to receive routes via BGP and then enact

their decisions via local mechanisms (*e.g.,* BIRD programs the Linux kernel via Netlink [77]). Requiring decisions to be enacted via OpenFlow or other non-standard interfaces would necessitate either the complex modification of existing routing engines, or development of a custom routing engine, both of which would decrease fidelity and neither of which is practical. In addition, BGP (or yet another protocol) would remain necessary to exchange route information.

Another approach involves using tags or tunnels to signal routing decisions. For example, Google's Espresso encapsulates packets with MPLS to convey which route should be used [104], and Transit Portal attempted delegation by having clients maintain multiple VPN tunnels, each corresponding to a single BGP neighbor (sending traffic via a specific tunnel would send it to the corresponding neighbor) [96]. However, these approaches introduce additional complexity and may not be supported by existing routing engines. For instance, using MPLS labels requires a separate label redistribution protocol, an MPLS enabled kernel (only recently available [95]), and a routing engine that supports MPLS (not natively supported by BIRD or Quagga). Likewise, using tunnels requires communicating a mapping of tunnel to BGP neighbor (or tunnel to route) via an out-of-band protocol and necessitates the use of a custom routing engine to select and install routes (existing routing engines do not support such mappings), or manual installation of routes.

In addition to their inherent challenges, these approaches only address delegation of the data plane. BGP or other custom protocols would remain necessary for the control plane. We conclude that maintaining conformity and compatibility by devising solutions to support delegation natively with existing BGP and IP is ideal given the overhead and challenges these alternate solutions present.

3 VIRTUALIZING THE EDGE WITH vBGP

To address the challenges of multiplexing control of a single BGP router for multiple experiments, we present vBGP, a framework to virtualize the data and control planes of a BGP router by providing (1) mechanisms for delegating complete visibility and control of data and control planes to experiments and (2) an architecture capable of enforcing sophisticated security policies required to prevent experiments from performing unsafe actions. Analogous to hypervisors in other virtualization domains, vBGP multiplexes experiments over the same BGP router to support parallel experiments; provides safety by isolating experiments from the underlying router and each other, and interposing on experiment interactions with the rest of the Internet; and exposes data and control plane interfaces to experiments that are equivalent to having sole control over the router's BGP process (akin to a hypervisor exposing x86).

We use vBGP at all PoPs in our implementation of PEERING, described in §4. vBGP is generalizable and compatible with hardware or software routers; our deployment instantiation of vBGP runs atop Linux and uses an open-source software router.

3.1 Key Design Decisions for vBGP

Three architectural decisions are key to realizing our goals:

Untether experiment logic from router infrastructure (§3.2). Virtualization for experimentation traditionally involves partitioning a machine's resources and then granting an experimenter control of a partition. In comparison, vBGP virtualizes a router's data and control plane decisions and *delegates* them to a system under the control of the experimenter. Decoupling experiment logic from the router enables vBGP to support a variety of experimental setups, letting researchers enact their experiment logic via hardware or software routers or SDN controllers, at their university, in the cloud, or in a container on the vBGP router.

Devise protocol-compliant mechanisms to natively provide complete data and control plane visibility and control (§3.2). As stated in §2.2, our interface between experiments and vBGP should be BGP and IP, since this is the interface any native (non-virtualized) experiment would have with the Internet. We developed mechanisms within BGP and layer 2 protocols to let experiments innately convey their decisions to vBGP without modifying protocol logic, or using out-of-band communication. From the perspective of an experiment, interactions with vBGP "just work" as they would if the experiment *was* the edge router, exposing the *complete range of (security-policy-compliant) data and control plane interactions with the rest of the Internet.*

Interpose on experiment data and control plane activity (§3.3). Experiments can exchange data and control plane traffic with the Internet, so vBGP must take measures to mediate between experiments and the Internet to enforce security policies, prevent dangerous activity, and perform logging necessary for attribution [48]. In particular, vBGP must be able to (1) enforce a wide range of security policies that are atypical (due to our use case) on the data and control planes and (2) to intercede in ways that let us prevent prohibited activities without otherwise affecting experiment control and capabilities. Supporting these requirements is challenging, as our security policies require functionality beyond what is provided by existing router policy frameworks. Our solution is platform-specific software that interposes between experiments and the Internet on both the data and control planes, separately, to enforce security.

3.2 Delegation to Experiments

vBGP delegates both the data and control plane decisions of a BGP edge router to experiments, which are logically (and can be physically) separate from the vBGP router. The control plane mechanisms we employ involve adapting existing BGP mechanisms for our setting and are not particularly groundbreaking on their own. However, they combine with our novel data plane mechanisms to delegate control in a way that addresses longstanding limitations in BGP routing.

3.2.1 Delegating the control plane. BGP has no intrinsic mechanisms for delegating visibility or control. We adapt two mechanisms, one for inbound announcements from the Internet and one for outbound announcements from experiments.

Announcements from the Internet to experiments. A BGP router may receive routes for the same destination from multiple BGP neighbors. For instance, Figure 1 shows E1 receives a route from both N1 and N2, but E1 would only forward its preferred route to X1 and X2, limiting their visibility (§2.2.2).

vBGP uses the BGP ADD-PATH extension [98] to send each experiment all received routes within a single BGP session. As a result, experiments see multiple routes coming from the vBGP node, as depicted in Figure 2a.

Announcements from experiments to the Internet. vBGP delegates control of which BGP neighbors an experiment's announcement will propagate to through the use of BGP communities. Communities are labels that a router can attach to a BGP announcement [24]. vBGP defines whitelist/blacklist BGP communities for neighbors at every PoP, and experiments label prefix announcements with communities that specify whether or not to announce the prefix to specific neighbors. If no communities are attached, vBGP forwards the announcement to all neighbors.

Experiments can couple this control with BGP ADD-PATH to send different announcements for the same prefix to different neighbors to support more sophisticated policies, such as those described in §2.2.2. For example, in Figure 1, X1 can announce an update for its prefix, 10.1.0.0/24, with AS-path prepending and tagged with a community to export the announcement only to N1. The experiment can also make an announcement for the same prefix, without any prepending, tagged with a community to export only to N2.

Communities could be used to signal behavior such as prepending or poisoning, but to maximize flexibility and realism we chose to allow experiments to directly announce the routes they want.

3.2.2 *Delegating the data plane.*

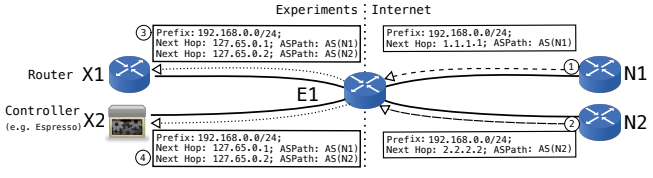
Routing traffic to the Internet. Although BGP ADD-PATH provides X1 and X2 with *visibility* of all BGP routes and updates at E1, it alone does not empower the experiments to *control* which route is used for traffic; outgoing traffic from X1 and X2 remains subject to the routing decision(s) made at E1 based on E1's configuration (§2.2.2).[1]

We want experiments to be able to control how their traffic is routed in a manner that "just works," and thus do not want to introduce additional protocols, encapsulation, or out-of-band communication that is incompatible with existing routers, BGP implementations, and production tooling (§2.3). Realizing a solution that operates within these constraints is non-trivial; previous work claimed that "forwarding traffic on different paths requires the data packets to carry an extra header or label" [45].

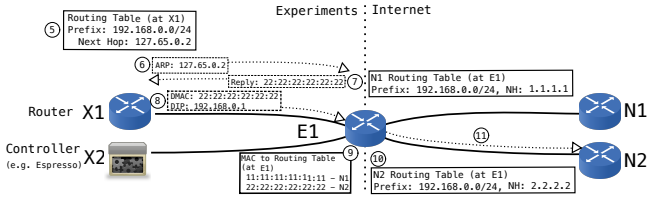
Our key insight is that routers already add an extra header to packets: the layer 2 header. We develop a technique that encodes an experiment's decision of how to route a packet via the layer 2 header and results in existing routers automatically encoding the decision via their normal behavior. While the technique only conveys the decision across a single layer 2 domain, our target setting naturally bridges a single domain, from an experiment router to a vBGP router. In §4.4 we extend the technique across multiple domains, although it is still not completely general.

Understanding our approach requires considering how a router typically enacts forwarding for its choice of a route. Although platforms can optimize the process to minimize lookups, the process is generally as follows: a router performs a lookup on the packet's destination to find its preferred route, which maps to a next-hop IP address from the route's BGP announcement. The router forwards the packet towards the next-hop, which it must be able to reach without BGP (*e.g.,* directly connected or via an IGP). Typically, a

[1]We have found that there is sometimes confusion over whether BGP communities can be used to address this challenge. BGP communities can be used to signal policies for routes announced *by* experiments, not select which route (received *from* an upstream neighbor) to use for forwarding traffic from an experiment to a neighbor.



(a) Control Plane



(b) Data Plane

Figure 2: Figure 2a shows how vBGP overwrites BGP next-hops to delegate control to experiments. The next-hop for announcements from the neighbors N1 and N2 (①, ②) are rewritten to IP addresses that are local to E1 (③, ④). Figure 2b shows how vBGP forwards packets from experiments. X1 prefers to route via N2 (⑤), so when sending a packet to 192.168.0.1, it first ARPs for the MAC of the next-hop (⑥), to which E1 responds with the MAC it locally assigned to N2 (⑦). When the frame arrives at E1 (⑧), it knows based on the destination MAC (DMAC) to look up the route in its local routing table for N2 (⑨, ⑩).

router announcing BGP routes to a neighbor either keeps the next-hops unchanged (if the neighbor can reach them, which they cannot in the vBGP setting) or sets them all to a specific local IP addresses (*e.g.,* a loopback address).

In our design, vBGP systematically modifies next-hop IP addresses and manipulates layer 2 interactions in a manner that results in the experiment's routing choices being naturally conveyed per packet to the router. Specifically, vBGP assigns distinct private IP and MAC addresses for each BGP neighbor. It also maintains one routing table per BGP neighbor. As depicted in Figure 2a, when a vBGP router receives an announcement from a neighbor (①, ②), it stores the route in the table for the neighbor, then rewrites the next-hop to the IP address it assigned to the neighbor before exporting the route to experiments (③, ④).

When an experiment selects a route to send a packet towards a destination, it resolves the next-hop's MAC address, just as any BGP router would to forward a packet. The vBGP instance offering the next-hop responds to an ARP or NDP query with the MAC, and the experiment forwards a layer 2 frame containing the packet to that MAC. Since the process is identical to standard BGP forwarding, the experiment can use a standard software or hardware router (X1) or a more sophisticated controller that uses BGP to interface with the Internet (X2). Once the vBGP router receives the frame, it inspects the destination MAC to determine which BGP neighbor's route the experiment selected. vBGP then routes the packet using the table corresponding to the neighbor.

Figure 2 illustrates the process. In Figure 2a, X1 and X2 receive routes from E1 with a next-hop of 127.65.0.1 and 127.65.0.2 which correspond to neighbors 1.1.1.1 (N1) and 2.2.2.2 (N2), respectively. X1 has configured policy to prefer routes to the destination network via N2. In Figure 2b, when X1 wants to forward a packet to 192.168.0.1, it looks up the next-hop in its routing

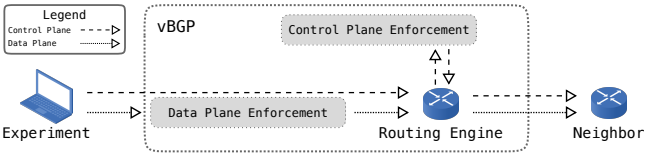


Figure 3: Logical locations of the enforcement engines as they interpose on the data and control planes between an experiment and the Internet.

table (⑤) and sends an ARP query for the next-hop (⑥), equivalent to what it would do if directly connected to N2. E1 responds with MAC(127.65.0.2) (⑦), which X1 sets as the destination for its frame (⑧). Upon receipt of the frame, E1 uses that MAC to determine which routing table to use (⑨). E1 performs a lookup in the routing table corresponding to N2 (⑩) and forwards the packet to next-hop 2.2.2.2 (⑪).

Although X2 uses a more sophisticated process to decide which route to use (such as deciding per application), it still uses BGP to exchange routing information and performs the same process as X1 to encapsulate a packet within a frame to forward. Because all routing decisions are delegated to experiments, the vBGP node does not need to make any routing decisions of its own.

Routing traffic to experiments. vBGP forwards traffic received from neighbors towards the experiment announcing the corresponding address space. Normally the source MAC address of the frame that arrives at the experiment will be MAC(E1), not the MAC address of E1's neighbor that delivered the traffic. To provide experiments with visibility into which neighbor delivered the traffic, vBGP rewrites the source MAC address of each packet received from a neighbor with the MAC address it assigned to the neighbor, *e.g.,* MAC(127.65.0.2).

3.2.3 Summary of contribution. Although the building blocks used in vBGP are standard (*e.g.,* BGP ADD-PATH, BGP communities) or equivalent to functionality available in routers (*e.g.,* Virtual Routing and Forwarding [31], Routing Instances [51], and Policy-Based Routing [30]), they have not been synthesized to achieve the same goals, and the delegation provided by vBGP does not follow from their simple combination, and the functionality we provide was previously seen as requiring additional protocols or mechanisms outside BGP [45]. Our approach does not rely on custom software, encapsulation, or changes to protocol headers, and thus supports experiments using existing hardware and software routers, and modern controllers that speak BGP. vBGP has operational uses outside of Peering and influenced the design of part of Facebook's BGP control system (§7.2).

3.3 Security and Isolation

In order to maintain safety and isolation, vBGP supports limiting experiment data and control plane activity based on any discretionary stateful or stateless policy, not just those supported by conventional routers. This approach supports more sophisticated policies that balance experimenter control with the need to maintain safety, enables evolution of policy to account for new capabilities or concerns, and allows capabilities to be enabled on a per-experiment basis, in keeping with the principle of least privilege.

Policy enforcement architecture. vBGP uses *policy enforcement* engines that operate alongside the routing engine and interpose on all experiment activities. The engines have non-volatile storage to maintain state.

vBGP separates policy enforcement from the router for two reasons. First, most router implementations can only support a limited set of policies. Decoupling the enforcement engine and implementing it separately allows vBGP maximum flexibility in the policies it supports, including stateful policies, and ensures vBGP is not tied to a specific router implementation. This allows vBGP to use a variety of industry standard, hardened software and hardware routing engines to communicate with neighbors without being limited by the routing engine's policy capabilities.

Second, it is difficult to validate the correctness/behavior of policies enforced by traditional router implementations; testing frequently requires setting up an emulated network with multiple BGP routers to create the desired test conditions [17, 37]. In comparison, we can validate the behavior of our decoupled implementation using unit tests that inject test conditions. Figure 3 depicts where the data and control plane enforcement engines fall logically in the vBGP architecture.

Control plane enforcement. The enforcement engine receives all routes announced by experiments from the router, evaluates whether each route is policy-compliant, and announces only compliant routes back to the router. The router only forwards announcements received from the enforcer to its neighbors. Our implementation uses ExaBGP [4], which is a BGP engine that allows execution of Python code inside the BGP pipeline. We capture the policy in Python, allowing a great deal of flexibility in what the administrator of the vBGP instance can enact and facilitating easier testing. For instance, state can be synchronized among vBGP instances to enable AS-wide policies, such as limiting the total number of times a prefix can be announced or withdrawn across all PoPs during a 24 hour period. Our current policies are defined in Section 4.7.

Data plane enforcement. vBGP's data plane is run in an isolated container, so it can either be collocated with a software router or run on a separate server. It interposes on experiment data plane traffic through the use of extended Berkeley Packet Filters (eBPF), which allows loading simple programs into the kernel to inspect packets. The eBPF program can make a stateless or stateful decision to allow, transform, or block each packet, enabling policies such as rate limiting experiment traffic on a per PoP or per neighbor basis.

4 PEERING: FROM A ROUTER TO AN AS

While the design of vBGP is generally applicable to different infrastructures, we used vBGP to build Peering, a platform for routing research that we make available to the community. Figure 4 shows an overview of Peering's architecture. Peering maintains infrastructure at PoPs around the world, and each consists of a commodity server running vBGP, from which we interconnect with one or more networks using BGP. We implement vBGP using common open source software, *i.e.,* the BIRD software router [9] for our BGP routing engine and OpenVPN [6] for VPN tunnels with experiments. Section 5 presents engineering aspects of Peering,

some of which would benefit other networks. vBGP adds manageable overhead, allowing a commodity server to virtualize a router at even the largest IXPs today and in the foreseeable future. Section 6 demonstrates scalability.

4.1 Key Design Decisions for Peering

Previous Internet routing research platforms were limited in the type of research experiments they could support, their ease-of-use and accessibility to experimenters, and their long-term maintainability. Our approach accounts for these challenges and focuses on addressing them in multiple ways.

Deploy at IXPs and universities (§4.2). To achieve a good representation of today's widely interconnected content and cloud providers [22, 29, 81, 104], our approach for deploying Peering focuses on a mix of both university and IXP sites. This allows us to sidestep the limitations of each by combining their strengths. In particular, IXP sites provide many interconnections and university sites allow easy federation with other resources that offer complementary functionality.

Federate with other platforms (§§ 4.3 and 4.4). To better approximate the cloud provider setting—interdomain connectivity at locations around the world, data centers providing computational resources, and a backbone connecting them all—Peering federates with CloudLab [2] to provide researchers with cloud-like data centers, and with educational networks to provide slices of their multiplexing backbones to interconnect Peering PoPs. These federations support a wider range of experiments while ensuring that the platform's resources remain predominantly focused on expansion of the AS and its connectivity.

Low-overhead, turn-key experiment and infrastructure setup and deployment (§§ 4.5 and 4.6). To democratize Internet routing research, we designed and implemented standardized workflows to allow easy provisioning and deployment of new experiments, new vBGP sites, and new peer networks. Further, we provide experimenters with a toolkit that can be used to instantiate a wide variety of experimental setups without requiring prior experience with BGP, vBGP, or Peering.

Follow the principle of least privilege (§§ 4.6 and 4.7). To balance our goals of maintaining safety while supporting a wide range of experiments, by default we tightly restrict what an experiment can do, especially in terms of the range of announcements it can make to the Internet. We carefully review experiments that need more functionality, including consulting commercial network operators for feedback as needed, and Peering supports per-experiment capabilities for those that can safely justify richer functionality.

4.2 Footprint and Connectivity

Numbered resources. Peering has 8 AS numbers (ASNs), including three 4-byte ASNs, and is allocated a total of 40 /24 IPv4 prefixes and one /32 IPv6 prefix. We dedicate one or more prefixes to each approved experiment for a specified duration.

PoPs. As of June 2019, Peering has thirteen operational PoPs on three continents, four at IXPs and nine at universities. Peering servers at five additional PoPs are projected to come online. At

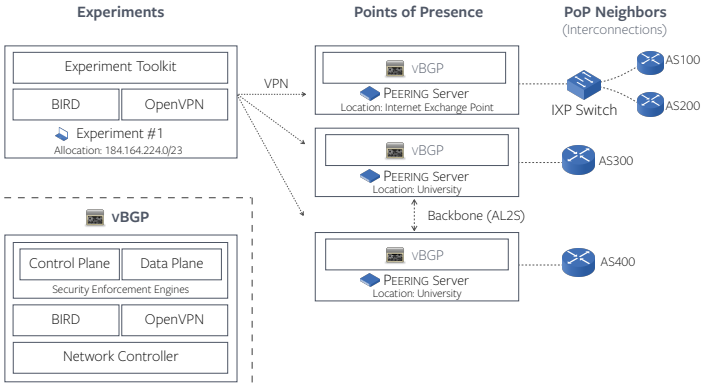


Figure 4: Peering's architecture. Experiments connect via VPN to one or more Peering servers, or PoPs, and use a software router to establish BGP sessions with a vBGP router at each PoP. Experiments exchange routes and traffic via the tunnels and the corresponding BGP session, where vBGP delegates data and control plane decisions while enforcing policy. All PoPs run vBGP, which consists of the networking controller (§5), routing engine (§3.2), OpenVPN (§§ 4.5 and 4.6), and enforcement engines (§3.3).

IXPs, we establish bilateral peering with tens or hundreds of networks and to many more via interconnections with route servers, and we pursue partnerships to obtain transit interconnections. At universities, the platform has a transit interconnection with the university's AS. The sites have different tradeoffs: IXPs offer richer connectivity, but universities can present opportunities such as our CloudLab federation. Universities add operational overhead to debug connectivity (Appendix A), whereas IXPs add operational overhead to negotiate hosting and transit.

Getting from the initial agreement to a server inside of a colocation facility with connectivity was the biggest challenge to getting connectivity at IXPs; the business aspects, bureaucratic procedures on both sides of placing a server within the colocation facility, and negotiating transit consumed months on average. Once the paperwork was completed, it took very little time to connect the server and to establish peering links.

Peer networks. Obtaining bilateral peering agreements with other members of the IXP was easier than we expected. Most member ASes at an IXP have open peering policies, but that is not a guarantee that they will peer blindly with everyone. Taking a direct approach and reaching out to the other IXP members resulted in several hundred direct peering connections. Peering currently has 12 transit providers and 923 unique peers (129 via bilateral BGP sessions and the rest only via IXP route servers [70]). We peer with 854 ASes at AMS-IX (106 bilaterally), 306 (63) at Seattle-IX, 140 (10) at Phoenix-IX, and 129 (6) at IX.br/MG in Brazil.

According to PeeringDB [7], our peers are balanced across diverse types of networks: 33% of our peers are transit providers, 28% are cable/DSL/ISPs, and 23% are content providers. Of the remaining 17%, 8% cannot be classified, and the rest are a mix of education/research networks, enterprise networks, non-profits, and route servers. An industry study published in 2016 reported that 60%

of traffic comes from a small number of content delivery networks. PEERING connects directly to 7 of the 10 named [63].

PEERING announcements can reach all ASes via transit providers, so experiments can exchange traffic with all ASes. If network P is a transit provider for network C (either directly or transitively), C is in P's *customer cone*. ASes in the customer cones of our peers receive announcements made by experiments to peers. This reach is of interest to researchers due to the importance of peering routes on today's Internet [12, 29, 44, 80, 81] and because it reflects ASes towards which PEERING has "extra" route diversity, as they are reachable both via all PEERING transits and via at least one peer.

4.3 Emulating a Cloud Provider

To support experiments in environments similar to those used by content and cloud providers, experimenters need to be able to pair PEERING's rich interdomain connectivity with backbone connectivity and compute resources.

4.3.1 Backbone connectivity. We worked with research and education networks, such as Internet2, to establish backbone connectivity between PEERING PoPs and configured vBGP routers on the backbone to exchange routes in a BGP mesh. Our US PoPs connect to Internet2's Advanced Layer 2 Services (AL2S) [1], and our Brazilian site uses RNP's equivalent in Brazil [74]. These services allow us to create VLANs between sites (including bridging between the US and Brazil), with provisioned capacity across the educational networks. In the future, we will use Geant [5] to integrate our European PoPs. An experiment connected to one PoP has visibility into routes at all other PoPs in the BGP mesh, and it can direct announcements and traffic across the backbone to BGP neighbors at any of the PoPs (§4.4). Section 6 provides an overview of TCP throughput over the backbone.

4.3.2 Federation with CloudLab. CloudLab provides researchers with access to bare-metal systems for establishing their own clouds to conduct experiments. PEERING has PoPs with backbone connectivity at all CloudLab locations. By colocating PEERING PoPs at CloudLab sites, CloudLab experiments can select from routes available at any PEERING PoP to reach destinations across the Internet, then route across the backbone to reach the selected PoP. Combined, PEERING and CloudLab provide experiments with edge PoPs, a backbone, and compute resources, enabling experiments to operate in environments that are qualitatively similar to those of large content or cloud providers.

4.4 vBGP Across the Backbone

An experiment needs the ability to use the backbone. For example, in Figure 5, experiment X1 is controlling vBGP instance E1 and should be able to send traffic to N2 via the backbone link between E1 and E2. For this to work, N2's BGP announcement must reach E1 and X1 with next-hops they can reach, either via layer 2 or via an IGP. Neither approach works out of the box. With an IGP, the next-hop would be an interface on E2, and X1 would make a forwarding decision by looking up the next-hop in its IGP table to learn that it should forward to E1. It would then forward a frame with a destination MAC belonging to E1. In order to avoid losing X1's decision, this MAC would have to uniquely encode

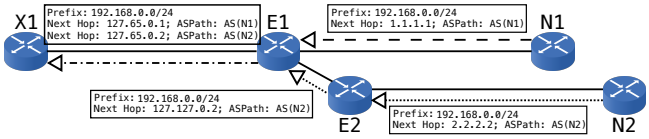


Figure 5: Example connectivity for an experiment (X1) using PEERING's backbone connectivity. E1 and E2 are vBGP routers in PEERING, and each has a single neighbor (N1 and N2). The prefix announcements demonstrate how vBGP's delegation rewrite's next-hops to enable X1 to control E2's connectivity to N2.

the next-hop and egress (E2 and N2), which would add significant complexity to the IGP configuration. However, making an interface on E2 reachable via layer 2 from X1 would require tunneling (*e.g.*, VLANs), additional complexity.

Instead, we extend our next-hop-based control hop-by-hop without requiring an IGP. We use a common pool of IPs to assign a unique global (to PEERING) IP to each external neighbor. This allows E1 to recognize E2's next-hop `127.127.0.2` and overwrite it with IP `127.65.0.2` from its local pool, prior to announcing the prefix to X1. It maintains a separate routing table for this local IP (and its corresponding MAC), containing the routes with next-hop `127.127.0.2`.

When X1 wants to send a packet to `192.168.0.0/24` via N2, it uses its routing table to find the next-hop `127.65.0.2` and sends an ARP request, to which E1 responds with the MAC address. X1 sets this as the destination for its frame, then E1 uses the routing table corresponding to the MAC to look up the route to `198.168.0.0/24`, which has next-hop `127.127.0.2`. The process now repeats as E1 sends an ARP request for `127.127.0.2`, and E2 responds with a MAC. E1 transmits the frame, E2 performs its lookups based on the MAC, and forwards the frame to N2.

4.5 Experiment Toolkit

We provide experimenters with a toolkit to connect to PEERING and execute experiments. Experiments establish BGP sessions with PEERING routers over VPN tunnels. The toolkit contains wrappers for OpenVPN and BIRD that implement a turn-key interface for common tasks such as establishing BGP sessions or making prefix announcements. Table 1 provides a full list of the functionality provided by the wrapper software. Advanced features such as per-packet routing (§3.2.2) must be configured by experimenters.

While the toolkit is open source and designed around OpenVPN and BIRD, experiments are free to use any software that can establish BGP sessions and VPN tunnels with PEERING servers (past experiments have used Quagga and ExaBGP for BGP). Because security policies are enforced at PEERING servers, we do not need to enforce any behavior at the experiment side.

4.6 Deploying Experiments

Experiments execute on infrastructure that is separate from PoPs. This decoupling promotes flexibility and enables the platform to support a variety of experimental setups. For instance, the BGP announcements of a measurement experiment can be managed by custom logic executed from a laptop. Experiments can also run applications such as a web server or a Tor relay, or combine PEERING connectivity with emulated intradomain topologies using tools

Category	Functionality
OpenVPN	Open/close/check status of tunnels
BGP/BIRD	Start/stop BIRD v4 and v6 sessions Status of BGP connections Access BIRD CLI
Prefix Management	Announce/withdraw prefix Manipulate community attribute Manipulate the AS-path attribute

Table 1: A list of capabilities provided by the PEERING experiment software to simplify and abstract basic tasks to configure and set up experiments.

such as *Mininet* [56]. Experiments requiring more computational resources can run on CloudLab [2] or cloud providers.

Before receiving access and prefixes to execute an experiment, experimenters must submit a proposal that outlines the experiment's goals, resource requirements, and execution plan via a simple web form. This process of manual approval mimics the one that was successful for PlanetLab, except PlanetLab outsources approval to the PIs at individual sites. We considered automatic approval and allocation of an IPv6 prefix (which are plentiful and will let experimenters start to use PEERING), since vBGP's security architecture and filters will prevent misbehavior. However, the current rate of proposals is manageable with manual review, so we have not invested the development effort to automate limited approval.

Once we approve the experiment via a simple management web interface, the management system we built automatically generates credentials for the experimenters that enable VPN connections to vBGP routers. The system updates the policies and configuration each vBGP router needs to allow the experiment (and to filter disallowed traffic and announcements that the experiment might try to send). It pushes the updates to vBGP routers without disrupting ongoing experiments or running BGP sessions, since we are only modifying configurations relative to individual experiments.

Although the number of experiments varies over time, during the past 12 months PEERING typically hosts from 3 to 6 concurrent experiments. Concurrency is limited by available IPv4 address space: although PEERING controls plenty of IPv6 space, most experiments to date concentrated on IPv4. Fortunately, no experiment has had to wait due to insufficient IPv4 address space thus far.

4.7 PEERING Security Policies

PEERING experiments exchange routes and traffic with other production networks on the Internet that are outside of our control. As a result, we cannot formally verify the safety of PEERING because we cannot guarantee that all possible interactions are safe, even if they comply with all relevant protocol specifications. This fundamental challenge exists in any environment in which there are interactions between varying implementations and configurations of a protocol standard [65]. For instance, it is fundamentally impossible to guarantee that BGP announcements will not trigger bugs in remote routers. Even announcements that are fully compliant with the BGP specification can cause widespread outages due to bugs in router implementations [58, 75, 94], and it is not feasible to test whether a given announcement may cause disruption given the plethora of implementations and configurations.

Because we cannot formally verify PEERING's safety, our approach is to define conservative security policies which match current best practices and verify correct enforcement. vBGP's design supports data and control plane policies. For PEERING we define two dimensions for security policies: the *rate* of traffic and BGP updates, and the *content* of packets and BGP updates.

Policing rate. PEERING shapes traffic at (two) sites with bandwidth constraints to the rates agreed upon with the sites' operators. To date, no experiments exercised these bandwidth limits, and experiments that would could still be deployed on sites without bandwidth constraints. To limit overhead on routers in the Internet, PEERING limits experiments to 144 BGP updates/day for each prefix and PoP pair. This corresponds to an average rate of one update every 10 minutes, which amounts to a small fraction of the "background noise" in the interdomain routing system [99].

Policing content. PEERING prevents experiments from performing activities that are harmful or preventing attribution back to the experiment. An experiment cannot announce a BGP update or source traffic using address space that is not part of the experiment's allocation (*hijacking* and *spoofing*), which also means experiments cannot transit non-experiment traffic; cannot originate announcements from an ASN it is not authorized to use; and cannot manipulate BGP attributes in ways not allowed by our capability framework. We do not currently police dataplane content beyond verifying the source IP address.

Capability framework. In keeping with the principle of least privilege, the management system has a capability-based framework that defaults to limiting experiments to "basic" announcements and supports adding capabilities on a per-experiment basis. Experiments request capabilities via the experiment web form, and admins can simply add the capability on the approval web form. Current capabilities include:

- Allow a limited number of poisoned ASes [21].
- Allow attaching a limited number of BGP communities or large communities to announcements [24, 46, 87].
- Allow optional BGP transitive attributes [75].
- Allow experiments to announce routes learned from one network to another network, for experiments that require *legitimately* providing transit for an experimental prefix.

Our existing capabilities suffice for most experiments. When an experiment requires novel capabilities, we work with experimenters to deploy them and add them to our capability framework for future experiments. For example, an experiment recently required the ability to announce 6to4 IPv6 addresses [23].

Implementation. On the control plane, we implement security policies in BIRD whenever possible, as it provides better performance than the general ExaBGP engine. We currently use the ExaBGP engine to limit the rate of announcements from experiments and to filter BGP updates with disallowed (any non-standard) BGP attributes. Similarly, we implement data plane security policies using Linux's built-in tools whenever possible.

Testing security policies. In the interest of safety, we do *not* verify enforcement of our security policies by executing adversarial tests against the production PEERING platform. Instead, we deploy our

production configurations and software stack in a test environment (§5) that includes (emulated) Peering experiments, servers, and BGP neighbors, and then use a custom framework to execute automated tests of our security policies and handling of experiment capabilities. Through this process, we verify that our security implementations correctly enforce the expected policies in terms of what traffic and announcements are (dis)allowed.

For each capability, we deploy two (emulated) experiments in our controlled environment: one that does not require the capability and one that does. We execute both experiments twice, with and without the capability. We check that the routes exported and traffic exchanged in each execution match the configured policy and are safely handled by the software routers we test them against within the test environment. For example, we deploy an experiment that makes announcement with BGP communities with and without the corresponding capability, and check that communities are stripped from exported announcements when the capability is missing.

Impact of misbehaving experiments. Peering's security engine and deployed policies protect the Internet from misbehaving experiments. However, misbehaving experiments may put strain on Peering's infrastructure and negatively impact its performance. For example, experiments sending an excessive rate of updates could overwhelm our control plane security engine (§3.3), impacting other experiments. To the extent possible, we designed Peering to isolate the performance experienced by our upstreams from that experienced by experiments, and our security mechanisms are designed to protect our upstreams and the broader Internet, even if it leads to an outage of the platform itself. For instance, if the security enforcement engine was to become overloaded due to a misbehaving experiment, the enforcement engine would fail closed, thereby blocking *any* experimental announcement from propagating upstream. However, as of November 2019, we have not experienced any scenario in which a misbehaving experiment caused a platform outage.

5 DEVELOPMENT AND DEPLOYMENT

Peering's design requires weaving together many components, and building Peering required addressing a number of technical and logistical challenges that arose as we scaled the testbed in terms of the number of PoPs, experiments supported, and experiment capabilities. With a growing number of heterogeneous servers deployed over multiple years into highly diverse environments, it has been essential for us to invest in tooling that enables us to maintain a standardized deployment and automate the numerous processes required to support experiments.

In this section we describe three pillars of Peering's engineering, some of which we believe can be applied to other networks to positive effect. Our solutions allow a small research team to develop and operate a distributed infrastructure with hundreds of interconnections that services a dynamic and sophisticated set of research experiments. These solutions were key to enabling a production platform and are distinguishing elements from the prototype designs and implementations in prior work.

Intent-based Configuration. Peering's components have complex configuration files; for example, the configuration files for BIRD alone can exceed over 10,000 lines at large PoPs. Peering configuration files are dynamic; for example, BGP sessions must be enabled and disabled on BIRD whenever an experiment connects and disconnects from a PoP. Finally, Peering configuration files have specificities; for example, only two Peering PoPs have traffic bandwidth limits. As a result, it is not possible to maintain these complex and dynamic configurations by hand.

We employ *intent-based configuration* best-practices [91] to transform a model containing desired configuration (such as experiment capabilities) into service-specific (*e.g.,* network controller, BIRD, OpenVPN, and policy enforcers in Figure 4) configuration files. The desired configuration is stored on a centralized database accessible through a web service. The database has information including approved experiments and their capabilities (§4.7), network configuration at each PoP, and interconnection information for BGP sessions with peers at each PoP. The desired configuration in the database is used to generate service configuration files automatically through a templating engine, and the resulting configuration files are used by the services.

As an example, consider how control plane capabilities vary by experiment. When we add support for a new experiment capability, such as AS-path poisoning, we add support for expressing the capability in our desired configuration model. Then we identify how the configurations of different services (Figure 4) need to be transformed so that the capability is enabled for authorized experiments (and blocked for others). We use the resulting insights to modify the configuration template, and then use the templating engine to generate configurations and test in an offline development environment (discussed later in this section) to ensure that the generated configuration works as expected. Following local testing, we update the production templating configuration and regenerate configuration files for all Peering servers.

All configuration files deployed to Peering servers are stored in a version-control system where they can be inspected and rolled back if needed. When we make templating changes, we *canary* the new configuration on a subset of our production fleet as a safeguard. We use Ansible to fetch configuration files from the version-control system, deploy them to a subset of servers, and then reload the impacted services. Once we are confident in the new configuration, we instruct Ansible to deploy updated configurations to all Peering servers. A similar templating process is used at the servers to update BGP session configuration as experiments connect / disconnect.

Network Configuration with Transactional Semantics. In order to maintain BGP sessions with PoP neighbors and delegate the data plane, vBGP must configure (1) physical interfaces used for interconnecting with upstream neighbors, (2) virtual interfaces used for delegating control of the data plane to experiments, (3) routing tables and rules, and (4) filters used to enforce security policies (§§ 3.2.2 and 4.4).

Given that vBGP network configuration is dynamic, we developed a network controller program that updates the server's network configuration such that it aligns with the high-level, intent-based description modeled in our centralized configuration database. However, the interface to configure Linux networking (Netlink) does not support expressing intents; it provides a request-response interface that allows querying, adding, and removing network configuration

(*e.g.,* routes and addresses). When the network controller receives a configuration update, simply resetting the network configuration and applying the new configuration from scratch would reset BGP sessions and VPN connections, interrupting ongoing experiments and interdomain connectivity with PoP neighbors. Instead, the controller attempts to minimize the amount of configuration changes by implementing logic, unavailable in Linux's networking tools, that (i) removes configuration that is incompatible with the intended state, (ii) keeps any configuration compatible with the intended state, and (iii) adds any missing configuration.

Two requirements complicate our network controller's design further. First, we enforce transactional semantics, where either all configuration changes are successfully applied or no changes are applied (e.g., partially complete changes are rolled back) to ensure that a server is never in an inconsistent state. Second, Linux network interfaces can have one primary address and an arbitrary number of secondary addresses. PEERING needs to control the primary address as it is used when generating ICMP error messages, particularly TTL Exceeded replies to traceroute probes. Because the Linux kernel does not support changing the primary address (it is set based on the order in which addresses are added), PEERING's network controller verifies each interface's primary address and, if incorrect, removes and readds the interface's addresses in the proper order.

Standardization and Isolation. We standardize configuration, deployment, and upgrades to our infrastructure by running servers with stripped down operating systems and packaging PEERING 's services (e.g., OpenVPN, BIRD, and PEERING's network controller service and enforcement engines) into containers. Linux namespaces isolate specific functionality of the Linux kernel, which container and lightweight virtualization technologies employ to control resource sharing. PEERING uses containers to isolate vBGP services' process, file system, and network namespaces from that of the host, allowing us to isolate each service and its dependencies, preventing conflicts and easing upgrades.

This isolation is key given that implementing vBGP requires tight integration with Linux's networking stack (§§ 3.2.2 and 4.4). If vBGP made configuration changes to the host's networking namespace, then configuration errors, software bugs, or failures in vBGP could put the host's networking stack in a dysfunctional state and prevent all in-band access. However, because vBGP configures an entirely separate networking namespace, it is logically isolated from the host's network configuration, significantly reducing risk and also enabling our tooling to reset the state of the namespace if needed.

We deploy containers to servers using Ansible. Our Ansible playbooks reset the server's operating system to a known, desired state, including managing the host's networking stack. Ansible is executed periodically and verifies that every PEERING server is in compliance, redeploying out of date containers and upgrading configuration files as needed (when configuration files are updated, BGP sessions are not reset and thus experiments are not impacted). This results in a single Ansible and operating system configuration for all servers, while supporting diverse PoPs.

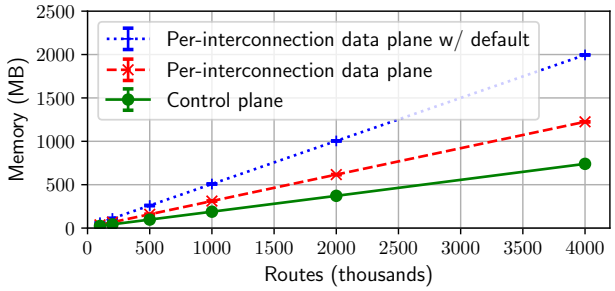
In addition to maintaining standardization in production, using containers simplifies platform development and testing. During development, we need to be able to execute experiments and test changes in an environment that is representative of a PEERING PoP. Thus, the environment must have the same packages and configuration that are used in production, must run the vBGP network controller, and must also have interactions between a PEERING PoP, its neighbors, and experiments. We accomplish this by using the same containers and automatically-generated configurations used in production on our personal development machines, and using additional containers running software routers and PEERING's client toolkit (§4.5) to emulate a PoP's neighbors and experiments. This allows us to systematically test changes to PEERING (such as a new experiment capability) in an environment that is guaranteed to be representative (in both software and configuration) of our production environments without the risk of problems on our development machines due to (for instance) the vBGP network controller reconfiguring the machine's network interfaces.

6 SCALABILITY OF PEERING

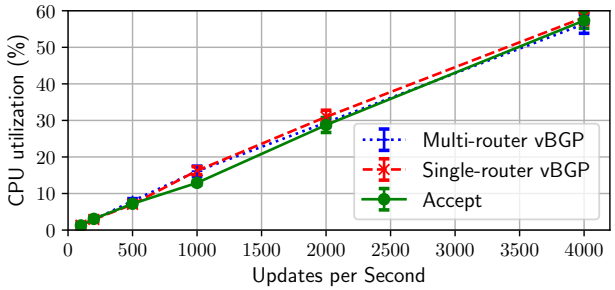
We evaluate the performance of our vBGP instantiation used in PEERING, implemented using the BIRD software router. Despite BIRD's limitations (most notably running a single thread), PEERING can support hundreds of peers and thousands of updates per second. Our implementation can virtualize routers in the largest IXPs today and in the foreseeable future. We also evaluate the achievable throughput of our interconnectivity across our backbone, and conclude that our backbone capacity can support a variety of experiments.

Known routes and memory utilization. vBGP employs BGP ADD-PATH to inform experiments of all available routes. This makes the number of routes managed by vBGP, and the memory requirements in PEERING, proportional to the number of routes learned across all upstream interconnections. Figure 6a shows the memory utilization of BIRD's routing tables as a function of the number of known routes. The *control plane* line corresponds to a minimal configuration with a single global Routing Information Base (RIB), required for BGP operation (but without the Forwarding Information Base (FIB) necessary to forward traffic). The *per-interconnection data plane* line adds in the overhead of vBGP, which maintains one FIB entry (in the Linux kernel) for each known route to allow experiments to choose their own routes when sending traffic. The *per-interconnection data plane with default* line additionally adds in the overhead of the router maintaining its own best-path routing table and keeping it synchronized with a kernel FIB. This additional overhead is not necessary for PEERING because vBGP experiments receive all routes via ADD-PATH and make their own routing decisions, but would be necessary if the vBGP node was also routing production traffic. Memory use in BIRD scales linearly with the number of routes, at a rate of approximately 327B/route, allowing a server with 32GiB of RAM to support 100 million routes. PEERING's vBGP router at AMS-IX, one of the largest IXPs in the world, exchanges routes with all 4 route servers, 2 transit providers, plus 235 routers in 104 member networks, and currently has 2.7 million routes from 854 ASes (combining route server and bilateral peers).

Rate of updates and CPU utilization. vBGP needs to rewrite the BGP next hop of announcements received from the Internet and filter invalid announcements from experiments (§3), which incurs additional CPU overhead in PEERING. We focus our analysis on



(a) Memory vs known routes



(b) CPU vs rate of updates

Figure 6: Memory consumption and CPU utilization grow linearly with number of routes and rate of updates. Results indicate vBGP can be deployed using commodity servers in the largest IXPs.

BIRD's CPU utilization for two reasons: (i) most of our route processing filters are implemented in BIRD (§4.7) and (ii) only experiment announcements are processed by the ExaBGP-based security engine (which is invoked infrequently as we limit experiments to 144 announcements/day per prefix).

Figure 6b shows BIRD's CPU utilization for different filter configurations as a function of the rate of BGP updates processed. We consider different filter configurations in a worst-case scenario where BIRD runs all filters to completion without rejecting any routes. The *accept* line shows results when BIRD is configured to simply accept all routes without any checks (the lower bound on CPU utilization). The *single-router* vBGP line shows the CPU utilization for the filters vBGP applies to announcements from experiments. This overestimates the complexity in actual vBGP deployments, as filters applied to updates received from the Internet (the majority) are significantly simpler than filters applied to announcements from experiments to the Internet. Finally, the *multi-router* vBGP line shows the CPU utilization for BIRD in the BGP mesh configuration on Peering's backbone, which requires a more complex handling of BGP next hops (§4.3).

The results show that CPU utilization grows linearly with the rate of updates and, more importantly, is not significantly impacted by vBGP's safety filters. During an 18h period in March 2018, Peering's vBGP router in AMS-IX processed an average of 21.8 updates/sec (with a 99th percentile of approximately 400 updates/sec). Although filters incur additional propagation delays for BGP update messages, this applies to any implementation and is small relative to global propagation delays [54] and delays imposed by BGP's built-in minimum route advertisement interval (MRAI) [102].

Data plane performance. Peering relies on Linux's networking stack for data plane forwarding, with Peering-specific configuration for virtual interfaces, multiple routing tables, and our BPF-based security framework. Peering's performance benefits from any improvements to the Linux networking stack. Significantly better performance could be achieved by techniques such as kernel bypass [92, 93] and offloading our security framework to BPF-capable NICs [64]; to date no experiments have required such capabilities and thus we leave these optimizations as future work.

To verify forwarding performance of our backbone between PoPs, we conducted throughput measurements using `iperf3`. Between sites, the average TCP throughput measurement was approximately 400 Mbps, with a minimum of 60 Mbps and a maximum of 750 Mbps between all PoP pairs. While not equivalent to the capacity available from the dedicated fiber connectivity for large content providers, the backbone connectivity between Peering locations has sufficient capacity available for supporting various experiments (including all experiments proposed to date) except those requiring enormous data transfers.

7 PEERING IN PRACTICE

7.1 How Peering Has Been Used

Since 2015, Peering has supported experiments with different goals, exposing an array of different behaviors to the Internet (including AS-path poisoning, AS-path prepending, and controlled hijacks (of Peering's own address space)), and from teams with varying degrees of experience with BGP and the Internet routing ecosystem. We rejected as risky an experiment proposal that required a large number of AS poisonings and one that planned to announce AS-paths with thousands of ASes. We granted all other requests for access to the testbed, with many experiments running for months at a time. The research performed on Peering was part of 15 publications: 3 at SIGCOMM, 2 at IEEE S&P, 2 at IMC, 2 at Usenix Security, and 1 each at ToN, SOSR, HotPETS, TMA, CCS, and CCR [13, 15, 18–20, 38, 57, 69, 79, 81, 83, 87–90]. In addition, Peering supported multiple projects at a BGP hackathon [33]. The majority of these experiments were conducted by researchers not affiliated with Peering. Earlier studies used a rudimentary version of our infrastructure that did not support multiplexing and helped inspire Peering's requirements [44, 49, 52, 54, 66, 97].

Almost all experiments announce routes (a small set have used Peering as a looking glass) and many exchange traffic. Several experiments have used BGP poisoning [13, 15, 18, 38, 90]. A few experiments have used BGP communities [20, 79, 87] and fine-grained control of announcements or traffic [20, 57, 69, 83], capabilities more recently added to the platform. We find that Peering offers the following key benefits to research using it:

Controlled experiments. Barriers exist to conducting controlled experiments on the Internet because it is a huge system with properties reliant on the complex interactions among tens of thousands of ASes with opaque topologies and policies. This challenge frequently manifests in two ways.

First, researchers rely on uncontrolled, natural experiments, in which conditions of interest vary outside of the control of the researchers and independent of the current research question. These

experiments can lead to challenges in isolating the cause of observations, as measurements are often consistent with multiple explanations, especially given researchers' limited visibility. For example, a recent study relied on passive observation of Internet routes to identify networks that deployed BGP security techniques to prevent prefix hijacks [41], but, because it relied on uncontrolled experiments without isolating security policy as the cause for routing decisions, it could misdiagnose unrelated traffic engineering as evidence of security policies [69].

Second, and closely related, researchers often lack ground truth to use in evaluating the accuracy of a system's inferences. Both cases are compounded in situations when the phenomena of interest are rare or hard to identify.

Peering gives researchers the ability to control aspects of routing as a means to conduct controlled experiments or systematically generate ground truth data. A recent study demonstrated how to use Peering to manipulate announcements in order to probe the security policies of ASes in a controlled manner, isolating the cause of decisions by varying only whether an announcement was valid [69]. Other work issued requests to a Content Delivery Network (CDN) over different paths while concurrently manipulating the performance of each path to measure the sensitivity of a traffic engineering system [81], and to generate known, ground-truth events for evaluating a system to protect Tor from routing attacks [89].

In-the-wild demonstrations. To have a better chance of adoption, extensions or alternate approaches to the Internet routing system must be incrementally deployable, backwards compatible, and should provide benefit to early adopters. Traditionally, however, evaluations are limited to emulation or simulation, and the community's limited ability to measure or model the Internet's topology or policies means that the fidelity of the evaluations can be unclear. Peering allows prototypes to interact with the real Internet to show their compatibility and capabilities, such as connecting an intradomain network to the Internet to measure the benefits of ingress traffic engineering for a multihomed AS [88], assessing a technique to identify and neutralize BGP prefix hijacking [83], or evaluating the BGP-compatibility of future Internet architectures [78]. Such demonstrations can lend credibility and encourage adoption, especially since network operators can often be conservative about change. They can also uncover pragmatic concerns that would not turn up in a lab setting.

The security community places value on real-world demonstrations of attacks, and researchers used our platform to demonstrate traffic interception attacks [20] and attacks on applications such as Tor [90], TLS certificate generation [18], and cryptocurrencies [15]. This line of work demonstrates how vulnerabilities in BGP can be exploited to create attacks on Internet-based systems for anonymity, security, and currency. The real-world demonstrations led to the adoption of countermeasures by Tor and by Let's Encrypt, the world's largest certificate authority.

Measurements of hidden routes. The design of BGP leads to routes only showing up in measurements if they are being used, providing limited visibility into backup routes, route diversity, or the underlying topology. Peering can manipulate which routes are available to reach it by using selective advertisements, AS-path prepending, BGP poisoning, or BGP communities for traffic engineering. Researchers used this ability to reverse engineer routing policy preferences at finer granularities [13] than was possible previously [40, 59].

7.2 Native Delegation is a Cornerstone for Generality

The decision to seek out flexible solutions using existing layer 2 protocols, IP, and BGP instead of non-native approaches (§§ 2.2 and 2.3) proved to be a good design decision that enabled a wide range of experiments. Many experiments focused on the interactions between applications and BGP and required flexibility and full delegation of both the data and control planes [15, 18, 19, 57, 81, 88–90]. Alternative interfaces, such as a custom out-of-band mechanism or an API with a BGP beacon [71], would not suffice for many experiments and would need to be extended for each experiment with new requirements. In addition, we find that experimenters are generally familiar with and expect to be able to use an existing routing engine (such as BIRD)—because our approach is inherently compatible with existing BGP deployments and tooling, Peering can support these experiments without modification.

A transit provider could adapt our approach to allow customers to choose among multiple routes. It is a possible deployment pathway for flexible routing schemes proposed in research [45, 101] or SDN control over BGP via existing devices, whereas other approaches, like Espresso [104], require replacing infrastructure. For instance, our approach inspired a variation that we helped deploy in production at Facebook to route traffic via alternate routes [81]. The variant uses per-packet data plane signaling and multiple routing tables at routers, but a centralized controller decides which routes to use and injects them into tables at routers.

7.3 Cooperation with Network Operators

Researchers must push boundaries to test and evaluate protocols, implementations, and potential solutions. Safely conducting experiments is challenging due to the large variation in implementations and configurations of routers [36, 75]. Conversely, the operational community's primary goal is stability and security. Although the goals seem diametrically opposed, operators are supportive and appreciative of research in the area, especially when researchers announce experiments and take feedback prior to execution.

Peering provides an environment for "white-hat" hackers to conduct experiments that rely on BGP manipulation [15, 18, 87, 90]. The experiment review process, conservative security policies, and cooperation with the operator community combine to enforce ethical use of the platform. When in doubt, we err on the side of safety. For example, one recent experiment requested feedback from NANOG in advance and proceeded to make (standards-compliant) announcements on a fixed schedule [58]. The announcements identified a vulnerability in an open-source routing daemon which caused BGP sessions to reset [94]. Although the majority of operator responses on the NANOG mailing list supported continuing the experiment, the experiment was halted until affected systems could be patched.

7.4 Experiments Peering Does Not Support

No direct control over other networks. Although Peering delegates its data and control planes to experiments, an experiment's announcements and traffic are subject to policies enforced by other networks in the Internet, which experiments have no control over. This limitation is not specific to Peering, it is intrinsic to the Internet's architecture. Experiments need to plan for the lack of control; e.g., an experiment that studied routing policies deployed hundreds of different announcement configurations to exercise and observe policies in different scenarios [13]. However, even without direct control over other networks, Peering can still influence their routes and traffic towards its prefixes.

Peering can support limited types of experiments with multiple ASes: Peering operates multiple ASNs, which allows experiments to emulate multiple networks that interact with the Internet as customers of Peering's main AS or of each other.

No high volume, production, or transit traffic. Peering employs servers as routers and does not operate any network links. Peering's backbone links are provisioned on top of other networks' infrastructures, and researchers connect their experiments to Peering routers using VPN tunnels over the Internet. Although peering can support experiments that exchange traffic at moderate rates (hundreds of Mbps, §6), capacity varies by PoP and is ultimately limited. Peering, as a research platform, also does not provide any guarantees on availability. Peering also blocks announcements and traffic for prefixes outside its IP space, and so experiments cannot transit traffic that is neither from nor to a Peering address.

Limited support for latency-sensitive or real-time experiments. Experiments connect to Peering PoPs via an OpenVPN tunnel. As a result, experiment traffic traverses the tunnels, incurring additional latency and impacting latency-sensitive or real-time experiments. Experiments desiring low latency can deploy on (and tunnel from) CloudLab (with which we federate and colocate PoPs) or cloud providers with low-latency paths to Peering PoPs (*e.g.,* Peering peers directly with some cloud providers). We also have a preliminary implementation of an extension to our platform to support lightweight applications on experiment-controlled containers running directly on Peering servers [50].

8 RELATED WORK

Existing tools such as BGP collectors [73, 76, 103] and looking glass services [42] directly measure the Internet control plane, while ping and traceroute indirectly measure control plane state via data plane measurements. In comparison, Peering enables experiments to *interact* with the Internet's control plane.

Prior work that interacted with the Internet's control plane established ad-hoc ASes that did not support parallel experiments [16, 21, 34, 96]. Each network had five to nine routers with a single upstream provider each. Establishing an AS per experiment represents an insurmountable barrier for many researchers, especially those who want rich connectivity and a global footprint similar to content and cloud providers.

FlowVisor [84] has similar goals to our work, but enables experiments within a single domain. Its OpenFlow-based design is not directly applicable to our interdomain setting, which introduces new challenges related to the need to interoperate with other networks via BGP, a protocol with intrinsic limitations. SDX [44] uses layer 2 signaling to forward traffic in an IXP switch, but the participants convey policy to the IXP to enact decisions. vBGP delegates decisions to the experiment and conveys the decision via the header.

Peering has been under development since 2014, and in parallel other network operators and researchers discovered opportunities to use layer 2 mechanisms (and specifically MAC addresses) for signaling [11, 14, 44]. However, these efforts use SDN solutions, which are applicable within a single domain, but require additional signaling mechanisms. Instead we sought seamless compatibility with existing routing engines using standard protocols.

9 CONCLUSION

Internet routing research has been limited by obstacles in executing experiments in the Internet. Without control of an AS, researchers are limited to simulations, which cannot realistically capture Internet properties, and measurements, which can observe routes as they are but cannot manipulate them to study the impact.

We presented Peering, a production platform that realizes our vision of enabling turn-key Internet routing research. Peering is built atop vBGP, a system that we designed to virtualize the data and control planes of a BGP edge router, while providing security mechanisms to prevent experiments from disrupting the Internet or each other. vBGP supports parallel experiments, each with control and visibility equivalent to having sole ownership of the router, using standard interfaces, which provide realism and flexibility.

With Peering, experiments operate in an environment that is qualitatively similar to that of a cloud provider, and can exchange routes and traffic with hundreds of other networks at locations around the world. To date, Peering's rich connectivity and flexibility have enabled it to support over 40 experiments and 15 publications in research areas such as security, network behavior, and route diversity.

Acknowledgements. Our work benefited from the contributions and vision of collaborators on earlier iterations, including Vytautas Valancius, Nick Feamster, Kyriakos Zarifis, and Christopher Hanford. Peering would not be available without the researchers and network administrators at the sites, Internet2, and RNP; the generous hosting provided by AMS-IX, Phoenix-IX, and the SIX; and transit provided by Randy Bush, Coloclue, and BIT.BV. We appreciate the valuable feedback from our shepherd Xenofontas Dimitropoulos, the reviewers, and network operators who helped with experiment design. This work was partly funded by NSF awards CNS-1740883, CNS-1835252, CNS-1413978, and CNS-1836872; RNP project 2955; CNPq award 311049; and CAPES award 88881.171646.

A CHALLENGES IN DEBUGGING AND OPERATION

Debugging route propagation. We found instances of PEERING announcements not being globally reachable due to improperly configured or out-of-date filters in other networks. Networks employ route import and export filters to prevent route leaks and prefix hijacks [32, 68]. When debugging these situations, our goal is to identify the network that is incorrectly filtering, but the process is manual and relies heavily on looking glass servers [42]. Although looking glasses help, they cannot accurately pinpoint filters because they only provide a restricted command line interface. Even in the optimistic scenario where two directly-connected networks A and B have looking glasses, if network A has the route and network B does not, the looking glasses do not allow us to disambiguate between (1) network A not exporting the route to B or (2) network B filtering the route received from A. In practice, debugging usually requires emailing our transit providers. They, in turn, may have to email their providers.

As future work, we plan to investigate the more general problem of identifying whether networks do not appear on routes to our prefixes because they are misconfigured or because they are less preferred than other providers. We plan to evaluate methods for automated filter troubleshooting.

Debugging Layer 2 Connectivity. Systems like AL2S promote automation in educational backbones. However, at university sites, PEERING servers may be deployed to facilities (e.g., 'server rooms' or labs) that are not managed by the university's core network operators. PEERING servers may also interact with equipment that is not under complete control of the university's networking team, such as when federating with other testbeds (e.g., CloudLab switches). These facilities are often out of the reach of automated management, and relatively straightforward tasks, like trunking a VLAN from a core router connected to Internet2 to a server at a (possibly unmanaged) location, can be surprisingly difficult. Operators would benefit from tools to ease debugging and network management systems suited to these environments.

REFERENCES

[1] [n.d.]. Advanced Layer 2 Services. https://www.internet2.edu/products-services/advanced-networking/layer-2-services/.
[2] [n.d.]. CloudLab. http://www.cloudlab.us/.
[3] [n.d.]. Emulab. https://www.emulab.net/.
[4] [n.d.]. ExaBGP. https://github.com/Exa-Networks/exabgp.
[5] [n.d.]. GÉANT. https://www.geant.org/.
[6] [n.d.]. OpenVPN. https://openvpn.net/.
[7] [n.d.]. PeeringDB. https://www.peeringdb.com/.
[8] [n.d.]. PlanetLab. https://www.planet-lab.org/.
[9] [n.d.]. The BIRD Internet Routing Daemon. http://bird.network.cz/.
[10] [n.d.]. XSEDE. https://www.xsede.org/.
[11] Kanak Agarwal, Colin Dixon, Eric Rozner, and John Carter. 2014. Shadow MACs: Scalable Label-switching for Commodity Ethernet. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*.
[12] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. 2012. Anatomy of a Large European IXP. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '12)*.
[13] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Italo Cunha, Phillipa Gill, and Ethan Katz-Bassett. 2015. Investigating interdomain routing policies in the wild. In *Proceedings of the ACM Internet Measurement Conference (IMC '15)*.
[14] Jo ao Taveira Araùjo. 2016. Building and scaling the Fastly network, part 1: Fighting the FIB. https://www.fastly.com/blog/building-and-scaling-fastly-network-part-1-fighting-fib.
[15] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy (S&P '17)*.
[16] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. 2006. A Measurement-Based Deployment Proposal for IP Anycast. In *Proceedings of the ACM Internet Measurement Conference (IMC '06)*.
[17] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.
[18] Henry Birge-Lee, Yixin Sun, Annie Edmundson, Jennifer Rexford, and Prateek Mittal. 2017. Using BGP to Acquire Bogus TLS Certificates. In *Privacy Enhancing Technologies Symposium (HotPETS '17)*.
[19] Henry Birge-Lee, Yixin Sun, Annie Edmundson, Jennifer Rexford, and Prateek Mittal. 2018. Bamboozling Certificate Authorities with BGP. In *27th USENIX Security Symposium (USENIX Security '18)*.
[20] Henry Birge-Lee, Liang Wang, Jennifer Rexford, and Prateek Mittal. 2019. SICO: Surgical Interception Attacks by Manipulating BGP Communities. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*.
[21] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. 2009. Internet Optometry: Assessing the Broken Glasses in Internet Reachability. In *Proceedings of the ACM Internet Measurement Conference (IMC '09)*.
[22] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. 2018. Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*.
[23] B. Carpenter and K. Moore. 2005. Connection of IPv6 Domains via IPv4 Clouds. RFC3056.
[24] R. Chandra and R. Traina. 1996. BGP Communities Attribute. RFC1997.
[25] Rocky K.C. Chang and Michael Lo. 2005. Inbound Traffic Engineering for Multihomed ASes Using AS Path Prepending. *IEEE Network* 19, 2 (2005), 18–25.
[26] Nikolaos Chatzis, Georgios Smaragdakis, Anja Feldmann, and Walter Willinger. 2013. There is More to IXPs than Meets the Eye. *SIGCOMM Comput. Commun. Rev. (CCR)* (2013).
[27] Kai Chen, David R. Choffnes, Rahul Potharaju, Yan Chen, Fabian E. Bustamante, Dan Pei, and Yao Zhao. 2009. Where the Sidewalk Ends: Extending the Internet AS Graph Using Traceroutes from P2P Users. In *Proceedings of Conference on emerging Networking EXperiments and Technologies (CoNEXT '09)*.
[28] Marco Chiesa, Daniel Demmler, Marco Canini, Michael Schapira, and Thomas Schneider. 2017. SIXPACK: Securing Internet eXchange Points Against Curious onlooKers. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*.
[29] Yi-Ching Chiu, Brandon Schlinker, Abhishek Balaji Radhakrishnan, Ethan Katz-Bassett, and Ramesh Govindan. 2015. Are We One Hop Away from a Better Internet?. In *Proceedings of the ACM Internet Measurement Conference (IMC '15)*.
[30] Cisco. 2005. Understanding Policy Routing.
[31] Cisco. 2008. Virtual Route Forwarding Design Guide.
[32] Jim Cowie. 2013. The New Threat: Targeted Internet Traffic Misdirection. Dyn Research Blog. http://research.dyn.com/2013/11/mitm-internet-hijacking/
[33] Alberto Dainotti, Ethan Katz-Bassett, and Xenofontas Dimitropolous. 2016. The BGP Hackathon 2016 Report. (2016).
[34] Wouter B. de Vries, Ricardo de O. Schmidt, Wes Hardaker, John Heidemann, Pieter-Tjerk de Boer, and Aiko Pras. 2017. Broad and Load-aware Anycast Mapping with Verfploeter. In *Proceedings of the ACM Internet Measurement Conference (IMC '17)*.
[35] Dyn. [n.d.]. Pakistan hijacks YouTube. https://dyn.com/blog/pakistan-hijacks-youtube-1/.
[36] Dyn. [n.d.]. Reckless Driving on the Internet. https://dyn.com/blog/the-flap-heard-around-the-world/.
[37] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.
[38] Julián Martín Del Fiore, Pascal Merindol, Valerio Persico, Cristel Pelsser, and Antonio Pescapè. 2019. Filtering the Noise to Reveal Inter-Domain Lies. In *Proc. of the Network Traffic Measurement and Analysis Conference (TMA '19)*.
[39] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd D Millstein. 2015. A General Approach to Network Configuration Analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*.
[40] Lixin Gao. 2001. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking (ToN)* 9, 6 (2001), 733–745.
[41] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. 2017. Are We There Yet? On RPKI's Deployment and Security. In *Proceedings of Network and Distributed System Security Symposium (NDSS '17)*.
[42] Vasileios Giotsas, Amogh Dhamdhere, and Kimberly C. Claffy. 2016. Periscope: Unifying Looking Glass Querying. In *Proceedings of International Conference on Passive and Active Network Measurement (PAM '16)*.

[43] Arpit Gupta, Robert MacDavid, Rüdiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, and Laurent Vanbever. 2016. An Industrial-Scale Software Defined Internet Exchange Point. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*.

[44] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. 2014. SDX: A Software Defined Internet Exchange. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '14)*.

[45] Jiayue He and Jennifer Rexford. 2008. Toward Internet-Wide Multipath Routing. *IEEE Network* 22, 2 (March 2008), 16–21.

[46] J. Heitz, J. Snijders, K. Patel, I. Bagdonas, and N. Hilliard. 2017. BGP Large Communities Attribute. RFC8092.

[47] Thomas Holterbach, Stefano Vissicchio, Alberto Dainotti, and Laurent Vanbever. 2017. SWIFT: Predictive Fast Reroute. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.

[48] Mark Huang, Andy Bavier, and Larry Peterson. 2006. PlanetFlow: Maintaining Accountability for Network Services. In *ACM SIGOPS Operating Systems Review (SOSR '06)*.

[49] Umar Javed, Italo Cunha, David R. Choffnes, Ethan Katz-Bassett, Thomas E. Anderson, and Arvind Krishnamurthy. 2013. PoiRoot: Investigating the Root Cause of Interdomain Path Changes. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '13)*.

[50] Brivaldo Junior, Ronaldo A. Ferreira, Ítalo Cunha, Brandon Schlinker, and Ethan Katz-Bassett. 2018. High-Fidelity Interdomain Routing Experiments. In *ACM SIGCOMM Posters and Demos (SIGCOMM '18)*.

[51] Juniper. 2017. Routing Instances Overview.

[52] Ethan Katz-Bassett, David R Choffnes, Ítalo Cunha, Colin Scott, Thomas Anderson, and Arvind Krishnamurthy. 2011. Machiavellian Routing: Improving Internet Availability with BGP Poisoning. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNETS '11)*.

[53] Ethan Katz-Bassett, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, David Wetherall, and Thomas Anderson. 2008. Studying Black Holes in the Internet with Hubble. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*.

[54] Ethan Katz-Bassett, Colin Scott, David R. Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V. Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. 2012. LIFEGUARD: Practical Repair of Persistent Route Failures. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '12)*.

[55] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. 2010. Internet Inter-domain Traffic. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '10)*.

[56] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software Defined Networks. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HotNets '10)*.

[57] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2018. Internet Anycast: Performance, Problems, & Potential. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*.

[58] NANOG Mailing List. 2019. BGP Experiment – Thread. https://mailman.nanog.org/pipermail/nanog/2019-January/098761.html

[59] Matthew Luckie, Brian Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and kc claffy. 2013. AS Relationships, Customer Cones, and Validation. In *Proceedings of the ACM Internet Measurement Conference (IMC '13)*.

[60] Ratul Mahajan, David Wetherall, and Thomas Anderson. 2005. Negotiation-based routing between neighboring ISPs. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*.

[61] Ratul Mahajan, David Wetherall, and Tom Anderson. 2012. Understanding BGP Misconfiguration. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '12)*.

[62] Zhuoqing Morley Mao, Jennifer Rexford, Jia Wang, and Randy H. Katz. 2003. Towards an Accurate AS-level Traceroute Tool. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '03)*.

[63] Stefan Meinders. 2016. In *RIPE NCC Regional Meeting: Eurasia Network Operators Group (ENOG 11)*.

[64] Metronome Systems Inc. 2018. eBPF Offload Getting Started Guide. https://www.netronome.com/m/documents/UG_Getting_Started_with_eBPF_Offload.pdf.

[65] Luis Pedrosa, Ari Fogel, Nupur Kothari, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. Analyzing Protocol Implementations for Interoperability. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*.

[66] Simon Peter, Umar Javed, Qiao Zhang, Doug Woos, Arvind Krishnamurthy, and Thomas Anderson. 2014. One Tunnel is (Often) Enough. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '14)*.

[67] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. 2006. Experiences Building PlanetLab. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*.

[68] Alex Pilosov and Tony Kapela. 2008. Stealing The Internet: An Internet-Scale Man In The Middle Attack. In *DEFCON 16*.

[69] Andreas Reuter, Randy Bush, Italo Cunha, Ethan Katz-Bassett, Thomas C. Schmidt, and Matthias Wählisch. 2018. Towards a Rigorous Methodology for Measuring Adoption of RPKI Route Validation and Filtering. *SIGCOMM Comput. Commun. Rev. (CCR)* (2018).

[70] Philipp Richter, Georgios Smaragdakis, Anja Feldmann, Nikolaos Chatzis, Jan Boettger, and Walter Willinger. 2014. Peering at Peerings: On the Role of IXP Route Servers. In *Proceedings of the ACM Internet Measurement Conference (IMC '14)*.

[71] RIPE NCC. 2018. Current RIS Routing Beacons. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/current-ris-routing-beacons.

[72] ripeatlas [n.d.]. RIPE Atlas. https://atlas.ripe.net/.

[73] riperis [n.d.]. RIPE Routing Information Service (RIS). http://www.ripe.net/ris/.

[74] rnp [n.d.]. Rede Nacional de Ensino e Pesquisa. https://www.rnp.br/.

[75] Erik Romijn. 2010. RIPE NCC and Duke University BGP Experiment. https://labs.ripe.net/Members/erik/ripe-ncc-and-duke-university-bgp-experiment

[76] routeviews [n.d.]. The University of Oregon Routeviews Project. http://www.routeviews.org.

[77] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. 2003. Linux Netlink as an IP Services Protocol. RFC3549.

[78] Raja R Sambasivan, David Tran-Lam, Aditya Akella, and Peter Steenkiste. 2016. Bootstrapping Evolvability for Inter-domain Routing with D-BGP. In *CMU-CS-16-117 Tech Report*.

[79] Raja R Sambasivan, David Tran-Lam, Aditya Akella, and Peter Steenkiste. 2017. Bootstrapping Evolvability for Inter-Domain Routing with D-BGP. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.

[80] Brandon Schlinker, Ítalo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. 2019. A View of Internet Performance from a Global Content Provider's Edge. In *Proceedings of the ACM Internet Measurement Conference (IMC '19)*.

[81] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.

[82] Brandon Schlinker, Kyriakos Zarifis, Italo Cunha, Nick Feamster, and Ethan Katz-Bassett. 2014. Peering: An AS for Us. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets '14)*.

[83] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. 2018. ARTEMIS: Neutralizing BGP Hijacking within a Minute. *IEEE/ACM Transactions on Networking* 26, 6 (Dec 2018).

[84] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. 2009. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep* 1 (2009), 132.

[85] Ben Treynor Sloss. [n.d.]. Expanding our global infrastructure with new regions and subsea cables.

[86] Neil T. Spring, Ratul Mahajan, and Thomas E. Anderson. 2003. The Causes of Path Inflation. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '03)*.

[87] Florian Streibelt, Franziska Lichtblau, Robert Beverly, Anja Feldmann, Cristel Pelsser, Georgios Smaragdakis, and Randy Bush. 2018. BGP Communities: Even more Worms in the Routing Can. In *Proceedings of the ACM Internet Measurement Conference (IMC '18)*.

[88] Peng Sun, Laurent Vanbever, and Jennifer Rexford. 2015. Scalable Programmable Inbound Traffic Engineering. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*.

[89] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. 2017. Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *IEEE Symposium on Security and Privacy (SP '17)*.

[90] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor. In *24th USENIX Security Symposium (USENIX Security 15)*.

[91] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. 2016. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '16)*.

[92] The Linux Foundation. 2018. Data Plane Development Kit. https://www.dpdk.org.

[93] The Linux Kernel Documentation. 2018. AF_XDP Overview. https://www.kernel.org/doc/html/latest/networking/af_xdp.html.

[94] The Mitre Corporation. 2019. CVE-2019-5892. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5892.
[95] Linus Torvalds. 2015. Linux Kernel 4.1—Release Notes. https://kernelnewbies.org/Linux_4.1.
[96] Vytautas Valancius, Nick Feamster, Jennifer Rexford, and Akihiro Nakao. 2010. Wide-Area Route Control for Distributed Services. In *USENIX Annual Technical Conference (USENIX ATC '10)*.
[97] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C. Snoeren. 2013. Quantifying the Benefits of Joint Content and Network Routing. In *SIGMETRICS*.
[98] D. Walton, A. Retana, E. Chen, and J. Scudder. 2016. Advertisement of Multiple Paths in BGP. RFC 7911.
[99] Robin Whittle. [n.d.]. [rrg] Geoff Huston's BGP/DFZ research. https://www.ietf.org/mail-archive/web/rrg/current/msg06163.html.
[100] Florian Wohlfart, Nikolaos Chatzis, Caglar Dabanoglu, Georg Carle, and Walter Willinger. 2018. Leveraging Interconnections for Performance: The Serving Infrastructure of a Large CDN. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*.
[101] Wen Xu and Jennifer Rexford. 2006. MIRO: Multi-path Interdomain ROuting. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '06)*.
[102] Y. Rekhter and T. Li and S. Hares. 2006. A Border Gateway Protocol 4 (BGP-4). RFC4271.
[103] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. 2009. BGPmon: A real-time, scalable, extensible monitoring system. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. IEEE, 212–223.
[104] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.