Logical Peering for Interdomain Networking on Testbeds

Yuanjun Yao† Qiang Cao† Paul Ruth§ Mert Cevik§ Cong Wang§ Jeff Chase†
†Duke University §RENCI †University of Georgia

Abstract—Research testbed fabrics have potential to support long-lived, evolving, interdomain experiments, including opt-in application traffic across multiple campuses and edge sites. We propose abstractions and security infrastructure to facilitate multi-domain networking, and a reusable controller toolkit (ExoPlex) for network service providers (NSPs) running in testbed-hosted virtual network slices. We demonstrate the idea on the ExoGENI testbed, which allows slices to interconnect and exchange traffic over peering links by mutual consent.

Each ExoPlex NSP runs a peering controller that manages its interactions with its linked peers and controls the NSP’s dataplane network via SDN. Our approach expresses policies for secure peering and routing in a declarative language—logical peering. The prototype uses logic rules to verify IP prefix ownership, filter and validate route advertisements, and implement user-specified policies for connectivity and path control in networks with multiple transit NSPs.

Index Terms—Networks, Testbeds, SDN, Policy Based Routing, Secure Routing, Internet Security

I. INTRODUCTION

Advanced network testbeds can serve as platforms to pilot new network transit services in testbed slices, and evolve them under real usage experience. This paper proposes an approach to secure inter-domain networking among testbed-hosted slices, and reports on experiments using ExoGENI [1] and the Internet-2/AL2S L2 circuit service. Our software and results generalize to testbeds with these enabling capabilities:

- **Dynamic slices with virtual dataplanes.** ExoGENI defines IaaS APIs to provision network topologies and program them with software-defined networking (SDN). In our model, slices may act as Network Service Providers (NSPs) that offer transit service for IP traffic.1

- **NSP peering.** ExoGENI slices may declare stitchports and interconnect (stitch) them by mutual consent [2], e.g., at an exchange site or by AL2S circuits. Testbed support for cross-slice stitching enables NSP slices to peer at L2 programmatically, even if they have different owners.

- **Customer opt-in.** A slice may peer with an NSP provider and exchange IP traffic over the link. In addition, campus networks increasingly support SDN bypass services for authorized subnets to route/accept selected traffic through a dynamic L2 network circuit, which may link to a testbed-hosted NSP. In this way a subnet owner may “opt in” to use an NSP as an alternate Internet Service Provider for selected prefixes.

1We program NSP dataplanes with OpenFlow SDN, which is limited to IP.

These capabilities enable experimental NSP services that can carry real user traffic across research fabrics. For example, we envision that NSPs can offer security-managed connectivity with policy controls to enable or disable flows; impose security scanning or other NFV service chains on specified flows; protect against spoofing, hijacking, and DDoS attacks; or configure other defenses that are lacking in the public Internet. This paper extends our previous work toward that goal [2], [3] with support for secure policy-based inter-domain routing among transit NSPs. It leads us to a vision of inter-domain traffic control within a network of NSPs, which may be experimental (e.g., user-managed), elastic, dynamic, and/or restricted to certain classes of traffic, e.g., high-priority data for a specific project. It could enable advanced network services as NSPs that weave into the fabric of the Internet over time through cycles of innovation and adoption.

This paper proposes and demonstrates policy-based inter-domain NSP networking in the ExoPlex toolkit—software elements that run within testbed slices and their controllers—to build NSPs and interconnect them securely. ExoPlex addresses common security needs for experimental interdomain networking (§II), including route security (§III) with custom policies for peering, route filtering, and path control, expressed in a logical trust language. §IV presents experiments.

The contributions of this paper are to: (1) expose security control abstractions for interdomain experiments with programmable security policy (a “testbed for trust”); (2) show how to support them in a reusable toolkit using a logical trust model; and (3) demonstrate them in proof-of-concept experiments with multiple testbed-hosted transit providers and custom policies for path control.

II. LOGICAL PEERING IN EXOPLEX

ExoPlex combines logical trust with functions for NSPs to manage an elastic network topology, control traffic in their dataplanes with SDN, and peer by stitching. Together, these functions enable a powerful platform for policy-driven interdomain networking with a compact implementation.

**The players (principals).** Each participating network domain (NSP or edge subnet) is controlled by a security principal with a keypair. Interacting network domains are necessarily embedded within some governance structure with additional principals, e.g., to assign addresses within a common space. For example, communicating subnets must own compatible IP prefixes delegated to them from common trust roots, and
policies may rely on security tags (attributes) of principals or networks asserted by various endorsing authorities. Principals use their keypairs to sign their requests, delegations, policies, endorsements, and/or advertised routes.

**Governance.** ExoPlex supports an open governance model for flexible experimentation. Each party specifies the trust roots and governance rules that it subscribes to using logic. Parties may interact only to the extent that their structures and rules are compatible. The experiments in this paper use a simple governance model in which common trust anchors—accepted by all participants—delegate IP prefix ownership and endorse/certify NSPs with security attributes (tags). Exoplex builds its secure control network over the existing public Internet, e.g., so that NSP controllers can invoke one another’s APIs for peering.

**Security model for peering.** NSP controllers expose APIs to negotiate link stitching. An NSP’s policies may limit the customers or peers that it accepts. Once a peering link is established, either side may advertise routes for subnet prefixes to the other. Secure interdomain routing requires that NSPs validate prefix ownership (origin authentication) and transitive route advertisements end-to-end (route validation), similarly to Internet security standards such as RPKI [4] and S-BGP [5] or BGPsec [6]. In this paper we add customer-specified policies for off-by-default connectivity and path control, which limit traffic and constrain eligible routes based on security attributes of the NSPs and subnets.

As an exemplary demonstration, we deploy an inter-domain network with ten ExoGENI slices representing edge providers (SDX), transit NSPs, and customer domains (Figure 1). In the demo scenario, customers specify path control policies that confine their traffic to compliant paths through qualified carriers—NSPs endorsed with specified tags anchored in trust roots that the customer accepts. For example, an endorsing authority might issue a signed assertion tagging the NSP with public key $K$ as “production-grade safe” or “classified secure”.

**Standards and interoperability.** Networks base routing and security functions on well-specified protocol standards that allow for multiple interoperable implementations. In this work we take a first step by defining a common software platform—ExoPlex—that NSP controller software may use to manage their interactions and program their internal dataplane networks accordingly. Logical peering in the control plane offers alternatives to relevant Internet standards (e.g., BGPsec and RPKI), but with a simpler deployment for SDN-enabled testbeds, no dataplane entanglements, flexible governance, and extended policy options (e.g., path control as in our experiments). Because security metadata propagates through the control plane APIs over the public Internet, all crypto operations occur off of the dataplane.

**Logical trust.** We use a logical trust language (datalog) to represent all security metadata, including endorsements, prefix delegations, routes, and policies. The SAFE [7] logical trust framework defines a certificate format for signed logic payloads, and a validation engine for policy checks incorporating an off-the-shelf datalog engine (Styla). The logic vocabulary is extensible and enables a wide range of policies and trust structures without changing the certificate format or platform.
implementation. Our approach is inspired by earlier work on networking using datalog, e.g., [8], [9].

The logical trust approach is a rapid prototyping vehicle for experimental approaches to secure peering. Although the power and flexibility of trust logic imposes substantial costs as prototyped, they are off of the dataplane and accrue only on changes to the network (e.g., peer link stitching, new prefix announcements) or its security policies. Importantly, the logic approach permits but does not require participants to write logic code: they can delegate their policies to others, take prepackaged policy off the shelf (e.g., from federation authorities), or use packaged logic for common structures and access control abstractions.

Threat model. We use logical peering to express rules that defend against IP spoofing, route hijacking, and unauthorized traffic. We provide standard logic rules for origin authentication and transitive route validation, modeling RPKI and BGPsec. We use path control to illustrate the potential for custom policies for logical peering. With path control, the transit path for each flow is compliant end-to-end with rules specified by the endpoints: the endpoints trust each NSP along the path to be faithful to the policies and to forward and accept traffic only along the trusted path, providing deep defenses against spoofing.

III. DESIGN OVERVIEW

Figure 2 depicts an ExoPlex NSP and its controller, which is layered above its SDN controller(s), the SAFE logical trust engine, and a testbed-specific IaaS plugin (slice controller). For ExoGENI, the IaaS plugin uses the Ahab library to build and maintain the NSP’s topology by invoking ExoGENI’s API for dynamic slices. ExoPlex includes OpenFlow SDN controller software to program the NSP dataplane, based on an extended Ryu rest-router module. NSP controllers may control an elastic topology or incorporate NFV and SDN-based traffic engineering. These elements are outside the scope of this paper’s focus on logical peering.

ExoPlex extends to testbeds other than ExoGENI. NSPs may replace the IaaS plugin for another dynamic slice API, or run without one over any static SDN-programmable dataplane topology. We have deployed ExoPlex NSPs over Corsa switch VFCs (virtual forwarding contexts) in the Chameleon [10] and ESnet testbeds.

An NSP controller exposes a northbound control plane API for its customers and peers to request peering links and notify the NSP of new policies and routes. Calls to these APIs drive all control plane interactions to propagate routes and policies across the interdomain network. The handler for an incoming call invokes a local SAFE engine to perform various validation checks, then optionally modifies its network state and propagates notifications to peers. Outgoing route advertisements are signed under the NSP’s keypair. NSP controllers are assumed to be reachable to one another, e.g., on the public Internet.

ExoPlex includes a standard set of controller API handlers and SAFE trust scripts, which together determine when and how to install or withdraw routes and filtering rules in the NSP dataplane via the SDN controller APIs. We extend the SDN controller for ingress filtering and source-specific routes to support policies for path control and anti-spoofing defenses. The trust scripts define logic templates, standard validation rules for incoming routes; hooks for custom authorization rules for peer requests and permissioned flows [3]; and custom policy rules to filter outgoing routes. We extended these rules to validate multi-hop paths through multiple transit NSPs.
A. Logical Policy

A *logical policy* is expressed as a set of logical facts and rules to govern and authorize routes and traffic. NSPs subscribe to standard rules to validate routes and authenticate IP origin prefixes. In addition, customer subnets may specify policies that guard connectivity to their prefixes and/or constrain the paths for inbound and/or outbound flows. Associated NSPs receive those policies and evaluate compliance. For example, a subnet’s direct provider (labeled SDX in Figure 1) receives connectivity policy from the subnet and blocks traffic from unauthorized senders on the last hop before delivery.

Policy rules may query statements and security attributes of other relevant parties. For example, connectivity rules may query attributes of the source. The policy also defines which authorities may assert/endorse these attributes. The standard route validation rules authenticate the origin as the owner of the prefix according to the NSP’s governance rules.

The logical trust approach makes it easy to express and share governance policy in logic, independent of other elements of the implementation. A policy might express a federation structure or, alternatively, a set of ad hoc trust agreements among the interacting parties. For example, the prefix ownership rules in our prototype express a structure similar to the public Internet, in which prefixes are delegated transitively through a hierarchy of owners, with range containment checked at each level. The participants must agree on the roots of authority, as in RPKI.

NSP controllers check policy compliance by issuing scripted queries to a local SAFE logic engine, passing a logic context—a set of certified facts and rules in datalog. The trust scripts construct each logic context and incorporate relevant assertions and policy rules extracted from signed SAFE certificates, and selected local logic. SAFE certificates may be passed by reference via a *link*, and a certificate may embed links to other certificates. Trust scripts retrieve and follow these links to construct the context for a compliance check. In particular, ExoPlex NSPs propagate links to customer-specified path control policies along with routes, and index them by prefix pairs in an Area-based Quad Tree (AQT) [11].

The logical approach allows any participant to check compliance with another’s policy on its behalf. For example, customers trust their edge providers (SDX) to enforce their connectivity policies. NSPs along a valid path cooperate to enforce customer-specified path control policies; the customer trusts these NSPs to be faithful to the policy.

B. Secure Routing and Prefix Ownership

ExoPlex includes off-the-shelf trust logic scripts for secure routing, including certified route advertisements modeled on BGPsec and prefix ownership modeled on RPKI.

**Route validation.** An NSP controller invokes a trust script to sign its route advertisements and to validate advertised routes. Each hop of a route is a logical assertion advertising to a peer NSP a route for a specified destination prefix, along with an ordered list of predecessors (PrincipalIDs) in the path: advertise(?DstPrefix, ?Path, ?Peer).

The issuing NSP invokes a script to encode the advertisement in a logical certificate and sign it under the issuer’s keypair. The certificate links to the next hop in the chain of predecessor advertisements.

**Prefix ownership.** The origin of a valid route must own the destination prefix. The origin links its initial advertisement to a certificate set with evidence that it owns the prefix. As the route propagates, each NSP in turn applies local policy rules to this logic set to validate the origin’s ownership of the prefix. ExoPlex includes a trust script to delegate a prefix to a named principal, linked to a predecessor as evidence that the issuer owns the containing prefix.

C. Path Control

**Path control.** For this paper, we added support for customers (subnet owners) to express logical policy rules for path control. These rules qualify which NSPs are eligible to carry their traffic, e.g., based on secure attributes of the NSPs. Interdomain routing in ExoPlex finds the least-cost paths that are compliant with registered policies of both the source and destination subnets, if such paths exist. A path (route) is compliant with the policy iff it traverses only qualified NSPs.

The subnet owner issues a path control policy as a certificate, and notifies its provider, passing the policy link. A policy notification associates each policy with a prefix pair (source, dest), which may be wildcarded. The route for a packet is governed by the policy with the most specific enclosing prefix pair, if any, for the packet’s source, dest addresses. If both source and destination assert a policy, then a compliant route complies with both.

**Inbound path control.** An *inbound* policy qualifies NSPs to carry traffic to a destination prefix, and originates from the owner of the prefix. The subnet owner trusts the qualified NSPs, for example, to block any traffic to the destination from spoofed source addresses. Inbound path control policies are attached to a route advertisement and propagate with the route advertisement. Each NSP propagates routes only to peer NSPs that are compliant with the route’s policy.

**Outbound path control.** An *outbound* policy qualifies NSPs to carry traffic from the source prefix S (whose owner specifies the policy) to the destination prefix D. A customer passes outbound path control policies to the provider in a separate API call, which it may invoke at any time.

Upon receiving an outbound policy event, an NSP N validates its default route for (S, D) (if any) for compliance with the new policy. If the route is not compliant, then N must find an alternative compliant route, even if it is longer than the current route, and then propagate it.

To do this, N considers other cached routes to D. Consider a cached route R. N received R previously from a peer, but N did not select or propagate R because N instead selected a shorter route (e.g., the current route). If R is the shortest known compliant route, then N selects R for (S, D), replacing the current route for any flows that are within the scope of the new policy. If N knows no compliant route R, then it propagates the policy to at least all compliant peers that have
advertised a valid route to D, indicating that the peer is also compliant with D’s inbound policy. These peers handle the event similarly.

Eventually, if a compliant path exists, some compliant NSP identifies a compliant R and advertises it as described above. The route propagates in the usual fashion and eventually the SDX for S receives it. Along the way, each NSP on the path chooses and installs a compliant sub-route R for matching flows. If an NSP later learns of a shorter compliant route it replaces the old route in the usual way.

**Policy conflicts.** A route must comply with both the outbound policy of the source and the inbound policy of the destination. Conflicts are not a concern, although restrictive policies might block traffic entirely. If one subnet owner publishes conflicting policies for different prefix pairs, then the longest prefix match takes priority. By convention, the source prefix dominates for an outbound policy and the destination prefix dominates for an inbound policy.

**Prototype source code.** The source code for ExoPlex and its SDN controller (in Python) is available at https://github.com/RENCI-NRIG/CICI-SAFE, which links to a separate repository for SAFE logical trust. The core modules of the ExoPlex NSP controller toolkit comprise about 6K lines of Java code and a few hundred lines of SAFE trust scripts.

## IV. Experiments

The ExoPlex prototype is suitable for experiments with secure routing and policy flexibility involving modest numbers of customer prefixes and NSPs. We conducted demonstration experiments on the ExoGENI testbed and I2-AL2S research fabric. The performance and scale of these experiments are bounded primarily by the current limitations of the fabric, the VM-based OpenVSwitch routers we use on ExoGENI, and the Ryu SDN controllers for each NSP. All compliance checks and crypto operations are off of the dataplane, and so affect only the setup times, and not the transit performance. There is an obvious tradeoff between policy granularity and scale: fine-grained policies lead to fragmentation of the prefix space and routing/SDN flow tables, which could be a scaling barrier.

We conducted several demonstration experiments based on the topology shown in Figure 1, involving ten ExoGENI slices. Each slice is instantiated under its own keypair and runs with its own controller, SDN, and SAFE logic engine, as in Figure 2. The customer networks A, B, C and D advertise subnet prefixes delegated to them through common governance authorities, which also endorse the networks with various security properties (tags), as described and shown in the figure. The NSP controllers interact via REST APIs to peer and propagate edge-to-edge routes and policies.

### A. Experiment 1: Inbound Path Control

We evaluated inbound path control for the scenario shown in Figure 1, but omitting NSPs N3 and N4. On ExoGENI it takes about 6.5 minutes to provision this topology and stitch the peering links, limited by provisioning times for I2-AL2S circuits. The customer subnets stitch to their SDX providers concurrently, and then advertise their routes and path control policies when the stitches complete. NSPs validate those advertisements and policies, configure their dataplanes via SDN accordingly, and propagate them to their peers.

For this experiment, customers A and C both authorize only the NSPs with secure attribute “tag0” to carry their inbound traffic, and authorize connectivity only with edge subnets bearing the same attribute “tag0”. Customers B and D similarly authorize only “tag1” for their network traffic. Upon receiving the route advertisement with linked inbound policy from A, S1 propagates the route to N1 only, based on A’s policy. N1 validates the route, adds it to its cache of known routes, and propagates it to its authorized neighbor. Other routes and policies are advertised and propagated throughout the network similarly. Then, customer pairs A-C and B-D each request a flow to the partner.

It takes about 3 seconds for each route advertisement to propagate throughout the network and enable flows between the pairs. After these requests complete, we ping between subnet pairs with 1 packet and dump the flow tables from NSP switches, shown in Figure 3, to verify that traffic follows the compliant paths as shown in Figure 1.

### B. Experiment 2: Outbound Path Control

For this experiment we extended the inter-domain network with NSPs N3 and N4, as shown in Figure 1, and added outbound policies. NSPs propagate route advertisements through the network to all NSPs that are compatible with both the inbound and outbound policies and have a route to the destination. It takes about 10 minutes to provision the topology and 30 seconds for routes to propagate and stabilize.

The customer subnets again constrain their inbound traffic to transit through NSPs with compatible tags. Customers A and C have additional inbound and outbound policies that only NSPs with secure tag “tag0” can carry traffic between subnet A (192.168.10.0/24) and subnet C (192.168.30.0/24). Similarly, customers B and D restrict the traffic between their subnets to pass through NSPs with secure tag “tag1”, and A and D restrict their flows to NSPs with “tag2”.

After the connections are set up, we ping between pairs of subnets and inspect the flow tables (not shown) as before to
verify that traffic takes the expected compliant paths.

C. SAFE Routing Authorization Performance

We conducted experiments to evaluate the cost of logical authorizations in isolation. We evaluated the inference performance of a SAFE server for validating routes and checking policy compliance under a throughput-limited synthetic workload. The figure of merit is authorization ops per second (authz-ops/sec) for the checks performed by an NSP when it receives an advertisement. We run the SAFE server on a machine with 16 2.6 GHz cores (Intel Xeon E5-2650 v2) and saturate it with concurrent authorization queries through its REST API. The evaluation measures the cost to process the network calls and the cost to run the logic query on logic content extracted from a linked certificate DAG.

We generated a synthetic topology of 1.8K NSP networks with a pattern of random peering among them: NSPs originate routes for their own IP prefixes and propagate routes to authorized neighbors randomly. We also generated synthetic governance principals to delegate NSP security tags and customer IP prefixes from a common root. As in the previous experiments, the customer path control policies query NSP attributes (tags) endorsed from the authorities. We chose a tag delegation depth of 3 and IP prefix delegation depth of 3.

Figure 4 shows route authorization throughput as a function of route length for three sets of policies. There is a fixed cost to verify IP prefix ownership and validate routes, and an additional cost to check compliance with inbound and outbound path control policies at each NSP in the path. Thus the most expensive policy is PBR-1 (Policy-Based Routing), which checks both inbound and outbound policies. PBR-2 checks inbound path control policy only. We compare the results to basic BGPsec-like route validation and prefix ownership alone without customer-specified path control (labeled as BGPsec).

The results show that even for the most costly workload an NSP controller can check more than 2K routes per second. These checks occur only when the NSP receives a new route or policy, and do not impact the dataplane. While Figure 4 does not include any crypto overhead (signature validation), these costs are fundamental for any routing security approach based on public-key cryptography (e.g., BGPsec). SAFE does impose additional costs to fetch linked certificates on demand, but the SAFE engine validates them once and caches their logic content until the TTL expires, which minimizes these costs for policies, governance endorsements, etc. (Figure 4 ran on a hot cache pre-warmed with all relevant certificates.) These results suggest that logical trust is fast enough to be practical at substantial scale.

V. Conclusion

We propose a logical trust approach to network security for testbed-hosted Network Service Providers (NSPs), implemented in the ExoPlex network controller platform, and extend it for secure policy-based routing for interdomain networks with multiple NSPs. ExoPlex can be the basis for a “testbed for trust” for inter-domain networks that are constructed on the fly and span multiple slices, testbeds, and campuses. NSP owners and customers may experiment with policy for peering, routing, path control, and governance by specifying custom policies in logic, without changing the ExoPlex code. In particular, the trust plane supports customer policies for permissioning the NSPs themselves, so that customer traffic does not pass through untrusted NSPs (path control). A secure foundation with at least these features is a necessary prerequisite for safe testbed opt-in by real customer traffic—an important aspirational goal.

REFERENCES