Toward Live Inter-Domain Network Services on the ExoGENI Testbed

Yuanjun Yao[†] Qiang Cao[†] Rubens Farias[†] Jeff Chase[†] Victor Orlikowski[†]
Paul Ruth[§] Mert Cevik[§] Cong Wang[§] Nick Buraglio[‡]

[†]Duke University [§]RENCI [‡]ESnet

***Abstract*—A key dimension of reproducibility in testbeds is stable performance that scales in regular and predictable ways in accordance with declarative specifications for virtual resources. We contend that reproducibility is crucial for elastic performance control in *live* experiments, in which testbed tenants (slices) provide services for real user traffic that varies over time. This paper gives an overview of *ExoPlex*, a framework for deploying network service providers (NSPs) as a basis for live inter-domain networking experiments on the ExoGENI testbed. As a motivating example, we show how to use ExoPlex to implement a *virtual software-defined exchange* (vSDX) as a tenant NSP. The vSDX implements *security-managed* interconnection of customer IP networks that peer with it via direct L2 links stitched dynamically into its slice. An elastic controller outside of the vSDX slice provisions network links and computing capacity for a scalable monitoring fabric within the tenant vSDX slice. The vSDX checks compliance of traffic flows with customer-specified interconnection policies, and blocks traffic from senders that trigger configured rules for intrusion detection in Bro security monitors. We present initial results showing the effect of resource provisioning on Bro performance within the vSDX.**

***Index Terms*—Networks, Testbeds, Intrusion Monitoring, SDN**

I. Introduction

Network testbeds have an important role as platforms to experiment with new approaches for networked systems. In the GENI [1] community, it is common to distinguish two aspects of this role: in the *wind tunnel* model researchers experiment with prototypes under controlled conditions and synthetic loads, while the *petri dish* model envisions the testbed as a platform for *live* network services that serve real users, and evolve under real-world conditions.

We focus on advancing support for GENI testbeds as a petri dish to culture new approaches to security-managed network services. Our approach is inspired in part by proposals put forward by leading researchers more than a decade ago for "pluralist" network architecture based on deep virtualization. Most notably, the authors of Plutarch [2] and Cabo [3], [4] build network service providers (NSPs) as a software layer over programmable infrastructure: pipes, programmable switching points, and in-network computation. They offer a compelling vision of NSPs that manage end-to-end interoperable connectivity, riding over an architecture-neutral underlay of infrastructure providers.

Reproducibility is crucial for testbeds that host live services, just as for fixed (wind tunnel) experiments. In particular, tenant NSPs must provide performance assurances to their customers and adapt their slice configurations to meet their targets as demands change. In this context, *reproducibility* means that any slice with a given declarative resource specification S delivers fixed performance under a given fixed workload W at any time. This property is essential for elastic performance control: it enables the tenant to predict the resources needed to stay on target as the workload changes, and adapt its slice accordingly. Stable and responsive elastic control based on high-fidelity virtualization is an active research topic (e.g., Pulsar [5]), which our testbeds should support. Success means demonstrating effectiveness under diverse conditions encountered in a live deployment.

Live tenant NSPs require performance assurances that are precise and strong at the foundation—the testbed infrastructure service. Recent post-GENI testbeds, including Chameleon [6], emphasize bare-metal control to achieve high fidelity. Our work uses ExoGENI [7] slices provisioned using open-source virtualization (e.g., KVM), similarly to commercial IaaS clouds. ExoGENI leverages advanced circuit fabrics (I2-AL2S, ESnet) for the cross-site network backplane. Thus, slices may instantiate end-to-end network topologies and evolve them by allocating and releasing VMs and dynamic circuits, with precise resource contracts from the infrastructure providers.

With these ingredients in place, our objective is to enable testbed-hosted NSPs that benefit real users—for example, to implement secure built-to-order virtual science networks that span campuses. Going further, we strive to support *inter-domain* networking experiments: traffic routing among NSPs that are controlled by different principals and that peer with one another as in today's Internet.

Our initial efforts focus on three key elements. First, enable controllable packet flow into and among GENI slices. ExoGENI *stitchports* allow slices to establish peering links with other slices at L2 by mutual consent [8], [9]. Moreover, Duke and other campuses have deployed SDN-enhanced edge networks that support *opt-in* redirection of real user traffic into the circuit fabrics and into locally hosted GENI edge points of presence (PoPs). Second, introduce an elastic slice controller architecture for slices to adapt their configurations over time—we might call them "software-defined slices". We recently reported on initial experiments with Ahab controllers and elastic NSP slices in ExoGENI [9]. Third, introduce tools for participants to authorize their peering connections, traffic

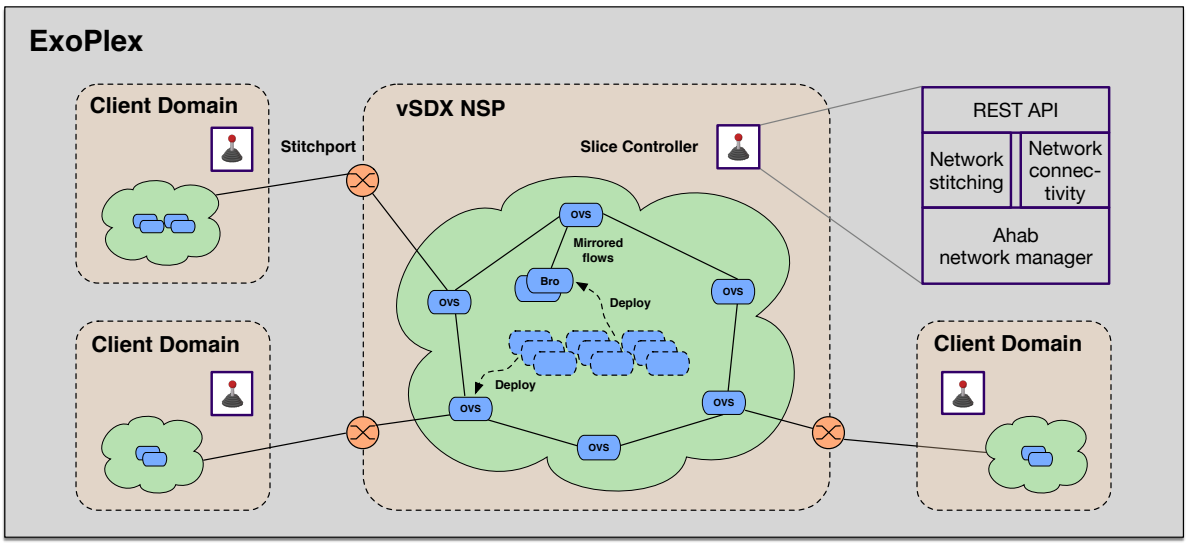


Fig. 1: The ExoPlex network service architecture. A vSDX NSP supported by ExoPlex uses an elastic slice controller to coordinate dynamic circuits and Bro security monitors via Ahab. The controller exposes REST APIs for clients to request network stitching and connectivity and uses a SAFE engine to check each request for compliance with logical trust policies.

flows, and related interactions. To this end we introduce a declarative assertion and policy language—a trust logic—and certificate transport for secure logical trust [10], [11].

This paper outlines these elements and presents experiments with an example tenant NSP —a security-enhanced network provider in a slice. We refer to the experimental NSP and the ExoGENI-hosted framework that supports it as *ExoPlex* (§II). An ExoPlex NSP offers edge-to-edge connectivity among attached customer slices (or campus subnets) with bandwidth assurances and declarative security policies. Our example NSP is analogous to a software-defined exchange (SDX [12]): customers attach their L2 networks to it at edge PoPs, stating policies to govern their traffic flow and their connectivity with other customers. The NSP then manages traffic flow in accordance with the union of the customer policies. We refer to the example NSP as a *virtual* SDX (vSDX) because its topology is elastic and it occupies many edge PoPs rather than a fixed exchange point (§III). The prototype vSDX also deploys Bro [13] appliances for integrated security monitoring. We investigate the performance of the Bro VMs as they filter suspect traffic passing through a vSDX NSP slice (§IV).

II. Overview of ExoPlex

Figure 1 illustrates the ExoPlex structure for hosting end-to-end tenant network providers (NSPs) with in-network services, such as elastic security scanning. ExoGENI is a good foundation for these NSPs because it provides on-demand computation and programmable (virtual) switching/NFV capacity at multiple sites/PoPs on advanced network circuit fabrics linking many campuses and research centers. The customers of a tenant NSP may be subnets in an SDN-enabled host campus network, external networks or testbeds connected through network circuits to edge interfaces (stitchports), or other ExoGENI slices connected via direct L2 peering of the slice dataplanes [8]. For example, the prototype vSDX (§III) provides network transport service to the Chameleon testbed.

An ExoPlex NSP is a testbed slice containing a dynamic virtual network topology. Each customer of an NSP attaches to it at L2 to form a peering link (a *stitch*) between at least one pair of interfaces on their network edges, via a circuit attached to a stitchport, or (if colocated at a site) by direct peering over an ExoGENI site interconnect. The customers route traffic onto their peering links, and the NSP transports the traffic onto an egress link to the destination customer—or drops it—according to its configured policies.

Any ExoGENI slice may have an optional external slice controller built with the Ahab toolkit [9]. Ahab controllers invoke ExoGENI IaaS interfaces programmatically to adapt the slice configuration and topology, and also make calls into VMs or service elements instantiated in the slice. An NSP slice controller may expose service-specific northbound REST APIs to the clients or customers of the NSP. Calls to these northbound APIs request network services, and may trigger changes to the NSP slice as needed to provide that service, according to an elastic control policy for the NSP.

ExoPlex is a tool to pilot traffic engineering, topology adaptation, and in-network security functions within tenant NSPs. Our initial vSDX experiment based on ExoPlex combines elements of three focus areas summarized below.

Authorizing interconnection and inter-domain traffic flow. We use datalog trust logic (SAFE) to authorize vSDX customer attachment and to validate both IP prefix ownership and advertisements at both sides of each stitch, thereby providing security assurances equivalent to RPKI and BGPSEC. In this way an NSP provides a secure and private interconnection service for authorized customers.

Network management languages. Recent network management languages (such as PGA [14], PANE [15], or Mer-

Policy	Category	Description	#Certificate types	Chain length
approveByUserACL	Stitch	Accept if the customer is on a vSDX access control list (ACL).	2	5
approveByUserAttr	Stitch	Accept if the customer is a member of an eligible GENI project.	4	10
approveBySliceAttr	Stitch	Accept if the customer slice is endorsed as eligible by a trusted party.	5	12
approveByProjectID	Stitch	Accept if the customer slice belongs to an eligible GENI project (by ACL).	3	6
approveByProjectAttr	Stitch	Accept if the customer slice is endorsed as eligible by a trusted party.	6	13
approveByUserACL	Transit	Accept if receiver agrees to accept traffic from peer requester (ACL).	2	4
approveByUserAttr	Transit	Accept if requester has a security attribute accepted by receiver.	3	6
approveByProjectID	Transit	Accept if requesting slice belongs to a project accepted by receiver.	5	11

TABLE I: **Summary of policies for network authorization in ExoPlex and their complexity: the number of types of logical statements (certificates) and the chain length of a typical logical proof of compliance for each policy.**

lin [16]) provides declarative policy tools to manage connectivity among network endpoints. While these systems are designed for use within an enterprise, an NSP might use similar tools to configure connectivity among customer domains across a wide area. All of these languages are based on an "off by default" whitelisting model for interconnection based on user-defined attributes (labels) and predicates for network endpoints. Our example vSDX allows customers to specify their interconnection policies to the vSDX declaratively using trust logic. Trust logic enables us to organize principal identities within an authority structure that governs which principals have authority to assign labels and enable flows. This approach is applicable to federated systems with no central point of policy control and multiple sources of trust labels, which may include security endorsements.

Deploying a Bro security monitor fabric. Network management languages also define syntax to impose path constraints on permitted flows, e.g., to mandate that flows transit through a service chain of one or more NFV functions. We focus on deployment of network security monitoring as an exemplary NFV. The vSDX performs intrusion detection/prevention based on Bro [13] appliances running in elastically deployed virtual machines (§III-B). Given strong reproducibility properties, the platform can serve as a testbed for scalable on-demand Bro deployments for the community.

III. Virtual SDX: A Prototype NSP

The prototype tenant NSP in this paper is a virtual software-defined exchange (a vSDX) with integrated security monitoring. The SDX concept was conceived as a physical facility (PoP) with direct attachment to customer networks. IaaS providers with broad reach, such as the ExoGENI federation, can host "virtual" SDX services that provide similar functions decoupled from fixed peering points. A vSDX can extend to many geographically distributed PoPs via a dynamic bandwidth-provisioned network backplane topology.

The vSDX Network Service Provider is an elastic slice managed by an Ahab controller. The controller runs outside of the vSDX slice and exposes a northbound API for operations on the slice by peer domains that are authorized to peer with the vSDX as customers. Customers invoke these northbound APIs to bind named subnets under their control to the vSDX via L2 stitching, request bandwidth-provisioned connectivity with other subnets (including subnets owned by other customers), and specify logical policies that govern approval of requests by other customers to connect to them.

The vSDX slice comprises virtual compute nodes running OpenVSwitch, OpenFlow controllers, and Bro traffic monitors. Traffic flow and routing within the vSDX slice are governed by a variant of the Ryu *rest_router* SDN controller similar to the *Plexus* SDN controller used within Duke's campus SDN network. The vSDX slice controller computes routes internally for traffic transiting the vSDX network, and invokes the northbound SDN controller API to install them. The SDN controller runs another Ryu module (*rest_ofctl*) to block traffic from offending senders. If a Bro node detects that traffic violates a Bro policy, it blocks the sender's traffic by invoking a *rest_ofctl* API via the Bro NetControl plugin.

As client requests for bandwidth-provisioned connectivity arrive at the vSDX, the slice controller instantiates slice resources as needed to carry the expected traffic. These resources include peering stitchport interfaces at each PoP, the OVS nodes that host these vSDX edge interfaces, Bro nodes to monitor the traffic, and backplane links to carry the traffic among the PoPs. The controller reuses existing resources in the slice if they have sufficient idle capacity to carry the newly provisioned traffic, and instantiates new resources as needed. In particular, it adapts the vSDX backplane topology by allocating and releasing dynamic network circuits as needed to meet its bandwidth assurances to its customers.

A. Authorization

A key difference from a classical SDX is that domains specify policy using trust logic rather than through OpenFlow directly. The vSDX customers specify logical policy rules that govern which other customers may interconnect with them; these policies may consider the security properties of peer networks and/or the accountable identities that control them. Customers trust the vSDX to evaluate compliance of any connecting peers with their security policies on their behalf. Customers also present logical certificates that represent their own identities, subnet ownership, and security properties when they attach to the vSDX.

Logical trust provides a simple and powerful basis for authorization of requests to the vSDX service. Clients authenticate their REST calls to the vSDX API using keypairs. Clients pass links to certificate chains that express their connectivity policies and various attributes and permissions granted to them by other parties. These include certificates assigning

identity attributes to their public keys, certificates from GENI authorities endorsing the slice and naming a GENI project that it belongs to, and attributes of the project and slice from these authorities or other trusted parties. (For the GENI attributes we use SAFE logical certificates from synthetic authorities rather than standard GENI formats, but their content matches the GENI trust model [17].) The vSDX slice controller runs an instance of the SAFE logical inference engine to validate each request against configured security policies before approval.

Stitching. A customer slice stitches to a vSDX provider's network at one or more named L2 peering points. The vSDX applies its own policies to validate each request before installing each peering link.

Connectivity. Connectivity among customers is off by default: flows are enabled among subnets only by customer request, and only when compliant with the stated connectivity policies of both affected customers.

Prefix ownership. The vSDX also uses trust logic to validate ownership of prefixes advertised by its customers. Following the model of RPKI, prefixes are delegated transitively through a hierarchy of owners, rooted in a trusted authority (e.g., ICANN) with range containment checked at each level. Following the model of BGPsec, customers must own the prefixes they advertise, or must link to certificates delegating the right to advertise those prefixes transitively from their owners. These delegations are represented in SAFE logic (rather than the RPKI or BGPsec standards), and are checked with logical SAFE validation rules.

The stitching and connectivity policies may include arbitrary attribute checks on the requesting client, slice, and/or project, and on the set of authorities trusted to assert these attributes. Table I lists some exemplary policies for vSDX stitching and customer connectivity.

If transit is approved based on connectivity policies, the vSDX slice controller finds a path for the connection and installs the routes via SDN. As a result, the established connectivity among customers is end-to-end and bidirectional, and limited to the authorized subnets.

B. Bro Intrusion Detection

In our vSDX, permitted flows are inspected by out-of-band Bro network security monitor appliances to detect intrusion. As a simple form of intrusion prevention, it uses Bro's *NetControl* framework to interrupt all traffic from the source of a suspect flow. The vSDX controller deploys Bro instances elastically to scale capacity as customers join. Bohatei [18] provides an elastic DDoS defense by deploying filtering appliances in a similar way.

Bro is a powerful, open-source out-of-band network monitoring and analysis framework supporting a wide range of traffic analysis tasks, including network activity logging, sanity checks for various protocols, and intrusion activity matching. It has been widely used by network infrastructures and service providers.

Typically, a Bro instance is deployed alongside an edge router, where intrusion detection is needed. A router or switch mirrors network traffic to the out-of-band Bro instance, which scans the network traffic without affecting performance. Scanning applies a script of match-action rules to identify and flag suspicious activities. If a sampled traffic pattern matches a rule, Bro triggers its event engine to take a corresponding action. These actions are selected from a library of connectors (event handlers) in Bro's *NetControl* framework. Typically, these actions add the sender's IP address (or subnet) to a blacklist and/or use SDN to cut flows or sandbox endpoints.

Bro may also be deployed in a closed loop to install rules that block (blackhole) traffic from a suspected attacker on ingress at the network edge. Bro tooling may also cross-reference with NetFlow data and install rules in access-filtering edge devices to block attack traffic.

In this paper we explore the performance of closed-loop traffic control based on Bro when operating within a vSDX slice on the ExoPlex platform. Our experiments use synthetic traffic, but we source and sink the traffic from a dynamic set of customer slices that stitch to the vSDX at L2, modeling a live deployment. Due to limitations of current testbeds, we are limited to virtual host-based network appliances and SDN-based access control within the vSDX slice. These functions run within OpenVSwitch (OVS) instances. The Bro and OVS instances run on elastically deployed ExoGENI VMs. An Ahab slice controller is responsible for elastic provisioning of the monitoring and filtering capacity within the vSDX slice.

IV. Experiment

In this section, we evaluate the effectiveness of the vSDX Bro nodes under varying traffic. This evaluation explores both the effectiveness and limitations of deploying Bro in the ExoGENI testbed, and stability (reproducibility) of the results across multiple deployments of the same declarative slice specification.

A. Design

Our vSDX NSP experiment consists of an ExoPlex slice deployed across two ExoGENI sites, which interconnects four client domains (see Figure 2). Two of the client domains are located on the Chameleon testbed, and connect to the vSDX using dynamic circuits and stitchports. The other two client domains are independent slices on ExoGENI, each connecting to the vSDX using slice-to-slice stitching at each PoP—the vSDX establishes a PoP on each ExoGENI site where it has an authorized customer. For simplicity, all client domains specify SAFE policies that allow incoming traffic from all other client domains, but require that the secure ingress service terminates flows identified as potentially malicious according to a pre-established set of Bro match-action rules.

Each link within the ExoPlex vSDX is allocated 2 Gbps of bandwidth, and the link between ExoPlex and each client domain is 1 Gbps. All ExoGENI nodes (clients and ExoPlex services) are VMs having 4 cores and 12 GB RAM (the "XO Extra Large" instance type). Two types of nodes comprise the vSDX: OVS nodes (running version 2.0.2) and Bro nodes (running version 2.5.2). On the Bro nodes, we load all policy

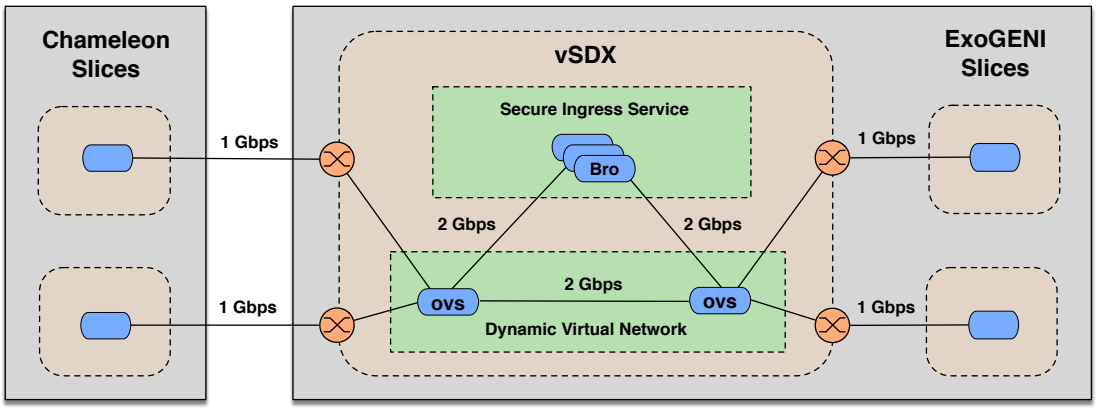


Fig. 2: **An experimental SDX network that exchanges traffic between customer nodes and mirrors traffic to Bro nodes for intrusion detection.**

scripts included in the distribution, and add one of our own that detects transfers of "malicious" files with specific signatures. Should a "malicious" file be detected, Bro instructs the SDN controller to drop traffic from the source using the *NetControl* framework as described above.

We measured the performance of Bro nodes deployed on several different sites within the vSDX. In each run a Bro node filters a traffic flow between a pair of clients interconnected via the vSDX. The flows are synthetic and have similar traffic profiles, including attack traffic. For each experiment we recorded selected metrics defined as follows:

- **Response Time**: We define response time as the period between detection of a "malicious" file transmission and the termination of the associated connection by the SDN controller. Thus, we are able to quantify the delay in protecting clients from an attack.
- **CPU Utilization**: As processing demand increases with the traffic flow at a given instance, each instance's monitoring ability is eventually saturated.[1]
- **Packet Drop Ratio**: With high load of CPU utilization and mirrored traffic, Bro begins dropping packets. We define the packet drop ratio as the percentage of dropped packets to the total that should have been mirrored.
- **Detection Rate**: As packets are dropped, the likelihood of Bro failing to identify an attack increases. We define detection rate as the percentage of malicious files detected by Bro, relative to the total that are present.

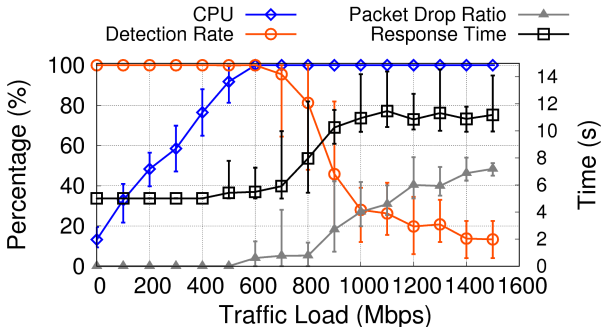
During each run of our experiments, pairs of clients (one each from Chameleon and ExoGENI) send measured amounts of UDP traffic through the vSDX using `iperf3` [19]. All traffic traversing the vSDX is mirrored to a Bro node. While this sample traffic is flowing, FTP is used to transfer 200 "malicious" files that should be detected by Bro. Each run reports the four selected metrics.

A Bro instance can only process a bounded traffic load before dropping packets—about 600 Mbps under our settings (§IV-B). To handle higher traffic loads, the vSDX NSP dynamically launches multiple Bro instances and balances flow processing across them based on measured flow rates and customer requirements. When the used capacity of a Bro pool exceeds a certain threshold (i.e., 60%)—or no Bro instance at the specific edge PoP has sufficient remaining capacity to process new flows— the controller launches a new Bro instance. To demonstrate the effectiveness, we have two pairs of connections, with each of them sending traffic at the same rate (300 Mbps). The initiation of one of these two flows is offset by 10 seconds, during a given run of this experiment. We evaluated Bro filtering performance against our metrics, both when the traffic was mirrored to a single Bro instance and when the mirrored traffic was diverted to different Bro instances on a per-client-pair basis. We do not have support for our OVS virtual routers to hash flows from the same (sender, destination) pair across multiple Bro nodes. For this purpose some switch vendors (e.g., Arista) have introduced high-speed switches with configurable *tap aggregation* to mirror traffic and distribute monitored flows across a cluster.

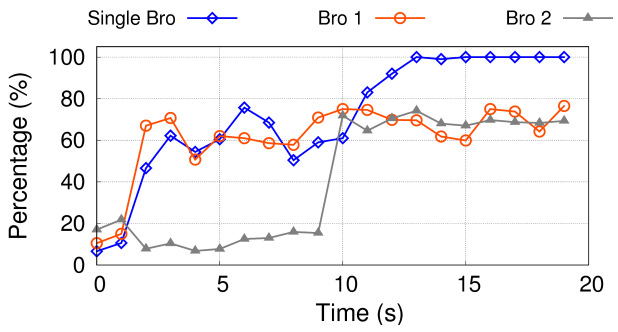
B. Results

Figure 3a shows the results measured from multiple runs on different host ExoGENI PoP sites. At ∼600 Mbps of mirrored traffic, a single core begins to saturate: Bro begins dropping packets and the detection rate decreases. When backgroud traffic is less than 600 Mbps, the response time is constant (∼5 seconds), due to Bro's event scheduling mechanism. Beyond 600 Mbps of mirrored traffic, Bro's response time increases rapidly, but the detection rate does not fall until it begins dropping packets. As we expected, even a minor increase in packet packet drop ratio (∼18% at 900 Mbps) can result in a significant decrease of the detection rate (∼55%), since a few consecutive dropped packets disrupt matching for file detection. For predictable and reliable performance, a single Bro instance's processing capacity is limited to ∼500 Mbps (given the Bro scripts, traffic features, and VM types used in our experiment). The error bars indicate the performance variability we experienced on the ExoGENI VMs hosting Bro at different times and across different sites. We believe that performance is sufficiently stable to support elastic performance control at some cost in efficient utilization to allow

[1]Since each Bro instance is single-threaded it can saturate at most one core, so we clip the CPU utilization to 100% in the graphs. It may reach a peak of 110% under a high flow volume due to other activity on the VM.



(a) **Performance of a single Bro instance on the VM, measured from two ExoGENI sites with 5 runs on each.**



(b) **CPU Utilization when mirroring two 300 Mbps flows to a single Bro instances versus two separate Bro instances.**

Fig. 3: **Bro performance in ExoPlex experiments.**

headroom for variable performance.

Figure 3b reveals the effectiveness of scaling across multiple Bro instances for reducing the CPU utilization of individual instances. When we mirror both 300 Mbps flows to a single Bro instance, CPU saturation occurs after the second flow is initiated, 10 seconds into the run. When the individual flows are mirrored to separate Bro instances, both instances exhibit stable and predictable performance that can be attributed to the decreased CPU utilization by each individual instance.

C. Repeatability

The experiments described in this paper can be repeated using the ExoGENI and Chameleon testbeds along with source code available on github. Detailed instructions are with the code in the *cnert-2018* branch of the SAFE project located here: https://github.com/RENCI-NRIG/SAFE.git.

V. Conclusion

These initial experiments are steps toward effective elastic configuration policies for scalable monitoring in the virtual SDX service. These policies require a performance model for Bro that predicts performance and effectiveness as a function of load and capacity. We can infer such a model and apply it for elastic provisioning in the vSDX slice controller only to the extent that the testbed exhibits reproducible performance.

VI. Acknowledgments

We are grateful to the Chameleon team for providing resources from the Chameleon sites as part of the experimenting infrastructure for ExoPlex. This work was supported partially by NSF awards OAC-1642140 and OAC-1642142.

References

[1] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *The GENI Book*, 2016.

[2] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: An argument for network pluralism," *SIGCOMM Computer Communication Review*, vol. 33, pp. 258–266, October 2003.

[3] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *SIGCOMM Computer Communication Review*, vol. 37, pp. 61–64, January 2007.

[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: Realistic and controlled network experimentation," in *SIGCOMM*, 2006, pp. 3–14.

[5] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska, "End-to-end performance isolation through virtual datacenters," in *USENIX OSDI*, 2014, pp. 233–248.

[6] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the Chameleon Cloud testbed, and Software Defined Networking (SDN)," in *International Conference on Cloud Computing Research and Innovation (ICCCRI)*, Oct 2015, pp. 73–79.

[7] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "ExoGENI: A multi-domain infrastructure-as-a-service testbed," in *The GENI Book*, 2016, pp. 279–315.

[8] Y. Xin, I. Baldin, A. Mandal, P. Ruth, and J. Chase, "Towards an experimental legoland: Slice modification and recovery in ExoGENI testbed," in *Testbeds and Research Infrastructures for the Development of Networks and Communities*, 2016, pp. 35–45.

[9] Y. Yao, Q. Cao, J. S. Chase, P. Ruth, I. Baldin, Y. Xin, and A. Mandal, "Slice-based network transit service: Inter-domain L2 networking on ExoGENI," in *IEEE INFOCOM Workshop on Distributed Cloud Computing (DCC)*, 2017.

[10] Q. Cao, V. Thummala, J. S. Chase, Y. Yao, and B. Xie, "Certificate Linking and Caching for Logical Trust," http://arxiv.org/abs/1701.06562, 2016, Duke University Technical Report.

[11] Q. Cao, Y. Yao, and J. S. Chase, "A logical approach to cloud federation," http://arxiv.org/abs/1708.03389, 2017, Duke University Technical Report.

[12] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined Internet exchange," in *ACM Conference on SIGCOMM*, 2014, pp. 551–562.

[13] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999.

[14] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using graphs to express and automatically reconcile network policies," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[15] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An api for application control of sdns," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 327–338.

[16] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with merlin," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, 2013, p. 24.

[17] M. Brinn, N. Bastin, A. Bavier, M. Berman, J. Chase, and R. Ricci, "Trust as the foundation of resource exchange in GENI," in *Proceedings of the 10th EAI International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, June 2015.

[18] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense," in *Proceedings of the 24th USENIX Conference on Security Symposium*, 2015.

[19] "Iperf - the network bandwidth measurement tool," https://iperf.fr/.