Leveraging SDN to Enable Short-Term On-Demand Security Exceptions

James Griffioen, Zongming Fei, Sergio Rivera, Jacob Chappell, Mami Hayashida,
Pinyi Shi, Charles Carpenter, Yongwook Song, Bhushan Chitre, Hussamuddin Nasir, Kenneth L. Calvert
Laboratory for Advanced Networking, University of Kentucky
Lexington, Kentucky 40506–0495
Emails: {griff,fei,sergio,jacob,mhaya2,pinyishi,charles,ywsong2,bhushan,nasir,calvert}@netlab.uky.edu

***Abstract*—Network security devices intercept, analyze and act on the traffic moving through the network to enforce security policies. They can have adverse impact on the performance, functionality, and privacy provided by the network. To address this issue, we propose a new approach to network security based on the concept of short-term *on-demand security exceptions*. The basic idea is to bring network providers and (trusted) users together by (1) implementing coarse-grained security policies in the traditional way using conventional in-band security approaches, and (2) handling special cases policy exceptions in the control plane using user/application-supplied information. By divulging their intent to network providers, trusted users can receive better service. By allowing security exceptions, network providers can focus inspections on general (untrusted) traffic. We describe the design of an on-demand security exception mechanism and demonstrate its utility using a prototype implementation that enables high-speed big-data transfer across campus networks. Our experiments show that the security exception mechanism can improve the throughput of flows by trusted users significantly.**

***Keywords*-software defined networking, security appliance, middleboxes, trusted flows**

I. Introduction

Network security devices that enforce policy have become highly sophisticated and are now pervasive across campus, corporate, and provider networks. Examples of these special purpose middleboxes include firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), network address translators (NAT), traffic shapers, rate limiters, and load balancers. These devices intercept and inspect all traffic moving through the network in order to apply and enforce policies—including security policy.

All such middleboxes add varying amounts of delay. Moreover, some middleboxes—especially those that perform deep packet inspection (DPI)—can limit throughput and become choke points when performance is a primary consideration. Middlebox functionality based on Software-Defined Networking (SDN) can be even worse, diverting packets to software controllers. For example, high performance computing (HPC) systems often encounter performance-limiting bottlenecks resulting from policy-enforcing middleboxes in the (campus) network.

In addition, network security devices often require policies to be expressed in terms of network-level abstractions such as IP addresses and ports. Some DPI-based services inspect every packet for well-known byte patterns that signal malicious content. In contrast, security policies should really be about higher-level abstractions such as users (or roles), applications, intent, or time of day. It is well-known that such higher-level policies are difficult to map perfectly onto network-level abstractions, so that a middlebox may not be able to precisely implement a desired high-level policy. The result is collateral damage, where the middlebox ends up either over-protecting (blocking traffic that does not need to be blocked, thereby limiting functionality) or under-protecting (passing traffic that should have been blocked, thereby increasing risk). The classic under-protection example is a "Science DMZ" [1] placed at the edge (outside) of the campus network that undergoes minimal policy enforcement/protection so that HPC resources and applications can avoid performance-limiting (but otherwise helpful) campus middleboxes.

Security is always a tradeoff among various costs and benefits, including performance, operational and capital expenses, and user (in)convenience. In this paper, we argue that the complex and dynamic policy needs of IT environments make it worthwhile to re-examine the tradeoffs involved in the coarse-grained, middlebox-based approach to network security—especially for campus networks. We are therefore exploring a new approach to network security by refining the middlebox approach using SDN.

The basic idea of our approach is to use traditional security appliances (that have been optimized and hardened) to provide a base level of coarse-grained policy enforcement on untrusted traffic, noting that the performance costs increase with increasing policy complexity as well as traffic volume. To address the security performance cost issue, we support the ability to establish trust relationships between users and network providers. These trust relationships can then be used to create fine-grained, short-term, trusted, *on-demand exceptions* to the base policies – implemented using SDN – that increase the performance of trusted traffic, while keeping the cost of the base-level down. In addition, some enforcement decisions (i.e., whether to grant an exception) can be moved from the data plane to the control plane, where they can be based on authenticated information provided by users indicating conformance (or not) to higher-level policy. In other words, if trusted users are willing to divulge their intent to network providers, their traffic can bypass the scrutiny—and associated

978-3-903176-15-7 © 2019 IFIP

costs—applied to the traffic of other (less-trusted) users. More precisely, negotiated security exceptions can allow users to bypass certain middleboxes, allow otherwise prohibited traffic to temporarily traverse the network, or offer some level of QoS to authorized flows. Exceptions might be made to stop normally-allowed traffic as well, for example to block or rate-limit unwanted traffic that would otherwise be delivered to a user.

The remainder of the paper is organized as follows. Section II presents a new model based on security exceptions. Section III describes an initial prototype implementing security exceptions, with the goal of improving the transmission of big data. Section IV provides performance improvement results when on-demand security exceptions are instantiated. Section V discusses related work and Section VI concludes the paper.

II. On-Demand Security Exceptions

Our proposed security approach splits the enforcement of high-level policy into two pieces. First, network administrators define general *base* network security policies to address common security issues and concerns, and deploy them to middleboxes much as they do today. These policies can be simple and even imprecise (e.g., erring on the restrictive side) or slow (e.g., employing extensive DPI) since authorized special-case traffic (e.g., traffic from trusted users requiring high performance) will be handled by exceptions and avoid these costly general checks. These base policies can be deployed over long timescales as they are not intended to change frequently.

The second part consists of short-term *on-demand exceptions* to the base policies. Providers (e.g., network operators, ISPs) grant policy exceptions to users—or their applications—that supply (trustworthy) information about their network traffic and intent. Unlike general policies, exceptions are intended to be narrowly limited in scope and defined and instantiated on short timescales, thereby allowing the network to quickly adapt policy to meet the current needs of applications, or to address the current security needs of the network. These dynamically created, flow-specific, limited lifetime exceptions can be implemented through recent advances in SDN [2] (although one could envision a scaled-back version of exceptions implemented on conventional switches/routers/middleboxes—e.g. SNMP or NETCONF/Yang).

A. Example Exceptions

As one example, consider an application that needs to move a large data set between a national supercomputer facility and the local campus HPC supercomputer. In this case, the user might present the network with information about the transmissions (e.g., type of data being transferred, the source and destination of the flows, or possibly things like the NSF project number associated with the flows). Based on this information, the provider might decide to allow a security exception in which such application flows are (temporarily) routed around the network's IDS/IPS system to avoid its throughput-limiting DPI, thus enabling the flow to operate at much higher transfer rates.

As another example, consider a policy exception that allows a highly interactive distributed application (e.g., a web conferencing application or an interactive game) to utilize low-latency network paths (as opposed to the default paths) among all participants in order to reduce delay, thereby improving the responsiveness of the application. Or consider a policy exception that is dynamically created to allow an authenticated collaborating researcher to use *ssh* from a specific end system in the Internet, to punch through a campus firewall and access a private *git* server containing shared data, without the need to set up a VPN.

All three examples above illustrate cases where general network security policy imposes costs by limiting bandwidth, causing sluggish behavior (high latency), or blocking legitimate users from accessing resources that should be shared. Exceptions allow users to advertise beforehand the details (e.g. type, requirements) of their traffic in order to justify their need for special treatment to the network provider. Given this information, providers no longer need to subject these flows to the general policy enforcement mechanisms.

B. Design Considerations

The notion of creating exceptions to security policy raises questions about how such exceptions are specified and how decisions are made. As noted in the examples above, exceptions must be vetted by the real-world authorities responsible for the network. In a sense, our approach moves some policy enforcement decisions from the data plane, where every packet is inspected by middleboxes, to the control plane. Whereas middlebox-based enforcement decisions must occur at (or near) line rate, control-plane decisions take place on the (slower) flow-initiation timescale.

One benefit is that the decision to grant an exception can be, and should be, tied to the abstractions of high-level flow instantiation policies (user IDs, role identifiers, classes of applications, etc.) rather than the addresses, ports, and byte patterns of low-level packet policies. Another is that the decision to grant an exception can be based on trustworthy information, because flow-initiation timescales make the use of cryptographic authentication of higher-level abstractions feasible. Moreover, because trust is with users and their applications, the number of trusted entities and their flows is relatively small and remains manageable (i.e., not tens/hundreds of thousands) – the term *exception*, after all, connotes something unusual occurring somewhat infrequently – preventing the exception mechanism itself from becoming a bottleneck.

Finally, the SDN infrastructure, as a means to bypass basic security policy enforcement, is part of the attack surface. Consequently, we take steps to protect and harden it against compromise. However, it should be noted that this problem exists for any SDN deployment, regardless of the way it is used, and we argue that adoption of SDN will require network elements that are at least as resistant to attack as existing middleboxes.

C. An Exception System

To support our security model, we divide the exception system into two parts: (1) a human interface for defining trust policies, and (2) an automated request/response system that applications can contact to request exceptions. The first involves formulating high-level policies that define trusted flows. These decisions are made on human timescales and often involve human validation of the policy. The second involves automatically changing the network state (using SDN) to implement the exception.

The goal of the first part of the exception system is to associate users/roles with specific flows—essentially to provide a "responsible party" for each flow granted an exception. To allow delegation of responsibility, the mechanism utilizes an *authorization tree* that arranges the set of all possible network flows into a hierarchy where each (child) node in the hierarchy represents a subset of the flows in the parent node. One or more users are then associated with each node in the tree, giving them authority to define allowable exceptions for that portion of the flow space. This allows users to delegate responsibility for certain flows to other users, creating hierarchical authorization schemes consistent with the organization of administrative responsibility in the Internet.

Each node in the tree identifies a portion of the possible flow space and has an associated *exception specification (ESpec)* that determines whether to grant or deny an exception. An ESpec is essentially a small piece of code that is executed by the automated exception service over the information provided in the exception request (e.g., who, what, when, where) as well as information about the current status of the network (e.g., load, other exceptions currently installed, available resources, etc). While one could envision these pieces of code being written in a general purpose language as "plugins" to nodes in the authorization tree, such a design would make it more difficult for humans to verify that a plugin is enforcing the policy correctly. As a result, we expect that ESpecs would be specified in a high-level policy definition language (or specialized markup) that could be easily compared against the intended policy exceptions without exposing the complexity of low-level implementation details to deploy an exception. To provide an example of what an ESpec policy definition language might look like, consider the following language used to define allowed exception requests (which we will use in later examples):

Request **Type**: `Add | Remove | Update`
Auth Credentials: `User ID | App ID`
Match → **Action**:
`<FlowSpec> → Max Bandwidth Path |`
`<FlowSpec> → Min Latency Path |`
`<FlowSpec> → Min Hop Count Path |`
`<FlowSpec> → Block`
Network **Condition**: (for example) `Path Load < 10% |`
`Current Time is in [22:00, 05:00]`
Exception **Lifetime**: `Flow Duration`

The automated part of our exception system accepts exception requests from trusted users or their application – specified in the above language to create (`Add`) on-demand exceptions (or `Update/Remove` existing exceptions). The information in the request is evaluated to determine if an exception should be granted (e.g., checking validity of credentials, ESpec format, valid request type, matches flowspec, etc). If the ESpec is valid, the system creates the exception by invoking SDN network management actions (e.g., computing OpenFlow rules, resolving domain names, etc) to deploy the exception to the appropriate network elements.

III. A Prototype System

As an example of applying the on-demand security exception model, we implemented a prototype system called *VIP Lanes* [3]. The objective of the VIP Lanes system is to support on-demand security exceptions that enable high-speed big-data flows to use paths that bypass campus network performance-limiting middlebox policy enforcement.

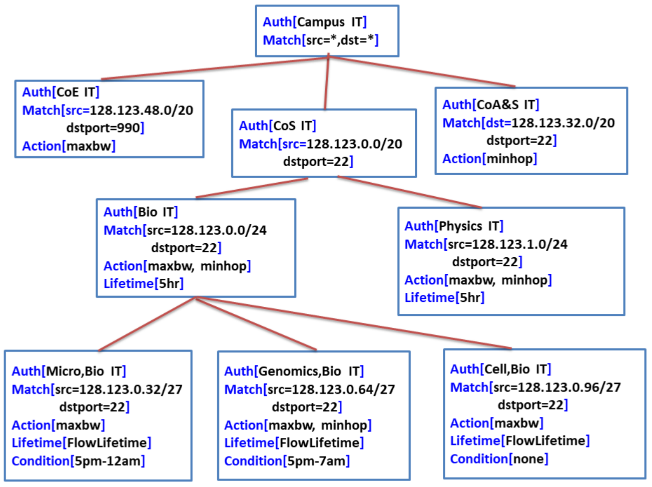

Fig. 1. An example Authorization Tree, delegating responsibility for the VIP Lane flowspace to the responsible parties.

A. A VIP Lanes Authorization Tree

The first step is creating the authorization tree described in Section II-C. Figure 1 shows an example VIP lanes authorization tree that specifies the trust relationships among network operators (providers).

Recall that the objective is to divide up the flowspace in a hierarchical manner, delegating the task of defining allowable flow exceptions to the (human) users responsible for those flows. In the context of a campus network, the root of the authorization tree would be defined by the Campus IT staff (Auth[Campus IT]) and would encompass all flows on campus (i.e., match[src=*, dst=*]). Campus IT might delegate the definition of exceptions for secure copy (scp) flows originating from the College of Science (match[src=128.123.0.0/20,dstport=22]) to the IT staff in the College of Science (Auth[CoS IT]). CoS IT staff members might further delegate the definitions of exceptions for scp flows originating in the Biology Department (match[src=128.123.0.0/24,dstport=22]) to the IT staff in Biology (Auth[Bio IT]). Bio IT staff might further delegate the definition of scp exceptions for traffic originating in the Genomics Lab (match[src=128.123.0.64/27,dstport=22]) to the Genomics project members (Auth[Genomics, Bio IT]) while retaining control of those definitions by Bio IT as well.

The tree presented in Figure 1 also defines the ESpecs used to grant exceptions on demand. For example, the ESpec for flows controlled by the Genomics Lab allows users in the Genomics group to instantiate exceptions that request (1) a (bypass) path that offers maximum bandwidth (max bw) or minimum hop count (min hops), (2) an exception lifetime matching the flow's lifetime, and (3) the stipulation (condition) that it be between 5pm and 7am.

B. Instantiating VIP Lane Policy Exceptions

Given a VIP Lanes authorization tree, users (or their applications) can request that specific exceptions be granted and instantiated via SDN in the network. Exceptions can be requested in one of two ways. In the first method users provide their institutional account credentials to login to the VIP Lanes web server, which authenticates the identity of the user, checks if the user is authorized to instantiate the exception, invokes the VIP Lanes path computation service to discover a middlebox-free path, and then uses SDN to push the exception into the network. The second method involves linking applications with a *wrapper library* that obtains socket information when a new connection is being established, and communicates with the VIP Lanes server to make the instantiation request, providing the user's credentials and flow characteristics needed to validate the request and ultimately install it in network devices along the desired path.

C. Securing the VIP Lanes Exception Mechanism

As we have shown, on-demand security exceptions opens up novel opportunities to improve performance, functionality, and privacy. However, it also opens up potentially dangerous new avenues of attack— including attacks where an attacker could "open up" the campus network with exceptions, or worse yet, gain complete control of the underlying programmable network. As illustrated in Figure 2, the VIP Lanes exception system consists of several components—including a VIP Lanes monitoring system and databases that are outside the scope of this paper—creating a reasonably broad attack surface. Consequently, it is critical that we secure the VIP Lanes exception system itself.

To ensure the security of the VIP Lanes system itself, VIP Lanes utilizes two levels of defense. First, it uses best-of-breed (web) practices to protect user-facing APIs. Second, it protects the SDN controller (which has complete control over the network) by employing a VIP Lanes proxy (a VIP Lanes-specific gateway) to tightly constrain the types of requests that can be sent to the controller.

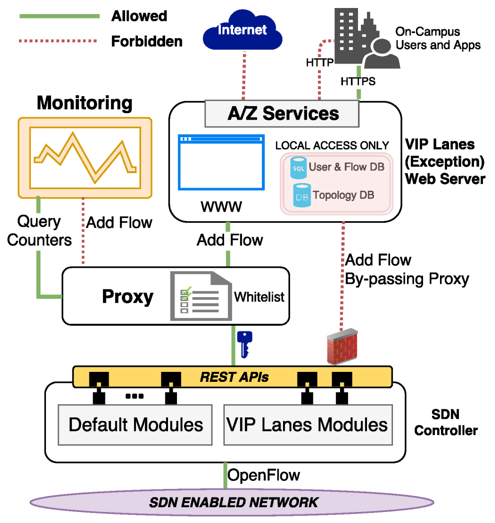

Fig. 2. Components of the VIP Lanes exception service, and allowed/forbidden communication among components.

On the user-facing side, only APIs from the VIP Lanes web server and the monitoring system are exposed. Other database services are protected by firewall rules, only allowing access from the VIP Lanes web server and monitoring system. Moreover, because the VIP Lanes web server and monitoring service have IP addresses that are only routeable internally, only applications running in the campus network can request VIP Lanes or view VIP Lanes traffic. The user-facing APIs are accessed over an encrypted channel (i.e. `HTTPS`) and require an industry standard username/password (web interface) or an identity key (wrapper).

To protect the SDN controller, we deployed a VIP Lanes Proxy (gateway) that inspects all requests sent from the VIP Lanes server to the SDN controller. Although the SDN controller supports a wide range of network management actions via its northbound interface (NBI), the VIP Lanes server only needs to use a small number of them to monitor and deploy security exceptions. It should be noted that the NBI of existing SDN controllers have widely varying access control mechanisms. Ideally they would offer a per-user or per-role authentication method. Unfortunately, robust access control mechanisms are often not included with current SDN controllers to the point that some of them (e.g., Floodlight, RYU, or POX) do not provide access control for REST-based APIs whatsoever. The Aruba VAN controller supports a very limited Role Based Access Control (RBAC) that currently provides a single role with access to all controller features (i.e. sdn-admin), giving far more control than is needed by the VIP Lanes exception server. If an attacker were to gain access to the sdn-admin role, they could bring ports up/down, capture any packet, inject traffic, or worse – all being capabilities not needed by the VIP Lanes exception service.

To reduce the risk of attack but yet work with existing controllers, the VIP Lanes Proxy is the only entity authorized to access the SDN controller's APIs. All calls to the controller

TABLE I
EXAMPLE WHITELIST ENTRIES IN A VIP LANES PROXY

Cert CN Field	*Authorized SDN Controller APIs*	*HTTP Commands Allowed*
vip-site.uky.edu	`^/sdn/viplanes/ab01[a-f0-9]{12}$`	`GET, POST, DELETE`
vip-site.uky.edu	`^/sdn/v2\.0/of/datapaths/[^/]+/ports/[^/]+$`	`GET`
vip-stats-db.uky.edu	`^/sdn/stcl/stats/counters$`	`GET`

must go through the VIP Lanes Proxy which inspects the API calls and blocks any calls that invoke controller capabilities that are not needed by the VIP Lanes exception server. In addition, the VIP Lanes Proxy serves as a certificate authority, signing client certificates (i.e. one for each component in the VIP Lanes system) so that clients can be identified and associated with a list of APIs they are authorized to invoke (i.e. a whitelist). (Note that if the SDN controller has no access control, a firewall – either standalone or on the controller, say via `iptables` – is needed to ensure packets cannot bypass the VIP Lanes Proxy to reach the controller.)

The data structure used to implement the VIP Lanes Proxy whitelist functionality is a map of clients (identified by the Common Name (CN) field of their signed certificates) to URLs (REST endpoints) that components are permitted to use (including the HTTP commands they are allowed to use per endpoint). Table I shows example whitelist entries, where the URLs are specified as extended regular expressions to narrow down the action field. For instance, the first entry enforces all VIP Lanes management calls to use our own structured identifiers, isolating on-demand exceptions from the default general policies controlling campus traffic, and obscuring the meaning of an identifier from would-be attackers.

On a similar note, if an attacker compromised the monitoring system and then asked the SDN controller (via the VIP Lanes Proxy) to make a change to the network, its connection to the VIP Lanes Proxy would be ignored by the VIP Lanes Proxy, reducing the risk of an attack on the monitoring system. Note that the monitoring system would still be able to invoke request for read-only information.

IV. EXPERIMENTAL RESULTS

We used VIP Lanes to deploy high-bandwidth on-demand exceptions at different locations on our campus network (Figure 3) to reach sites that are known to be used for research activites and therefore, trusted. Specifically, we ran experiments to measure throughtput to ESnet sites located in different geographic regions of the United States (San Diego, Washington D.C., and Chicago) and the Data Transfer Node (DTN) of the University of Kentucky which is located in the Science DMZ hanging off of the university's edge router.

We ran all the tests on a Macbook Pro with an Intel Core i5 processor 2.4 GHz, 16 GB RAM, and an external Thunderbolt2 10G adapter. In order to maximize the performance per test, some variables of the system's TCP/IP stack (e.g. TCP window scale factor or receive buffer) were tuned following the recommendations published by ESnet [4].

For each site and building we measured two throughputs using the `bwctl` tool. First, we recorded the performance obtained by letting the *Normal* campus network security

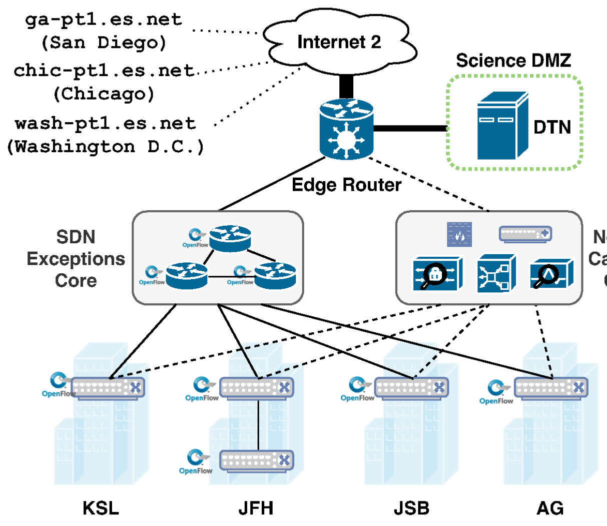

Fig. 3. SDN-enabled campus topology used to deploy exceptions

appliances inspect packets to enforce policies. Afterwards, we deployed short-lived security *Exceptions* from the laptop to the trusted sites. Lastly, we started the second set of performance measurements using the allocated path. We observed that on average, it takes 314 ms to deploy each exception in the data plane. Since this step happens before a flow is initiated, it has no impact on the performance gains obtained.

Table II shows the data collected after running the above experiments. At a first glance, it is clear that using the VIP Lanes exception mechanism researchers on our campus in most cases could benefit from a performance boost from tens of megabits per second (under normal conditions) to multiple gigabits per second (using exceptions). Unsurprisingly, the improved throughput was affected by the geographic location of the trusted site, e.g., speeds to the DTN reached close to 7.2 Gbps whereas measurements at San Diego (on the opposite coast of our campus) were below the 700 Mbps mark. Nonetheless, as can be seen in Figure 4, the speedup factor, i.e., how much faster the throughput is by using exceptions, was not necessarily bound to geographic location. For instance, the improvement from AG to the DTN was only of 11x the normal throughput, whereas from that same location to

TABLE II
THROUGHPUT FROM DIFFERENT BUILDINGS TO TRUSTED SITES*

Site	*KSL*	*JFH*	*JSB*	*AG*
San Diego, CA	31.3 (**669**)	28.8 (**671**)	31 (**669**)	19 (**663**)
Chicago, IL	182 (**3959**)	36.4 (**3129**)	95.4 (**3974**)	74.1 (**3707**)
Washington, D.C.	70 (**1289**)	29.4 (**1400**)	69.4 (**1570**)	56.7 (**1532**)
DTN	300 (**7120**)	67.7 (**7140**)	320 (**7200**)	644 (**7123**)

*The numbers are shown as *Normal* (***Exception***) throughput in Mbps

Chicago ESnet site the factor jumped to 50x. In most of the cases, the speedup factor was higher than 20x with only two data points sitting below such value.

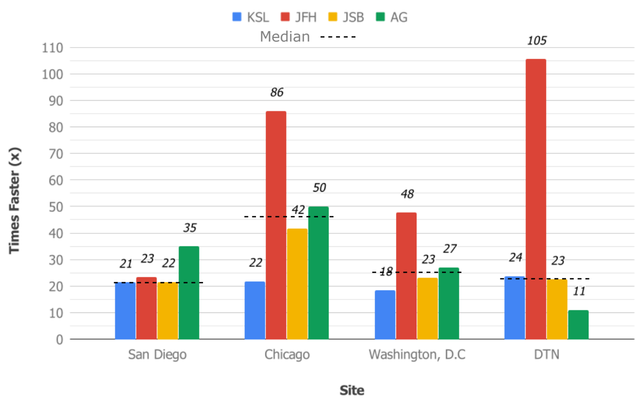

Fig. 4. Speedup factors at different sites using exceptions

V. Related Work

Different stakeholders in the network often have interests that may be adverse to each other, but accommodating these competing interests is crucial to the design of future networks [5]. In our previous work [3], we laid the groundwork to support alternative paths for approved research traffic. This paper describes a way to specify on-demand security exceptions that leverage those paths. The Internet Architecture Board's "Stack Evolution" program [6] considered a similar problem and proposed a solution to endpoint-middlebox communication where application and network provide *hints* to each other to convey intent in return for better service. However, their focus was on evolving the transport layer to address ossification issues, as opposed to developing a generalized model that includes real-world (human) policies to improve communication performance, functionality, and privacy.

The Switchboard project [7] allows for the creation of campus SDN paths to achieve high-speed data transfers, but uses a heavy-weight approval process involving the managers of the networks traversed. The DANCES project [8] targets data transfers between HPC sites and uses OpenFlow to create QoS paths. Requests are handled by HTTP calls to a centralized covering authority using authentication keys.

Various projects have proposed approaches that use SDN to implement access control mechanisms in the network [9]–[12]. These approaches focus on access control as opposed to secure high-speed path exceptions. Moreover, they are often tightly integrated with the internals of a particular SDN controller and its southbound API. In contrast, the VIP Lanes Proxy is controller-agnostic and only relies on a controller's northbound API.

VI. Conclusion

We proposed a new framework for handling security issues based on the on-demand security exceptions model. Short-term, on-demand, fine-grained exceptions provide an opportunity for trusted (and authenticated) users to declare the intent of their traffic, and in return get better service. The approach also allows service providers to define simpler, long-term policies and to focus data-plane security scrutiny on general traffic. We described a prototype implementation of the security exception mechanism to improve the high-speed big-data transfer in our campus network. Our experimental results demonstrate that the transfer rate of trusted users can be improved significantly when on-demand exceptions allow them to bypass middleboxes. Future work includes refining the exception specification mechanism and investigating other contexts in which the exception model can be usefully applied.

Acknowledgment

The work of Kenneth L. Calvert was supported by the National Science Foundation during his assignment there. The work of other authors was supported in part by the National Science Foundation under Grants ACI-1541380, ACI-1541426, and ACI-1642134.

References

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The Science DMZ: A Network Design Pattern for Data-intensive Science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 85:1–85:10.

[2] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[3] J. Griffioen, K. Calvert, Z. Fei, S. Rivera, J. Chappell, M. Hayashida, C. Carpenter, Y. Song, and H. Nasir, "VIP Lanes: High-speed custom communication paths for authorized flows," in *Proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN 2017)*, July 2017, Vancouver, Canada.

[4] Energy Sciences Network, "Host tuning," https://fasterdata.es.net/host-tuning/.

[5] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in cyberspace: defining tomorrow's Internet," *SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 347–356, Oct. 2002.

[6] Internet Architecture Board, "IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI)," https://www.iab.org/activities/workshops/semi/, January 2015.

[7] "Duke University SDN," https://sites.duke.edu/dukesdn/.

[8] V. Hazlewood, K. Benninger, G. Peterson, J. Charcalla, B. Sparks, J. Hanley, A. Adams, B. Learn, R. Budden, D. Simmel, J. Lappa, and J. Yanovich, "Developing Applications with Networking Capabilities via End-to-End SDN (DANCES)," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, ser. XSEDE16. New York, NY, USA: ACM, 2016, pp. 29:1–29:7.

[9] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, "Access Control for SDN Controllers," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN'14)*. New York, NY, USA: ACM, 2014, pp. 219–220.

[10] S. T. Yakasai and C. G. Guy, "FlowIdentity: Software-defined network access control," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 115–120.

[11] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based Network Access Control," in *2014 Third European Workshop on Software Defined Networks*, Sept 2014, pp. 79–84.

[12] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software Defined Network Control Layer." in *NDSS*, 2015.