Architecting Programmable Data Plane Defenses into the Network with FastFlex

Jiarong Xing
Rice University

Wenqing Wu
Rice University

Ang Chen
Rice University

ABSTRACT

This paper is motivated by the ever increasing scale and diversity of attacks that are best handled by the network infrastructure. FastFlex builds upon recent progress, which has developed a variety of network defenses in programmable data planes, and takes this trend one step further: it aims to develop *architectural* support for these defenses as a first-class citizen. We envision that the network architecture would support these defenses as naturally as it does routing—as the network routes traffic end-to-end, it also turns the defenses on and off as needed for attack mitigation. We propose a key abstraction: *the multimode data plane*. Normally, it operates under optimal configurations computed by centralized control, but upon attacks, it performs distributed mode changes entirely in data plane for mitigation. Mixed-vector attacks would trigger co-existing modes at different regions of the network, and attacks that rapidly change would be met with equally fast mode adaptations. We sketch this vision, discuss the opportunities and challenges it involves, and present a use case on link-flooding defense.

ACM Reference Format:
Jiarong Xing, Wenqing Wu, and Ang Chen. 2019. Architecting Programmable Data Plane Defenses into the Network with FastFlex. In *HotNets '19: ACM Workshop on Hot Topics in Networks, November 13–15, 2019, Princeton, NJ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3365609.3365860

1 INTRODUCTION

Network attacks are escalating [3–7, 9–11]. While some attacks can and should be handled by endpoints, an increasing number of them are best handled by the network infrastructure [79]. Notable examples include large-scale DDoS attacks against small endpoints [3, 5, 6, 11], and link-flooding attacks that specifically target the network core [43, 44, 50, 71, 74, 80]. Worse still, attackers usually have control over a "programmable attack infrastructure"—i.e., the compromised machines—so they can launch dynamic attacks with rapidly changing locations and vectors [44, 71]. For these attacks, defenses at the last mile are at a severe disadvantage or inherently insufficient.

Supposing that the network infrastructure treats security as a first-class goal just as it does routing, then these attacks are much easier to handle. The network infrastructure already forms a massive backbone, whose power can naturally match that of all network attackers *combined*. Since any attack traffic has to first flow through the network, an architecture with direct security support can detect, throttle, or drop suspicious traffic at any location and time. This would allow the endpoints to focus their defenses on endpoint-specific threats, of which there are already many.

Up until very recently, the network infrastructure was only able to provide limited forms of security support, such as access control lists in network switches, or deep packet inspection in middleboxes. The key reason is that the network data plane used to be fixed in function, which in turn could only provide fixed-function security. Deploying programmable defenses in SDN controllers [43, 68, 76, 80] can only alleviate this problem to a certain extent, because software controllers cannot directly handle the volume and velocity of data plane traffic. As a result, there has come to exist a gap between the diversity and dynamicity of attacks and the simple and static nature of the network infrastructure.

But as of late, researchers have seized the opportunity provided by *programmable data planes* to develop a wide range of in-network defenses [31, 34, 51, 55, 56, 69, 78]. Recent switches can be programmed to support user-defined protocols, customized packet processing, and sophisticated hardware state. These new features have enabled many powerful defenses, such as heavy-hitter detection [34, 69], DDoS detection [31], hop count filtering [51], topology obfuscation [55], enterprise access control [56], covert channel mitigation [78], likely with more to come. These programmable data plane defenses are in a class of their own: they can mitigate high-volume attacks with low overhead, they can perform attack detection over every single packet, and they can respond to changing attacks at hardware speeds.

HotNets '19, November 13–15, 2019, Princeton, NJ, USA

ACM ISBN 978-1-4503-7020-2/19/11…$15.00
https://doi.org/10.1145/3365609.3365860

This paper considers an *architectural* question: *Can we architect these defenses into the network paths, so that as the network routes traffic end-to-end, it also turns the defenses on and off as needed for real-time attack mitigation?* We envision a data plane that self-adapts to changing attacks in multiple modes. It operates by default under optimal configurations computed by centralized control, e.g., using traffic engineering over a stable traffic matrix [18, 37, 40], but it can perform fast mode changes entirely in data plane upon attack. Mixed-vector attacks would trigger different modes at different regions of the network, and rapidly changing attacks would be met with equally fast-changing modes. By architecting defenses into network paths, we can reduce (or eliminate!) the need for traffic detour to security checks, and by optimizing the degree of distribution, we can prevent chokepoints from forming. We can even repurpose switches at runtime to dynamically scale certain defenses as needed. This architecture would blur the boundaries between "networking" and "security", with the eventual goal of transforming the network into a "programmable defense infrastructure".

FastFlex does *not* aim to replace legacy middleboxes, but to work together with them in a complementary manner. Features like cryptography or payload inspection go beyond the capability of today's switch hardware, and they can still run in middleboxes. The FastFlex architecture simply advocates for defenses that can run in the data plane—which are becoming increasingly abundant [31, 34, 51, 55, 56, 69, 78]—to be architected into the routing infrastructure to achieve a whole-network defense. In other words, legacy elements can still be part of the "default" mode, while programmable elements can enter and exit the defense modes dynamically.

The FastFlex architecture presents interesting research challenges. Since we need to multiplex switch resources between defenses and routing, we may need to decompose a defense into smaller modules, so that they can be packed more efficiently, and that we can identify shared modules for consolidation. Moreover, although some attacks can be detected locally at one switch (e.g., link flooding [43]), others are only detectable in a distributed manner (e.g., global rate limits [62]); we need to carefully design synchronization and mode change protocols to accommodate co-existing and fast-changing modes while maintaining stability. Finally, when we repurpose a switch at runtime, we need to ensure that its functions are correctly and efficiently handled elsewhere, and that relevant state is transferred in and out reliably. We also need to ensure that critical state is properly replicated, so that the defense would work correctly in the presence of switch failures.

The rest of this paper motivates this vision further, discusses the new challenges and opportunities in more detail, and presents an initial case study.

2 OVERVIEW

Next, we further motivate why programmable data planes could enable a new class of defenses. We then describe our vision and discuss the key challenges.

2.1 Motivation

Why in-network? In proposing to support security inside the network, we must give due consideration to the classic "end-to-end arguments" [64], and evaluate whether a security function should be placed in-network or at endpoints. We believe there are three classes of defenses where in-network support is indispensable. We describe them below, drawing on some examples provided by Zave and Rexford [79].

First, if an attack targets the network, then it should be mitigated in-network; in this case, the network effectively becomes an "end" [64]. Examples of this include 1) link-flooding attacks (LFA) [44, 74], where an adversary causes congestion in a few critical network links for denial of service, and 2) network reconnaissance attacks [55]. Second, if the threat model explicitly considers compromised endpoints, then the network is a last line of defense. Examples in this category include 3) mandatory access control from the network on the endpoints [56], and 4) prevention of data exfiltration from compromised hosts [78]. Finally, network-based availability attacks (e.g., volumetric DDoS [31, 34, 70])—even if they target endpoints—are best handled by the network because the endpoints often have fewer resources for the defense.

Why programmable data planes? The backdrop of our vision is a fruitful line of work in developing *programmable control plane* defenses using OpenFlow-based SDN [25, 33, 43, 68, 76, 80]. Programmable data planes provide three key security advantages over OpenFlow: per-packet visibility, per-packet dynamicity, and scale-free defense. *Per-packet visibility* means that we can develop attack detection algorithms in switch hardware and apply them to every single packet, instead of sampled or aggregated traces at software controllers. *Per-packet dynamicity* means that, upon attack detection, we can immediately take actions to mitigate the attack at the switch, instead of incurring a round-trip time delay to remote controllers. Furthermore, *scale-free defense* means that, since the defenses reside directly in the switches, they naturally scale with network size and speed; centralized controllers are no longer a bottleneck. We believe that these advantages will enable a new paradigm for network security, which we call *programmable in-network security*.

2.2 Programmable in-network security

In our vision, a switch would eventually become a defense platform running many "defense apps", and it would dynamically swap in the right defenses as it forwards traffic. On that note, switch hardware has constrained programming models

and limited resources, which certainly impose constraints on the possible defenses we could develop. But interestingly, existing work has already developed a range of programmable data plane defenses *within* these constraints [31, 34, 51, 55, 56, 69, 78]. For instance, NetHide [55] prevents an attacker from learning the network topology by obfuscating traceroute responses, mitigating network reconnaissance attacks; Poise [56] enforces network access control on enterprise devices, protecting against compromised endpoints; HashPipe [70] defends against volumetric DDoS attacks. These are encouraging evidence that the classes of defenses best suited for in-network deployment seem to be a good match for programmable data planes.

Moreover, a recent project NetWarden [78] has further observed that, we could *overcome* the limitations of the hardware by co-designing it with control plane software. At a high level, we can split a defense algorithm into a fastpath component, which runs in the data plane hardware to achieve high efficiency, and a slowpath component, which runs in control plane software to achieve high generality. As long as the slowpath is only occasionally involved, the defense algorithm can still run efficiently. This would potentially enable even more defenses to be developed in the switch.

Taking this vision one step further, we envision that a programmable network would eventually become a defense fleet, where programmable data plane defenses are architected into the network paths and synchronized for whole-network defense. The need for orchestrating standalone defenses for a network-wide defense has been extensively motivated and studied in SDN-based middlebox architectures [26, 28, 35, 61], and similar rationale holds for programmable data plane defenses: Individual defenses working in silo only have local views, they may produce uncoordinated (or even conflicting) decisions, and they may duplicate each other's work. Furthermore, programmable data plane defenses also need to co-reside and interact with routing. Therefore, whereas existing work has mainly focused on developing individual defenses, our primary investigation in this paper is to develop architectural support for these defenses.

2.3 Key challenges

Architecting programmable data plane defenses into the routing infrastructure raises several interesting challenges.

Challenge 1: Resource multiplexing. Switches have resource constraints in terms of memory, hardware stages, and ALUs (arithmetic logic units), which now need to be multiplexed between security and routing functions. Fortunately, unlike traditional middleboxes, data plane defenses are just another kind of switch programs. We can decompose them into smaller modules to enable easier packing, and we can reassemble these modules to enable sharing. For instance, shareable components in existing defenses include examples like probabilistic data structures such as sketches and bloom filters [34, 69, 78], packet parsers/deparsers [34, 51, 69], and tables that maintain per-flow/per-destination state [51, 56, 78].

Challenge 2: Optimal placements. A second challenge is to find optimal defense placements in the network. On the one hand, we have a unique opportunity to distribute the defenses *pervasively* to avoid traffic detour and hotspots. But on the other hand, finding an optimal placement for unpredictable attack patterns is challenging. Traditional middleboxes are usually placed at a fixed number of locations based on a stable traffic matrix [26, 35, 61], but FastFlex cannot exhaustively anticipate all possible attacks.

Challenge 3: Distributed control. Existing work on middleboxes have developed centralized SDN architectures for network-wide control [26, 28, 61]. However, in FastFlex, interposing an software controller on data plane defenses would considerably offset one of their greatest advantages—dynamicity—as responses must now again go through a software feedback loop. Therefore, we need a way to synchronize the defenses in a distributed manner in data plane, so that they can exchange information with each other and perform mode changes without centralized control.

Challenge 4: Dynamic scaling. Since attacks can be unpredictable, FastFlex may also need to dynamically scale certain defenses at runtime, e.g., by repurposing certain switches to run different programs. However, this also raises challenges such as handling disruptions and tolerating faults.

3 THE FASTFLEX ARCHITECTURE

In this section, we discuss these challenges in more detail, and sketch tentative solutions. Figure 1 shows the overall workflow of FastFlex. In the ensuing discussion, we will refer to a defense app as a "booster".

3.1 Resource multiplexing

Our first **challenge** comes from the *limited switch resources*. A typical programmable switch today has 10–20 hardware stages, each of which has a fixed amount of memory and ALUs [19]. These resources need to be shared across boosters and routing programs. To achieve this, we can model each switch using a vector of resource constraints $<\Theta_1, \Theta_2, ...\Theta_k>$, where Θ_i represents the amount of resources of the i-th type. Similarly, we can model a switch program as a vector of resource requirements, e.g., the j-th program requires $<\theta_{j1}, \theta_{j2}, ...\theta_{jk}>$. Our problem is then to *pack* a number of items (i.e., programs) into sets (i.e., switches), while ensuring that $\sum_j \theta_{ji} \leq \Theta_i, \forall i \in [1..k]$, i.e., programs on a switch collectively stay with the available resources.

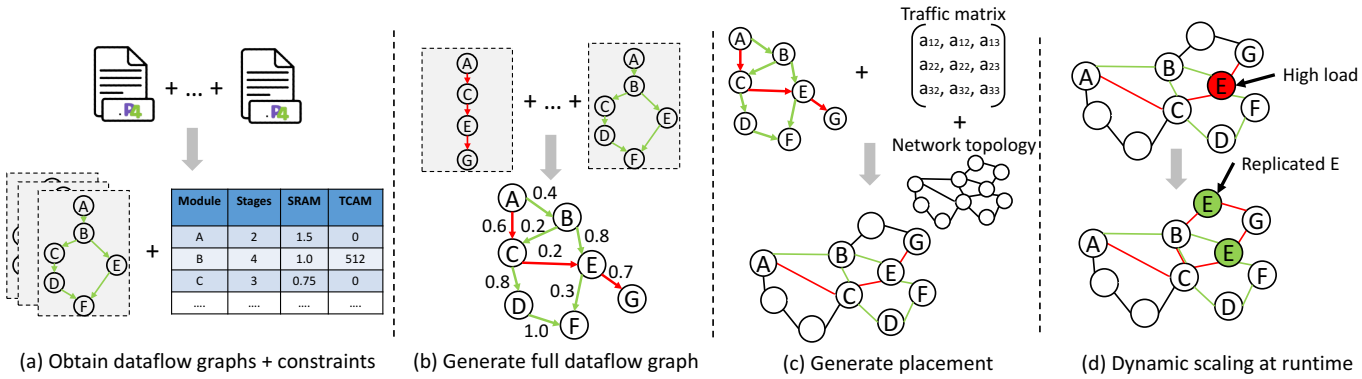

Figure 1: FastFlex transforms boosters into dataflow graphs (a). The program analyzer jointly analyzes all dataflow graphs to identify opportunities for module sharing, and produces a merged dataflow graph (b). The scheduler maps the merged graph to the network while optimizing for some performance objectives (c). At runtime, FastFlex can dynamically scale boosters based on real-time attack patterns (d).

Opportunity: Decomposition. Since smaller programs are easier to pack than larger ones, we can use a *program analysis* engine to decompose programs into smaller modules to enable a tighter packing. We call these modules *packet processing modules*, or PPMs. A program then can be transformed into a dataflow graph, where the set of vertexes V are the PPMs and the set of edges E represent traffic direction. An edge $v \rightarrow v'$ also has a weight, which represents the amount of state sharing between v and v'. For instance, if v' needs to read a packet counter x in v, then a packet leaving v needs to carry the value of x to v', e.g., as a header field. Therefore, ideally, we should identify clusters of PPMs, where intra-cluster edges are dense and have heavy weights and inter-cluster edges have opposite properties.

Opportunity: Sharing. This decomposition also enables opportunities to share PPMs across boosters [14, 32, 81]. Common components in existing boosters include probabilistic data structures, such as sketches and bloom filters, connection tables for maintaining per-prefix, pre-IP, or per-flow state, as well as packet parsers and deparsers. An interesting challenge here is that, boosters may implement the same function differently, e.g., using different variable names and code structures, so how does FastFlex tell whether two PPMs are shareable? A recent project [24] has shown that switch programs are simple enough to determine *equivalence*. This result comes in handy and provides a way to identify functionally equivalent PPMs despite implementation differences.

3.2 Finding an optimal placement

Our second step is to place the PPM graphs to a network. Suppose that we have a stable traffic matrix and a network topology (e.g., switches with different resource constraints, and links with different rates), then we can directly apply existing results in placing middleboxes [26, 28, 35, 61]. However, FastFlex has a new opportunity and a new challenge.

Opportunity: Pervasive distribution. Existing work assumes the number of middlebox instances as a given [26, 35, 61], but here, the boosters are akin to software-based "network functions" [29, 47, 48, 57, 58, 77], and can potentially maintain a pervasive presence to *maximally prevent hotspot from forming* and *reduce traffic detour*. Therefore, the scheduler needs to decide on the optimal number of instances for each PPM, as well as how to place them directly on path when possible. The optimization goal, for instance, could be to minimize the maximal link load across the network [17]. As traffic flows through their optimal routes, they simultaneously pass through the boosters.

Challenge: Planning for the unpredictable. Another difference from existing work on middlebox placement is that, while we could obtain a stable traffic matrix for the FastFlex default mode, traffic matrices in attack modes are *unknown* to the scheduler and can be *highly unpredictable*. Unlike software NFs, which can be elastically scaled up and down with low overheads [29, 47, 48, 57, 58, 77], runtime scaling of switch-based PPMs is much more costly (more details later in Section 3.4). One potential approach is to perform best-effort planning for attack modes, by distinguishing between two types of PPMs: detection and mitigation modules. We could distribute detection modules as widely as possible, ideally on all paths, since they need to inspect and classify traffic and trigger mode changes. Mitigation modules, on the other hand, are placed as close as possible to their respective detectors, e.g., at their downstream, so that attacks can be mitigated quickly.

3.3 Distributed control at runtime

FastFlex has an **opportunity** to perform fast mode changes entirely in data plane to mitigate *changing attacks*. The corresponding challenge is that, we need to synchronize the boosters in a distributed manner without a central controller.

Challenge: Mode change protocols. To achieve a fast response, detectors can initiate mode change requests in data plane immediately upon attack detection. These requests can be carried in special probes, and they should specify the detected attack type so that mitigation modules can enter the corresponding defense mode. To handle mixed attack vectors, FastFlex could activate multiple modes for different regions of the network. For changing attacks, FastFlex would perform rapid mode changes in response. FastFlex can build upon existing work on mode change protocols [20, 59, 60], which has developed formal frameworks for reasoning about stability and mode transitions.

Challenge: Distributed detection. Another interesting angle is whether it is sufficient to detect attacks locally at one detector, or whether we would need to detect attacks in a distributed manner across detectors. As examples, link-flooding attacks are locally detectable, but other problems, such as network-wide heavy hitters [34] or global rate limits [62], may require a network-wide detection. In these cases, FastFlex needs to additionally synchronize different detectors' views periodically, e.g., similarly using probing packets. Our goal would be to detect network-wide attacks while minimizing the amount of synchronization across detectors.

3.4 Dynamic scaling at runtime

Unlike fixed-function middleboxes [26, 61], FastFlex has a unique **opportunity** to repurpose a switch to *dynamically scale* at runtime. This is helpful when attack strengths exceed the "best effort" planning in the setup/placement phase.

Challenge: Handling transient disruption. In contrast to software NFs with full elasticity [29, 47, 48, 57, 58, 77], repurposing a switch at runtime is much more costly. In some switch models (e.g., Barefoot Tofino [13]), this currently requires installing a new switch program, which depending on the program size could take several seconds; [1] on other switch models (e.g., Broadcom Trident 4 [2]), certain parts of the switch program can be reconfigured at runtime without incurring downtime. Either way, a switch needs to inform its neighbors before it goes through a reconfiguration, so that neighboring switches can perform fast reroutes along other paths [38, 46] until the reconfiguration completes.

[1] Obtained by an experiment installing a complex switch program; based on feedback from industry vendors, this latency could be significantly optimized.

Challenge: Transferring and replicating state. When we reconfigure a switch, we may also need to transfer its state to other switches and potentially migrate some of it back later. As data plane state could be updated per packet at Tbps, FastFlex cannot use software controllers to perform this transfer [53]. A recent project has considered a similar problem: it developed program analysis techniques to identify variables whose values need to be transferred, and then tagged these values to normal traffic to be piggybacked across the network [53]. FastFlex can leverage a similar mechanism, but at the same time, it also needs to handle new problems such as fault tolerance. For instance, to tolerate packet drops, we should be able to temporarily increase the reliability of state-carrying packets, e.g., using FEC (forward error correction) codes and redundancy. FEC encoding and decoding are bitwise operations over special header fields, therefore implementable in data plane. To tolerate switch and link failures, which may cause state loss and/or disruption to the distributed protocols, FastFlex may need to replicate critical state across the network.

4 CASE STUDY

In this section, we use link-flooding attacks (LFA) [44, 49, 54, 74] as a concrete case study to discuss how FastFlex can enable fast responses to changing attacks. An LFA attacker can perform traceroutes to public servers near the victim to map the topology, pick a few critical links that carry all or most of the victim's traffic, and then overwhelm these links by initiating many connections to these public servers. The victim destination can be cut off from the Internet without even seeing attack traffic. Worse still, advanced attackers can enhance these attacks in two ways.

Indistinguishability [44]: While some LFAs use high-volume attack flows [74], which are easy to detect and block, an adversary with large-scale botnets can launch many legitimate TCP flows (e.g., low-rate web requests) to overwhelm a link [44]. Detectors have high false positive/negative rates on such traffic patterns [71]. To be conservative, state-of-the-art defenses reroute traffic from congested links to others, e.g., using SDN controllers that perform dynamic traffic engineering (TE) [50], instead of simply dropping the packets.

Rolling attacks [44, 80]: However, an attacker can launch a *rolling* attack that changes its target link(s) dynamically to evade TE [50, 71]. Since TE is performed in a centralized software controller, new configurations can only be computed and deployed at the timescale of minutes [37, 40]. A rolling attack, therefore, can dynamically shift its target links within this timescale, and persist for a long period of time [44, 80].

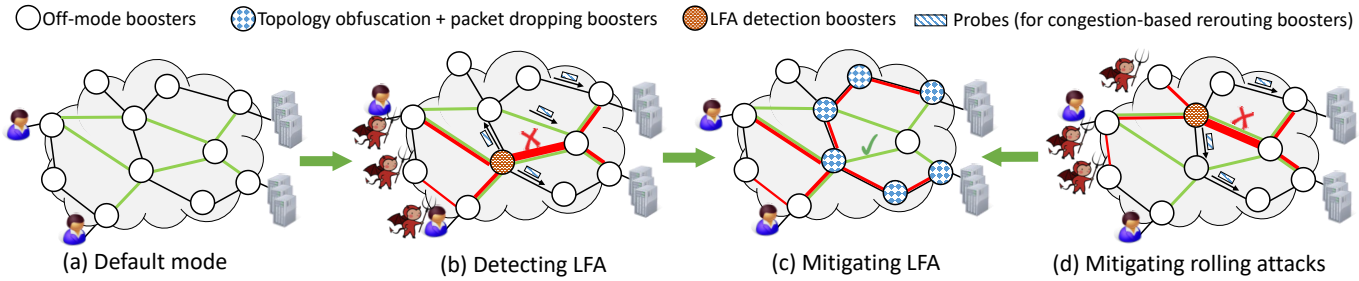

Figure 2: FastFlex provides the abstraction of a multimode data plane. FastFlex defenses are default off in the normal case (a). Upon detecting LFA, FastFlex starts propagating probe packets to activate congestion-based rerouting (b). FastFlex pins normal flows to their original paths, and only reroutes suspicious traffic to longer paths, and it creates an "illusion of success" to the attacker by causing packet drops and obfuscating traceroute responses to suspicious flows (c). FastFlex is robust to rolling attacks (d). This abstraction applies to other types of attacks as well, e.g., multi-vectored DDoS attacks [12] or short-lived pulsing attacks [1].

4.1 Building block boosters

We first present several building blocks that we can use for different aspects of LFA defense. FastFlex will then control them collectively to achieve, to the best of our knowledge, *the first effective defense against rolling attacks.*

LFA detection. First, we need to detect a) high link loads and b) persistent, low-rate flows to a destination prefix. a) is quite straightforward, and b) can be achieved by adapting algorithms that monitor per-flow TCP state in the data plane [30, 36].

Packet-dropping defense. One simple defense is to rate limit or drop packets from suspicious flows [43, 50, 80]. Since this may cause collateral damage to normal flows, such a defense should be applied only to highly suspicious flows.

Routing around congestion. A more conservative defense is to reroute traffic from congested links to other parts of the network. This can be naturally achieved from SDN controllers [43], but more relevant here are the recent projects on performance-aware routing entirely in data plane [16, 38, 46]. Hula [46] and Contra [38] are two solutions that work for Fattree and arbitrary topologies, respectively. At a high level, programmable switches disseminate probes that carry path utilization metrics, and they optimize their routing decisions on the fly to forward traffic to the least-congested paths. Since forwarding decisions are made entirely in data plane, this may lead to suboptimal solutions compared to centralized TE [16]; but they enable much faster responses and are good candidates as defenses against rolling attacks.

Topology obfuscation. An attacker can easily change the target links if she detects that her attack has triggered a defense. NetHide [55] is a data plane defense that can respond to traceroute probes using obfuscated IP addresses, trading some diagnostic utility of traceroutes for security.

4.2 The FastFlex data plane defense

Putting them all together is the FastFlex defense, which routes traffic along these boosters and mitigates attacks in real time. Figure 2 shows the data plane mode changes.

(1) In the default mode, only LFA detectors are turned on. Routing follows an optimal strategy as computed by centralized TE. (2) Upon detecting an attack, the detector propagates the alarm across the network using probe packets, which turns on the congestion-based rerouting boosters; switches start to probe for available bandwidths along longer but less congested paths. This rerouting will reduce congestion therefore loss rates at the victim links. However, packets forwarded along less congested but longer paths would experience a tradeoff between shorter queuing delays (i.e., less congestion) and longer propagation delay (i.e., longer paths); the end-to-end latency could fluctuate or even increase [38, 42]. (3) Therefore, FastFlex further applies different modes to normal and suspicious flows based on the detector outputs. It only reroutes suspicious flows, but pins normal flows to the original paths as determined by optimal TE; this relieves the congestion while only causing minimal disturbance to normal traffic. (4) Furthermore, FastFlex turns on topology obfuscation for the suspicious flows, so that attackers cannot detect rerouting of their traffic. (5) Finally, FastFlex can create an "illusion of success" by dropping packets in the most suspicious flows. If attackers mistakenly perceive the attack to be successful, they are even less incentivized to change their attack further. (6) FastFlex would return to the optimal default mode as soon as attacks subside.

Since the mode changes take place at RTT timescale without central control, FastFlex can effectively mitigate highly dynamic, rolling attacks. Also, some of these boosters could be shareable for other types of defenses beyond LFA, e.g., for volumetric DDoS attacks. Individually, these boosters target

very different aspects, but FastFlex can synchronize them as a whole for network-wide defense.

4.3 Initial validation

We present an initial validation using a customized version of `ns3` with P4 `bmv2` [8] model support. We have set up the network topology in Figure 2, and implemented much simplified versions of the boosters discussed above.

FastFlex vs. baseline. The baseline system uses an SDN controller that performs centralized TE to reconfigure the network every 30 seconds, which is modeled after a state-of-the-art LFA defense [43]. Our FastFlex defense uses the same configuration as computed by centralized TE in stable mode. However, it can detect attacks and change defense modes entirely in data plane at RTT timescales.

Attacks. In the topology, there are two critical links that an LFA attacker can target. The adversary mapped the critical links using traceroute, and she launched rolling attacks to different links whenever she detected a routing change.

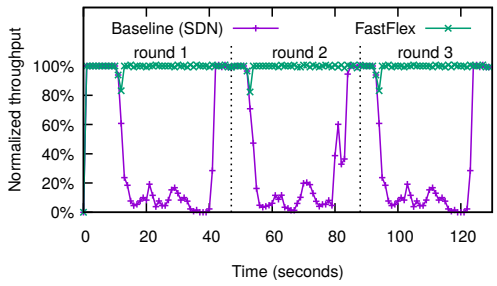

Figure 3: FastFlex outperforms the baseline defense.

Effectiveness. Figure 3 shows the throughput of normal user flows during the attack (normalized by the stable throughput without attack). We found that the baseline defense constantly falls behind when the adversary performs rolling attacks that dynamically change targets. The FastFlex defense, on the other hand, can present an obfuscated topology to malicious flows; moreover, even if the attacker changes her target link dynamically, FastFlex can disperse the traffic almost instantaneously by data plane mode changes.

5 RELATED WORK

We have already discussed many related projects; below, we briefly summarize the most relevant threads of work.

In-network middleboxes. Existing work has studied placing and routing around middleboxes inside the network [26, 28, 35, 41, 61]. FastFlex could build upon these work in terms of formulating the placement and routing problems, as well as the optimization techniques they have developed. However, in FastFlex, the security boosters are neither fixed in function or location; their orchestration is also performed in data plane for fast responses, instead of at an SDN controller [26, 28].

Software NFs. Another line of work has studied software NFs running in general-purpose servers, which has produced fruitful results on sharing and migrating state, as well as dynamic scaling [22, 23, 29, 39, 47, 48, 57, 58, 67, 73, 77]. FastFlex is also inspired by these work. However, architecting boosters into network paths involves new challenges due to their switch-based nature and in-network deployment.

Active networking. Research on network programmability can be traced back to active networking [15, 66, 75]. Recent work on OpenFlow SDN, NFV, and programmable data planes share similar design and architectural insights, although they also revisit these concepts in light of new technological developments and obtained significantly different results [27]. Our architecture is based on programmable data planes and shares these similarities and differences.

6 DISCUSSION

Securing the boosters. FastFlex must make sure that the individual in-network defenses, as well as their composition, are secure. Since switch programs are much simpler than general-purpose programs, it should be possible to achieve high assurance by formally verifying them [52, 72]. In a related project, we have also outlined a first step towards discovering and mitigating attack vectors in switch programs in an automated manner [45].

Stability. Since FastFlex performs fast mode changes in data plane, it is important to ensure that these mode changes—individually and collectively—do not lead to instability. In particular, we should defend against an attacker that intentionally causes mode changes frequently. FastFlex should be able to borrow from the literature on self-stabilizing systems [21, 65] for this purpose.

Federation. So far, we have discussed FastFlex as operating in a local network without cross-domain federation. If multiple domains deploy FastFlex, they would be able to collaboratively detect and mitigate more advanced attacks [63]. At the same time, federation would raise new challenges in both technical and non-technical aspects, such as trust, authentication, and privacy.

7 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. We also thank Jennifer Rexford, Vincent Liu, Qiao Kang, and Kuo-Feng Hsu for their insightful comments. This work was partially supported by an NSF grant CNS-1801884.

REFERENCES

[1] Attackers Use DDoS Pulses to Pin Down Multiple Targets. https://www.imperva.com/blog/pulse-wave-ddos-pins-down-multiple-targets/.

[2] Broadcom Trident 4 delivers disruptive economics for enterprise data center and campus networks. https://www.globenewswire.com/news-release/2019/06/11/1866927/0/en/Broadcom-Trident-4-Delivers-Disruptive-Economics-for-Enterprise-Data-Center-and-Campus-Networks.html.

[3] GitHub survived the biggest DDoS attack ever recorded. https://www.wired.com/story/github-ddos-memcached/.

[4] An interview with Lizard Squad, the hackers who took down Xbox Live. https://www.dailydot.com/debug/lizard-squad-hackers/.

[5] Mirai: what you need to know about the botnet behind recent major DDoS attacks. https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks.

[6] No sooner did the ink dry: 1.7 Tbps DDoS attack makes history. https://www.netscout.com/blog/security-17tbps-ddos-attack-makes-history.

[7] NSFOCUS identifies DDoS attack trends in new 2018 insights report. https://nsfocusglobal.com/nsfocus-identifies-ddos-attack-trends-new-2018-insights-report/.

[8] P4 behavioral model. https://github.com/p4lang/behavioral-model.

[9] A PoC hash complexity DoS against PHP. https://github.com/bk2204/php-hash-dos.

[10] Slow DoS on the rise. https://blogs.akamai.com/2013/09/slow-dos-on-the-rise.html.

[11] TCP flag DDoS attack by Lizard Squad indicates DDoS tool development. https://blogs.akamai.com/2015/01/tcp-flag-ddos-attack-by-lizard-squad-indicates-ddos-tool-development.html.

[12] Three out of four DDoS attacks target multiple vectors. https://www.helpnetsecurity.com/2017/09/26/ddos-attacks-target-multiple-vectors/.

[13] Tofino: World's fastest P4-programmable Ethernet switch ASICs. https://www.barefootnetworks.com/products/brief-tofino/.

[14] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, and A. Akella. P5: Policy-driven optimization of P4 pipeline. In *Proc. SOSR*, 2017.

[15] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. j Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *IEEE Network*, 12(3):29–36, 1998.

[16] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proc. SIGCOMM*, 2014.

[17] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. SNAP: Stateful network-wide abstractions for packet processing. In *Proc. SIGCOMM*, 2016.

[18] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine grained traffic engineering for data centers. In *Proc. CoNEXT*, 2011.

[19] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[20] T. Chen and L. T. X. Phan. SafeMC: A system for the design and evaluation of mode change protocols. In *Proc. RTAS*, 2018.

[21] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[22] M. Dobrescu, K. J. Argyraki, and S. Ratnasamy. Toward predictable performance in software packet-processing platforms. In *Proc. NSDI*, 2012.

[23] M. Dobrescu, N. Egi, K. J. Argyraki, B.-G. Chun, K. R. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. In *Proc. SOSP*, 2009.

[24] D. Dumitrescu, R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu. Dataplane equivalence and its applications. In *Proc. NSDI*, 2019.

[25] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic DDoS defense. In *Proc. USENIX Security*, 2015.

[26] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. NSDI*, 2014.

[27] N. Feamster, J. Rexford, and E. Zegura. The road to SDN: An intellectual history of programmable networks. *ACM SIGCOMM CCR*, 44(2):87–98, 2014.

[28] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. In *Proc. HotNets*, 2012.

[29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *Proc. SIGCOMM*, 2014.

[30] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data plane performance diagnosis of TCP. In *Proc. SOSR*, 2017.

[31] G. Grigoryan and Y. Liu. LAMP: Prompt layer 7 attack mitigation with programmable data planes. In *Proc. ANCS*, 2018.

[32] D. Hancock and J. van der Merwe. HyPer4: Using P4 to virtualize the programmable data plane. In *Proc. CoNEXT*, 2016.

[33] R. Hand, M. Ton, and E. Keller. Active security. In *Proc. HotNets*, 2013.

[34] R. Harrison, Q. Cai, A. Gupta, and J. Rexford. Network-wide heavy hitter detection with commodity switches. In *Proc. SOSR*, 2018.

[35] V. Heorhiadi, M. K. Reiter, and V. Sekar. Simplifying software-defined network optimization using SOL. In *Proc. NSDI*, 2016.

[36] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *Proc. NSDI*, 2019.

[37] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. SIGCOMM*, 2013.

[38] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, P. Tammana, and D. Walker. Contra: A programmable system for performance-aware routing. In *Proc. NSDI*, 2020.

[39] M. A. J. Martins, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the art of network function virtualization. In *Proc. NSDI*, 2014.

[40] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proc. SIGCOMM*, 2013.

[41] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *Proc. SIGCOMM*, 2008.

[42] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. SIGCOMM*, 2005.

[43] M. S. Kang, V. D. Gligor, and V. Sekar. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. In *NDSS*, 2016.

[44] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire attack. In *Proc. IEEE S&P*, 2013.

[45] Q. Kang, J. Xing, and A. Chen. Automated attack discovery in data plane systems. In *Proc. USENIX CSET*, 2019.

[46] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proc. SOSR*, 2016.

[47] J. Khalid and A. Akella. Correctness and performance for stateful chained network functions. In *Proc. NSDI*, 2019.

[48] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the way for NFV: Simplifying middlebox modifications using StateAlyzr. In *Proc. NSDI*, 2016.

[49] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants. In *Proc.*

SIGCOMM, 2003.

[50] S. B. Lee, M. S. Kang, and V. D. Gligor. CoDef: Collaborative defense against large-scale link-flooding attacks. In *Proc. CoNEXT*, 2013.

[51] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan. NetHCF: Enabling line-rate and adaptive spoofed IP traffic filtering. In *Proc. ICNP*, 2019.

[52] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, C. C. Robert Soulé, Han Wang, N. McKeown, and N. Foster. p4v: Practical verification for programmable data planes. In *Proc. SIGCOMM*, 2018.

[53] S. Luo, H. Yu, and L. Vanbever. Swing state: Consistent updates for stateful and programmable data planes. In *Proc. SOSR*, 2017.

[54] X. Luo and R. K. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *NDSS*, 2005.

[55] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev. NetHide: Secure and practical network topology obfuscation. In *Proc. USENIX Security*, 2018.

[56] A. Morrison, L. Xue, A. Chen, and X. Luo. Enforcing context-aware BYOD policies with in-network security. In *Proc. HotCloud*, 2018.

[57] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A framework for NFV applications. In *Proc. SOSP*, 2015.

[58] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. NetBricks: Taking the V out of NFV. In *Proc. OSDI*, 2016.

[59] L. T. Phan, S. Chakraborty, and I. Lee. Timing analysis of mixed time/event-triggered multi-mode systems. In *Proc. RTSS*, 2009.

[60] L. T. Phan, I. Lee, and O. Sokolsky. A semantic framework for multi-mode systems. In *Proc. RTAS*, 2011.

[61] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *Proc. SIGCOMM*, 2013.

[62] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud control with distributed rate limiting. In *Proc. SIGCOMM*, 2007.

[63] J. Reed, A. J. Aviv, D. Wagner, A. Haeberlen, B. C. Pierce, and J. M. Smith. Differential privacy for collaborative security. In *Proc. EuroSec*, 2010.

[64] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4), 1984.

[65] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[66] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart packets for active networks. In *Proc. OpenArch*, 1999.

[67] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. NSDI*, 2012.

[68] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proc. CCS*, 2013.

[69] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proc. SOSR*, 2017.

[70] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proc. SOSR*, 2017.

[71] J. M. Smith and M. Schuchard. Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive BGP routing. In *Proc. IEEE S&P*, 2018.

[72] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu. Debugging P4 programs with Vera. In *Proc. SIGCOMM*, 2018.

[73] R. Stoenescu, M. Popovici, V. Olteanu, J. Martins, R. Bifulco, F. Huici, M. Ahmed, G. Smaragdakis, M. Handley, and C. Raiciu. In-Net: In-network processing for the masses. In *Proc. EuroSys*, 2015.

[74] A. Studer and A. Perrig. The Coremelt attack. In *Proc. ESORICS*, 2009.

[75] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *ACM SIGCOMM CCR*, 26(2):5–18, 1996.

[76] H. Wang, L. Xu, and G. Gu. FloodGuard: A DoS attack prevention extension in software-defined networks. In *Proc. DSN*, 2015.

[77] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker. Elastic scaling of stateful network functions. In *Proc. NSDI*, 2018.

[78] J. Xing, A. Morrison, and A. Chen. NetWarden: Mitigating network covert channels without performance loss. In *Proc. HotCloud*, 2019.

[79] P. Zave and J. Rexford. Network security. *http://www.cs.princeton.edu/courses/archive/fall18-cos561/papers/Security18.pdf*, 2018.

[80] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu. Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security*, 13(7):1838–1853, 2018.

[81] P. Zheng, T. Benson, and C. Hu. P4Visor: Lightweight virtualization and composition primitives for building and testing modular programs. In *Proc. CoNEXT*, 2018.