Extensible and Scalable Network Monitoring Using OpenSAFE

Jeffrey R. Ballard *ballard@cs.wisc.edu*

Ian Rae *ian@cs.wisc.edu*

Aditya Akella *akella@cs.wisc.edu*

Abstract

Administrators of today's networks are highly interested in monitoring traffic for purposes of collecting statistics, detecting intrusions, and providing forensic evidence. Unfortunately, network size and complexity can make this a daunting task. Aside from the problems in analyzing network traffic for this information—an extremely difficult task itself—a more fundamental problem exists: how to route the traffic for network analysis in a robust, high performance manner that does not impact normal network traffic.

Current solutions fail to address these problems in a manner that allows high performance and easy management. In this paper, we propose OpenSAFE, a system for enabling the arbitrary direction of traffic for security monitoring applications at line rates. Additionally, we describe ALARMS, a flow specification language that greatly simplifies management of network monitoring appliances. Finally, we describe a proof-of-concept implementation that we are currently undertaking to monitor traffic across our network.

1 Introduction

Contemporary computer networks have an interesting issue; at the same time the dependency we place upon these networks is increasing, the bandwidth is also increasing. With this, the speed of computer networking is out-pacing our ability to effectively monitor them for safety. Ideally network security is best applied with defense in layers: at the border, at the administrative boundary, and at the edge.

The last of these, the edge, is the only place where reasonable security solutions currently exist for most applications. End-hosts have a diverse array of appropriate solutions to solve their security problems. However, as end-hosts are also very complicated, they cannot be solely trusted for ensuring their safety.

Monitoring network traffic at administrative and border-level boundary ingress and egress points are the traditional mainline defenses. Security happens in two phases: active protection (firewalls, for example) and network monitoring.

In this work, we are looking at network monitoring. Typically, this monitoring is provided either via middleboxes placed directly on the network path or via inspection of copies of traffic at interesting points in the network.

On campus networks there are specific challenges to middleboxes—the largest of these is typically high fan-out. Middleboxes would be placed either at aggregation points—requiring a handful of extremely high-bandwidth appliances—or further down the fan-out—requiring a large number of normal-bandwidth appliances. As of February 2010, the University of Wisconsin—Madison College of Engineering has 22 $\times$ 1 Gbps connections and 2 $\times$ 10 Gbps connections to the University backbone routers. Clearly, in this environment, middleboxes at any level are infeasible due to cost.

Another issue confronting traditional middleboxes is network interruption. Adding or removing middleboxes requires changes to the physical path of the network. Altering the physical path of the network dramatically increases the difficulty of network maintenance.

Copy the Traffic

State of the art network monitoring uses *span ports* to create copies of traffic traffic at various *interesting* points in the network. Typically, interesting points are at administrative boundaries, either before or after a network firewall. This allows for complete copies of all network flows to be seen and inspected.

Span ports are typically directed into a single computer running some sort of IDS (Intrusion Detection System) such as Snort [7], as shown in figure 1. However, the challenge is that these network connections are typically 10 Gbps or more. To solve this problem, kludges like careful NIC driver manipulation and aggressive on-host queuing on the host need to be employed.

Another constricting factor is that the number of span ports on network equipment is often extremely limited. For example, the Cisco Catalyst 6000 series is limited to two span ports per device. Making it worse, enabling multicast on a Cisco FireWall Services Module (FWSM) consumes one of those two, leaving only one for monitoring.

Presenting even more difficulty, this 10 Gbps is a unidirectional span of a bidirectional 10 Gbps connection. While limited queuing appears to occur, ultimately the span merely needs 5 Gbps of bidirectional traffic to saturate the 10 Gbps span port. In other words, the span port will be twice as busy as it sees both directions of traffic.

Managing multiple devices on a span is a difficult task. Either all traffic must be forwarded to every device on the

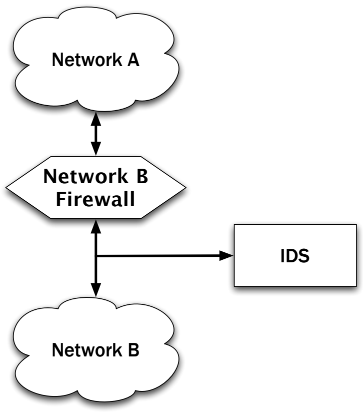

Figure 1: A typical configuration for network monitoring today.

span, or special care must be taken to direct subsets of traffic to different devices.[1] This makes it much more difficult for administrators to add new monitoring devices, which can be troublesome when more specialized appliances are acquired or when the load becomes too great for the current monitoring hardware.

We propose OpenSAFE (Open Security Auditing and Flow Examination), a unified system for network monitoring, to solve this problem. Leveraging open networking technology such as OpenFlow [6], OpenSAFE can direct spanned network traffic in arbitrary ways. OpenSAFE can handle any number of network inputs and manage the traffic in such a way that it can be used by many services while filtering packets at line rate.

OpenSAFE consists of three important components: a set of design abstractions for thinking about the flow of network traffic; ALARMS (A Language for Arbitrary Route Management for Security), a policy language for easily specifying and managing paths; and an OpenFlow component that implements the policy. We present the overall design of the system in section 2, describe ALARMS in section 3, and show the OpenFlow component of our system in section 4. Finally, we list related work in section 5, discuss future work in section 6, and conclude in section 7.

2 Overall Design

To make the management of routes for network monitoring both flexible and easy, OpenSAFE is designed around several simple primitives. We use the notion of *paths* as the basic abstraction of describing the selection of traffic and the route this particular traffic should take. Fundamentally, we wish to support the construction of paths that allow desired traffic to enter the system and be routed to one or more network monitoring systems.

[1]There are commercial products that also accomplish this—GigaMon [2] sells a line of switches that offer modular filtering, replication, and forwarding of network traffic to several monitoring devices. However, configuration of monitoring policy lacks an expressive, high level language; the primary interface is a web-based user interface for connecting ports and specifying filters.

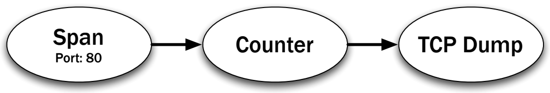

Figure 2: A basic monitoring path.

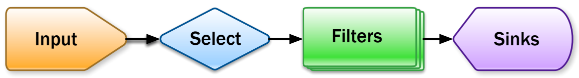

Figure 3: Abstractions used to describe monitoring paths.

A basic example of this is shown in figure 2, where HTTP traffic is routed through a counter appliance and finally to a TCP dump appliance. This could be easily implemented as a static route; however, our goal is to enable the construction of much more complex systems.

2.1 Path Abstractions

In OpenSAFE, the articulation of paths occurs incrementally along the desired route of the path. As shown in figure 3, paths are composed of several parts: *inputs*, *selections*, *filters*, and *sinks*.

At a high level, each path begins with an input, applies an optional selection criteria, routes matching traffic through zero or more filters, and ends in one or more sinks. Inputs can only produce traffic, sinks can only receive traffic, and filters must do both.

If we take figure 2 and view it in this way, it becomes figure 4. This shows traffic entering on a span port (input), being selected for port 80 (selection), routed through a counter (filter), and finally sent to a TCP dump (sink). A more complicated example involving more than one filter is shown in figure 5, demonstrating how paths can be extended.

The overall design of OpenSAFE is shown in figure 6. The input is a connection from the span path at the chosen network aggregation point to an OpenFlow switch port. Some number of filters are in use, attached to various OpenFlow switch ports. Finally, output is directed into some number of sinks.

2.2 Parallel Filters and Sinks

To monitor large networks at line rates, it is quite possible (and quite likely) that a single filter or sink will not be able to cope with all the network traffic. To address this problem, we allow traffic to be routed to multiple filters or sinks operating in parallel within a path. Figure 7 shows such a path, with HTTP traffic routed to multiple IDS appliances. We provide a variety of methods to distribute traffic between multiple filters or sinks, described in more detail in section 3.3.

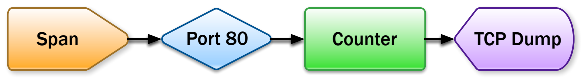

Figure 4: A basic logical monitoring path (figure 2) with coded abstractions.

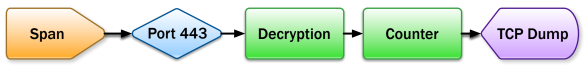

Figure 5: A logical monitoring path with multiple filters.

2.3 Waypoints

Our final abstraction is one that is added largely as a convenience to ease the creation of multiple semi-redundant paths. In a system of a reasonable size, it is possible—even probable—to have multiple paths configured with common attributes. For instance, suppose that an administrator wants to perform some degree of processing on traffic, then send the result to a specific filter and sink, as shown in figure 4 and figure 5. This quickly becomes a maintenance problem as modifying the common end-component of the paths may involve editing many different rules.

We alleviate this issue by introducing a new abstraction: *waypoints*. Waypoints serve as "virtual destinations," allowing administrators to aggregate policy rules and reduce repetition. A path using a waypoint is displayed in figure 8, where HTTP and HTTPS traffic is sent to a "web" waypoint before being passed to a counter filter and TCP dump sink.

3 ALARMS: A Language for Arbitrary Route Management for Security

To enable network administrators to easily manage and update their monitoring infrastructure, we introduce ALARMS, a language for arbitrary route management for security traffic. ALARMS utilizes the abstractions mentioned in section 2 to create a simple policy language syntax to describe paths. Paths are defined between *named components*, and each component may be subject to a *distribution rule* in the case of multiple, parallel components.

ALARMS is a high-level programming language and relies on a low-level programmatic interface to a network switch. We use OpenFlow [6] as the programmable switch technology. OpenFlow is an Ethernet switch with a programmatic interface to add and remove entries in its flow table from a centralized controller. By default, new flows that do not match existing entries in the flow table are sent to the OpenFlow controller. This way, the OpenFlow controller can manipulate the flow tables dynamically, based on network activity.

OpenFlow allows entries in the flow table to be based

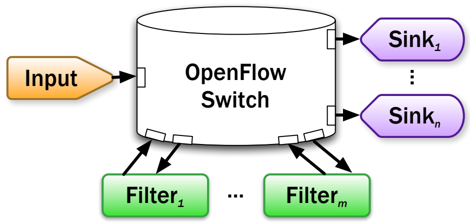

Figure 6: The overall design of OpenSAFE, using our abstractions.

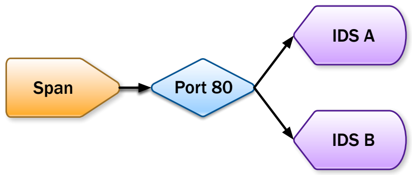

Figure 7: A monitoring path with parallel sinks.

upon up to ten entries in a packet (including source/destination IP address, source/destination port, and so forth). This is known as the OpenFlow *10-tuple*. While we have implemented ALARMS with OpenFlow, this language should be generic enough to handle any arbitrary programmable network layer.

In this section we will present some of the core concepts of ALARMS; details about the interaction between ALARMS and OpenFlow are described more fully in section 4.

3.1 Naming

In ALARMS, all components of a path are given unique types and names. Specifically, the policy file names the following components:

- OpenFlow switches (`switch`)
- Inputs and Sinks (`input` and `sink`)
- Filters (`filter`)
- Selections (`select`)
- Waypoints (`waypoint`)

We describe the language specification and features for each of these components below.

3.1.1 OpenFlow Switches

Each OpenFlow switch is given a unique name that corresponds to its OpenFlow datapath ID. This is accomplished using the `switch` statement:

```
switch of = 0x00000021;
```

Multiple switches may be defined, although it is assumed that each ALARMS-controlled switch is either directly connected to another ALARMS-controlled switch or connected through a number of ALARMS-controlled switches. Routes may be defined from a port

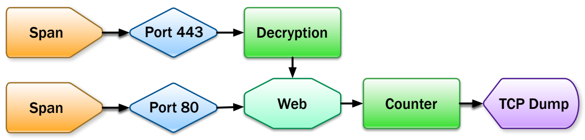

Figure 8: A logical monitoring path with a waypoint.

on any ALARMS-controlled switch to a port on any other ALARMS-controlled switch.

3.1.2 Inputs and Sinks

As shown in figure 3, paths begin with inputs and end with sinks. Inputs and sinks are simply named OpenFlow switch ports (as in figure 6), defined like so:

```
input span = of:0;
sink tcpdump = of:1;
```

Since inputs can only transmit traffic and sinks can only receive traffic, each named input or sink is restricted to a single port. Traffic, however, can be directed to multiple sinks—see section 3.3 below.

3.1.3 Filters

Filters are middleboxes within an OpenSAFE network. They are shown as the third item in figure 3.

A filter is a combination of a sink plus corresponding inputs. As such, filters are defined similarly to inputs and sinks, but with somewhat more flexibility as they are able to transmit and receive traffic. Each filter must define single `tofrom` switch port (to both receive and transmit) or both a `to` and a `from` port (to delegate receiving and transmitting, respectively, to separate ports). As with inputs and sinks, filters are named OpenFlow switch ports, and are specified in the policy language as follows:

```
filter to counter = of:2;
filter from counter = of:3;
```

Each port may be given its own unique name, or the same name may be used for both the `to` and `from` ports. Multiple `to`, `from`, and `tofrom` ports for a single name are not permitted.

3.1.4 Selections

Selections are named instances of the traditional OpenFlow 10-tuple, with some limited boolean syntax. Specifically, ALARMS permits syntax like:

```
select http = tp_src: 80 || tp_dst: 80;
```

This selection produces only unencrypted HTTP traffic. Any fields in the OpenFlow 10-tuple that are not specified in the selection are treated as wildcards.

3.1.5 Waypoints

As waypoints are not physical destinations, they only exist within the ALARMS language. Defining waypoints is very simple:

```
waypoint web;
```

This allows path rules to reference the waypoint as either a source or a destination.

3.2 Paths

Now that all named components have been specified, we can connect these components to form paths. The simplest form of a path connects an input directly to a sink:

```
span -> tcpdump;
```

This can be modified to include, for example, a filter and a selection:

```
span[http] -> counter -> tcpdump;
```

And the implementation of figure 8 involving a waypoint,

```
span[http] -> web;
span[https] -> decrypt -> web;
web -> counter -> tcpdump;
```

As OpenFlow has no concept of waypoints, ALARMS will unroll the waypoints to specify each path in its entirety to OpenFlow.

This provides almost all of the functionality we mentioned previously in section 2, with the exception of handling multiple filters or sinks, which we will address in the next section.

3.3 Distribution Rules

In order to enable features like load balancing, each portion of a path may be distributed between several components. The distribution of traffic between these components is handled by one of the following distribution rules, applied on a per-flow basis:

- *ALL* (duplicate)
- *RR* (Round Robin)
- *ANY* (random)
- *HASH* (apply a hash function)

These distribution rules are represented in the policy language like so:

```
span[http] -> {ALL, counter1, counter2} ->
tcpdump;
```

The first three rules are quite straightforward. The *ALL* rule sends incoming flows to all components in the list. The *RR* rule distributes flows to each component in a round-robin order. Finally, the *ANY* rule forwards flows to one of the components selected randomly.

The *HASH* rule is a special case. It takes an additional argument—the name of the hash function—and relies on this function to determine the destination. The hash function is provided with the first packet of the flow as well as the entire distribution list, and it is expected to return the component to which flows should be forwarded.

4 OpenFlow Observations

We created a prototype implementation of OpenSAFE in Python, using the OpenFlow reference software switch version 0.8.9 and NOX [3], an OpenFlow controller, version 0.6.

To preserve high performance, we attempt to precompute as many routes as possible and install them in the flow table of the OpenFlow switch on startup. This is done to prevent the controller from being overloaded with traffic—a distinct possibility when operating at high line rates—as well as to deal with potentially limited space in the flow table.

OpenFlow-enabled switches are hardware Ethernet switches that are able to use the OpenFlow protocol. As these are physical products, they often have limitations that vary between vendors and products. There are three distinct concerns when using OpenFlow-enabled switches: flow table exhaustion, matching ability, and insertion latency. The number of flow table entries is often very limited on OpenFlow-enabled switches, typically from 1500–3000 entries per line card. With these entries, often not all fields are able to be matched in hardware on the switch. Finally, when inserting a flow into the OpenFlow-enabled switch, if new flows need to be evaluated by the controller, it can take up to several hundred milliseconds for the initial packet from the flow to go to the controller and then for the controller to insert a new entry into the flow table.

As the OpenFlow version 0.8.9 specification does not have explicit hashing functions, latency can be an important issue. To emulate ALARMS rules such as `ANY`, `RR`, or `HASH`, flows need to be dynamically handled by the controller. This will result in a relatively long round-trip time to the controller for each hashing function along the path that a flow will take.

4.1 Waypoints

Since OpenFlow does not have the concept of waypoints, we recompute any flow containing a waypoint. By creating the cross product of the path rules that terminate and initiate from a waypoint, we create the representative set of OpenFlow rules. In this way, we are able to pre-install any routes involving waypoints that use only static rules. However, paths using dynamic distribution rules are more complicated and are described in section 3.3.

4.1.1 Virtual Sinks

Waypoints also serve another purpose in our implementation—that of "virtual sinks." Virtual sinks can apply any OpenFlow action to a particular flow.

One OpenFlow action of particular interest is `discard`. ALARMS implicitly creates a virtual discard sink. Traffic sent to this virtual sink is discarded immediately.

This allows for simple network flow sampling. For example, if a network administrator wished to drop 50% of the incoming traffic flows, he could install a `RR` distribution rule with two destinations, one of which would be the `discard` sink. Other fractions could be created by setting the `RR` distribution accordingly—for instance, one-third of the network traffic could be sampled by having a rule of {`RR, discard, discard, destination`}. By adding virtual sinks, this type of functionality is both easy and seamless; however, it is still subject to the performance concerns of distribution rules detailed in the next section.

4.2 Default Drop

OpenFlow defaults to sending all network traffic to the controller if it does not match an active flow. However, ALARMS starts with the premise that we are explicitly stating which traffic we would like to route. Therefore, any traffic that does not match an entry in the policy file is dropped.

We accomplish this by pre-installing a wildcard OpenFlow rule that drops traffic with low priority. Network administrators can still specify paths which act on all traffic in the policy file, and this will take priority. Also, this traffic could all be sent to the controller as part of a distribution rule. To achieve reasonable performance, we simply do not want traffic to be sent to the controller unless this behavior is explicitly desired.

5 Related Work

Casado et al. describe using Ethane [1] switches to enforce middlebox policies and propose a language, *Pol-Eth*, for describing these policies. However, their work on *Pol-Eth* is primarily designed around reachability and the idea that middleboxes would still be on the logical path of a flow (even if not explicitly on the physical path). Our work differs in that OpenSAFE implements a policy language, ALARMS, which handles a copy of the network traffic instead of on a middlebox inserted into the network. As such OpenSAFE does not handle end-to-end connectivity but rather a unidirectional flow.

Joseph et al. propose a similar architecture to Ethane in their work on policy-aware switching [5]. However, they do away with OpenFlow's concept of a centralized controller, instead relying on each switch to individually determine the next hop and forward packets immediately. This improves throughput, especially with large quantities of brief flows (where the overhead of contacting the controller is significant), but makes some aspects of network management more difficult, as no single entity has a complete view of the network. Additionally, the policy specification language described in their work is still centered around deciding appropriate paths for a flow, rather than a higher-level concept of what network monitoring needs to be applied.

A Flow-Based Security Language (FSL) [4] for expressing network policy has been suggested by Hinrichs et al. FSL, a variant of Datalog, allows specification of policies such as access controls, isolation, and communication paths. This specification is flexible and fast, capable of performing lookup and enforcement at high line rates. Again, however, the language is generally focused on end-to-end reachability and path selection, without specific thought to network monitoring.

6 Future Work

ALARMS

To broaden the capabilities of ALARMS, it should be investigated as to whether implementing dynamic control as a first-class primitive in the language would be useful. In the current version of ALARMS, feedback from filters and sinks is implemented purely through `HASH` distribution rules. While this is sufficient for some dynamic control capability, it could be beneficial to allow filters and sinks to explicitly modify paths by adding or removing path components. This would enable components to directly influence OpenSAFE—because of being overloaded, being taken offline, and so forth. In the future, we would like to explore this using a programmatic interface, such as XML-RPC, that allows filters and sinks to use all of the naming primitives to modify paths in the system.

OpenFlow

As we mentioned, when ALARMS uses constructions that are not available natively in the OpenFlow specification, OpenSAFE must send the flows to the OpenFlow controller to be processed. This potentially has a significant performance issue if too many flows are sent to the controller. First, until flows are entered into the flow table, packets will be sent to the controller, potentially overwhelming it. Second, OpenFlow-enabled switches typically process entries in the flow table in hardware, but process exceptions in software where performance is directly impacted.

We avoid these problems by attempting to carefully construct OpenFlow entries that minimize the number of flows that are sent to the controller. Additional study should be done in the area of pre-computing more dynamic distribution rules. It is possible that a particular hash function could be covered by a specific set of static OpenFlow rules; this is obviously not general to all hash functions, but it could be used to improve performance in some cases. Additionally, simple dynamic distribution rules that don't require any state, like `ANY`, could be added to the OpenFlow specification to reduce activity on the controller.

Deployment

Our test of OpenSAFE and ALARMS was entirely done using a software implemention of OpenFlow using virtual machines. Next we will be implementing OpenSAFE using a physical switch and machines. This will further illustrate the practical issues of using OpenFlow on a high-traffic link.

7 Conclusion

Network security monitoring in today's large-scale networks is a difficult task. We focused on the area of how to route traffic to monitoring appliances, rather than attempting to solve all parts of the problem, including how to analyze network traffic.

Current solutions for routing monitored traffic are expensive, difficult to manage, and have problems scaling to high line rates. OpenSAFE uses OpenFlow to scale to line rates by utilizing supporting hardware. Management is facilitated by ALARMS, our simple language for routing copies of network flows. Finally, the expense of the system is greatly reduced as OpenSAFE runs on commodity hardware, with the most exotic component, the OpenFlow-enabled switch, available from many vendors.

OpenSAFE makes monitoring large scale networks easier than ever before, and it has a rich area for future work.

Acknowledgments

This work is supported in part by a grant from the GENI Project Office ("Campus Trials of E-GENI"). We thank the anonymous INM/WREN reviewers whose comments helped improve this work.

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In J. Murai and K. Cho, editors, *SIGCOMM*, pages 1–12. ACM, 2007.

[2] GigaMon GigaVue Switch. `http://www.gigamon.com/gigavue-2404.php`.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[4] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker. Expressing and Enforcing Flow-Based Network Security Policies. Technical report, University of Chicago, 2008.

[5] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. In V. Bahl, D. Wetherall, S. Savage, and I. Stoica, editors, *SIGCOMM*, pages 51–62. ACM, 2008.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[7] M. Roesch. Snort—Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, page 238. USENIX Association, 1999.