Whole Home Proxies: Bringing Enterprise-Grade Security to Residential Networks

Curtis R. Taylor, Craig A. Shue and Mohamed E. Najd
Worcester Polytechnic Institute
{crtaylor, cshue, menajd}@cs.wpi.edu

ABSTRACT

While enterprise networks follow best practices and security measures, residential networks often lack these protections. Home networks have constrained resources and lack a dedicated IT staff that can secure and manage the network and systems. At the same time, homes must tackle the same challenges of securing heterogeneous devices when communicating to the Internet. In this work, we explore combining software-defined networking and proxies with commodity residential Internet routers. We evaluate a "whole home" proxy solution for the Skype video conferencing application to determine the viability of the approach in practice. We find that we are able to automatically detect when a device is about to use Skype and dynamically intercept all of the Skype communication and route it through a proxy while not disturbing unrelated network flows. Our approach works across multiple operating systems, form factors, and versions of Skype.

I. Introduction

While there are many more residential networks than enterprise networks, most network security measures are directed at enterprises. Residential networks have constrained resources, in hardware, connectivity, IT expertise, and funding, that limit the viability of most protective measures for these networks. While some security measures are straightforward in an enterprise network, they can be challenging to execute across residential networks.

One example class of applications is that of a "whole home" proxy solution that is tailored to specific applications. Enterprises often employ proxies to detect and block access to potentially malicious destinations or content. By employing this protection at the perimeter, enterprises can provide protection to many hosts at once. This goal is shared by residential networks. In particular, residential networks have numerous heterogeneous devices, including desktop and laptop computers, mobile devices, and embedded devices (e.g televisions, receivers, and video game consoles). Some devices, particularly for mobile or embedded devices, may not have options to allow users to configure proxy settings or other advanced networking features.

Residential users have limited options for a whole home proxy solution. Many commodity routers lack options for setting up proxy servers or site-to-site VPN end-points in their manufacturer-provided firmware. Even if users replace their routers with high-end devices or install custom after-market firmware (which can be daunting even for technical users [14]), the controls are too coarse grained. Many VPN setups allow the tunneling of all network or none at all. "Split tunnel" VPNs can allow partial rerouting of traffic, but those tunnels are created on a per-destination basis rather than on a per-flow basis. Finally, the complexity of managing these VPN tunnels may be cumbersome for users.

We propose to change the network model. Rather than require home users to become experts, we focus on outsourcing security management to expert service providers. We explore modifications to commodity residential routers to allow them to export management to a remote controller, using the OpenFlow protocol [18], and a series of device proxies. Unlike traditional OpenFlow, we will examine the payload of network traffic and use remote cloud nodes to protect residential users.

In exploring this concept, we focus on the Skype video conferencing application. Skype is commonly used, with over 300 million users worldwide [2], with support on devices ranging from computers to mobile devices and video game consoles. Skype uses a peer-to-peer connection between communicating parties which can reveal the IP address of a Skype user to others, whether they are aware of an established connection or not [17]. Some blackmarket providers offer to denial-of-service attack users when provided with a target Skype username since the Skype directory service leaks the IP addresses of connecting parties [4]. We believe Skype is a particularly good example application because it is known for breaking through common network barriers (like firewalls), uses a proprietary protocol that cannot be altered, and has complex infrastructure. Simply put, a technique that works for Skype will likely work for many simpler network applications.

We make the following contributions:

1) **Proxies on a per-flow basis:** We combine an OpenFlow approach with proxies and a tunneling agent on the router to proxy communication on a per-flow basis.
2) **Demonstration of a utility of an application-specific proxy:** We create an SDN controller application, agent at the router, and proxy configuration for Skype that demonstrates the viability of per-flow proxies that are tailored to applications in a residential network.
3) **Evaluation of the effectiveness of the approach:** We evaluate the approach using 5 different devices running Skype on a home network with a cloud-based OpenFlow controller and proxy.

II. Related Work

Our approach is related to work in detecting if a network flow is associated with the Skype protocol, to measures that influence Skype privacy, and to work in residential network innovation using software-defined networking techniques. We now describe each of these areas.

A. Distinguishing Skype Network Traffic

Skype has a complex peer-to-peer (P2P) infrastructure with supernodes (which are used for routing), ordinary nodes (such as end-user machines), and a login server [7]. Many researchers have tried to characterize and understand how the underlying Skype protocols work [7], [21], while others have focused on detecting Skype traffic in networks [8], [9], [19].

Since we do not decode Skype's proprietary protocol in this work, we must simply detect and proxy all messages associated with the Skype program to ensure the user's real IP address is not leaked. SkyTracer [26] has a similar goal of detecting Skype traffic at the flow-level. SkyTracer uses a mixture of flow tuple and byte-level packet characteristics to identify Skype traffic within the first few packets. While such approaches may work well for identifying ongoing or new Skype calls, we must be able to detect Skype activity *before* the associated network traffic leaves the network. We must proxy all communication to Skype servers, supernodes, and ordinary nodes to hide the user's IP address.

B. IP Address Privacy in Skype

Since Skype uses a P2P connection to directly establish a connection between communicating hosts, each host naturally learns the IP address of its communicating counterpart during a call. However, Le Blond *et al.* [17] describe a method to passively obtain the IP addresses of thousands of Skype users without alerting the user. They further describe linking a user's IP address to other Internet activity such as BitTorrent traffic. While Le Blond proposes infrastructure changes, their approach does not completely address the issue. The Skype client (SC) application could simply only allow added contacts to establish direct connections; however, this is only enabled on the iPhone and not any of the other devices we tested. The Xbox One likewise only allows immediate contacts to connect, but it does so without determining whether a connection is direct or not. These features could easily be undermined with a social engineering attack in which the attacker is added as a contact.

In other work, Ehlert *et al.* [11] found that even when manually configuring Skype to use a proxy server in the client's settings, Skype will still try to establish a direct connection with the peer and will only use the proxy as a last resort if the earlier efforts fail. As a result, users may believe they are masking their actual IP addresses behind a proxy only to have Skype bypass the proxy.

C. Residential Network Innovations

Feamster [12] noted that residential networks are well-known for being insecure and hard to manage. In a position paper, he proposed outsourcing home network security to a third party. Yiakoumis *et al.* [25] suggested "slicing" home networks to allow independent third-parties to manage individual slices. In later work, Yiakoumis *et al.* [24] took a different approach to empower users to control the network. In that work, they create an agent, called *Skype+*, to implement quality of service for Skype. Kim *et al.* [15] implemented an extension to the Project BISMark suite [23] that enforces data caps on residential networks. Kumar *et al.* [16] explored an approach where residential users adjust their network experience by manipulating ISP-owned OpenFlow switches and controllers.

While these prior efforts have focused on outsourcing network management, they do not address significant security concerns. In particular, they have not considered approaches to outsource security controls in an incrementally deployable way, nor approaches that allow users to be self-sufficient in doing so. In this work, we instead focus on using SDNs to create an immediately deployable solution for specific applications. By sharing these applications and tools, we demonstrate that experts can create and share security tools with less technologically sophisticated users.

III. Approach: Tailored Proxying

A user may run many programs, each with their own workflow and associated security concerns and goals. To ensure these security goals are met, we enable security experts to write tailored control applications to manage the network traffic of the user's applications. We then create a general platform and an API that allows those experts to run their control application across many different types of residential networks.

Our general platform consists of four components: a commodity residential router running custom firmware, a cloud-based OpenFlow controller that directs the router's behavior, a cloud-based proxy/middlebox that monitors traffic, and a GRE tunnel between the router and proxy. These components are common across applications and services. To tailor the system to a particular user program, a security expert will create a custom application on the OpenFlow controller to manage the features. Further, the expert may run custom software on the proxy/middlebox to enforce these goals.

We instantiate this general approach with a specific application for the Skype video conferencing application. We now describe each of the components in the general platform and the customizations needed to meet Skype's security goals.

A. Platform: Router, Controller, Proxy

We modify a consumer-grade router to support the OpenFlow protocol by installing the OpenWrt's [5] firmware and enabling the Open vSwitch [20] module. Unlike prior work, we control the router remotely with an OpenFlow controller that is hosted at a cloud provider. This controller has the ability to vet all of the new connections established through the router, including traffic within the LAN and Internet traffic. The router establishes a connection to the controller upon boot and requests instruction for new network flows.

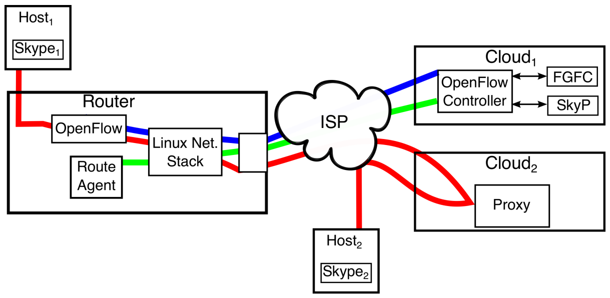

Fig. 1. Overview of how our Skype proxy approach works using multiple cloud providers for controlling OpenFlow and proxying traffic. Our controller uses fine-grained flow control (FGFC) and the Skype Proxy (SkyP) module to detect Skype calls and update the route agent to send traffic through the proxy.

We then create a cloud virtual machine that operates as a middlebox or proxy server. In its most basic form, the proxy simply uses network address translation (NAT) and forwards packets from the consumer's router to the requested destination and vice versa. To facilitate communication with the proxy, the router creates a GRE to a list of eligible proxies upon booting. When ordered to do so by the OpenFlow controller, the router simply uses the appropriate GRE tunnel as the destination for selected flows, causing them to be sent via the proxy.

B. Tailored Control: The SkyP Module

While the basic platform provides a mechanism to send arbitrary traffic via a cloud-based proxy, there must be a module or application that indicates which traffic should be sent to the proxy and what the proxy should do with the traffic once it receives it. This module may be different for each type of application protocol to provide tailored control.

For Skype traffic, we create a custom controller application, which we call the *SkyP Module*. This module uses Skype network characteristics to detect what traffic is likely associated with Skype and directs that traffic via the proxy. Since Skype is a complex proprietary protocol, we do not know which messages are used to register the user's IP address in the Skype directory. To prevent the user's real address from being leaked, our SkyP Module must take a series of steps to determine what traffic is Skype-related.

There are two main features the SkyP module must consider: 1) communication with known Skype infrastructure or 2) direct P2P communication. We use DNS features and IP ownership to identify and proxy the connections to the Skype infrastructure. However, for P2P communication, we leverage the fact that the client initiates its P2P connections using a randomly-generated port number that is created upon installation of the client. Using a few approaches, we can learn the client's P2P source port. Once we have done so, we watch for any peers the client contacts using the the P2P source port and proxy all traffic to those discovered peers (since traffic subsequent to the rendezvous may communicate using random ports). We now describe each of these approaches and features in detail.

1) Skype DNS Requests: When the Skype client (SC) first starts, it initiates a series of DNS requests to hardcoded domain names that are included as part of the Skype executable. Some DNS host names, such as `ui.skype.com`, are fixed while others, such as `dns13.d.skype.net`, appear to be members of a load balancing group that the SC may rotate amongst. To create a complete list of DNS host names associated with Skype, we examined a diverse set of devices and operating systems as shown in Table I.

TABLE I
LIST OF DEVICES USED IN OUR EXPERIMENTS

Device	Operating System	SC Version
iPhone	iOS 8.4	6.1.0.210
Macbook Pro	OS X 10.10.5	7.10
Dell Laptop	Windows 7	7.10.0.101
Dell Laptop	Ubuntu 14.04.3	4.3.0.37
Xbox One	Xbox OS 6.2.13332.0	1.9.0.1003

For each device, we launch the SC, initiate a roughly five second long voice call, and close the SC. We repeated this process 20 times for each application, flushing the device's DNS cache. Each client was behind a NAT device since Skype is known to exhibit different behave when operating behind NAT [7].

From these trials, we created a list of 32 host names that appeared to be related to Skype. Of the 32 host, 6 had distinct patterns that could be generalized into a regular expression. For example, there are 18 different host names that match the pattern `dsn[0-17].d.skype.net` [21], allowing us to easily construct a regular expression to match the hosts. We combine our empirically discovered addresses with important host names discovered in prior work [21].

We configured the SkyP module to monitor DNS requests for these host names. Since the SkyP module can receive all packets elevated to the OpenFlow controller, including DNS packets, it can analyze these requests and their responses. Each time a client initiates a DNS request, OpenFlow controller sends a copy of the DNS response to the SkyP module. The SkyP module searches the DNS response for replies containing any of these known host names. If an entry is found, the module extracts the all of the IP addresses and directs the router to send all traffic to those IP addresses via the proxy.

2) Skype's Use of NAT-PMP: When the SC is first installed, it randomly generates a port number that it will use when it later attempts to create P2P connections [7]. To facilitate communication even through NAT middleboxes, the SC uses the NAT Port Mapping Protocol (NAT-PMP) [10] to request that certain ports be mapped to the SC via the NAT device's public IP address. By simply monitoring for these NAT-PMP requests, which are elevated to the SkyP module, we can learn what port the SC uses for P2P connections and subsequently direct any traffic originating from the host using that port to be sent via the proxy. Since other unrelated applications could also initiate NAT-PMP requests, we only learn ports from NAT-PMP if they are within a delta of 4 seconds of a SC-related DNS request. This approach was effective for each of the devices in Table I across 80 call sessions, excluding the Xbox One (which does not use NAT-PMP).

3) Skype's Interactions with Supernodes: Since some devices, such as the Xbox One, do not use NAT-PMP, we use another SC characteristic to learn the SC's P2P port. When started, the SC makes multiple connections to supernodes. The first such connection uses the SC's dedicated port. Accordingly, by knowing the identity of all supernodes, or features associated with those supernodes, we can watch for any connections to those supernodes to learn the SC's P2P port. Prior work found that connections to supernode IP addresses typically use the port range 40001-40047 [21]. Further, all supernodes are now operated by Microsoft [1] [3], so we can examine whether the destination IP address belongs to Microsoft-owned IP space to determine if the connection is to a supernode.

IV. Implementation

We implement our approach using a consumer-grade router and elevating flows to a remote OpenFlow controller on a server in a cloud data center. We flash a TP-LINK TL-WR1043ND v2 router with a custom build of the OpenWrt (Chaos Calmer 15.05) image. To enable OpenFlow support, we selected the kernel-level Open vSwitch package.

To ensure continued operation in the event of connectivity issues when reaching the cloud controller, we ran NAT, a recursive DNS resolver, and DHCP services locally along with OpenFlow. We had to create a virtual interface to act as an intermediary between the router's WAN interface and the router's internal LAN. To enable NAT functionality, we created static rules in `iptables` for masquerading. We did not have to make any special changes for the DHCP or DNS services. When used for production, we will conceal these complex routing configurations by including them inside the firmware.

We then created two cloud virtual machines (VMs) to host the OpenFlow controller and anonymizing proxy. Each VM was an Ubuntu 14.04 Linux server micro-VM instance in the Amazon EC2 compute cloud and was eligible for Amazon's free tier. Each VM has a single 2.5 GHz core with 1 GByte of RAM and uses a dynamic global IP address. We ran a script to install and launch the POX OpenFlow controller with our own fine-grain flow control and SkyP modules.

The anonymizing proxy is configured to implement a source NAT using `iptables`. With this configuration, the proxy automatically translates and forwards traffic to and from the GRE tunnel connected to the home router. It only performs network-layer translations, so the Skype P2P port will be exposed in network communication. We did not explore performing port address translation at the proxy.

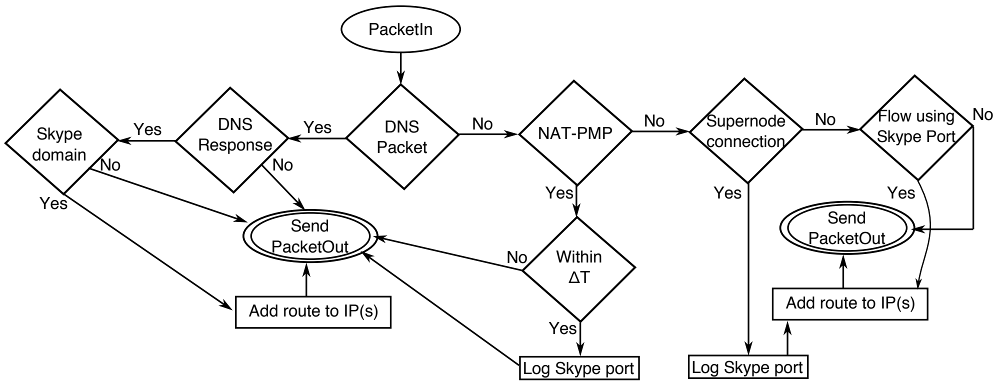

Fig. 2. The above diagram shows the decision-making process in the SkyP module for proxying traffic.

V. Evaluating SkyP

We evaluated our approach by performing Skype voice calls and verifying functionality using third-party Skype IP address lookup applications, such as Skype Resolver [4], and via Wireshark captures. The specific devices and software versions are listed in Table I.

A. Evaluation Setup

Our evaluation setup is shown in Figure 3. We position the client using our whole home proxy in a residential network behind NAT. In our evaluation, the call initiator and responder are already contacts.

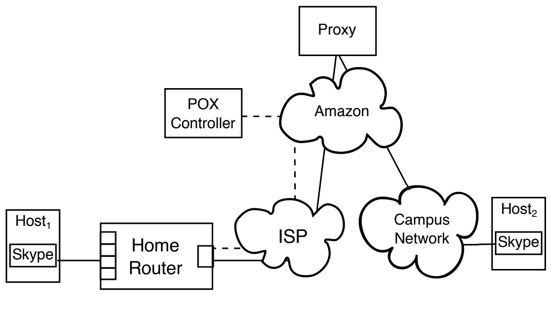

Fig. 3. Our evaluation setup for testing SkyP on different devices.

For each device except the Xbox One, we performed the following steps. First, the SC using SkyP (Host$_1$ in Figure 3) attempted a voice call to Host$_2$. After establishing the call, Host$_1$ sent a chat message, transmitted an image file 3 MBytes in size and ended the call after approximately 2 minutes. We then had Host$_2$ attempt a voice call to Host$_1$ to ensure the proper IP address was used to establish the P2P connection through the proxy. Since the Xbox One's version of Skype does include chat or file transfer, we only tested VoIP calls on it.

B. Verification

We verified our approach works in two ways. First, we used an online third-party tool, Skype Resolver [4], to ensure the only IP address associated with our username was the IP address of the proxy. Because hosts can be associated with multiple IP addresses at once, such as a mobile device and an office computer, we waited until no IP addresses were cached.

We performed packet captures at each host to verify correct proxying. For each device being tested, we verified each packet capture individually to ensure our IP address was never leaked to Host$_2$. We observed that all VoIP call, chat, and file transmission traffic established connections to Host$_2$ using our cloud proxy or were transmitted via an anonymizing supernode (for chat and file transmission). The Skype Resolver only learned the proxied IP address; it was never able to detect the real IP address of the proxied user.

VI. Discussion and Future Work

Performance and security are two important considerations for deploying systems that leverage SDN and the cloud. Existing research focuses on the OpenFlow protocol and relevant performance considerations such as the controller placement problem [13]. In our experimentation, there was no noticeable delay in call setup or call quality. Other research has considered performance of proxies and proxies within the cloud [6]. OpenFlow controller security [22] is also an important area of research that lies outside the scope of our work.

In evaluating our Skype setup, we found an interesting edge case. When the two communicating parties are not already contacts in the Skype system, a direct connection can occur if the adversary uses an unrestricted publicly routable address. In this case, the adversary sends a request through the Skype supernode to the internal host. That request causes the internal host to directly connect to the adversary. This particular workflow bypasses DNS, NAT-PMP, and supernodes and thus we do not proxy the connection correctly.

This approach, of requesting the other party to initiate the connection, is particularly useful for Skype to bypass NAT. Since one of the machines uses a publicly routable address, it can act as a server to have the other machine connect. By sending a request to this effect via the Skype supernode, the machines establish a connection.

Since the Skype protocol is encrypted, we cannot detect the IP address for these new requests and simply proxy all connections to that IP address. However, the connection request packet appears to use a packet size in the range of [329-339] bytes. As a workaround, we add a function not shown in Figure 2 to proxy any new network flows that occur within 200 milliseconds of these requests. As such, the requirement of peers being pre-existing contacts is no longer necessary.

VII. Conclusion

We proposed an approach that uses network function virtualization to enable a "whole home" proxy for residential networks. Using a cloud-based controller and proxy, we are able to control traffic on a per-flow basis that is immediately deployable. Using Skype as a motivating example, we found that even a complicated proprietary protocol can be singled out and selectively proxied. In doing so, we have highlighted the potential and discussed other applications for application-specific cloud-based proxies in residential networks.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1422180. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Skype at 10: How an Estonian startup transformed itself (and the world). https://www.microsoft.com/en-us/stories/skype/skype-chapter-4-are-you-smoking.aspx. Accessed: 2015-09-13.

[2] Skype audience stats - Microsoft advertising. https://advertising.microsoft.com/en-us/WWDocs/User/display/cl/brand_subproperty/1589/global/Skype-Audience-Stats.pdf. Accessed: 2015-10-09.

[3] Skype ditched peer-to-peer supernodes for scalability, not surveillance. http://www.zdnet.com/article/skype-ditched-peer-to-peer-supernodes-for-scalability-not-surveillance/. Accessed: 2015-09-13.

[4] Skype resolver. http://mostwantedhf.info/. Accessed: 2015-09-13.

[5] OpenWrt wireless freedom. https://openwrt.org, 2014.

[6] J. Almeida and P. Cao. Measuring proxy performance with the wisconsin proxy benchmark. *Computer Networks And ISDN Systems*, 30(22):2179–2192, 1998.

[7] S. A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *arXiv preprint cs/0412017*, 2004.

[8] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi. Detailed analysis of skype traffic. *Multimedia, IEEE Transactions on*, 11(1):117–127, 2009.

[9] P. A. Branch, A. Heyde, and G. J. Armitage. Rapid identification of skype traffic flows. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, pages 91–96. ACM, 2009.

[10] S. Cheshire and M. Krochmal. NAT Port Mapping Protocol (NAT-PMP). RFC 6886 (Informational), Apr. 2013.

[11] S. Ehlert, S. Petgang, T. Magedanz, and D. Sisalem. Analysis and signature of skype voip session traffic. *4th IASTED International*, 2006.

[12] N. Feamster. Outsourcing home network security. In *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, pages 37–42. ACM, 2010.

[13] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM, 2012.

[14] M. Horowitz. A router firmware update goes bad. http://www.computerworld.com/article/2692514/a-router-firmware-update-goes-bad.html, October 2014.

[15] H. Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, 2013.

[16] H. Kumar, H. H. Gharakheili, and V. Sivaraman. User control of quality of experience in home networks using sdn. In *Advanced Networks and Telecommuncations Systems (ANTS), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[17] S. Le Blond, C. Zhang, A. Legout, K. Ross, and W. Dabbous. I know where you are and what you are sharing: exploiting p2p communications to invade users' privacy. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 45–60. ACM, 2011.

[18] 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.

[19] M. Perényi, A. Gefferth, T. D. Dang, and S. Molnár. Skype traffic identification. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pages 399–404. IEEE, 2007.

[20] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[21] P. Qi, C. Du, Y. Ren, and Y. Xue. The secrets of skype login. 2013.

[22] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 78–89. ACM, 2014.

[23] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato. Bismark: a testbed for deploying measurements and applications in broadband access networks. In *2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC 14)*, pages 383–394, 2014.

[24] Y. Yiakoumis, S. Katti, T.-Y. Huang, N. McKeown, K.-K. Yap, and R. Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1114–1119. ACM, 2012.

[25] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, pages 1–6. ACM, 2011.

[26] Z. Yuan, C. Du, X. Chen, D. Wang, and Y. Xue. Skytracer: Towards fine-grained identification for skype traffic via sequence signatures. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 1–5. IEEE, 2014.