Residential Network Security:

Using Software-defined Networking to Inspect and Label Traffic

by

Yu Liu

A Ph.D. Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

by

December 2021

APPROVED:

Prof. Craig A.Shue, Dissertation Advisor

Prof. Mark L. Claypool, Committee Member

Prof. Tian Guo, Committee Member

Dr. Kerry A. McKay, External Committee Member

Abstract

With more households subscribing to Internet services, and the growth of the Smart Home paradigm and the Internet of Things (IoT), there are more assets that need protection in home networks. However, many manufacturers produce devices with weak security. From hardware to software, these devices have attack surfaces that can be easily utilized by attackers. Attacks may cause sensitive information exposure and even physical device damage. Further, compromised devices can join a botnet to launch attacks on other targets, such as distributed denial-of-service attacks (DDoS). Those attacks, under today's Internet infrastructure, are hard to effectively track and block from the victim side, and they also harm residents' service at the origin. Finally, home network owners usually lack expert computer security knowledge. With the limited security features provided by home network gateways, home network security tools are unable to provide adequate protection.

Prior work proposes to increase home network security by outsourcing management to the cloud, where experts can help. With the assistance of software-defined networking (SDN) technology, enterprise-grade security benefits can be achieved in home networks. This dissertation focuses on utilizing SDN and network function virtualization (NFV) techniques to achieve such benefits. To optimize the deployability, we try to minimize modifications to legacy network systems, the Internet backbone and Internet Service Providers (ISPs).

Along with prior works, we continue using SDN for home network security. This dissertation focus on traffic inspection and labeling. These methods can secure home network security from two angles: inside the home (the individual network) and outside (the collaboration of home networks in the Internet environment). Looking at the inner

home network, we propose a series of methods to enhance security for devices and their communications. First, we provide additional authentication between the controlling devices and IoT devices. Since the SDN controller is able to inspect inside networks, we show that vetting simple IoT device protocols can be used as a traffic filtering technique.

Mobile devices such as smartphones, tablets and laptops are common in a home network. To better secure this part of the home network, we explore host-based SDN architectures to achieve fine-grained network management on these devices. With system-level information infused networking data provided to the controller, the controller can understand the network better.

Thinking individual home networks as a part of the Internet, we explore methods to evolve home networks into responsible and collaborative traffic origins. We provide the basis for effective authentication and control in modern network environments. Since SDN controllers are able to see into the NAT of home networks, we use this feature to insert identifiers into network flows, based on the destination, to provide device-level and flow-level access control. These identifiers could be unique tokens or cryptographic cookies. Since home network devices can be compromised, and then export unwanted traffic to other networks to launch DDoS attacks, we design a protocol that uses the identifiers in the flows to allow the home network controller to stop the attacks upon receiving a trusted filter request from the victim. This method reduces the defense burden on the victim's end and effectively stops the DDoS attack.

Further, this flow identifier that is inserted by the SDN controller can provide information that is only known by the user and enterprise network. The enterprise network can then utilize this information to perform better access control. An account lockout attack is a concern of home network users who need to access enterprise networks remotely. Attackers may abuse the account lockout mechanism to deliberately lock accounts and launch denial-of-service attacks. With our mechanism, when an account is under attack, we verify the inserted flow identifier, and the enterprise network can unlock the account to ensure the largest resource accessibility for this trusted user. Our flow control method outperforms VPN since we provide lightweight computation

and flexible flow control, while keeping a similar level of security.

Table of Contents

Chapter 1

Introduction

With the development of information technologies, more home networks are connected to the Internet. Resources managed by home networks are taking a more significant role in Internet security. Smart home systems, for example, cause the inclusion of more Internet of Things (IoT) devices in the network. Their various functionalities form a large pool of computation resources and collect comprehensive data from users. Thus, converting those resources from security risks to security assets is not only beneficial to home network users, but also necessary for hygiene on the Internet as a whole. This dissertation mainly focuses on two techniques, traffic inspection and traffic labeling, to transform home networks from neglected spaces where attackers thrive to well-monitored areas that can help identify budding attacks. We will make these networks trustworthy and accountable, both for users within the networks and for remote Internet users.

Increased Internet service can improve people's lives. People rely on Internet service for communication, information, and online shopping. As a result, around 82% of the U.S households are subscribed to Internet services [141]. With the development of Wi-Fi technology, more than 442 million Smart Home devices are projected to be connected to the Internet in 2021 [10]. Those devices range from smartphones, laptops and tablets to smart cameras and smart electrical switches. These devices, which provide their services while continuing to collect data from users, are attractive to attackers.

Unfortunately, unlike enterprise networks, home networks usually lack protection. Most home network users have little network knowledge or awareness of security. For example, users may forget to update default passwords after installing IoT devices, or they may pick simple passwords. Users may neglect to upgrade firmware or to obtain patches for vulnerabilities. The lack of security awareness also impedes users from spending money to deploy security technologies. Insufficient hardware in home networks offers limited security. Home networks are also unlikely to deploy firewalls as effectively as an enterprise. Although most consumer-grade routers provide Network Address Translation (NAT) technology, which can hide devices behind a single network address, it is insufficient to defend current networks from attacks.

When home networks lack protection, home users are under threat and have the potential to contaminate others on the Internet. Attackers may get opportunities to infiltrate home networks using various attacks [150], and then they identify and infect vulnerable devices. The compromised devices can breach user privacy by collecting users' data. They can learn the protocols used by IoT devices and prevent use of those devices. For example, the communication between a paired smartphone and IoT device may not employ additional authentication and encryption. Attackers could thus observe and learn the communications. Further, compromised devices may execute malicious scripts to infect other devices in the same home network. Attackers can operate these devices to join a botnet and launch an attack on other network targets. The increasing number of devices and networks can make the combined attack volume exceed 1 Tbps, which is hard for a destination network to block. Attack traffic can also deplete the upstream bandwidth of the home network owner. In another sense, since regular work-at-home has grown 173% since 2005, compromised home devices may infiltrate into well-protected enterprise networks [91].

To solve these challenges, this dissertation contributes in two directions. First, we try to secure devices and their communication inside a home network to ensure a smart home safely serves the network owner. Second, since home networks generate personal data for various applications, we provide more precise flow identification technologies to help achieve fine-grained flow control from the origin of the traffic. This technique

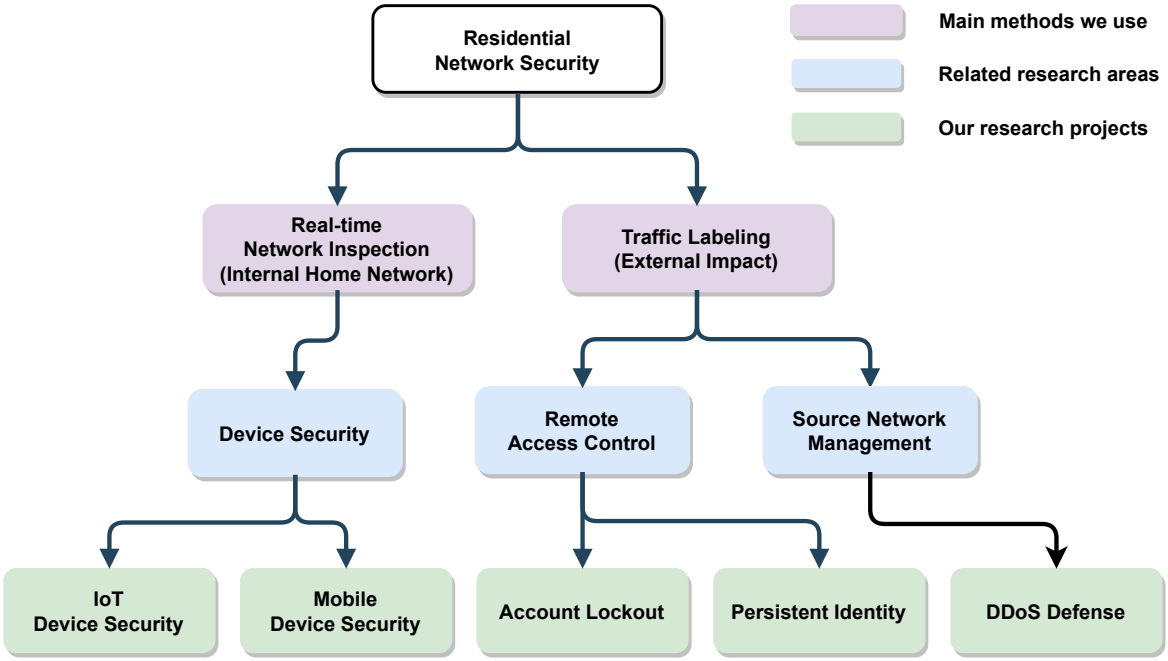

Figure 1.1: A tree indicates the research scopes, associated with our research projects.

intends to improve the accountability of home networks by helping remote Internet users to recognize and manage the network traffic. Each of our contributions follows certain principles:

- We automate our methods with minimal human intervention.
- Our methods are backward-compatibity for legacy systems or require minimal effort to incorporate.
- We can support some abandoned technologies with our platform.
- Rather than relying on Internet Service Providers (ISPs) or Internet backbone infrastructure, which are not incentivized to implement state-of-art technologies, we focus on platforms that incentivize users and third-party service providers to deploy new technologies.

These metrics provides a basis to further clarify the boundary of this dissertation. First, we focus on network-based techniques to secure home network environment. Research fields like hardware analysis and software vulnerabilities can also help, but

they are reserved for reverse engineer and embedded devices researchers. Second, since enterprise network security has been well-studied before, we consider to adapt those existing technologies to home networks, such as real-time traffic inspection for intrusion detection. However, home networks usually have constraints on users' knowledge and computation resources. Thus, while adapting enterprise network security techniques to home networks, this dissertation further reduces the scope – we focus on techniques that achieve the largest automation and minimal additional costs, as mentioned above. Taking a counter-example, we do not consider methods like deploying firewall on a separate Raspberry Pi in a home network. This method requires users' efforts to maintain firewall rules, and it needs additional costs on extra devices. With these considerations, Figure 1.1 visualizes the scope of this dissertation by a tree structure, with that how our research topics fit in. More specific descriptions about our techniques are as follow:

- **Outsourcing Security Functionality:** Our methods outsource network functionalities to the cloud. The cloud offers rich computational resources and security tools that are designed and maintained by experts.

- **Home Network Device Protection:** We identify and solve common vulnerabilities from IoT devices, such as lack of encryption and authentication. To enhance device protection in home networks, we consider bringing host-based SDN technologies to Android devices, which are common mobile devices in home networks.

- **Home Networks as Responsible Origins:** Since attacks may originate in home networks, we investigate methods to create network monitors at origins. We configure origins to support the destination networks by providing information for authentication and filtering unwanted traffic upon request.

This dissertation is organized as follows: Chapter 2 introduces fundamental technologies we discuss throughout the dissertation. Chapter 3 explores how IoT devices can be vulnerable to an attack inside a home network and how SDN and real time

traffic vetting can enforce either authentication or traffic inspection for those simple devices. Chapter 4 presents a protocol to mark home network flows at the source to stop DDoS traffic at the attacks origins. Chapter 5 measures the potential threat of account lockout and uses a traffic labling technique to help enterprises secure their network portals. Chapter 6 considers the development of Carrier-Grade NAT (CGN) and 5G networks, further uses a lightweight traffic labeling technique at the source, while providing the same level of security with virtual private network (VPN). Chapter 7 proposes to implement SDN on Android devices, which largely appear in a home network, for more robust home network analysis and control.

Chapter 2

Background and Related Work

This chapter provides background knowledge for this dissertation. We introduce the security challenges faced by home networks and prior work discussing these problems. We then introduce software-defined networking technologies that we use in our countermeasures, and network performance metrics that are needed for understanding the system evaluations in this dissertation.

2.1 Residential Network Security Challenges

More households are connecting to the Internet. A recent survey shows that 82% of households subscribe to Internet services and 89% of households have multiple computers at home [141]. More than 76% percent [10] of US home networks include a wireless router to share network service. Although each home network has limited network resources, by 2020, there could be 442 million home devices connected to the Internet [10]. These devices could be vulnerable to certain attacks. Zhang et al. [185] explored the vulnerabilities and privacy issues of IoT devices and the challenges to fix them. Barbar et al. [14] investigated the security model of IoT devices and their security threats. Lolias et al. [82] explored how a compromised device can join a botnet, like Mirai, to launch attacks on other targets. The security of a home network is usually poorly maintained, since home network users often lack security awareness. For example, they may forget to reset the default password after installing a Smart Home

device or they may choose a weak password. Unlike enterprise networks, which are protected with powerful firewalls and strict security policies for employees, home network gateways provide only simple NAT service to hide devices, which is insufficient to defend against network attacks. Moreover, the limited resources of a home network make it difficulty to bring powerful analysis to home networks. Deploying strong firewalls in home networks requires additional hardware costs, thereby decreasing the deployability of these technologies.

2.2 Distributed Denial-of-Service (DDoS) Attacks

DDoS attacks, in which attackers exhaust a victim's resources through a large volume of requests, are a common and on-going phenomenon [134]. Botnets of compromised machines can be large, encompassing millions of machines [23]. A recent botnet, Mirai, is designed to run on compromised Internet of Things (IoT) devices [5] and continues to evolve [134].

Most denial-of-service (DoS) availability attacks target a bottleneck resource and overwhelm it to prevent legitimate user access. Network-based flooding attacks, for example, attempt to saturate the bottleneck bandwidth between the Internet and a targeted victim. Application-layer DoS attacks exploit a bottleneck in the host software to deny access. Moore et al. [112] describe an application-layer threat between an HTTP sever and a backend database resource. The account lockout attack is a variant of an application-level availability attack [110].

The research community has explored methods to detect and mitigate DDoS attacks. Prior detection efforts have included statistical and mathematical methods to distinguish DDoS traffic from benign traffic [45, 74, 116], an analysis of network distance between the destination and origin [182], the application of deep learning [119], and even entropy comparisons of flows [32]. Some approaches have looked at ISP-wide data to detect botnets [63], though botnets typically operate globally across ISPs [171].

Once an attack is detected, defenders can try to mitigate it. Some endpoint filtering approaches have used IP history and reputation to determine malicious and legitimate

senders [125, 128]. However, with IP rotation and sharing in DHCP and NAT environments, IP history and reputation may have limited value. Other endpoint solutions include the work by Buragohain et al. [21], which performs flow-modeling on a SDN controller, and the work by Rebecchi et al. [136], which performs stateful anomaly detection on traffic at a router. A common limitation for these approaches is filtering at the endpoint (or at network devices in the same LAN as the endpoint), which is ineffective when the saturated bottleneck network link is between the LAN and its ISP. Such "last mile" bottleneck links are common on the Internet.

The industry also provides methods to protect its customers from DDoS attacks. However, as content providers, the industry can manage a limited area of the network infrastructure. For example, Google Cloud, Amazon, Cloudflare provide DDoS protection on the edges of their Autonomous System (AS) [4, 28, 94]. Srinivasan et al. [149] summarised three mainly used methods to defend Cloud services from DDoS attacks: resource scaling, victim migration, and software-defined networking. These methods are deployed at the victim's end, and are usually expensive. These service providers are considered as victims of DDoS attacks in this dissertation.

Other approaches have sought to employ packet filtering before the traffic reaches the victim's bottleneck link. Weniger et al. [176] propose a "moving target" mechanism in which ISP customers have multiple addresses and can unsubscribe from an address upon detecting an attack. This allows the upstream ISP to filter traffic to an unsubscribed address. A survey of ISPs performed by Steinberger et al. [153] shows that the ISPs have limited methods to defend DDoS, and the ISPs have limited collaboration with each other, which is important for DDoS defense. Mahajan et al. [96] propose a method to send filtering requests to upstream routers. Likewise, in their work on AITF, Argyraki et al. [7] designed a filtering protocol for routers at both the victim and attacker networks to cooperatively filter traffic. These approaches have powerful filtering capabilities, but they require cooperation from a set of ISP routers, ideally some at high-traffic peering points. The reliance on ISP cooperation and lack of incentives for ISPs to deploy this approach have resulted in little adoption of these techniques. This dissertation aims at shifting the filtering load to SDN controllers

operated by any service providers that have incentives to report and filter malicious traffic (Chapter 5).

2.3 Software-defined Networking (SDN)

This section provides background on software-defined networking (SDN), one of the main technologies applied in this dissertation. Figure 2.1 shows a basic software-defined networking paradigm. Unlike traditional packet processing, SDN separates the packet forwarding hardware, the data plane, and the control plane, which decides how a packet should be processed. Such separation brings certain benefits to a network:

- **Convenient deployment of new network functionality:** Rather than upgrading a hardware switch, SDN abstracts these functions into software modules that a network operator can easily deploy.
- **Dynamic, flexible network management:** Each module in the management plane can obtain network status in real-time. After analysis, a module can send commands to manage the network based on the result.
- **Centralized view of connected networks:** The management plane can connect to the control planes of more networks.

OpenFlow [177] is a standardized protocol for SDN to coordinate these different components. OpenFlow defines a framework of network functionalities that a management plane can apply to a connected network and its communication principles. An SDN-enabled network must deploy an OpenFlow agent module in the control plane. With OpenFlow, the control plane can provide packet information to the controller for analysis via a `PACKET_IN` message. The controller can send a packet to the control plane and responds via a `PACKET_OUT` message. A controller can also send a `FLOW_MOD` message to tell the switch how to process packets that match certain rules. Accordingly, the control and data planes also need to understand OpenFlow.

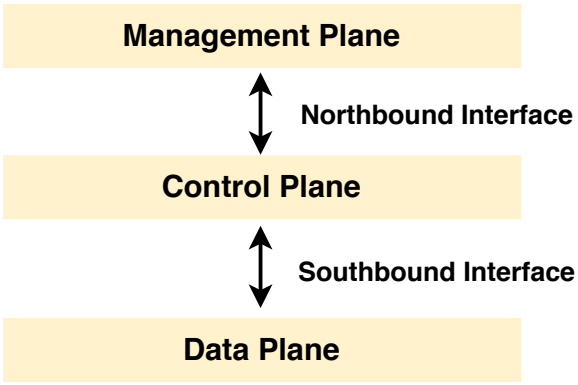

Figure 2.1: A basic SDN architecture consisting of three planes and two interfaces.

We explain the concept of each component of SDN, based on Figure 2.1, starting at the bottom. OpenFlow also defines the interfaces that the control plane can interact with the data plane.

2.3.1 Data Plane

The data plane forwards packets through various devices that are connected with the switch. Traditionally, it forwards packets according to the forwarding table. However, the data plane of an OpenFlow-enabled switch processes packets according to flow tables. Each rule in the flow table consists of entry, action and statistics. The rules are invoked by packet matches, i.e., the IP addresses, port number and protocol of a packet are the same as a flow entry. Then the data plane processes this packet according to the action specified by this rule. It can forward or drop a packet and even modify the packet inline. Finally, the statistics field can collect information for analysis, such as the number of packets matching certain rules.

2.3.2 Southbound Interface

The southbound interface defines how the control plane can manage the data plane. SDN can use OpenFlow protocol to achieve this communication. The control plane may install OpenFlow rules into the flow tables maintained by the data plane. The data plane must then process incoming packets according to these rules.

2.3.3 Control Plane

The control plane manages the network by installing rules to the data plane. In an SDN implementation, the control plane needs to understand the OpenFlow protocol. On the southbound interface, it installs OpenFlow rules to the flow tables maintained by data plane. On the northbound interface, it talks with the management plane, the SDN controller, via the API defined by OpenFlow. The control plane may intercept and elevate packets to the SDN controller for a decision. In particular, if a packet received by the data plane does not match any existing rule, the control plane configures it to send this packet to the controller for a decision. After receiving a decision (for example, a `FLOW_MOD` message sent from the controller), the control plane parses it and installs this flow to the data plane. Accordingly, the SDN controller is able to process network functions in the control plane and implemented them in the data plane.

2.3.4 Northbound Interface

The northbound interface of SDN is the API used to communicate between the SDN controller and the control plane. Rather than having network owners configure the control plane locally and manually, this interface outsources network management to achieve automation, since home network owners usually lack such expertise.

2.3.5 Management Plane

The software that connects to the control plane by OpenFlow protocol is called the management plane. There are various implementations of the SDN controller framework. For example, `Floodlight` [131] is a controller developed in Java, and POX [114] is another one developed in Python. SDN allows network functions to be achieved by higher-level programming language, which is more convenient for developers.

Various applications, such as firewalls, network topology visualization, and load balancers, can register with a controller and work simultaneously. The packets received by the controller via `PACKET_IN` go through each registered application. An application can alter the packets inline and tell the control plane how to process this network flow.

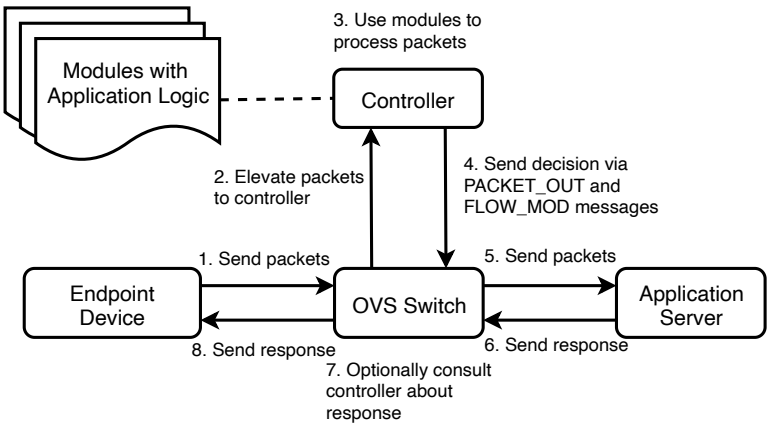

Figure 2.2: The basic components of SDN and packet processing.

The overall delay introduced by the management plane is the sum of the delay introduced by each application. Accordingly, the performance of each application should be efficient. For better performance, the management plane usually installs `FLOW_MOD` rules in the data plane for subsequent packets after processing the first packet in the flow. This mechanism avoids packets being unnecessarily passed through controller again, since the elevation causes network delay, especially if the controller resides in a remote cloud server.

Since multiple networks may connect to a controller, the controller can acquire information from distributed locations. The aggregated information on the controller can help with network analysis and threat diagnosis, such as the distribution of a botnet.

2.3.6 SDN Procedures in a Home Network

Figure 2.2 shows an example of a basic SDN system. The device using the SDN service is required to connect to an OpenFlow-enabled switch. Some enterprise-grade switches have a built-in OpenFlow agent. However, consumer-grade residential routers usually need to flash their firmware to OpenWRT [167], an open source, Linux-like system designed for resource-constrained devices. OpenWRT comes with rich widgets to help with configuration. In particular, OpenWRT is needed to install Open vSwitch [132], which configures network interfaces with an OpenFlow-compatible bridge. It trans-

forms traditional router functions to SDN data and control plane. It also maintains a local flow table.

As Figure 2.2 shows, a client device connected with the network sends out a network request (Step 1). If the first packet fails to match any flow entry in the flow table, by default, the packet is forwarded to the controller (Step 2). This packet goes through each module on the controller. Each makes a decision and tells the OVS switch how to process this flow via a `FLOW_MOD` message (Steps 3 and 4). The controller may also want the switch to send out the original packet (Step 5). Thus, later packets of this flow, or even traffic in the reverse direction, are not necessarily elevated to the controller again (Steps 6 and 7). Finally, the endpoint device receives the response (Step 8). In certain cases, the controller can also inspect each packet of a flow without installing a `FLOW_MOD` rule.

2.3.7 SDN Considerations

The placement of an SDN controller may vary by situation according to different needs. An SDN service can be provided by a third-party company with a profit incentive. As more users connect to the controller, it gains a global view across the connected networks. This view provides the controller with more accurate network insight, so it can give better security to users. Controllers can also be managed by an Internet Service Provider (ISP). The ISP location is close to home networks and thus introduces less latency. However, both of these positions require a constant Internet connection. A standalone SDN controller can help in the event of Internet disconnection. The modules without Internet requirements, on a standalone controller, can monitor the network activities inside the network. The controller can be hosted on a separate and relatively powerful device, like a Raspberry Pi.

The placement of the SDN agent can also vary. An SDN-compatible network can deploy its OpenFlow agent on an OpenFlow-enabled physical switch. Since a gateway monitors all network traffic in and out, the SDN agent on a switch is able to manage all traffic in a device-agnostic way. It requires no modification on endpoint devices.

Alternatively, the OpenFlow agent can be set up on an end-device itself. Then the

agent is able to extract more information from the machine. For example, it can relate network flows with process and graphical user interface (GUI) information [115, 164] to distinguish whether a network flow was initiated by a human activity, or a malicious script. This information is helpful for intrusion detection. Lei et al. [89] deploy SDN systems on desktops in an enterprise network for traffic engineering. Other work by Lei et al. [90] shows how correlated signals from endpoints can reveal compromises on those endpoint devices. Further, SDN has also been implemented in Android: meSDN [88] and PBS-Droid [68] propose to deploy SDN agents in the Linux kernel on Android. However, this method requires recompiling the OS, which is an obstacle for many users. HanGuard [34] proposes to install SDN technology on both Android endpoints and home routers to enforce fine-grained access control. Zorigbattar et al. [26] fused SDN with Windows for UI based network management.

2.3.8 Residential Networks with SDN

To achieve enterprise grade security for home networks, researchers have proposed to outsource home network security. In 2010, Feamster [44] suggested outsourcing the management of home networks to services in cloud providers using SDN. Haq et al. [63] proposed using an ISP-hosted SDN controller to achieve better network performance. However, Taylor et al. [163] measured and found that 90% households in the United States have a data center within a 50-millisecond round trip time, which suggests deploying home network services in the cloud would be viable for many. Taylor et al. [165] explored a system to verify certificate revocation on a cloud middlebox for residential networks. Given these prior efforts in security applications based on SDN, we proceed to secure home networks.

2.4 Performance Evaluation and Metrics

This section discusses performance evaluation, the metrics and why they are important.

Performance matters in most network-based applications in home networks. For example, online video games may need a low delay of the network since it refreshes

frames frequently. It requires each of its frames is delivered with minimal propagation time. The video conference also needs a low network latency to allow participants to communicate smoothly. For most other activities, such as web browsing, IoT device operations, like turning on a smart bulb, and chat, only require latency that is low enough to meet user expectations.

Next we describe our performance evaluation and merits in more detail. Generally our experiments are performed in a local network environment, which means the client machine, modified router and the controller are located in the same subnet. The Persistent Identity paper (Chapter 6) evaluated the performance where the controller is located at WPI's server, for evaluating the practical situation. We usually measure additional network delay introduced by our system compared to the original system. The goal is to demonstrate that the additional latency is acceptable to users. We use a previous result, 90% of the U.S households have a RTT to a nearby data center, where can host SDN controllers, within 50 milliseconds [163], to support our performance evaluation. In our projects, the applications that need to outsource packets to a remote controller usually need to add 50 millisecond delay, plus the time needed by our SDN module. We use DNS as an example to compare with our system and justify our network latency, since DNS works similarly with our SDN approach: it makes query once per flow if there is not an local entry cached, in addition to the original traffic. A survey [18] measured that the median DNS query time to a com server is 42.62 ms. Since DNS lookups are usually accepted by users, we believe network propagation delay around 50ms, once per request, can likewise be accepted by users.

For most of our designs in this dissertation, an SDN application requires only one packet elevation per flow, like DNS. Further, for applications like email checking, logging in to Microsoft Teams, and operating an IoT device, where the flows are not generated frequently, the once-per-flow overhead may be acceptable to users.

Some of the designs in this dissertation, including IoT traffic inspection and host-based SDN agents for mobile devices, require processing for each packet. For this case, we recommend deploying the application locally, to avoid the network propagation delay between the network and SDN controller. For example, in terms of our host-

based SDN on mobile devices work described in Chapter 7, the added delay for each subsequent packet is only 2.6 ms on average, which we believe it will be acceptable to users.

Bandwidth and throughput are two significant metrics for network-related applications. Most of our methods are based on SDN, and usually only the first packet is elevated and evaluated by the SDN controller. The overall throughput overhead should focus on subsequent packets, which are processed by the SDN data plane on the local router, according to the FLOW_MOD previously installed by the SDN controller. In this dissertation, we only need to evaluate throughput for the methods that affact all packets in a flow, such as the Android-based SDN research (Chapter 7).

The SDN data plane may also introduce overhead. It could be affected by various elements, like different types of OpenFlow table implementations (hash vs. linear) [16]. Regarding the data plane performance, there is work to improve OpenFlow switch performance [139], but this is out of this dissertation's scope.

Another performance metric is resource consumption on endpoint devices and the SDN controller. On endpoint devices, we perform CPU, memory and battery consumption measurements on Android SDN project (Chapter 7). SDN controller optimization is not our focus. On the other hand, if the overhead is high, SDN security service providers has incentive to deploy load balancer and Content Delivery Network (CDN) technologies to optimize the performance for customers.

Chapter 3

Authenticating Endpoints and Vetting Connections

3.1 Introduction

Most residential networks are created and managed by end-users that may lack computer security expertise. These users may employ weak security practices, such as not changing default usernames and passwords on devices, which allow attackers to easily compromise systems on the network. Further, the end-devices themselves may introduce weaknesses, such as failing to encrypt traffic or to patch vulnerabilities. This is a particular challenge in Internet-enabled embedded devices, commonly referred to as "Internet of Things" (IoT) devices, in which only the device manufacturer can deploy updates. As a result, residential networks may also be home to compromised devices that enable attackers to persist, pivot to other devices, and control the home's environment.

One way to combat the insecurity of residential networks is to outsource the security management to experts. We envision a service provider that hosts systems outside the network, potentially in a cloud data center, to manage the devices inside the residential network. With recent developments in software-defined networking (SDN), it is now possible for an off-site controller to remotely manage a network's infrastructure [163].

To capitalize on the benefits of outsourced network management, we need new

approaches to distinguish legitimate traffic from potentially malicious activity. In this work, we examine whether 1) the initiator is an authorized party 2) the exchanged messages conform to expectations (i.e., they follow protocol and historical interactions). When properly implemented, these requirements could greatly reduce a network's attack surface.

Our work revolves around two research questions: *What mechanisms can effectively authenticate client communication without requiring changes to server or client applications? What are the performance overheads associated with implementing this in consumer-grade hardware?* We explore these questions by installing OpenFlow software on an existing consumer-grade router and installing custom modules on an OpenFlow controller to implement device-agnostic, network-level multi-factor authentication and strict enforcement of IoT device communication using a historically-derived rules.

We make the following contributions in this work:

1. Enable Device-Agnostic Authentication: We incorporate Google Authenticator into the control plane defined by an SDN controller. Only devices presenting a valid one-time-use passcode can access the IoT registered device. Otherwise the requests are dropped. Thus, even if other built-in methods that come with the devices are subverted by an attacker, they cannot access the target without authentication. We present our approach in a lab environment and the results show that our system adds less than 50 ms of delay to 90% of the connections.

2. Enforce IoT Device Protocols: IoT devices are typically purpose-built and provide a small set of services via relatively simple protocols. As a result, we can exhaustively enumerate the packets used by these IoT devices during standard operation. We build a state machine model of each protected device and only allow packets to the device in which the payload follows known transitions within that state machine. This prevents attacks such as buffer overflows or other specially-crafted messages that could exploit a vulnerability on the device. The approach can detect and discard packets that violate the protocol, preventing attacks from reaching the device, usually with less than 200 ms of delay.

3.2 Related Work

The security challenges of IoT devices have been explored from many angles. Babar et al. [14] explored the security model of IoT devices and the threats that they face. Zhang et al. [185] explore the challenges for IoT device research and the privacy challenges that some devices introduce. Kolias et al. [82] investigated the role that vulnerable IoT devices can play in large-scale attacks, such as those enabled by the Mirai botnet. While the topic of IoT device security is being studied by the community, little work focuses on residential networks.

3.3 Threat Model

Residential IoT devices may have unpatched vulnerabilities, default passwords, or simply weak passwords. These devices may have built-in authentication mechanisms; however, we consider these built-in mechanisms to be second-factor authentication which may offer only limited security benefits in practice. Further, within the LAN, some devices may implicitly trust all other devices and communicate without any authentication mechanism at all. This approach is common for IoT devices and video players, in which physical presence and/or knowledge of wireless network keys is considered sufficient evidence of authorization.

We assume our router is directly connected to each end-point device and there are no routes to these devices which bypass our router. We consider our authentication server, OpenFlow controller, and modified router to be part of the trusted computing base (TCB). All other devices on the network are considered untrusted and potentially compromised.

3.4 Approach: Vetting New Flows

This section we introduce the approaches we use on the residential network: mutlifactor authentication and traffic vetting.

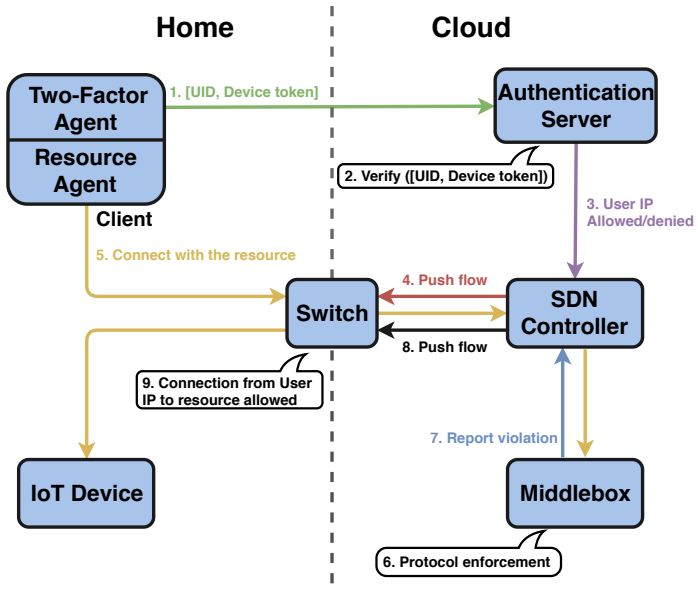

Figure 3.1: The Open vSwitch (OVS) gateway ensures effective two-factor authentication. Before each request, each client needs to send an authentication request to the authentication server. The authentication server sends the result back to the SDN controller, which informs the OVS router to allow or deny later request by installing a `FLOW_MOD`. The traffic may go through a middlebox for other flow inspections.

3.4.1 Mutli-Factor Authentication (MFA)

Multi-factor authentication requires two or more separate forms of authentication information as proof of identity. Common forms of authentication include passwords, possession of random values from previous interactions (e.g., web browser cookies), possession of cryptographic keys, possession of specific hardware security tokens, or information transmitted out-of-band, such as messages to another computing device. In our approach, the second factor is similar to the possession of a cryptographic key since authentication requests require appropriate keying data to be present on the initiating client.

A client must send a request and resource-specific authentication token, potentially via a separate application, before connecting to a given IoT device, as shown in Figure 3.1. If the authentication server is able to verify the client's token, it notifies the OpenFlow controller of the result. When the client attempts to access the IoT device,

```
Content-Type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1
#SetBinaryState"
Content-Length: 383
Host: 192.168.3.157:49153
User-Agent: CyberGarage-HTTP/1.0
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.
org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/">
 <s:Body>
  <u:SetBinaryState xmlns:u="urn:Belkin:
service:basicevent:1">
   <BinaryState>1</BinaryState>
   <Duration></Duration>
   <EndAction></EndAction>
   <UDN></UDN>
  </u:SetBinaryState>
 </s:Body>
</s:Envelope>
```

Figure 3.2: Example payload from the tested Smart Switch (some line-wraps were added to fit within a column). The interaction is via a SOAP request using XML encoding allowing the payload and structure to easily be enforced programmatically.

the OVS router elevates the request to the controller. If the controller has received an authentication notification from the server, it orders the OVS router to cache a rule allowing that client IP address to temporarily reach the IoT device. Otherwise, it orders the OVS router to drop the flow's packets.

3.4.2 Traffic Vetting

Some IoT devices have a more restricted API than generic computing devices. This restricted API allows us to enhance our middleboxes to implement communication models for each IoT device type and use these models to restrict the type of messages allowed when communicating with each IoT device. We build regular expressions for matching the packet payload that constrain the messages to the small set that we observed during a training phase. These expressions allow the variability needed for session-specific values. In Figure 3.2, we provide an example of this approach

for the payload between a smartphone and an IoT device. The request orders the device to activate or deactivate. Our regular expressions allow these two states and allows variation in the content length associated with the packet. The expressions require other fields to remain static. This approach enables the middlebox to allow only authorized messages and to easily detect unauthorized messages. In the instance an unauthorized message is detected, the middlebox can discard the packet and alert the OpenFlow controller. The controller can then alter the flow entry at the OVS router to discard subsequent packets in the flow.

3.5 Implementation

We implement our system using a TP-LINK Archer C7 consumer-grade router to directly connect our protected end-points. We install OpenWrt [167] with Open vSwitch (OVS) [132] on the router. The router communicates via the OpenFlow protocol to a local controller running the Floodlight [131] Java-based OpenFlow controller software. The controller runs on Mac laptop that is connected to one of the OVS router's LAN ports. The controller laptop has four 2.6GHz cores and 16 GB RAM. We add a custom module to the Floodlight controller that subscribes to all packets processed at the controller.

When hosts communicate within the same subnet, the OVS router normally connects the two using its hardware switch, bypassing the need for the packet to be examined in software. However, our approach requires that each flow be examined in order to provide access control, so we needed to avoid this behavior. We configured the router to place each of its physical interface ports on a separate virtual LAN (VLAN) and allowed the router's main processor to route packets across VLANs. We likewise use wireless isolation for VLAN radio communication [169].

Our two-factor authentication server uses the Google Authenticator library [61] in a C program that runs on an Ubuntu 14.04 laptop that is connected via Ethernet to the OVS router. That laptop has four 2GHz cores and 8GB of RAM. Using a one-way hash function based on the current time and a pre-shared secret, the server dynamically

generates one-time use secret tokens for each registered device. Each registered device obtains a copy of the pre-shared secret. With this value, the client can use the current time and pre-shared secret to compute its own version of the hash output and send it to the server, which can verify its authenticity by comparing it with its own locally computed value. Upon successful verification, the authentication server communicates with the controller over a network socket to send the client's IP address and authorized destination. The controller stores a record for the authentication result to authorize subsequent flows to that destination for a short period.

3.5.1 Protecting IoT Devices

In our experiments on IoT devices, we focus on the Belkin WeMo Smart Switch [15]. The Smart Switch is a electrical power outlet adapter that plugs into a standard electrical outlet and exposes another outlet that devices, such as a lamp, can use. The Smart Switch can be controlled through a smartphone application in both the iOS and Android operating systems. Through the smartphone application, a user can activate or deactivate the power flow, gaining the ability to control the power to the connected device remotely.

To test the protocol enforcement, we use the WeMo smartphone application and manually toggle the switch setting 1,000 times. For the two-factor authentication system, we automated our experiments using a Python script using Google Authenticator on a wirelessly-connected Mac Mini, which had two 2.6 GHz cores and 8 GB of RAM.

To model the types of communication between the smartphone and IoT device, we consider a state machine. To move from the initial state, the smartphone must send one of a small set of valid messages. Upon receiving such a message, we advance the state of the device and consider the new messages available. By continually repeating this process, we are able to block any messages that are not valid for the given state and prevent any malicious messages from arriving. We build our state machine over a series of runs with an uncompromised device and then use the state machine in a middlebox to enforce its actions. Each path through the state machine can be considered a packet sequence and each transition is determined by matching the payload of a given packet

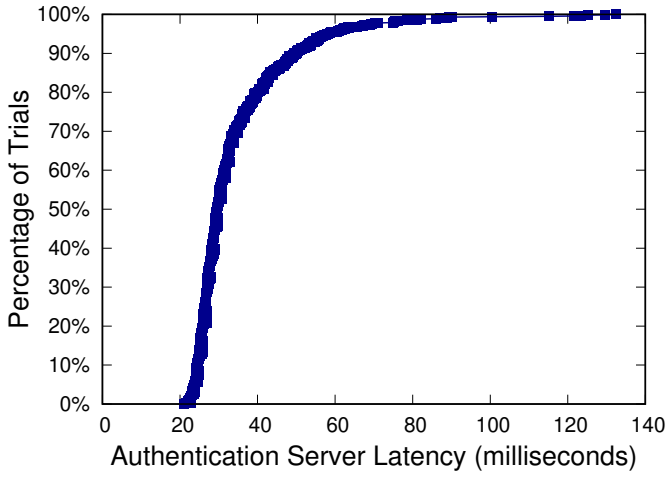

Figure 3.3: Two-factor authentication request delay (1,000 trials).

Table 3.1: Security evaluation results for each approach

Approach	Request	Trials	Allowed	Blocked
Two-Factor Authentication	legitimate	20	20	0
	malicious	20	0	20
IoT Protocol Enforcement	legitimate	20	20	0
	malicious	20	0	20

with a known regular expression for that packet.

In Figure 3.2, we provide an example of an initial request with HTTP payload. It begins with the HTTP header, followed by the *envelope* structure. The *content-length*, *HOST*, and *BinaryState* fields may vary across devices and actions. We build regular expressions that match these dynamic parts while requiring exact matches for the static components. When a packet matches a sequence, we push the packet information plus a time stamp into a packet list. The timestamp allows us to reset our state machine after a timeout occurs.

3.6 Security and Performance Evaluation

To evaluate the security of our approach, we made connections to our Smart Switch IoT device. To create legitimate behavior, we followed the protocol of providing two-factor authentication connections and then connected to the device. Likewise, the interactions

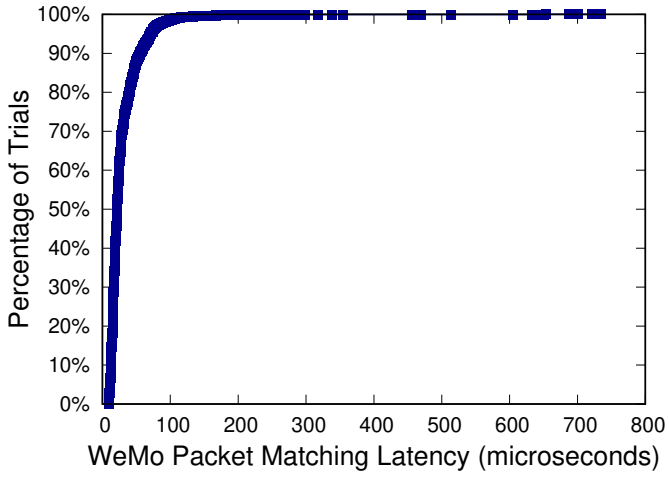

Figure 3.4: Delay from the protocol enforcement module for the IoT device traffic. These results are from 67,335 packets produced during 1,000 trials. We omitted 30 outliers (0.04% of data) for readability.

with the WeMo device conformed to the protocol expected for that device. In the two-factor authentication approach, we create malicious connections simply by initiating a connection without completing the two-factor authentication steps. In the protocol enforcement experiment, we send packets that contained random bytes as payload to create malicious packets. Each approach was tested in isolation, with only one module active.

In Table 3.1, we show the results of these experiments. In each case, the approach allowed the authorized connections and denied the connections that were malicious. These results show that each of these approaches can offer significant security advantages. While the IoT protocol enforcement system must be customized to each type of IoT device, the two-factor mechanism is more generally applicable.

For our performance evaluation, we consider the performance of the two-factor authentication system and the IoT protocol enforcement module. We consider the performance of each individually. Our results are for a controller and middlebox in the LAN. Additional propagation delays would be incurred for remote deployments.

To determine the timing overheads of the two-factor authentication system, we used a script on the client that recorded the results of the `time.time()` function in

Python, which returns the current Unix epoch time with microsecond resolution. We measured the time before we issued the request to the authentication server and after we received the server's response. By subtracting these values, we could determine the amount of time elapsed. In Figure 3.3, we see that the majority of requests are satisfied within approximately 30 milliseconds and 90% are satisfied within about 50 milliseconds. These delays are unlikely to be noticeable to an end user.

For all the modules running on the controller, including the IoT protocol enforcement, we used the `System.nanotime()` function in Java to record the timestamp when the module started and when it ended and calculated the overhead as the difference.

3.7 Discussion

Our work uses a controller and middlebox located in the LAN. When these components are further away, latency could become a concern. However, our prior work has explored residential connectivity with public cloud data centers and found that over 90% of residential networks in the United States were within 50 milliseconds of a public data center [163]. That study found such latency would have only a small impact on the user experience, even in applications like web browsing.

The two-factor authentication system could be automated using a resource like the `netfilter_queue` library which could intercept a packet, send the authentication request, and requeue the packet. Prior work has used a similar technique [164]. This would essentially create a device-level authorization system.

Our IoT device state machine was based off of a Smart Switch with a limited API. Other IoT devices may have more functionality and require more involved state machines to replicate their protocols. However, other modern IoT devices may have constrained behavior.

Chapter 4

Account Lockouts: Characterizing and Preventing Attacks

4.1 Introduction

In an attempt to gain unauthorized access to a system, attackers may try to guess the credentials associated with a legitimate user's account. These attackers may vary in sophistication, from brute-forcing passwords on default usernames to using a list of known usernames at an organization and lists of most commonly used passwords. Prior analysis of password data sets has shown that end-users often select weak passwords that are vulnerable to such attacks [175]. Further, many organizations consider usability and memorability to be key goals in username generation. As a result, usernames are often generated that match email addresses and use parts of a user's real name [178].

Given this threat, many systems implement an account lockout mechanism in which all authentication attempts are denied after a certain number of failed attempts in a predetermined time window. NIST, which sets standards for US government systems, recommends an attempt threshold of 100 attempts or less with a lockout period of between 30 seconds and 60 minutes [59]. The SANS institute recommends a threshold of five attempts with a 30 minute lockout period [142]. To be PCI compliant, which is required for organizations handling consumer payment information, accounts must be

locked out for 30 minutes after six failed attempts [127].

This account lockout mechanism can be used by attackers to create a denial-of-service (DoS) attack that prevents legitimate users from gaining access to their accounts [110]. Such an attack is easy to launch: an attacker can issue authentication attempts at a rate that would keep an account perpetually locked. With the aforementioned thresholds, such an attack would consume minimal attacker bandwidth and computational resources. Even if simple IP address blocking is used for repeated failed attempts, an attacker could use a network of compromised machines to distribute the attempts.

With the deployment of single-sign-on (SSO) services, account lockouts can transform from a simple nuisance to a crippling attack. Recent work has explored web-based SSO systems and the relationships between identity providers that authenticate users and other websites, called relying parties, that use those identity providers to authenticate their own users [49]. In one example, a single identity provider was used by 42,232 relying parties. Further, recent reports [133] estimate that Active Directory—Microsoft's prominent SSO identity provider—is used in more than 90% of companies in the Fortune 1000. With an account lockout attack on a single identity provider, a targeted user could be denied access to thousands of other services.

In this chapter, we ask two key research questions: *To what extent are organizations vulnerable to account lockout attacks? What countermeasures can be effectively deployed to address these attacks in a way that supports even legacy systems and devices?* Given its widespread deployment and integral nature at organizations, we focus our investigation on Microsoft's Active Directory service. In doing so, we make the following contributions:

1. **Vulnerability Measurements:** We examine 2,066 organizations, including Fortune 1000 companies and universities, to determine the extent to which attackers can systematically identify vulnerable authentication portals and lock accounts. We find that roughly 58% of the universities and roughly 77% of the companies examined expose a vulnerable authentication portal. Lockouts targeting these portals can potentially deny users access to thousands of applications [111].

2. **A Suite of Proposed Countermeasures:** Rather than relying on changes to Active Directory, we propose countermeasures that can be deployed immediately on legacy architectures. The suite of options, based on the concept of distinct authentication pools, includes mechanisms that work across devices with end-user involvement to completely transparent options, such as those using web browsers or modified home routers.

3. **Evaluation of the Countermeasures:** We evaluate the security effectiveness and performance of the proposed countermeasures. We find that each has clear availability advantages while introducing minimal performance costs. Notably, we find that existing authentication mechanisms—such as multi-factor authentication—are insufficient to stop account lockout attacks because the root problem lies with the lockout policy, not the mechanism.

4.2 Background and Related Work

The combination of a username and password is a ubiquitous method of user authentication. Attackers try to obtain such sensitive information to infiltrate computer systems. The sophistication of theses attempts vary. The most basic attack, a brute force attack, exhaustively enumerates all possible character combinations until a valid sequence allows access. The success rate of brute force attacks is dependent upon the underlying strength of user passwords [179].

Other approaches are more sophisticated and use information about end-user behavior to increase success rates [65]. Dictionary attacks, for example, form the password guesses by using a large database of popular passwords or words in a targeted language's dictionary. Prior research has found that many end-users select passwords that could easily be discovered by a dictionary attack [175]. Further, discovered passwords from a compromised service may be used to guess passwords for the same user at other sites due to password reuse [72].

To combat password guessing attacks, standard bodies recommend account lockout thresholds [59, 127, 142]. After a specified number of failed login attempts, the

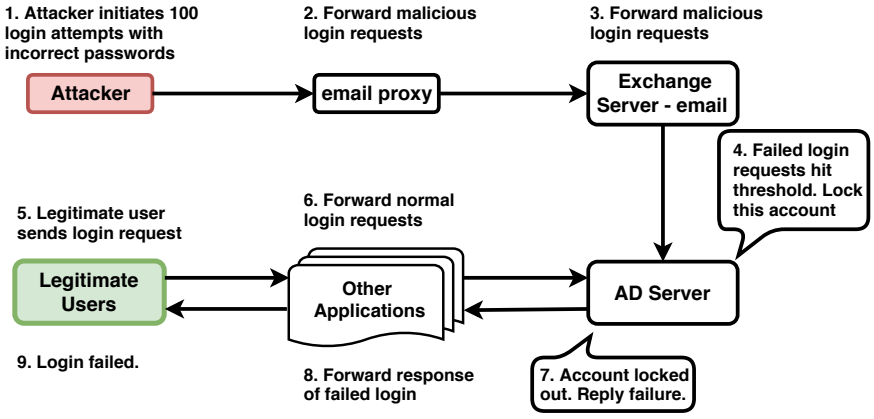

Figure 4.1: In an account lockout attack, the attacker selects a username and tries to authenticate with invalid passwords. Each failed attempt causes the server to increment the failed attempt counter for that specific account. When the legitimate user attempts to authenticate, the account may already be locked.

account lockout approach denies access to a given account even if valid credentials are provided [37]. This simple mechanism makes brute force password guessing infeasible and limits the rate at which attackers can make attempts using dictionaries. Unfortunately, account lockouts provide a natural avenue for denial-of-service attacks: an adversary can simply make numerous failed authentication attempts for a given username, causing the account to lock, and thereby preventing the legitimate account user from authenticating, as shown in Figure 4.1.

Other techniques attempt to limit malicious authentication attempts without using a lockout. A common approach is a form of automated Turing test before each login attempt that will purportedly distinguish a human from an automated adversary. A prominent approach is the CAPTCHA [130], which requires a user to decode an image or audio signal in a way that is challenging for computers to do. Such approaches may help deter dictionary attacks, but they do impose usability costs upon users [2]. Unfortunately, with innovations in machine learning, some previously-effective CAPTCHAs may be defeated automatically [173]. Further, hardware or legacy systems may be unable to support CAPTCHAs.

Aura et al. [13] propose the use of client-side puzzles to defend against denial-of-

service (DoS) attacks by slowing the attack rate. Each time the client makes a request to the server, it is asked to solve a cryptographic puzzle provided by the server. These puzzles must require significant client effort to solve and are unpredictable. The verification of the result should be inexpensive. Dean et al. [33] incorporate cryptographic puzzles into the TLS protocol to protect servers from DoS attacks. Koh et al. [80] evaluated a high performance puzzle algorithm.

Unfortunately, puzzle-based defenses may not be compatible with some existing systems and applications. For example, the use of a CAPTCHA may not be feasible when logging into a legacy video conferencing system. Some prior versions of mail software, such as Outlook 2010, not support tools like CAPTCHAs when authenticating. Similar limitations may occur for Skype for Business and applications without a browser-based interface.

4.2.1 Active Directory (AD)

Active Directory (AD) is a service from Microsoft for managing user accounts and system resources belonging to an organization. It groups users, workstations, servers, and policies and organizes them into hierarchies that facilitate management. This service allows user management to be logically centralized by an organization in a set of domain controllers. Application servers may authenticate users via these domain controllers rather than managing accounts and passwords locally, as shown in Figure 4.2. For example, Microsoft's email server, Exchange, uses an AD server for authentication.

Organizations may host their AD domain controllers on-site, host them in the cloud through Microsoft's Azure AD service, or use a hybrid of both options. The Azure AD service is essential for Microsoft-hosted online services like Office 365 and Skype for Business Online. Those Azure-based services communicate directly with the Azure AD domain controllers rather than using the on-premise servers. In hybrid deployments, on-site AD domain controllers may configure a unidirectional synchronization channel with the Azure AD servers. For the purposes of account lockouts, an account locked by an on-site domain controller will result in a lock in all domain controllers and, depending upon the configured settings, may propagate to Azure domain controllers.

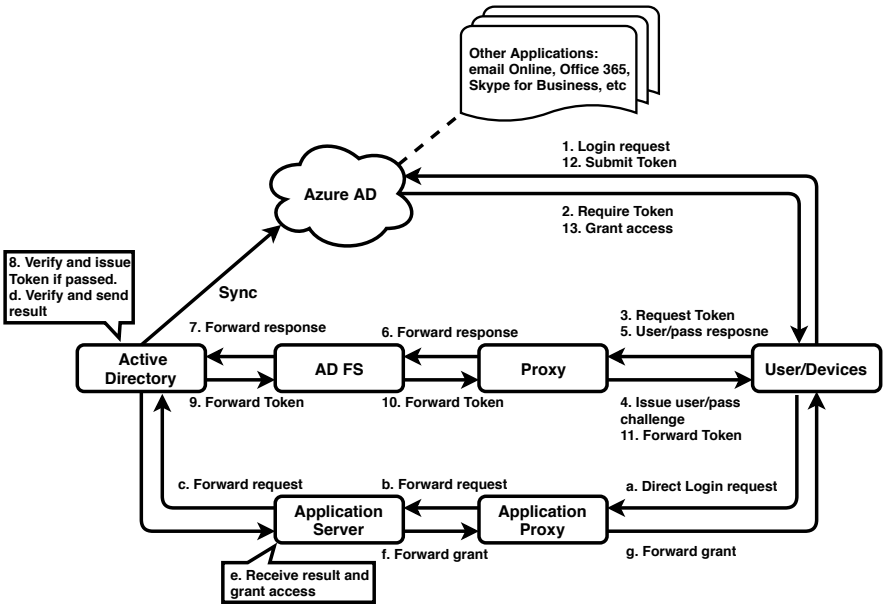

Figure 4.2: This diagram depicts the components and interactions between an on-site AD domain controller, an Azure AD domain controller, and the ADFS connector.

In contrast, an account locked by an Azure domain controller will not propagate to on-site domain controllers.

Microsoft also provides an Active Directory Federation Services (ADFS) interface for applications to interact with Active Directory when they cannot use the integrated Windows authentication service. ADFS has its own account lockout mechanism, but that lockout only affects ADFS services.

For an attacker to maximize the impact of an account lockout, the best option would be to target a service that authenticates to an on-site domain controller, if such a domain controller exists, in addition to targeting an Azure AD domain controller. An account lockout in ADFS or in an Azure AD domain controller may result in a lockout that affects only a subset of the organization's services. However, in some attack scenarios, a subset may be acceptable to an attacker if it includes a critical service the attacker wishes to make unavailable.

4.2.2 Middleboxes for Security

Middleboxes, such as firewalls, intrusion detection systems, and proxies, have regularly been used for security purposes in the enterprise. Recent techniques have leveraged the cloud for enterprise security [144].

Other work has extended middlebox techniques to residential networks, including for whole-home proxies [166], validating TLS connections [165], and verifying IoT device communication [92]. As demonstrated by Taylor et al. [163], these residential middleboxes are feasible in countries like the United States since most residential users are within 50 milliseconds of a public cloud data center, causing middleboxes to only incur minor latency costs.

Our work shows how middleboxes can address account lockouts on an enterprise network in a backwards-compatible manner. We further show that middleboxes at the home (e.g. via a modified home router) can further enable robust account lockout protections.

4.3 System Overview

Active Directory is inherently flexible and scalable, which can lead to deployments that vary greatly in terms of complexity and redundancy. In the simplest case, an Active Directory setup involves a primary domain controller, one or more dependent application servers, and a set of client machines that wish to use the application server. Organizations may deploy other infrastructure, such as secondary domain controllers, proxy servers, and middleboxes, to support legacy systems or to achieve resiliency or security goals. Such infrastructure has little impact on the account lockout threat and we omit it for simplicity.

4.3.1 Assumptions and Threat Model

In the context of this work, the goal of the adversary is to deny a legitimate user access to services and resources through an account lockout attack. These adversaries may perform reconnaissance on an organization ahead of an attack to obtain email addresses,

usernames, or to locate public-facing authentication portals. With the availability of botnets, an adversary may have significant computational and network resources. These resources afford the attacker significant flexibility in devising her attack strategy. For example, the attacker may send a high-volume of authentication requests from geographically-diverse machines and rapidly switch between IP addresses to avoid IP-based blacklisting.

This work does not consider attempts to compromise the Active Directory server, its dependent servers, or other hardware such as the organization's switches and routers. If these servers fall under the control of the adversary, it would be impossible for an organization to guarantee the accuracy of a user's identity or the availability of authentication services. Similarly, we assume an adversary lacks valid user credentials.

The defender's goal is to provide legitimate users with the ability to authenticate even under an ongoing lockout attack. For our proposed countermeasures, we assume that the organization's IT staff can insert one or more middleboxes into an organization's infrastructure, but they cannot modify the Active Directory server or the services that authenticate against Active Directory.

4.4 Characterizing the Account Lockout Problem

In this section, we explore the following research questions:

1. Are the public authentication portals feasible to attackers to launch account lockout? With permission from a cooperating organization, we explore their attack surface from public portals and test the attack effect with a production account setup for this test. This organization strictly follows the setup process offered by Microsoft, thus this case study is likely representative of many other organizations.

2. Can attackers automatically discover attack surfaces that can be used for account lockout from organizations' public portals? How many organizations can be affected by this attack?

4.4.1 Case Study: Identifying the Attack Surface in Production

We contacted a multinational organization with over 5,000 employees that uses Active Directory extensively and gained their approval to assess the impact of account lockouts across their environment. This organization used Active Directory for authentication for the vast majority of their IT services. From this case study, we created an Internet measurement strategy to characterize the risks at other organizations to determine the broader applicability of our findings.

Our partnering organization made an Active Directory administrator available to provide feedback on our tests, but the organization required anonymity as part of their participation. The organization created a test account for our use which was modeled after a standard employee account at the organization. The organization set a secure password on the account and ensured it was not shared with the authors performing the authentication attempts.

In our testing, we independently gathered information that was available publicly without use of organizational insider knowledge. In our experiments, we found that the organization used `mail.[organization domain]` to forward to a themed Outlook Web App (OWA) portal, which is a Microsoft-provided interface for web-based email. Since the portal used IP addresses that were not associated with Microsoft, we determine that the OWA portal was not Azure-hosted and thus was not using an Azure AD server for authentication.

With many Office 365 services, Microsoft provides a centralized authentication portal that leverages the user's email address to determine the appropriate Azure AD server to use to process the authentication. That authentication page compares the host portion of the email address to its list of registered organization domains. Accordingly, we went to the Office 365 authentication page [109] and entered a randomly constructed username, followed by the '@' character, and then the organization's domain name. The website redirected to the organization's account authentication page, where we were prompted to enter a password. This interface appeared to be Azure-hosted, indicating that login attempts would be directed to an Azure AD server. Account lockouts generated on this service would likely only affect services authenticating against the Azure

AD server while not affecting user access via the OWA page we previously discovered.

We then examined Skype for Business (SFB, formerly known as Microsoft Lync). Based on Microsoft's documentation, the `lyncdiscover.[organization domain]` host name is typically used for this service. We performed a `CNAME` query on that host and the response indicated that the organization was not using an Azure-hosted SFB service. We then performed a DNS `A` record query, which returned a valid IP address that is not associated with Microsoft's Azure data centers, which suggests that the organization uses an on-site SFB service.

4.4.2 Case Study: Testing Account Lockouts in Production

After identifying the attack surfaces of the measured organization, we began testing account lockouts. Microsoft's documentation for Windows Server 2012 [99] and 2016 [100] recommends an account lockout of 10 attempts with a lockout period of 15 minutes. For Azure's AD service, Microsoft's documentation indicates a threshold of 10 attempts with a 1 minute lockout period. The most generous lockout policy was suggested by NIST with up to 100 attempts and a lockout period as short as 30 seconds. Based on these thresholds, we created an attack that would try authenticating as our test account with randomly-generated passwords around 200 times per minute. This attack is relatively low bandwidth at only 13 KBytes/second, which poses little burden on the attacker or on the organization's infrastructure. However, under the most conservative guidance, the attack would keep the targeted account perpetually locked.

We first targeted our attack at the organization's OWA portal. Our organization contact confirmed that the attack caused the account to be locked at the organization, preventing the account from logging into the organization's resources for the duration of the attack. We discontinued the attack and the organization contact removed the account lockout.

We next performed an attack targeted at SFB. Using the fake account, we use a tool provided by an open source project on Github named `lyncsmash` [123]. It provides an option to discover the SFB servers and an option to launch an account lockout attack. We manually went to the URL found in the tool, entered the username supplied by the

organization, and entered an inaccurate password 10 times. Our organization contact then confirmed that the account was locked. We note that the `lyncsmash` tool can automate these attempts.

In these tests, we used the same source IP address for each query. While a simple IP rate limit or blacklist would stop our attack, an actual attacker could easily perform the attempts using a botnet to ensure no IP address queried more than once. This would easily keep the account locked without an obvious defense. The measured organization's contact confirmed that the organization lacks a mechanism to combat such account lockouts.

4.4.3 Characterizing the Risk with Internet Measurements

While our partner organization was vulnerable to an account lockout attack, we now focus on determining the extent to which other organizations are likewise vulnerable. We begin by making non-invasive measurements of the public-facing infrastructure of a set of organizations. While we focus on Active Directory in this work, most organizations avoid directly exposing their AD servers to the public for security reasons. However, in many cases, these organizations expose their application servers to boost productivity. To allow employees to access their email outside the office, these organizations may expose Exchange email servers or website interfaces, such as the popular Outlook Web App (OWA) that Microsoft provides. Unified messaging services, like Microsoft Skype for Business (SFB), allow employees, customers, and partners to instant message, call, and join video conferences remotely. In some cases, the devices joining these calls may be mobile phones or dedicated video conferencing hardware.

Given the popularity of email and unified messaging, our measurement study focuses on determining the extent to which authentication portals for Microsoft-specific email and messaging servers are exposed publicly since we know such servers must use an AD server for authentication. We perform our measurements by using a list of domains associated with the Fortune 1000 companies [64] and with 1,066 universities [151]. We focus on these organizations because their domains can be easily obtained. Further, these larger organizations likely have need for centralized authentication services like

Active Directory.

Using our list of domains, we perform a DNS `MX` record lookup on the provided domain to determine the identity of the organization's public SMTP server. The host names of the SMTP servers provide some insight into the underlying infrastructure. For example, host names ending with `.protection.outlook.com` are indicative of an organization using Microsoft's cloud-hosted email service. Since these organizations necessarily use Active Directory in Microsoft's Azure cloud, these servers can be used to initiate an account lockout for all Azure-hosted solutions at the organization. Other `MX` records may indicate that the mail server is located on-site at the organization or is hosted by another provider.

Our second measurement uses information related to email auto-discovery [106]. We issue `CNAME` queries for the host `autodiscover` associated with the organization's domain (e.g., `autodiscover.example.com`). In some cases, the `CNAME` result was `autodiscover.outlook.com`, indicating the mail services use Microsoft's Azure-hosted Exchange server. In the case when another host name was returned, the mail server was not Azure-hosted. We then issued a web request on port 80 or 443 to the host name returned in the `CNAME` record. In some cases, the server required valid credentials to proceed. In some cases, the credentials would be validated by an Active Directory server, enabling the account lockout attack. However, in other cases, the authentication credentials could be independent of a user account (e.g., a username and password shared across the organization for relatively weak protection).

The discovered mail server's default web page could reveal information about the infrastructure. In some cases, the servers presented a default or themed version of Microsoft's Outlook Web Application (OWA) page, which is commonly associated with an on-premises Exchange server. When web servers return 403 forbidden, it means there could be a portal which requires authentication. We simply append "/owa" or "/autodiscover" and we found half of them redirect to an OWA login page. In other cases, the web server returned pages containing the string "Microsoft Corporation" indicating this server runs Microsoft's software. These authentication portals provide an avenue for the account lockout attack.

Organizations	Exchange Email		Skype For Business		Extent Vulnerable
	On-site	Azure-hosted	On-site	Azure-hosted	
Fortune 1000	190	339	360	345	765 (76.5%)
Universities	126	416	124	395	616 (57.8%)

Table 4.1: Our measurement study results show the majority of each group uses Microsoft services and has at least one exposed authentication portal, enabling account lockout attacks. The final column shows unique organizations vulnerable, even if an organization has multiple exposed attack surfaces.

Some domains did not use an auto-discovery service or did not provide an obvious account authentication page. For these domains, we issued an `A` record DNS query for the `mail` host name associated with the domain (e.g., `mail.example.com`), which follows the examples provided in Microsoft's documentation for configuring mail servers. We found that nearly half of organizations provide such a server for their employees to authenticate, though few of them used a default interface such as OWA or Microsoft's Azure-hosted email portal.

We next focused our measurements on the Skype for Business (SFB) service. Microsoft's SFB client automatically searches for an organization's servers using a mechanism similar to email auto-discovery. For all the Fortune 1000 and university domains, we perform a `CNAME` DNS query on the `lyncdiscover` host associated with the organization (e.g., `lyncdiscover.example.com`), which can reveal which organizations use SFB services. We also query for `dialin.example.com` and `meet.example.com`, which are other commonly used SFB host names. When organizations use Microsoft's Azure hosted systems, the `CNAME` query returns an answer associated with the `webdir.online.lync.com` host name. For all the non-Azure responses, we performed a `A` record DNS query to obtain the IP address of the on-site SFB service.

In Table 4.1, we show the result of the measurements. Roughly 77% of companies and 58% of universities had servers that would be affected by some form of account lockout attack. For organizations that use Azure-hosted services, an account lockout attack targeted at these servers would affect other services that consult the Azure Active Directory server, but they would not affect services that communicate with an on-site

Active Directory server because the uni-directional Azure AD server connection with an on-site AD server does provide the capability to share this information. However, attacks against services that communicate to a non-Azure AD server would affect all services, since non-Azure AD servers propagate an account lockout organization-wide, including to the Azure AD server. Accordingly, attackers looking for the biggest impact may target non-Azure AD servers when possible.

4.5 Discussion of Potential Countermeasures

Our measurements demonstrate that account lockout attacks can be crippling for an organization and that many large organizations are vulnerable to these attacks. However, there are a variety of mechanisms that may be effective at mitigating such attacks. Each method has strengths and limitations in terms of ease of deployment, legacy compatibility, visibility and impact on end-users. We discuss potential methods and implemented two of them, one which modifies a residential router and another that leverages user provided secret information, to show to what extent we can prevent account lockout attacks.

Countermeasure: Private Usernames. An account lockout attack requires knowledge of the target username. In practice, gaining this knowledge is often trivial. For example, Alice's username might be `alice` and her email address may be `alice@example.com`. Intuitively, if the username becomes harder to guess then lockout attacks become commensurately harder for the attacker to execute. Private usernames offer tangible benefits. The approach is backwards-compatible with all existing infrastructure, it avoids lockout attempts on the username, and incurs no additional computational overheads or infrastructure. However, the approach may sacrifice end-user convenience for this computational efficiency. In particular, end-users will now need to manage multiple identifiers and know when to enter their private username and when to use their public email address. Further, organizations may need to reconsider how access control systems and resource sharing will work when a username is

intended to be kept private from an employee's coworkers. Finally, transitioning to a private username schema may be prohibitively disruptive for organizations that have a large number of users and legacy systems.

Countermeasure: Multi-Factor Authentication. Another countermeasure is to ask the user to provide additional secret information as part of a multi-factor authentication (MFA) scheme, such as biometrics, hardware tokens, or one-time pass codes that are transmitted via a smartphone application. MFA-based approaches are effective at distinguishing legitimate users from attackers, assuming the attacker has not compromised all the factors. Further, they are widely deployed so users are already familiar with the process and the usability cost is relatively low compared to the security benefits. Unfortunately, multi-factor schemes alone cannot solve the problem of account lockouts. Intuitively, this problem is not necessarily a limitation of multi-factor authentication but of the lockout policies themselves. In other words, most lockout polices only account for the *number* of failed attempts and not the *kind* of information used in the attempt. Consider Active Directory's multi-factor authentication interface; this workflow allows a username and password to be used in conjunction with a second factor verification via smartphone application or text message. Failed authentication attempts still lead to an account lockout as the second factor is only used if the provided username and password are valid. In short, the second factor does not influence the server's decision to lockout an account.

Countermeasure: Observed Characteristics. The above approaches rely on the user to provide private information as proof. An orthogonal approach is for the authenticating server to use historical information related to the user's behavior or observable connection characteristics. For example, Eriksson et al. [40] studied geographic detection based on IP addresses. The primary limitation of such approaches is they must be tuned carefully to balance between false negatives (allowing an attacker to authenticate) and false positives (preventing a legitimate user from authenticating).

4.5.1 Distinct Authentication Pools

The goal of this work is to incorporate and augment existing authentication approaches. Our proposed countermeasures are based on the following observations. First, existing authentication mechanisms fail to stop account lockout attacks because the problem lies with the lockout policy not the mechanism. Second, account lockout policies should base lockout decisions on the totality of information rather than a simple boolean log of attempts. Third, the proliferation of legacy systems means that organizations are more likely to adopt defenses (at least in the short term) that do not require changes to the end-user software or existing authentication servers.

We codify these observations into a proposed authentication scheme based on *distinct authentication pools*. This scheme is designed to leverage historical activity, network proximity, and secondary credentials to maintain separate authentication risk pools with their own lockout thresholds and failed authentication attempt counts, as shown in Figure 4.3. Each pool maintains a separate counter and threshold which can be configured to meet different security requirements. To make authentication pools immediately applicable to existing systems, we use security middleboxes to implement the key functionality without requiring changes to the services or Active Directory servers. Importantly, this scheme allows a user to authenticate even if there is an on-going lockout attack.

We also propose and implement two novel, and orthogonal, authentication mechanisms to serve as the basis for two of the authentication pools. The first is a token-based mechanism that transparently authenticates requests originating from a user's residential network. The second proposed mechanism leverages a user-supplied credential that effectively turns a public username into a private username. We discuss the design of both mechanisms below.

4.5.2 Protecting Requests from Residential Networks

Websites commonly utilize HTTP cookies to recognize a previously authenticated user. Our proxy server authenticates users by such cookie values and places them in a sep-

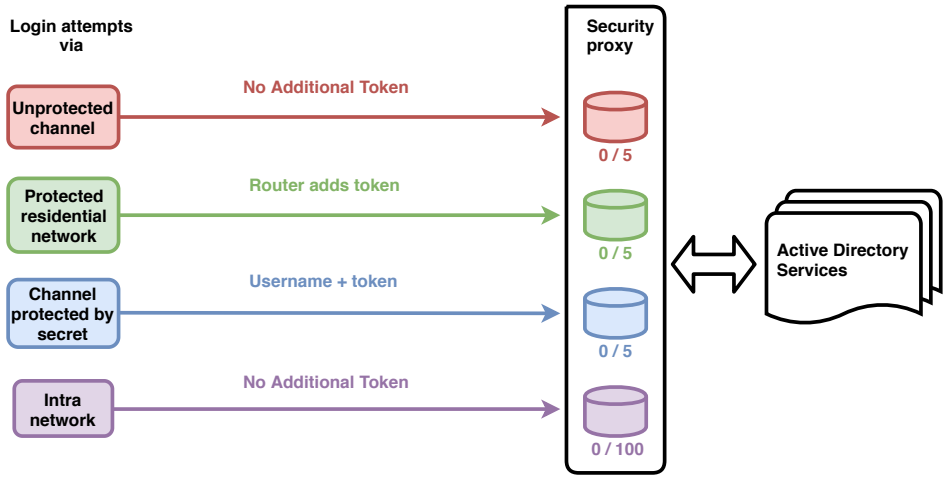

Figure 4.3: With a security middlebox or proxy, an organization can create separate authentication thresholds and authentication attempt counts based on factors that may indicate the user's legitimacy. This diagram depicts four authentication pools based on the presence of tokens, manual authenticators, and on-site presence. The proxy can send commands to help account management in Active Directory, such as unlocking an account, restoring the original account lock state, and checking the account status.

arate pool, which is with different level of security than the pool for unauthenticated users. However, cookies-based methods work only for web applications. We propose using the approach from TCP Fast Open (TFO) [24] standard, which leverages TCP options in the SYN packet to help a server recognize a client and fasten the TCP re-connection. Similar to TFO, the cookie-based method we propose is not an authoritative authenticator. It is not resistant to on-path attackers. However, our method provides sufficient information to put users in the corresponding risk pool.

The authentication pool proxies can be implemented using the TLS "peek and splice" technique [140], in which the proxy is on the route to the protected application servers and has the private TLS keys associated with each server. This allows the proxy to decrypt the traffic, extract the username, and validate tokens and cookie values. If a token is present, the proxy server can then issue commands to the Active Directory server to determine whether the account is locked out, and if so, temporarily lift the lock. It can then re-encrypt the request with the tokens extracted and send it to

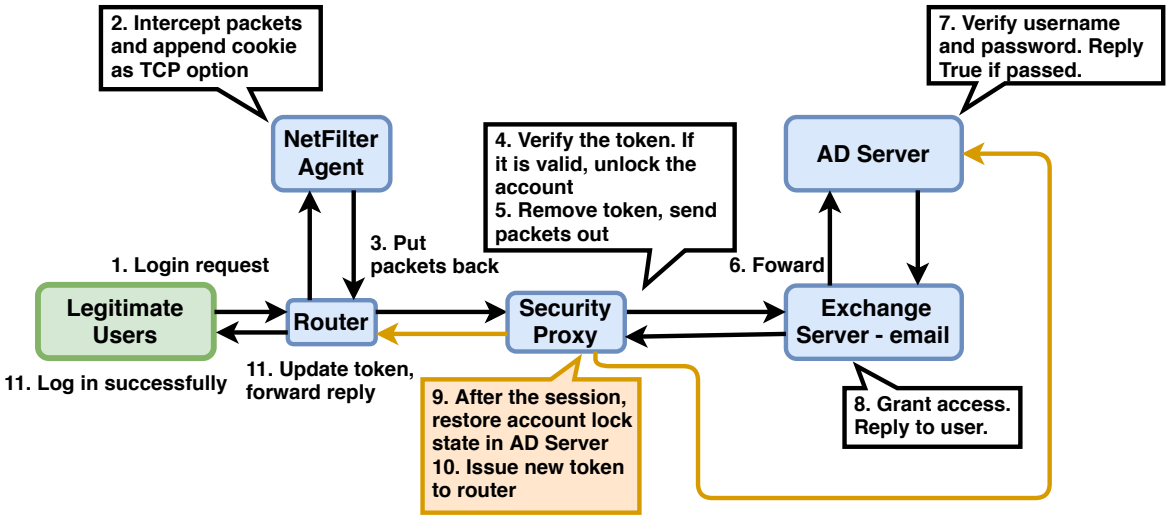

Figure 4.4: With a residential middlebox, tokens can be automatically supplied and stored by examining packets to and from application servers.

the application server to process the authentication attempt. Once the authentication result is sent back to the proxy, it can re-lock the account, if it was previously locked.

In our implementation, to check if an account is locked and obtain the number of failed authentication attempts, the proxy speaks the LDAP protocol with the AD server. Using the `python-ldap` library [36], the security proxy checks the account status via the `ldap.search()` function on the `badPwdCount` attribute with administrator privileges. To unlock the account, the proxy server uses the `ldap.modify_s()` function and sets value of `lockoutTime` to 0. Finally, to restore the failed authentication attempts, the proxy server can use an invalid password to send the same number of prior authentication attempts.

For cookies using TCP options, the process can proceed in a fashion similar to TCP Fast Open (see Figure 4.4). Organizations can provide some employees with a modified router that will act as a middlebox that manages TCP cookies for the user. When the user accesses the organization's servers, the router checks to see if it has a cookie for the destination. If so, it adds the cookie as a TCP option with the request. The security proxy can then extract the cookie and perform the appropriate account unlocking operations before sending the request to the application server. Upon receiving a positive authentication response from the application server, the security proxy can

generate a cookie value and insert it as a TCP option. The user's router will then extract the cookie, store it locally, and then forward the response to the user.

4.5.3 Supporting Private Usernames

A token can also be user-supplied. The secret code can be shared via email, on an employee's badge, or in new employee orientation materials. When a user supplies the username for authentication, they can insert a delimiter followed by a non-public value that the proxy device can detect (e.g., `username+code`). The non-public value can be arbitrary set by the proxy administrator and changed if it was ever learned by an adversary and used in an account lockout attack. Each account should have unique secret token used for login. Since the value is only used to circumvent a lockout attack, the value could be one the user could readily access or remember, such as the user's associated employee ID or badge number. When processing authentication attempts, the proxy can search for the delimiter in a username, extract the non-public value, and verify it. The proxy can then forward the authentication request with the delimiter and code stripped from the username field for authentication by the AD server.

The user-supplied token approach has the value of being easily implemented and supported in legacy systems with the help of a proxy. When a user's account is not being attacked, they need not provide the token since the default authentication pool will be unlocked. After their account is locked due to an attack, the user only needs to type a short addition to their username to gain access. While this approach does require user training, during support calls to IT staff, the helpdesk staff can quickly remind users of the override. Finally, when setting up automated clients, such as email programs or smartphone applications, the user can choose to enter the token to ensure continued access during attacks without incurring any inconvenience. The user-supplied token avoids the complications of legacy usernames and access control that are associated with the private username approach while attaining similar benefits.

4.6 Evaluation of the Authentication Pools System

For our evaluation, we consider both the security effectiveness and the performance of using authentication pools. We focus on the two authentication mechanisms proposed in the preceding section: 1) authenticating requests from residential networks and 2) providing support for private usernames with tokens.

4.6.1 Implementation and Experimental Setup

For our baseline experiments, we configured a Windows Server 2016 Standard server to run the Active Directory service on a virtual machine with two cores and 8 GB of RAM. We configured an Exchange 2010 server on another Windows Server 2016 Standard VM with two cores and 8 GB of RAM. The Exchange server used the Active Directory server VM for authentication and the POP3 service for checking emails. To test the proposed countermeasures, we implemented two different middleboxes: the authentication pool proxy and the residential router. Our client is another Ubuntu 16.04 server VM that runs a POP3 Python script client that attempts to use our Exchange email server. While the deployed enterprise configurations will differ from our experimental setup, we believe this setup is sufficient for evaluating the security and performance characteristics.

The pool proxy implementation supports both the TLS "peek and splice" and the private username authentication mechanisms described in the preceding section. We use an Ubuntu virtual machine with 2 cores and 4096 MB of RAM and the `tcpproxy` library [70], which allows the interception and modification of packets. The `tcpproxy` library provides the functionality to wrap normal socket communication into TLS protected communication when a private key is imported. The pool proxy uses a copy of the Exchange server's private key. The decrypted payload contains the account information. However, when `tcpproxy` uses SSL-wrapped sockets, it only provides the decrypted packet payload without network or transport layer headers. Using the `libnetfilter_queue` library and the `iptables` tool, the pool proxy intercepts the packets and extracts any TCP options from the packet headers before forwarding, to

`tcpproxy` to see any token TCP options. If they are present, as shown by the green line in Figure 4.3, the proxy switches the request to a separate authentication pool.

To support private usernames, the pool proxy also checks each username for a `+` character and extracts the subsequent code. If the code is valid, the system recognizes the connection as associated with the blue line in Figure 4.3. When the verification of a security token succeeds, the security proxy issues commands to the AD domain controller to unlock the account if additional attempts are permitted for that pool group. After the credential verification completes at the domain controller, the security proxy sends commands to restore the account lock status. When the TCP option is used, the security proxy generates a new token and appends it to the reply packet as a TCP option. The router can thus extract the new token and store it for future use.

For the residential router, we use an Ubuntu 16.04 server VM configured with single core and 2048 MB of RAM. We then created a C program that uses the Linux `libnetfilter_queue` library and `iptables` to intercept traffic to and from the residential network. The program is designed in a fashion to allow it to be ported to commodity router hardware. The program uses the packet's destination address to determine if it is a known organizational application server. If so, supplies any associated token as a TCP option. The program also looks at the packet's source address to determine if it is an application server, and if so, the program looks for TCP options containing a token. If one is found, the router stores it for future out-bound packets and removes the option before sending the packet towards its destination. In Figure 4.4, we provide a diagram of this process.

4.6.2 Security Effectiveness

Using a methodology similar to our measurements in Section 4.4.2, we create a tool that emulates an attacker trying to trigger an account lockout. We use a Python script with the `poplib` library [1] to create a POP3 client. That script initiates 100 authentication attempts in rapid succession. Even with the relatively permissive NIST guidance, that volume triggered a lockout.

We unlocked the account and reset the failed attempt counter to zero. We then

replicated this process using web requests to the Outlook Web Application (OWA) interface on the Exchange server manually. The outcome was the same: the legitimate user was unable to authenticate to either OWA or POP3 because the account was locked in AD.

We note that the account lockout through the OWA portal may be affected by credential caching in Microsoft's Internet Information Services (IIS). A parameter, `UserTokenTTL`, defines how long the IIS server should cache authentication tokens. The default cache flush delay is 15 minutes [108]. With that default, an attacker has 15 minutes to make unlimited password guess attempts. During that attack, the failed authentication attempts counter increases. After it hits the threshold, access via services like POP3 is denied because the account is locked, but the cached credential still allows a user to authenticate via the OWA portal. In effect, this delays the account lockout attack from affecting the OWA portal, but still allows an account lock to propagate throughout the rest of the organization. After the cache period ends, the account will also be locked on the OWA portal. Accordingly, this caching does not ultimately affect the attack's success.

For the residential router mechanism, we primed the router by performing a legitimate authentication attempt. We then cleared the account lock status and authentication attempt counts. We then ran the attacker script without presenting a token value. However, when the legitimate user attempted to login, the router supplied the previously obtained token and the security proxy correctly unlocked the account temporarily to process the request before re-locking it. We found that the legitimate user was able to authenticate without impediment despite the ongoing attack.

We repeated the TCP option process with an adversary that tried to forge TCP tokens, by supplying random values. As expected with the low likelihood of guessing a 10-byte token value, we found that adversary never generated a correct token. The security proxy accordingly ignored these tokens and the attacker remained locked out. However, when the legitimate user attempted to authenticate, the proxy recognized the token and properly allowed the attempt. Finally, we repeated these experiments using the user-supplied tokens. The legitimate user provided a username of the for-

Table 4.2: Our security evaluation determined the effectiveness of the TCP option and embedded code countermeasures. Across 20 trials, both approaches correctly allow legitimate requests and deny malicious attempts.

	Router TCP Option		Username + secret	
	Token Valid	Token Invalid	Token Valid	Token Invalid
Allows access	20	0	20	0
Denies access	0	20	0	20

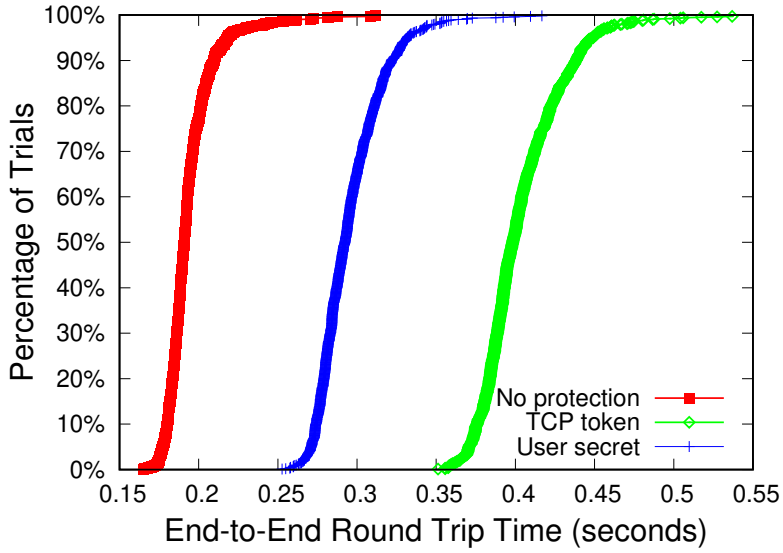

Figure 4.5: We evaluate and compare end to end delay among three cases: red line shows without any middlebox or proxy, green line shows inserting tokens via TCP option and blue line shows username and secret token.

mat `username+token` when attempting to authenticate. In our first experiment, the attacker did not provide tokens and in our second, the attacker attempted to guess tokens randomly. As with the TCP-option experiments, we found the attacker was unable to unlock the account while the legitimate user had unimpeded access to authenticate. In Table 4.2, we show the numerical results of our experiments. In each experiment, we conducted 20 separate trials and the results were consistent.

4.6.3 Performance Evaluation

To determine the performance impact of the approach, we use end-to-end timings. We measured the amount of time required to complete an authentication request using the `time.time()` function, provided by Python, in the legitimate client's script. We measured the timing without our countermeasures implemented and with them in place. This allows us to determine the sum of all overheads present in the system. We show the results in Figure 4.5.

When comparing our the two authentication mechanisms, we found that the TCP option countermeasure added around 180 milliseconds of latency while the user-supplied token added 80 milliseconds of latency, compared to the same system without a countermeasure present. These overheads are so small they are unlikely to be perceived by an end-user. We believe the user secret scenario is faster in our implementation because it does not need to examine TCP options, extract tokens or append TCP tokens.

Based on these results, we find our countermeasures provide effective security benefits without introducing noticeable latency.

4.7 Outcomes

In this work, we explored the extent to which organizations are vulnerable to account lockouts and the impact that the lockouts could have. Looking only at the deployments of Microsoft Active Directory, we found that the majority of top companies and universities had an exposed authentication portal that would enable an attacker to launch an account lockout. Through our experiments with a partnering organization, we demonstrated the feasibility of such an attack in a production environment. We then introduced a suite of countermeasures and compared the benefits. We found that both user-supplied tokens and middlebox-added tokens would be effective and would add acceptable delays or performance overheads.

Chapter 5

Incentivizing Network Hygiene via Distributed Attack Reporting

5.1 Introduction

Residential networks typically lack dedicated security hardware. Instead, they typically leverage network address translation (NAT), which has the side-effect of blocking inbound connections by default, as a security measure [38]. However, NAT was not designed for security and has substantial limitations when used in this fashion [86]. Further, attackers are well aware of NAT implementations and design their attacks to circumvent them [150].

The presence of vast swaths of insecure networks makes the Internet, as a whole, less secure. Mobile devices may be compromised on residential networks and later physically move into corporate networks where the contamination spreads [183]. Insecure networks also offer opportunities for adversaries to anonymize their traffic and attack with impunity.

Although distributed denial-of-service (DDoS) attacks are a challenge for residential networks, they also represent an opportunity to inform home users that their security has been breached. Typically, residential users are unaware when their devices are compromised. Owners of devices participating in the Mirai botnet had no way of knowing their systems were attacking others on the Internet. If informed of a compromised

device, many owners would remediate the issue to avoid negative consequences for themselves and others. In DDoS attacks, a victim learns the IP addresses of a large set of attacking hosts, which are typically compromised machines organized in a botnet. If victims could easily report these attacks, the device owners could learn about the compromise and remediate the devices, lessening the flood and preventing the devices from being involved in future attacks

To address these problems, the research community has explored ways to connect experts with home networks. Feamster [44] recommends outsourcing all security and network management to experts. With Project BISmark [60], a remote service provider is able to periodically measure network performance across a collection of residential networks. Taylor et al. [163] found that cloud-based software-defined network controllers would have acceptable latency for the vast majority of residential networks in the United States even when deploying fine-grained flow control. This suggests external providers could feasibly manage network access control for residential networks.

In their Active Internet Traffic Filtering (AITF) work [7], Argyraki and Cheriton proposed a cooperative filtering scheme in which victims could issue filtering requests to Internet Service Providers (ISPs) close to attacking hosts, preventing unwanted traffic from saturating bottleneck network links. A significant challenge for the AITF work was a lack of incentives for deployers: the filtering ISPs would have to modify their routing infrastructure to support AITF, but those ISPs did not directly receive benefits in return.

In this work, we explore opportunities to provide all deployers with incentives to participate. We combine fine-grained SDN access control with AITF's cooperative traffic filtering techniques. We explore a system where SDN controllers from different service providers, which can either be hosted by ISP or third party, could automatically discover each other, report attacks, and cooperate on network filtering. In doing so, we make the following contributions:

- **Exploration of Incentives:** We examine how each of the stakeholders in the system benefit from this approach. These stakeholders include the owners of compromised devices, the attacked victims, and SDN service providers for both

the attackers and victims (Section 5.3).

- **Architecture Design and Implementation:** We provide an overview of how the approach works, a protocol that allows the SDN controllers and consumer-grade residential routers to identify and verify each other, and implement a distributed attack reporting system (Sections 5.2 and 5.4).

- **Security and Performance Evaluation:** We evaluate the approach's reporting from a security perspective and analyze the performance at each device (Section 5.5). We find that the approach can report and verify a compromised host within seconds and introduces few performance overheads in the controllers and routers.

5.2 Architecture and Design

We now describe the architecture and components we will use to combine SDN techniques with distributed attack reporting. We describe the incentives for this approach in Section 5.3 and describe the implementation details of this design in Section 5.4.

In Figure 5.1, we show an example network configuration. In this diagram, we have an attack target, which we call the "victim host," that is is connected via a router to the Internet. The victim router can be a commodity consumer-grade wireless router, such as the variety used in 76% of households [10]. The router is configured to run Open vSwitch [132], a popular OpenFlow agent implementation. It communicates with an OpenFlow controller, labeled the "victim controller," which is run by a security service provider. The victim controller could run in a public cloud data center, ISP-hosted data center, or other location. In our model, we assume that the controller is not in the same LAN with the victim host or router. The victim router and controller are connected to the Internet via ISPs, which we merge together in a simple box labeled "Internet." We depict the bottleneck bandwidth link as being between the victim router and its ISP, since this is the common "last mile" link that constrains throughput.

Since network attacks are illegal in many jurisdictions, the actual perpetrator of a

DDoS attack will often use compromised systems to actually send the attack traffic to the victim. We call the compromised device a "pawn" to indicate that it is working for an attacker (though the device owner is unaware that this is occurring). In our example network, we show a pawn host connected to its own router, which connects with its own controller. The pawn could share an ISP with the victim or use an ISP that is distant in the topology. Multiple pawns could be working in concert as part of a botnet. We consider the pawn to be fully controlled by the attacker. However, the pawn's router and pawn's controller are managed by a security service provider that works for a legitimate device owner who does not realize the device has been compromised. Accordingly, the pawn's router and controller may curtail the pawn's activities upon receiving evidence of a compromise.

In the event of an attack, the victim's controller will initiate communication with the pawn's controller to request filtering of the traffic. The two controllers may filter the traffic with an OpenFlow flow modification (`FlowMod`) message that indicates that packets matching the unwanted flow's IP addresses and ports should be discarded. The mechanisms to discover and verify the responsible controllers will be described later in this section.

5.2.1 Threat Model

The defender's goal is to report and stop unwanted traffic as quickly as possible. We assume the defender has an existing technique to determine which traffic is unwanted. The victim can request filtering from the victim router–explicitly or via continual connection reset messages–trusting the security service provider and considering the victim router and victim controller to be a trusted computing base (TCB).

The owner of the compromised device likewise trusts the security service provider operating the pawn router and pawn controller and considers them in a TCB. The pawn's owner wants to know if the pawn is ever compromised and to prevent it from engaging in attacks. The pawn's owner does not consider anyone else to be trusted. If a given destination will block communication with its own hosts anyway, the pawn's owner gains upload bandwidth capacity by filtering that traffic at its local router when

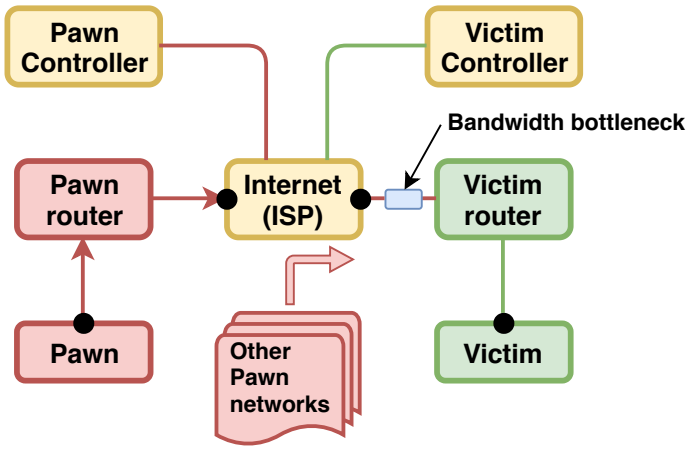

Figure 5.1: An overview of the components in our system. The victim and pawns are connected via their respective routers and managed by their respective controllers. The black dots represent measurement points in our evaluation.

participating in an attack.

The security service providers for the victim and pawn may be mutually distrusting. In that case, the providers will need a way to securely identify and verify each other. A service provider only needs to confirm that the other service provider is an agent for the remote communicating party. In other words, if the other service provider can prove it can intercept packets destined to the remote communicating party, then it is allowed to request that filtering be done closer to the source. The service providers assume the links between the two controllers, between the two routers, and between the routers and controllers are uncompromised. If this assumption is violated, fake reports and filtering requests could be introduced. However, those on-path adversaries would already have the capability of simply discarding traffic.

5.2.2 Protocol Design

In Figure 5.2, we provide a sequence diagram of communication between a pawn and a victim. When the pawn initiates its connection to the victim, the request reaches the pawn's router. Since this is the first entry in the flow, and because the pawn's router uses fine-grained flow entries, the pawn router does not find a matching flow entry and elevates the request to the pawn controller. The pawn controller decides whether

to allow the traffic or not. If it is allowed, the pawn controller generates a special value, `SRC_COOKIE`, that can be used by recipients or other controllers to prove that it received the packet. The controller then sends the packet back to the pawn router using a standard OpenFlow `PacketOut` along with any `FlowMod` messages needed to authorize the flow.

Before transmitting the packet as ordered, the pawn router first inspects the message in the `PacketOut` to search for any cookie values the controller may have inserted. If it finds one, it stores the `SRC_COOKIE` in a local database for future consultation. It then transmits the packet towards its destination.

The packet is then transmitted across the Internet until it reaches the victim router. The victim router detects any new flows and searches for any cookie values associated with them. If one is detected, it records the `SRC_COOKIE` to a local database. If it ever determines that the flow needs to be filtered in the future, it can use the `SRC_COOKIE` as an authenticator value.

When a victim router recognizes that traffic should be filtered for a victim host, it first filters the packet locally by adding a drop rule. It then contacts the victim controller and reports the filtering request, along with the `SRC_COOKIE`, if any, that the victim router previously stored when the flow was being created. The filtering request contains the full flow information (the IP addresses of the pawn and victim, the transport protocol and associated transport layer ports). The victim router sends this request to a pre-defined port on the victim controller.

The victim controller listens for filter requests on its pre-defined port. It examines each part of the filtering request and then attempts to contact the controller associated with the pawn. It does so by sending a packet to the pawn's public IP address using another pre-defined, globally-known port value for filtering requests. The victim controller supplies all the information it received from the victim router and generates its own value, `DST_COOKIE`, that will serve as a nonce and authenticator for the pawn controller. It then sends the request to the pawn router.

The pawn router listens for filtering requests on the well-known request port. If it sees requests to its own IP address (if the router uses NAT) or to IPs associated

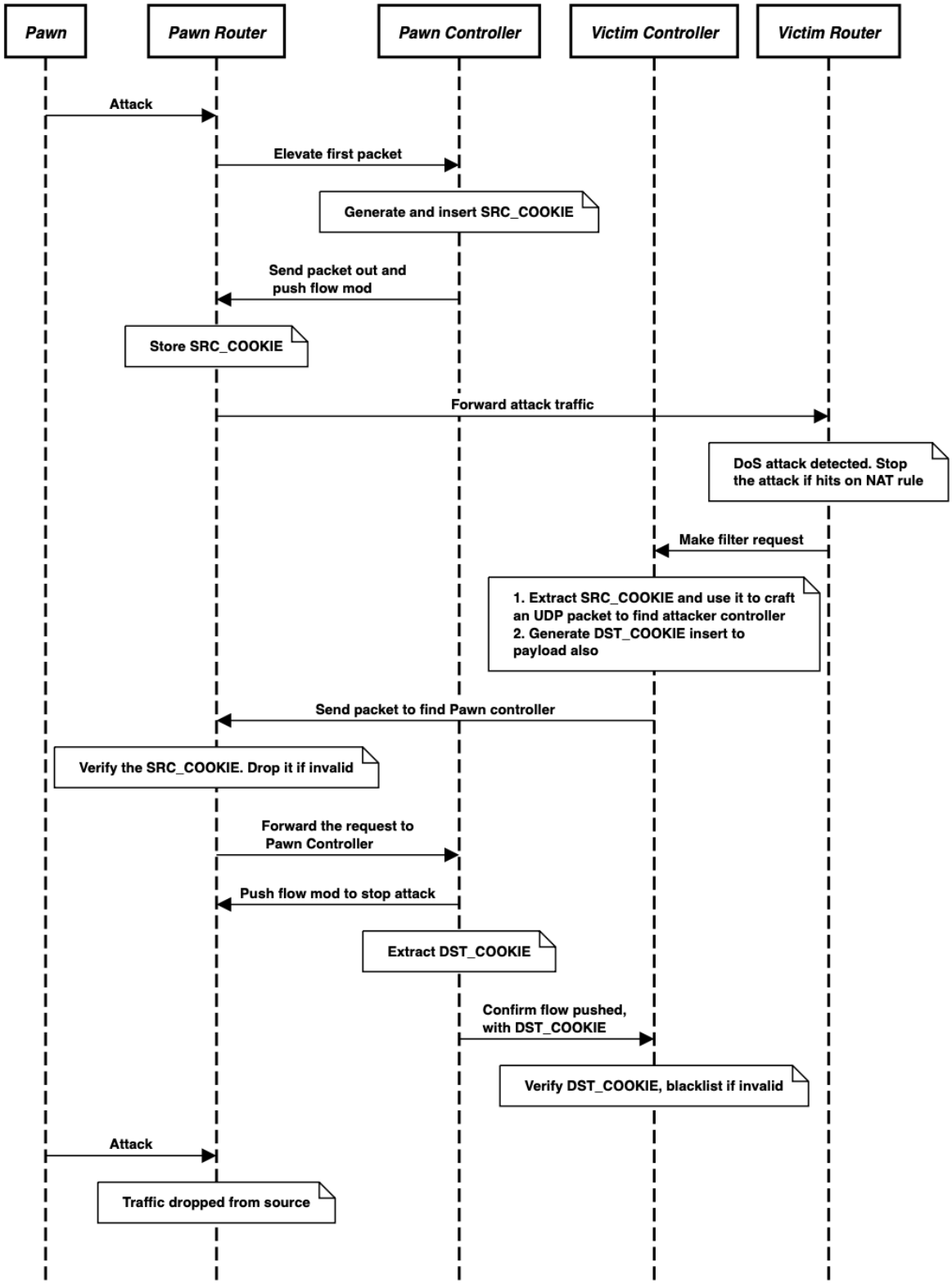

Figure 5.2: A sequence diagram overview of the reporting protocol

with its own hosts (when NAT is not in use), it extracts the packet and determines whether it is a valid filtering request. It examines the packet to determine the flow that should be filtered and the associated `SRC_COOKIE`. If it finds the extracted cookie in its database, and the cookie is associated with the indicated flow values, it knows the filtering request is valid and sends it on to the pawn controller. If there is a mismatch in cookie value and flow, or if either of the fields is invalid, the router simply drops the filter request packet.

Upon receiving the filter request packet from the pawn router, the pawn controller also verifies the `SRC_COOKIE` and flow details match. If so, it sends an OpenFlow `FlowMod` to the pawn router that discards any subsequent packets in the flow. It then uses the victim controller's IP address, which is the source of the request, to send an acknowledgement of the filtering request. By including the `DST_COOKIE` that the victim controller supplied, the pawn controller confirms its agency for the pawn. When the victim controller receives this acknowledgment, it can verify the `DST_COOKIE` and know that the pawn controller has received the report and may start filtering the traffic.

5.3 Exploration of Incentives

While residential users care and are willing to make sacrifices to improve their computer security [85, 105], they need tangible benefits to encourage their use of security technology. For security service providers to help these residential users, they need a way to prove their value to these end users to justify their service charges. In this section, we explore the incentives for both these stakeholders.

5.3.1 Incentives for Residential Users

Home network users want to trust their device security (i.e., devices are not compromised). Our approach lets victims report abnormal network events, which are linked to attack source devices and can inform the owners of any compromised devices. Our system has the ability to timely track attacks and report compromise to device owners.

Bandwidth is an important resource for home network users and their network ex-

perience. However, when a device is compromised, it usually uses bandwidth resources to launch attacks, which cause network congestion. Upstream bandwidth, in particular, is limited for a home network [48]. Based on the victim's report, the attacker's router stops the attack at the origin, thus preventing the compromised devices from abusing limited upstream bandwidth.

Although the victim of a DDoS attack would gain all the benefits that come with remote filtering, the effectiveness can be weakened by the pawn networks that do not deploy this technique. Unfortunately, it is unrealistic to assume such a perfect deployment in practice. However, since our method can effectively block malicious traffic sent by bots, with wider deployment, the costs of a DDoS attack will increase. The scarcity of those unblocked bots will command a price premium. Over time, network service providers begin prioritizing traffic from the networks that employ the approach. It can further incentivizing users to deploy the technique. The method will affect on the DDoS attack economics, while unlikely to yield tangible benefits for the victim of any given attack in the short-term.

5.3.2 Incentives for Security Service Providers

To provide more convincing and accurate information to home network owners, service providers may need more information. For example, service providers can inform network owners about the destinations that their IoT devices interact with and the number of networks. They can track the collected reports and associated information. By showing users that they are collecting attack reports from distributed spots, users are more likely to trust that the service provider provides quality security services.

When network traffic is denied out the home network, the service provider can then provide more meaningful reasons and feedback, such as the reported attackers, the device involved and the number of reports, to device owners. Although it is still hard for average users to understand the technical details, they may be concerned by the quantified reports and the number of victims. Accordingly, the reports may make users take action. Service providers can provide guidance to help their customers remove the threat from their devices. Further, with aggregated information on controllers, service

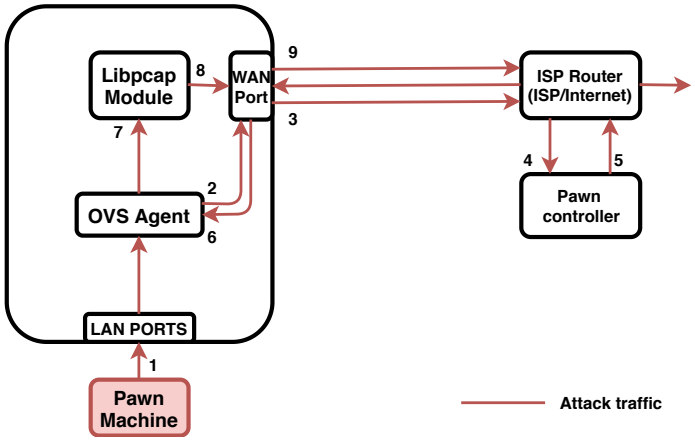

Figure 5.3: This figure shows the interactions between pawn host and the OpenFlow router and controller. The process begins with the attack traffic from the pawn (line 1) which the pawn OVS router elevates to the pawn controller (lines 2, 3, 4). The pawn controller then generates a `SRC_COOKIE` and sends it back to the OVS router (lines 5 and 6). The OVS agent then transmits the packet to the victim (line 7), which is intercepted by a `libpcap` agent that stores the cookie value before sending it onward to the victim (lines 8 and 9).

providers can identify command-and-control network traffic with network analysis.

5.4 Implementation

To explore the distributed reporting approach, we create an implementation on consumer-grade residential network routers. We flash three TP-Link Archer C7 v2.0 routers with the OpenWrt [167] firmware, which is a popular open source router operating system. We configure two of the three router to run the Open vSwitch (OVS) OpenFlow agent. We host the pawn controller and victim controller in VMs on a laptop running OS X. That laptop has four cores and 16GBytes RAM. Each controller runs POX [114], a Python-based OpenFlow controller and stores records using a MySQL database.

To model an upstream ISP router, we place the router that is not running OVS in the center of the network to link the other two via LAN ports. We further connect the

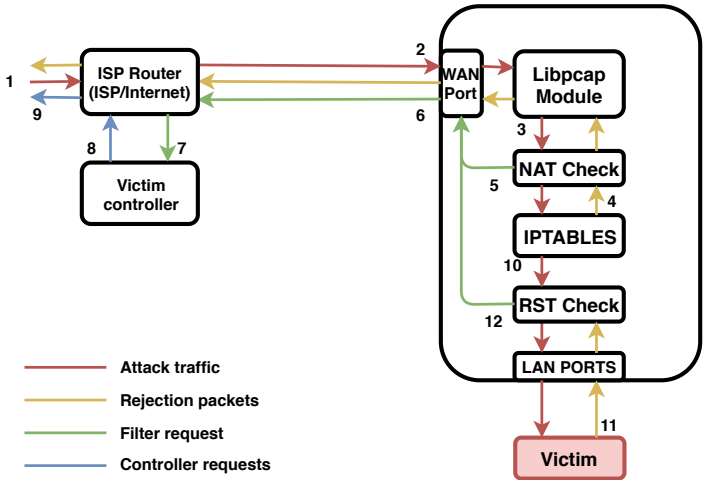

Figure 5.4: The pawn's transmission is forwarded to the victim router (lines 1 and 2). On the victim router, a `libpcap` agent extracts and stores the `SRC_COOKIE` locally before they are processed by the router's NAT and firewall components. If the packet is sent to a port without a NAT mapping, the router issues a filter request to the victim controller (lines 4, 5, 6, 7). If a NAT entry exists, the router sends the packets to the victim (line 10). If the destination issues a large number of resets (line 11), the router likewise requests a filter request (lines 12, 6, 7). When processing a filter request, the controller inserts a `DST_COOKIE` and reports the attack to the pawn controller (lines 8 and 9).

laptop running the controller VMs to a LAN port on the ISP router. This upstream ISP router is configured so that each physical LAN port is on a separate VLAN and we use the `tc` command to constrain the downsteam bandwidth to 5Mbps to allow attack modeling without any testing hardware being a limiting factor. We manually configure the IP addresses, subnets, and routes on the routers and configure IP aliases on the pawn host to enable a single system to simulate multiple attacking hosts.

One of the two remaining TP-Link routers is designated as the pawn's router and the other is designated as the victim's router. The pawn's router elevates new flows to the pawn controller to obtain the `SRC_COOKIE` as described in Section 5.2.2. Upon receiving a response, the pawn router stores the `SRC_COOKIE` locally in memory and then sends the packet on to the victim (Steps 2-6 in Figure 5.3).

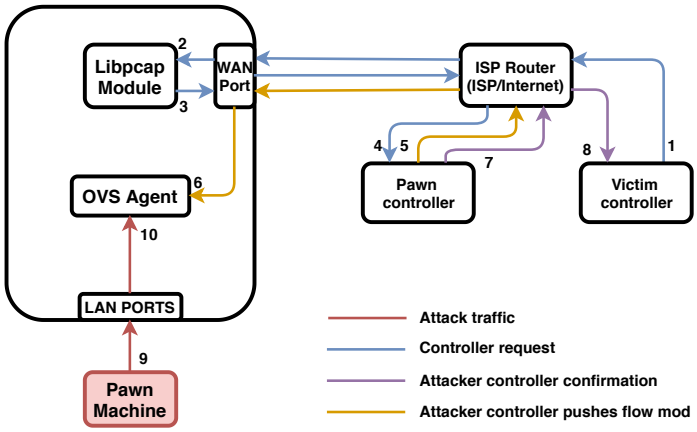

Figure 5.5: The victim router's attack report (line 1) is sent to the pawn router's `libpcap` module (line 2). That module extracts and verifies the `SRC_COOKIE` in the request and compares it to the supplied flow information. If the request is valid, the router sends the request to the pawn controller (lines 3 and 4). The pawn controller again verifies the `SRC_COOKIE` and flow information. If valid, the controller records the report and pushes a `FlowMod` to the pawn router to filter the flow (lines 5 and 6). Meanwhile, it sends an acknowledgment to the victim controller (lines 7 and 8). Subsequent packets to the victim from the pawn are thus dropped due to the new rule (lines 9 and 10).

The victim router is configured to examine incoming packets for IP options that could be associated with a `SRC_COOKIE`. Using the `libpcap` library, the router identifies cookies and associates them with flow data, including the source IP address, the transport layer protocol, and the transport layer source port. To test our system, we implemented an anomaly detection program that runs on the victim router. The program monitors packets to identify any indications the traffic is unwanted, such as ICMP errors or TCP reset packets, and filters the flow when a threshold is reached (e.g., 20 rejections/second in our experiments). Prior work analyzing TCP reset rates by Arlitt et al. [8] and Bilal et al. [17] shows some variation in reset rates based on client activity, but a high volume of resets when bandwidth is saturated is a potential attack indicator.

While the victim router's downstream bandwidth may be saturated during a DDoS

attack, the victim router can control the upstream channel and prioritize attack reports over any other messages. Accordingly, the victim router can locally filter the unwanted flow and send a request to the victim controller to filter the traffic at the source, if possible. To do so, the victim router supplies the controller with the `SRC_COOKIE` that it previously logged, if any, along with details about the flow, including addresses, ports, transport protocol.

Upon receiving a filtering request, the victim controller crafts an UDP packet with the pawn's IP address as the destination and a globally-known filter service port as the destination port. In the UDP packet, the victim controller provides the pawn's IP address and port (which may be empty for `ICMP`), the victim's IP address, the transport layer protocol, and the `SRC_COOKIE`. The victim controller also generates and includes a random nonce value, the `DST_COOKIE`, that can be used to acknowledge the filter request. The victim router caches this information in a local database and then sends the UDP packet towards the pawn.

The pawn router monitors traffic on its globally-known filter request port. If it receives a request, it determines if the contained `SRC_COOKIE` is valid and matches the flow, and if so, elevates it to the controller, as shown in Step 4 in Figure 5.5. The pawn controller again verifies the request, and if authentic, sends an OpenFlow `FlowMod` message to the pawn router to deny traffic associated with the flow's IP addresses, ports, and protocol. The pawn controller then sends a message to the victim controller to acknowledge the request, as shown in Steps 7 and 8 in Figure 5.5. Once the pawn router applies the `FlowMod` rule, the attack traffic is filtered in the pawn's LAN.

5.5 Evaluation

We evaluate our system from both a security effectiveness and a performance perspective. From a security standpoint, the approach must rapidly respond to attack reports in a manner that ensures the reports are authentic. From a performance perspective, the approach must minimize overheads and ensure it can scale in the event of an attack.

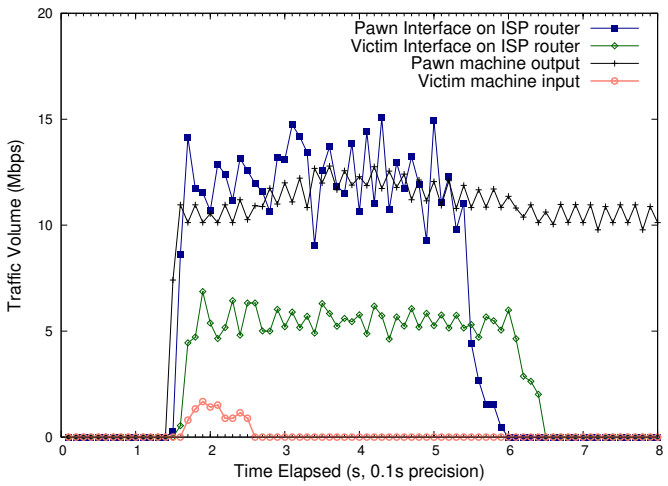

Figure 5.6: When five pawn systems simultaneously DDoS a victim, the traffic spikes and saturates the link between the victim router and its ISP. When pawn routers and controllers respond to source filtering requests, the attack traffic quickly drops off, despite the continued transmissions from the pawn host.

5.5.1 Security Effectiveness

We begin by examining the size of reporting packets that are sent from the victim router to the victim controller. Each report is 100 bytes, of which 58 bytes is the payload of the filtering request. Accordingly, even a 5Mbps upload bandwidth capacity would allow a victim router to report roughly $6,250$ unique malicious flows per second. This would allow a victim to identify and report a large number of compromised machines while a DDoS attack is ongoing.

We next empirically examine the potential attack traffic reduction possible if pawns perform source filtering. For our pawn host, we use a Thinkpad S3 machine with 4 cores and 8 GBytes RAM. We configure the pawn with five IP aliases to simulate multiple attackers that must be filtered individually. On the pawn, we use the `hping3` tool to launch a TCP SYN flood attack at the public IP address of the victim router. Since the victim router does not run public services, the router OS will send back a TCP RST packet in response. We use a separate `hping3` process for each IP alias on the pawn and we set each `hping3` process to send 2 Mbps of attack traffic in order to saturate the bottleneck link between the ISP router and the victim router.

We monitor the traffic volume at multiple points between the pawn and the victim router, as represented by the black dots in Figure 5.1. These measurement points include a sample on the pawn's network interface, the pawn-connected interface on the ISP router, the victim-connected interface on the ISP router, and the victim's interface.

In Figure 5.6, we show the results of our filtering experiments when all five pawns launch an attack simultaneously. The pawn traffic starts around the 1.5 second mark, averaging around 10Mbps (represented by the black line). A queue builds at the pawn router, leading to bursty results (represented by the blue line). The ISP router constrains the throughput of the traffic, which averages around 5 Mbps, as expected (represented by the green line). The victim router quickly notices the attack and applies filters locally, preventing the victim from ever experiencing the full bandwidth of the attack (represented by the red line). The victim router elevates the requests to the victim controller, which starts implementing the filters around 5.5 seconds into the experiment, resulting in a rapid drop of traffic volume at the ISP router on both the victim and attacker interfaces, despite steady out-bound traffic at the pawn host. The outbound queue causes some residual traffic to be sent to the victim router around the 6 second mark.

5.5.2 Performance evaluation

We measure the overhead introduced by our system from different viewpoints. We first measure how much overhead is introduced by inserting cookies into the first packet of each flow. On the pawn machine, we create a UDP client program that sends UDP packets with different source ports to ensure that each packet represents a new flow that is elevated by the router's OVS agent to the pawn's controller. We implement a UDP server at the victim router to echo back each requested packet. We then measure the round trip time (RTT) of 1,000 UDP packets when both the pawn router and pawn controller are processing packets. We configure the controller to create `PacketOut` messages immediately upon receiving a packet without any packet processing. Accordingly, the RTT includes the bidirectional propagation time plus two packet elevations to the attacker controller since both the outbound packet and its reply are elevated

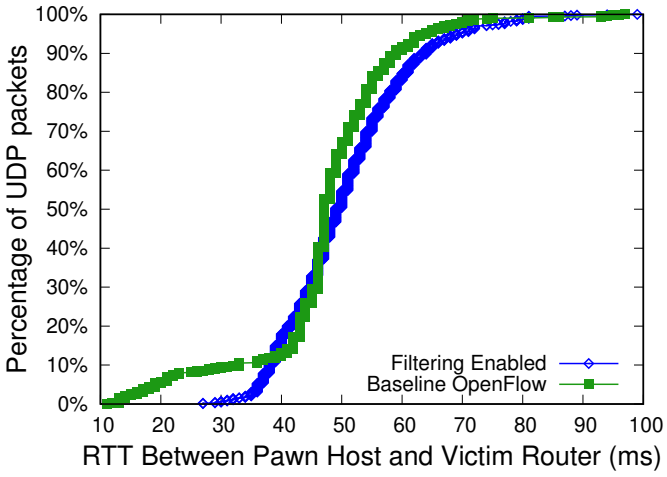

Figure 5.7: The delay between a skeleton OpenFlow implementation and our distributed reporting averages only 3ms across our 1000 trials. This overhead is less than 10% of the overhead associated with basic flow elevation.

to the controller. In Figure 5.7, the RTT is represented by the green line. We see that 90% of packets have an RTT of less than 60 milliseconds. We compare this with the OVS router and OpenFlow running our distributed reporting approach, depicted by the blue line in Figure 5.7. These results show minor differences between the two OpenFlow implementations, with the version running our code taking approximately 3 milliseconds longer than a skeleton OpenFlow elevation. While OpenFlow naturally introduces delays with reactive flow elevation, the delays are only on the first packet in a flow and can be accommodated in production environments. The small differences in RTT introduced by our approach are dwarfed by the base OpenFlow delay and are unlikely to be apparent to an end-user.

5.6 Outcomes

This chapter introduced a distributed reporting approach that allows third-party service providers to detect and filter DDoS traffic for residential users. This model incentivizes each stakeholder in the system: residential users save bandwidth when under attack and learn about compromised systems quickly, while security service providers

respond rapidly to events and provide notification to residential customers. With incremental deployment and immediate incentives, collaborative filtering can motivate users to adopt a platform for security providers to host residential SDN security applications.

Chapter 6

Client Identity with Widespread IP Address Sharing

6.1 Introduction

Virtual private network (VPN) protocols are often used by organizations to allow remote users to access the organization's network as if they were on-site. These protocols have been around for decades [12] and were designed for an Internet that needed cryptography to protect the confidentiality, integrity, and authenticity of the communication payload. In essence, VPNs allow organizations to treat the traffic from a remote worker the same as that from a local worker.

In recent years, the deployment of end-to-end cryptography has grown substantially, with over 90% of web servers supporting the TLS/SSL application-layer protocol [53]. When remote users access an HTTPS server via a VPN, the traffic is encrypted by TLS both between the application-layer endpoints and again between the VPN termination points. Since confidentiality, integrity, and authenticity can be reasonably assured by either of the protocols, having both forms of encryption is redundant. Further, it comes at a cost: 1) VPN servers act as an aggregation point that can become a bottleneck [29], 2) a greater portion of each network packet is used for headers for the cryptographic protocols, and 3) VPN licenses for remote users can be expensive, costing up to $13 per user per month [11].

When reviewing the traditional motivations behind a VPN [50, 75, 101], we find that organizations choose VPNs in order to 1) protect data confidentiality and authenticity, 2) manage communication at the remote endpoint, and 3) simplify access control. With redundant cryptography, the first goal is decreasingly important. The second goal, as we discuss in Section 6.3, can be better achieved with endpoint filtering. This leaves only the third goal of access control, which we explore in detail in this work.

Organizations often implicitly use a host's position within a network perimeter as a factor in determining whether to trust the host or not. As examples, the configuration instructions for web servers [6, 42], email servers [3, 54, 107], and firewalls [152] often describe how administrators can configure the systems to permit only traffic within an organization's IP prefix. Organizations may also use network address translation (NAT) within their networks to assign hosts and devices to private, unroutable address space so that the infrastructure cannot be reached by outsiders without going through NAT devices [137].

While organizations can employ such address filtering within their networks, it may be impractical to do so with remote users. Internet Service Providers (ISPs) often use dynamic addresses for their customers, with lease times that vary greatly [84]. Residential users often deploy NAT in their home networks to share addresses, so authorizing a particular user's IP address would allow others at the same residence access to the organization's resources. Even worse, some ISPs have adopted a competing technology called carrier-grade network address translation ("carrier-grade NAT" or CGN). CGN is used in 92% of cellular networks [138]. These carriers plan to use 5G cellular networks to provide residential network connectivity, with some providers estimating 30 million home networks will be connected in this fashion [170]. In some cases, hundreds of users share an IP address [118] and in others, a single customer's traffic may be simultaneously associated with multiple public IP addresses.

Organizations may use VPNs to mitigate the problems that accompany address sharing. The VPN tunneling approach allows an organization to provide remote access to systems that are highly sensitive or have weaker protections, such as printers, Supervisory Control and Data Acquisition (SCADA) or Internet of Things (IoT) in-

frastructure, or other embedded devices. However, a heavyweight VPNs approach may not be necessary to achieve these goals.

Organizations need a quick and easy way for a network-level identifier to validate a remotely-connecting end user. This validation can be a quick, "first-factor" authenticator that provides evidence of the connecting machine's or network's likely legitimacy. This factor can then be combined with other authentication factors, such as application-layer credentials, on the server endpoint. It must be cooperatively used by both the end-user and the organization to avoid abuse by malicious parties. While identifiers have been used by network providers to identify clients in the past, such as "super cookie" deployments in cellular networks [77], the use of the factor in authentication requires it to be cross-application, dynamic, and under the associated user's control.

In this work, we explore a practical and deployable approach to allow end-users to create dynamic network-level factors for access control. Our contributions include:

- **Dynamic Identifier Insertion:** Using software-defined networking (SDN) techniques, we allow endpoints and residential routers to insert authentication identifiers into traffic in an application-agnostic manner. This approach allows organizations to determine if a user is connecting from a known device or network location. We find that it also allows cryptographically-secure techniques while introducing minimal latency overheads, affecting only the first packet in each network flow.

- **Gateway and Endpoint Validation:** Using the popular `iptables` implementation, we allow organizations to validate clients at either a gateway or endpoint, providing functionality similar to a VPN without creating unnecessary bottlenecks. This approach is able to verify each flow in around 90 milliseconds (of which around 0.8ms is spent at the validator) and adds no on-going performance costs, unlike VPNs.

6.2 Background and Related Work

We provide background and related work on CGN, user identity, and techniques to encode identifiers.

6.2.1 Carrier-Grade NAT (CGN) and Address Sharing

To help with the scarcity of IPv4 addresses and transition from IPv4 to IPv6, providers deploy CGN to share addresses across users [138]. CGN typically combines network addresses and transport layer ports to map traffic to the appropriate end-user. The number of public addresses needed for a given number of end-users can be estimated with formulas; for example, some guides recommend 30,517 public IP address for every 1,000,000 subscribers [118]. In measuring CGN behavior, Richter et al. [138] observed that some CGNs used the same public IP address and varying transport layer ports for subsequent TCP sessions from the same subscriber. In Netalyzr [84], the authors found that more than 60% of the TCP sessions for a given subscriber used different public IP addresses. These guides and measurements show that providers use highly dynamic and widely shared public IP addresses for their subscribers.

With their Revelio tool, Mandalari et al. [98] performed Internet measurements and found around 10% of ISPs deployed CGN. With web server logs and multiple measurement points, Livadariu et al. [93] estimated that around 4.1k of the 17.4k ASes they measured deployed CGNs. Richter et al. [138] found that while only 13.3% of non-cellular ASes use CGNs, 92% of cellular networks use them [138]. In their plans to deploy 5G cellular connectivity, some providers estimate they will serve 30 million residential networks through 5G [170].

These measurements show that address sharing is already widespread on the Internet with a trend towards continued growth.

6.2.2 Conveying Host Identity

Address sharing introduces challenges for a wide range of public IP address-based applications. For example, enterprise-grade firewalls often utilize public IP addresses

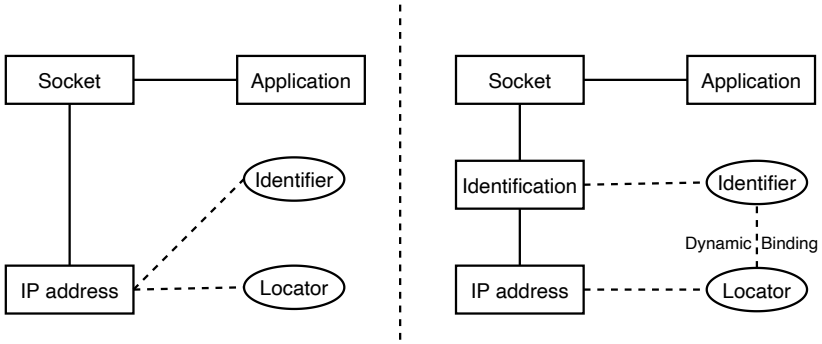

Figure 6.1: Left: The IP address serves as both the identifier and locator. Right: The IP address serves only as a locator.

in policies [152]. Efforts to mitigate DNS amplification attacks use IP addresses in response rate limiting [95]. IP reputation systems, which are used by major email providers, are often used to determine the threat associated with incoming email messages [3]. For example, Outlook indicates that Microsoft's SmartScreen technology is designed for protecting their services [107]. Gmail adjusts email delivery rates according to the sender's reputation [54]. Cloudflare identifies users with a bad IP reputation and challenges them with CAPTCHAs [27]. Such tools may mistakenly assume that IP addresses change infrequently and are unlikely to be shared. This leads to false negatives when attackers move across IP addresses and false positives when innocent people happen to use an IP address previously involved in an attack [66].

Komu et al. [83] surveyed Internet architecture efforts that aim to split the "locator" functionality from network addresses from the "identifier" associated with the system. Figure 6.1 shows how addressing may be used to locate a host while a separate long-term identifier is associated with the system. This separation is important for mobile hosts or for times when addresses change. HIP is a protocol to maintain persistent identity even with dynamic IP addresses [113]. HIP uses a public key to identify end-hosts and uses IPSec for packet tunneling. Since HIP requires HIP-aware gateways to forward packets to the correct destination, the deployment of HIP requires infrastructure changes.

MILSA [124] proposes a Hierarchical URI-like Identifiers (HUI) system to name the objects in a network. As with DNS, MILSA entities can perform lookups in a hierarchical system to map between HUI addresses and host names. Since it intro-

duces a mapping shim layer, MILSA requires endpoint entities to support the protocol. MAT [71] applies a naming space and assigns each host a persistent identifier, the Host Address (HA). When packets are sent and received, the OS network stack maps HA and Mobile Address (MA) to and from, respectively, in the Network Layer. In i3 [154], the packet sender issues an ("id", "data") pair, where "id" is the receiver identifier. The server raises triggers in ("id", "address") pairs and accepts data sent by "id" from "address." The mappings are managed by a rendezvous server, which forwards packets according to the triggers. TCP migration [161] and multipath TCP [46], which were designed for mobile TCP connections and multihomed TCP connections, associate TCP connections with cryptographic keys negotiated during the handshake process. They associate the host's TCP connections despite changing IP addresses.

At the application layer, web applications can track user identities with cookies, supported by browsers. Unfortunately, cookies are application-specific and only work with HTTP traffic. In some cases, network providers create persistent identifiers, called "super cookies," to identify systems at the device-level [41]. These super cookies are outside the end-user's control. They enable tracking that could violate end-user privacy and be used for profit, resulting in fines for some ISPs [77]. Our approach is mindful of these potential privacy concerns. A key design goal is to allow end-users to have control over persistent identifiers while supporting multiple applications, which we discuss in Section 6.4.4.

Organizations often use virtual private network (VPN) protocols, such as IPSec [12], to authenticate remote users and then leverage the VPN server's position inside a local area network (LAN) to provide access to LAN resources. As aggregation points, VPN servers can become throughput bottlenecks since they must be involved in the entire network flow and use cryptography to encrypt and authenticate traffic, even if the application-layer already offers that support (e.g., in HTTPS or SSH). Application-layer software lack options to configure IPSec tunnels for specific flows or destinations. Instead, current VPN software works across all applications on a device-wide basis. All network traffic is forwarded to the VPN server, which increases overhead and decreases performance. In practice, VPNs can increase organization costs [20], reduce

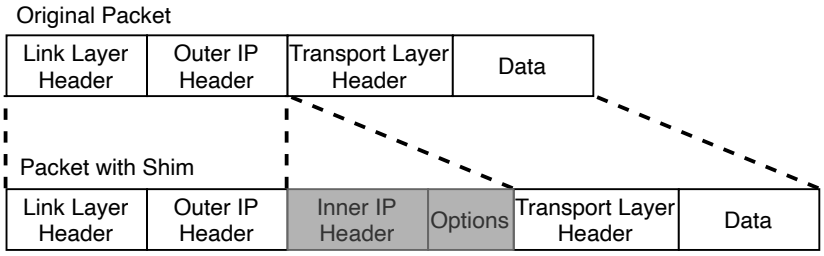

Figure 6.2: IP-in-IP packet format.

performance [29], create single points of failure [18], and add complexity [117]. In our approach, we focus on easy-to-deploy techniques that avoid these limitations.

6.2.3 Mechanisms to Encode Application-Agnostic Identifiers

Protocols such as TCP and IP support options for communicating information, such as identifiers and authenticators. Options in the IP header can be used for all transport layer protocols, rather than just TCP. However, these IP options may not be supported by intermediary routers, resulting in the packet being dropped [52]. In their measurement study, Medina et al. [104] found that only 45% of packets with the "record route" option, and 36% of packets with the "IP timestamp" option, reached the destination. Their study found that using a previously-unknown option caused all packets to be dropped. Given these limitations, prior work has examined repurposing IP header fields, such as fields associated with IP fragmentation, to transmit identifiers [30, 148]. However, this repurposing can negatively impact other network features.

Prior work has examined using "shim" layers between the IP header and transport layer headers to encode data [120]. IPSec does so using the ESP or AH headers to encapsulate protected traffic [78, 79]. Special-purpose shims have the downside of requiring support from endpoints and the risk that they will be discarded by firewalls or routers that do not understand the shim layer headers. However, the IP-in-IP tunneling technique, standardized in RFC 1853 [145], essentially provides a second IP header as a shim layer (shown in Figure 6.2).

The use of an IP-in-IP shim gives us a straightforward way to add options in a backwards-compatible manner. We can insert a second IP header in front of the original

transport layer header using the standardized approach. Mobile IP [129] has been designed to use IP-in-IP tunnels. It uses the outer IP as the locator and the IP addresses in the inner IP header as the persistent identity, which bonds to a device's home network. However, it assumes a device has a fixed home network identity and considers only the roaming issue of a mobile device. While we could repurpose header fields inside the inner IP header to encode identifiers, this may cause intermediate routers that inspect the inner header to drop the packets if they are considered malformed. We leave these fields unchanged. When gateways validate the tokens, the IP addresses included in the inner IP header can be used to automatically create NAT mappings at the gateway to recreate the "local address" feature present in VPN tunnels.

6.3 Motivations for VPN Deployments

Commonly cited reasons for organizations to use VPNs with their employees include [75, 101]:

1. protecting packet confidentiality, integrity, and authenticity via cryptography,
2. controlling communication to remote systems, and
3. simplifying access control by signalling a host's "insider" status.

In this section, we explore these usages and describe how they may be affected by changing Internet trends.

6.3.1 Confidentiality, Integrity, and Authenticity

Most network traffic uses web protocols. Based a recent survey, over 90% of web traffic is protected by Transport Layer Protocol (TLS) [31, 53]. However, the most recent detailed study of residential network traffic was in 2009 [97]. TLS is commonly used in business applications, including web and email systems. Applications such as remote desktop tools, file transfer and secure shell widely use application level cryptography. Over 98% of network printers support protocols with encryption and authenticity [162]. Accordingly, most of the applications, and thus network traffic,

already have application-layer protection, from a confidentiality, integrity and authenticity perspective.

For those legacy protocols that do not support cryptography, organizations may use the reverse proxy model [43] to enhance security without using VPNs.

6.3.2 Controlling Communication

VPN client-side software can allow network operators to split tunnels, which selectively sends traffic through VPN, by implementing routing table rules. VPN client software without this functionality need to tunnel all network traffic through the VPN. Accordingly, split tunneling improves performance and saves VPN's bandwidth bottlenecks because less unneeded traffic transits through the VPN server. However, some organizations, for security reasons, require uniform tunneling from end devices to their security tools, such as firewalls, IDSes, and content filters.

However, network defenders may have concerns that uniform tunnelling techniques could be ineffective. Attackers may utilize the bridge between a remote machine to reach the enterprise network. Thus, a malware on the client machine may infiltrate the enterprise network [19]. However, in attacking mobile devices, previous malware has examined the device's network connectivity and changed the attack behavior as a result [121, 168]. A device could be compromised when the VPN is not connected. When the compromised machine connects to a VPN, the malware can launch attacks by running previously downloaded attack instructions.

Endpoint security tools can better allow organizations to ensure uniform security and policy goals for remote endpoints, since those tools do not depend upon a specific network configuration. Naturally, such endpoint software does not require or preclude the use of a VPN.

6.3.3 Simplifying Access Control

Organizations usually manage their network by deploying filtering strategies on their network perimeter, which is usually the network gateway. These devices inspect and

filter network traffic crossing the network. They can deny unauthorized access. Organizations can also implement NAT technology and assign private IP addresses to their own hosts, preventing outsiders from reaching the internal resources unless utilizing an existing NAT mapping initiated by an internal host.

Organizations use VPNs to help their employees with crossing such a network perimeter. An enterprise network can configure a VPN server as a network proxy. An authorized employee is allowed to use traffic tunnelling techniques to traverse this perimeter and access the resources inside the enterprise network, even if they are assigned private IP addresses.

Usually the organization's VPN servers require authentication before connection, potentially requiring multi-factor authentication. Once authenticated, VPN tunnels regard remote users as they are on-site. Therefore, VPN can be regarded as a rough, network-level single-sign-on (SSO) system for organization services. This SSO functionality plays a key role for organization VPNs. However, in the remainder of this chapter, we demonstrate that this functionality can be implemented in a lighter weight technology to eliminate unnecessary costs and improve network performance.

6.4 Approach: Indicating Authenticity Validation

Inspired by the Kerberos approach [81] and HTTP cookies, we explore a token-based design that allows users to provide evidence showing that they are likely an authenticated user in the system. We describe the goals of such a system and how those differ from the robust authentication present in end-to-end applications. We then describe our threat model and the techniques we use in our approach.

6.4.1 Design Goal: Evidence Supporting Legitimacy

VPN servers can robustly authenticate their remote VPN gateways. For gateway-to-gateway VPNs that interconnect two LANs, this approach may not uniquely identify the connecting end-user. The VPN server may be able to uniquely identify the connecting end-user if the remote VPN gateway runs on the remote user's endpoint. However,

the VPN server does not share that identification with the endpoints that the user then connects to through the VPN server. The VPN server often performs NAT to proxy the connection between the remote user and server, but the IP addresses may be randomly selected from the available IPs. The server endpoint may be able to infer that the end-user authenticated with the VPN server by determining if the remote IP belongs to an IP address associated with the VPN server's pool, but it lacks a unique identifier or strong authenticity guarantee for the user.

Our goal is to re-create this weak authentication evidence for an organization's servers. We want to present the server with evidence that an end-user interacted with an authentication server at the organization and is likely to be legitimate. Our goal is to allow an organization to easily filter traffic that is likely associated with legitimate end-users, while leaving robust user identification to the application layer. In other words, we aim to simplify first-pass network connection filtering through a simple authentication token.

6.4.2 Threat Model

Our approach enables clients to provide an application-agnostic device-level authenticator. This authenticator is not designed to be authoritative about the identity of a client. Instead, it is a quick "first-pass" authenticator that can be used to separate out "likely legitimate" traffic from completely unknown traffic. It can be used in combination with application-layer authentication (e.g., a mechanism to address brute-force guessing on SSH servers). Enterprises can also calculate reputation based upon these identifiers, with better reputation leading to better services, as incentives.

The approach is designed to be effective against "on-the-side" adversaries, such as other clients that happen to share a CGN gateway and are multiplexed onto the same IP address. The authenticator value is constructed to make it difficult for an adversary to guess the value and impersonate the client. However, the approach is not designed to be robust against a "on-path" adversary, such as a network provider. Such an adversary would be able to inspect or alter the IP-in-IP shim layer and discover the identifiers. Further, the shim is only inserted in the first packet in the flow, so an

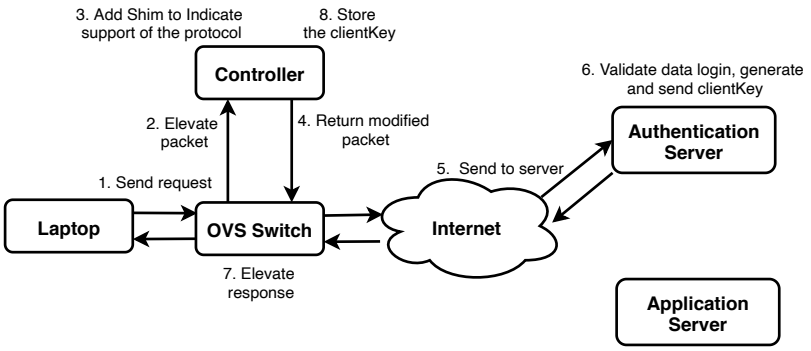

Figure 6.3: The key transfer from the authentication server to the OpenFlow controller.

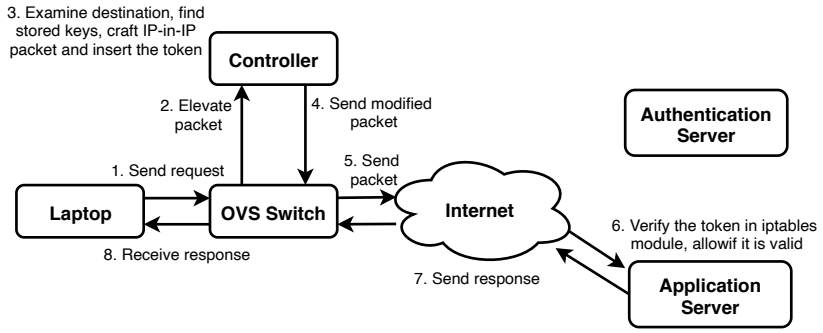

Figure 6.4: The process for the client to authenticate to the application server.

on-path adversary could simply hijack the flow for malicious purposes after the initial verification. However, this threat model is consistent with our design goals of keeping robust authentication at the application-layer.

6.4.3 Leveraging Authentication Servers

In our approach, we create a mechanism that allows an organization to authenticate its remote users using a lightweight device-level authentication factor. The end user can authenticate to the organization using a pre-existing authentication system, such as a web-based authentication page. This authentication system can use multi-factor authentication to robustly verify the end-user. Upon successful verification, the authentication system provides a token that can be used by the user's device as an authentication factor. If verification is unsuccessful, the server simply does not provide a token.

Our approach requires both the device and authentication server to automatically determine each others' support for the protocol, as well as for the other systems that are able to support the device-level authenticator. We enable automated deployment discovery using specially-crafted DNS records. When a client requests certain DNS records associated with the organization's domain (e.g., `A` records for `www.example.com`), the DNS server provides a `TXT` record in the Additional Records section of the response indicating the authentication server that is authorized to create authentication tokens for the domain (e.g., `login.example.com`). The `TXT` record also indicates which servers at the organization support the authentication scheme by listing public-facing IP address or CIDR prefixes.

6.4.4 Using OpenFlow to Manage Tokens

To avoid requiring support for the authentication mechanism in each application, we use an SDN technique to engage in the protocol on the client's behalf. An OpenFlow agent in the client's network, which could be on the client endpoint device itself (e.g., via Open vSwitch) or on a network gateway (e.g., a residential router), intercepts new flow requests and directs them to a SDN controller. The controller examines the related DNS responses for any `TXT` records that indicate support for the protocol. It also manages the authentication factors on behalf of the end-user on the device or on multiple devices in the network.

Our approach is designed to grant users full control over their identifiers. Unlike the "super cookies" approach [41], our system allows users to configure their identifies and choose when to use tokens with a remote party. Since the OpenFlow controller manages the database, it can allow users to manage the entries via interfaces such as a dedicated application or web page. Based on the user's configuration and the destination of each flow, the controller determines whether to insert identifiers.

We use the standardized IP-in-IP tunneling approach [145] to create a "shim" layer. This creates two IP headers, allowing the outer IP header to be processed normally by routers while the inner IP header contains options that might otherwise result in the packet being dropped. We use those IP options to communicate the token that

provides evidence of authentication.

When a client first interacts with an authorized authentication server, the OpenFlow agent intercepts the initial packet in the flow and sends it to the OpenFlow controller. The controller modifies the packet to signal that the client supports the scheme and sends it back to the OpenFlow agent, which then transmits it to the authentication server. Since the DNS records signal the server's support for the approach, we can construct packets that require the server to engage in custom parsing. The controller alters the packet to insert the IP shim layer, resulting in an IP-in-IP packet. In the inner IP header, the controller includes an option indicating that the client supports the protocol. Upon successful login, the authentication server replies with its own IP-in-IP packet, with the authentication data contained in an option field in the inner IP header. The OpenFlow agent elevates this response to the OpenFlow controller, which extracts the authentication factor, removes the IP shim header, and orders the OpenFlow agent to send the decapsulated packet to the client application.

When a client subsequently creates a new network flow to a server that supports the scheme, the OpenFlow agent intercepts the request and elevates it to the controller. The controller again performs the necessary alteration of the packet to create the IP-in-IP shim that contains the pre-determined authentication data. The controller then returns the modified packet to the OpenFlow agent for transmission to the server. The server, or a gateway or middlebox on the path to the server, processes any packets containing the IP-in-IP shim to verify and strip the authentication data. In doing so, the server or middlebox can record that the flow is verified and, depending on policy, allow the flow where unverified flows may be denied.

Importantly, the inner IP header addresses can be used by a gateway or middlebox validator to implement NAT translations that allow a remote user to have local IP addresses in the same manner as VPN servers. However, unlike VPN servers, the inner IP header need only appear in the initial exchange to create the appropriate NAT mapping to translates the remote machine's address to a local one.

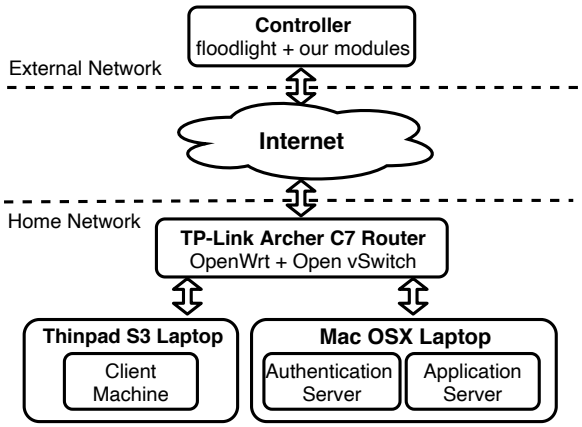

Figure 6.5: Our experiment architecture.

6.5 Implementation

We implement our system in a home network to allow us to evaluate its security and performance. As shown in Figure 6.5, we explore the modifications that would be needed to include a regular client, an authentication server (e.g., a single sign-on identity provider), an application server (e.g., a relying party), and a SDN controller to coordinate it all.

We create our client host in a virtual machine (VM) running on a Thinkpad S3 laptop with four cores and 8 GBytes of RAM. Both the application server and the authentication servers are VMs that runs on a Macbook Pro laptop that has four cores and 16 GBytes of RAM. Each VM is assigned a single CPU core and 4 GBytes of memory. For our SDN gateway, we use a TP-Link Archer C7 router running OpenWrt with the Open vSwitch module. The two laptops are connected via two LAN ports of the router. The VMs bridge their network interfaces and receive their DHCP leases from the router. Both the two laptops and router are located in a residential network. The controller runs on a dual-core VM with 4 GBytes of memory on a university campus, emulating the use of a remote third-party SDN service provider. We implement our SDN controller and module using the Floodlight SDN framework.

6.5.1 Identity Provider Interactions

In Section 6.4.4, we described how the OpenFlow switch and controller cooperate to detect that the application and authentication servers support the approach. The Kerberos approach provides a shared-key architecture for identity providers to construct keys for relying parties [81].

The SDN controller learns about the authentication server through DNS records and signals the client's support via an IP-in-IP shim. When the authentication server replies, it includes a shared secret that the client can use to authenticate itself.

The shared secret can be calculated using a pre-shared value that the server shares with each of the application servers. Then, for each user authentication request, it creates a unique identifier for the user and a nonce value. It then produces a shared secret that is a hash of these values (i.e., `clientKey=SHA224(uniqueIdentifier || nonce1 || applicationServerSecretKey)`). The authentication server then sends both the `clientKey` value and the concatenation of the `uniqueIdentifier` and `nonce1` value to the client.

The authentication server communicates the key and identifier by crafting an IP-in-IP packet. The inner IP header includes an option field in which both the `clientKey` and `uniqueIdentifier` are encoded. The protocol field differs between the two IP headers (the outer header indicates the protocol is another IP header, while the inner header indicates the transport protocol used), but otherwise the two headers contain identical values.

The OpenFlow switch simply elevates any packet with IP-in-IP shims to the controller for review, allowing the controller to obtain the `clientKey` and `uniqueIdentifier`.

6.5.2 Application Server Interactions

In our implementation, when the client initiates a connection to the application server, the OpenFlow switch elevates the request to the OpenFlow controller. The controller consults its database, determines that a token is needed, and creates an IP-in-IP shim.

In the inner header, it creates an IP option that contains the user identity, the authentication server's nonce (`nonce1`), the client's own nonce value (`nonce2`), and a SHA224 value constructed from the concatenation of the identity, `nonce2`, and `clientKey` (i.e., `SHA224(uniqueIdentifier || nonce2 || clientKey)`). The token we construct is 37 bytes total (5 bytes for the user ID, 2 bytes for the `nonce1`, 2 bytes for `nonce2`, and 28 bytes for the SHA224 output). The controller sends this modified packet back to the switch for transmission using an OpenFlow `PACKET_OUT` message.

The application server must parse the IP-in-IP message, validate the token, and then remove it before delivering it to the actual destination application. To do this, we develop an open source `iptables` module using the Xtables-Addons framework [73]. This module efficiently performs the interception and validation before the packet reaches the destination application.

Our functionality is divided into an `iptables` match module that specifies user-defined conditions. It passes any matching packets on to a target module for processing. We configure our match module to examine any IP-in-IP packets. For all such packets, it searches for our specific IP options type inside the inner IP header. If found, it parses the options to obtain the unique identifier and nonces. The match module then uses the client-supplied information and the key shared by the application server with the authentication server to calculate the corresponding `clientKey` (i.e., `clientKey=SHA224(uniqueIdentifier || nonce1 || applicationServerSecretKey)`). The application server then constructs a SHA224 digest using this `clientKey`, the `uniqueIdentifier`, and the client's nonce (i.e., `SHA224(uniqueIdentifier || nonce2 || clientKey)`) and compares it with the SHA224 value that is contained within the IP option. If the digests match, it knows the client interacted with the authentication server to obtain the `clientKey`. The match module returns true if a match is found and returns false in all other cases.

We next implement a target module that is used if the match module successfully validates a packet. Our target module modifies the `skb` buffer, which is the data structure used in Linux for packet processing. The target module must decapsulate the packet to remove the shim. It does so by removing the inner IP header and IP options,

updating the protocol field in the outer IP header and recalculating the checksum. It then sends the packet to the application for processing. This allows the destination to validate the communication across applications. Importantly, the `iptables` tool can be run on a middlebox or on the application server itself to avoid bottlenecks.

We have couple reasons to choose SHA224 as our "good enough" hash function. First of all, the length of the inner IP header is limited. The length of SHA224 output fits well with our space limitation. Second, SHA224 provides 112 bits of security. Although SHA224 is not robust like SHA256 and SHA512 hash functions, there is no collisions detected by SHA224 so far. Plus, if the authentication and application servers pick long and complicated enough server keys, an on-path attacker is unlikely to guess out the server key.

6.6 Evaluation: Security and Performance

We evaluate our approach from both a security and performance perspective using the configuration depicted in Figure 6.5. We focus on the performance of the token validation and shim layer operations between the client and application server, since these same operations are used in the interaction between the client and authentication server.

6.6.1 Security Evaluation

To evaluate the security of the approach, we create new connections from the client machine to the application server. The application server runs our `iptables` modules locally to allow validated packets with a default deny rule to drop any new flows that are not approved by our match module.

In our first scenario, we disable the OpenFlow controller module, causing the client to send packets without a token. As shown in Table 6.1, all such attempts were blocked by the default deny rule.

In our second scenario, we use the OpenFlow module that intercepted only the first packet in each flow to insert the properly constructed header but with an invalid token.

Result	No Token	Invalid Token	Valid Token
Access Allowed	0	0	20
Access Denied	20	20	0

Table 6.1: Result of effectiveness evaluation. We performed experiments 20 times for three scenarios: 1) network request without a token, 2) network request with an invalid token, and 3) network request with a valid token.

As shown in Table 6.1, all such flows were properly blocked.

In our third scenario, we use the OpenFlow module to construct a proper header and a valid token value. As shown in Table 6.1, all such flows were properly matched and authorized. While only the first packet in the flow was processed by the `iptables` rule, future work could use a packet capture listener to detect any new flows that pass the filter and add a filter rule for that host to allow subsequent packets in the flow.

6.6.2 Performance Evaluation

Our approach affects only the first exchange in each flow. The controller elevation to insert the shim, and the `iptables` processing to validate and remove the shim, are only needed on the initial message from the client to the server. Subsequent packets in the flow are not modified and or inspected by our custom `iptables` module. Those packets will proceed through standard packet processing. Therefore, the only significant overhead in our approach is incurred in the initial round-trip, so we focus our measurements accordingly.

During key transfer, the client initiates a TCP connection to the authentication server. Since the authentication server program is essentially unchanged, we use a simple echo reply to omit its overhead. For our measurements, we transmit a TCP `SYN` packet to a port without an associated application server, resulting in a TCP `RST` packet that refuses the connection. This simple exchange allows us to monitor any overheads at the OpenFlow controller and `iptables` modules to signal support for the protocol, encode keys into packets, and extract those keys. The client sends 1,000 TCP requests and measures the round-trip time (RTT) that includes all the overheads.

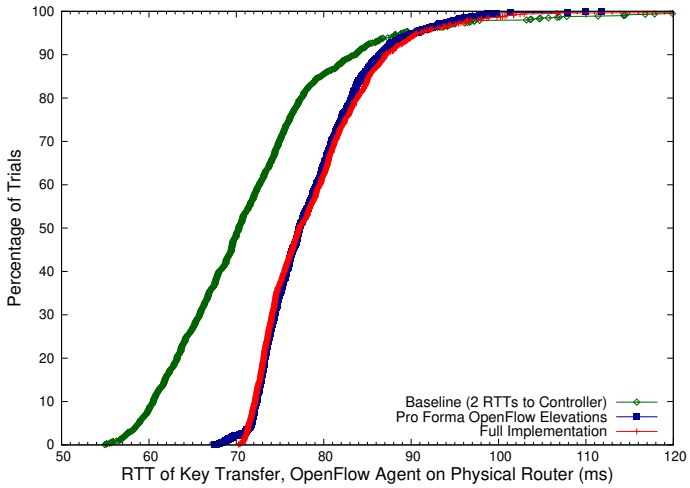

Figure 6.6: Key transfer between client and authentication server across 1,000 trials, with the TP-Link router executing the OpenFlow agent.

Our evaluation examines two deployment scenarios: 1) where the OpenFlow agent runs on a separate, physical router for a local network deployment and 2) where the OpenFlow agent runs on the client machine itself, in which the endpoint natively supports the use of a controller for persistent identity. For the router-based experiment, we use a consumer-grade TP-Link Archer C7 router, as shown in Figure 6.5.

During the key transfer at the authentication server, two elevations to the OpenFlow controller are required. Since this includes significant propagation delay to and from the controller, we measure that RTT and refer to it as the baseline in our method. In Figure 6.6, we explore the scenario where the OpenFlow agent runs on a physical router. In the diagram, the leftmost (green) line indicates baseline of two RTTs with the controller. The 90th percentile is 83ms. The middle (blue) line shows the measured end-to-end RTT for the client to the server using *pro forma* OpenFlow elevations. In our pro forma elevations, the OpenFlow agent is configured to elevate each packet for approval, but the controller simply approves each packet without changes (i.e., the controller simply approves packets using `PACKET_OUT` messages without using `FLOW_MOD` rules). In the pro forma exchanges, no packet encapsulation or `iptables` verification occurs. In this scenario, the 90th percentile of the RTT is 86ms. In our full key transfer implementation (the rightmost, red line), the 90th percentile rises to 87ms.

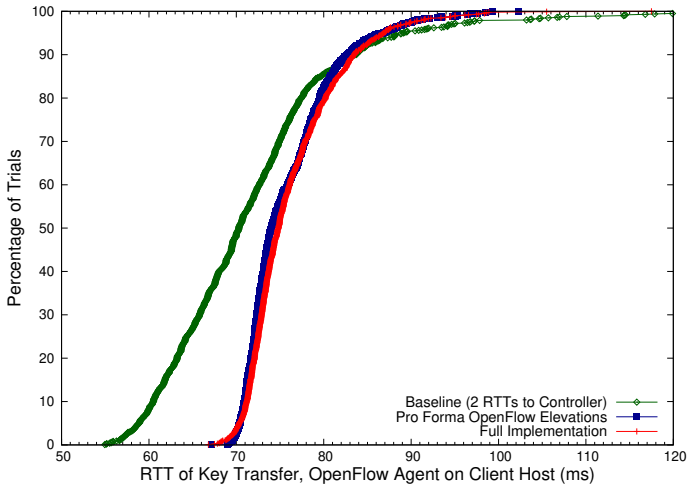

Figure 6.7: Key transfer between client and authentication server across 1,000 trials, with the client host executing the OpenFlow agent.

The similarity of the results of the pro forma and full implementations indicate that the overheads for encapsulation, `iptables`, and the OpenFlow controller are not significant.

We show the results where the OpenFlow agent runs on the client host in Figure 6.7. The baseline remains same. When the pro forma scenario has a 90th percentile of 82ms. When we enable the full key transfer functions, the 90th percentile is 83ms. The OpenFlow overheads are lower when the client runs the OpenFlow agent rather than a router. However, the overheads of encapsulation, `iptables`, and controller processing remain similar.

To measure the performance evaluation of the key validation associated with the application server, the client sends a UDP packet and measures the RTT. We conduct 1,000 trials of this experiment. In each trial, the packet is elevated to the controller twice: once to perform the IP-in-IP encapsulation and once to handle the UDP server's response, which results in the addition of a `FLOW_MOD` in the switch for subsequent packets.

In Figure 6.8, the baseline of two RTTs with the controller remains same (green, leftmost line) with a 90th percentile RTT of 83ms. In the pro forma scenario (blue, middle line), the 90th percentile RTT is 90ms. For the full implementation scenario (rightmost, red line), the 90th percentile RTT is around 93ms. When we moved the

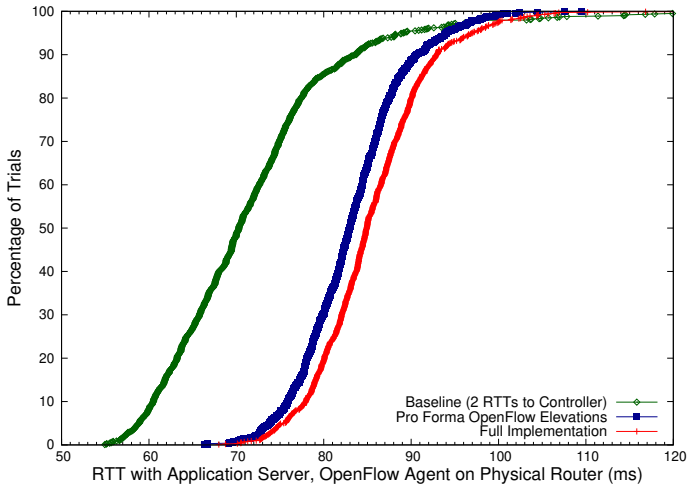

Figure 6.8: In key validation, the client interact with application server in 1,000 trials. The OpenFlow agent runs on the physical TP-Link router.

OpenFlow agent to the client machine, the pro forma 90th percentile drops to 84ms and the full implementation drops to 86ms, as shown in Figure 6.9. We again see that the full implementation has only modest overheads over a basic OpenFlow elevation approach. Further, the time spent at the validator in the full implementation was around 0.8 milliseconds, indicating the verification overheads are low.

Importantly, the latency overheads incurred here occur only on the first round-trip between the client and the application server for each flow. Since they do not affect ongoing flows, they are unlikely to have a major impact on the end-user's experience. A subsequent optimization, to insert a `FLOW_MOD` during the first encapsulation, would cut the propagation time in half, reducing the RTT by roughly 40ms in these experiments. However, if the initial packet were lost in transit, the `FLOW_MOD` would prevent the insertion of the token in any retransmissions sent by the client. This provides a configurable tradeoff between performance and resiliency. End-users could further reduce their delay by hosting the controller closer to the client, such as in the LAN or in nearby ISP-hosted data centers.

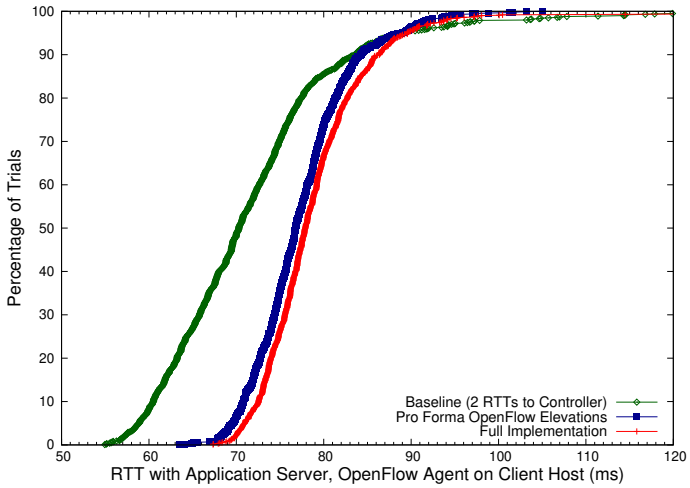

Figure 6.9: In key validation, the client interact with application server in 1,000 trials. The OpenFlow agent runs on the client host.

6.6.3 Throughput Evaluation

Our approach aims to improve bandwidth performance when an enterprise deploys its VPN server on a general-purpose machine. So in this section, we simulate real environments and design experiments to create that bottleneck. We demonstrate the extent to which our approach can remove the bandwidth bottleneck associated with many VPN deployments. We explore five scenarios with some employing TLS, VPNs, and our approach.

In these experiments, we use `Strongswan` as our VPN software and `apache2` to host a large file for download via HTTPS. For each case, we run a varying number of clients concurrently to determine the per client throughput in each scenario. We measure the time used for downloading for each client, as well as CPU usage on the servers. In the experiments, we explore up to five clients since this degree of parallelism is sufficient to expose bottlenecks and demonstrate trends. Except as noted, we transfer a 1 GByte file in our trials. Each data point is the result of 30 trials.

In Figure 6.10, we show the results of these scenarios. The first scenario, `Plain TLS`, represents a baseline in which the enterprise network has a TLS server that clients directly access for file transfer. This essentially represents an upper-bound on performance capabilities of the system. The traffic is constrained by the web server's ability

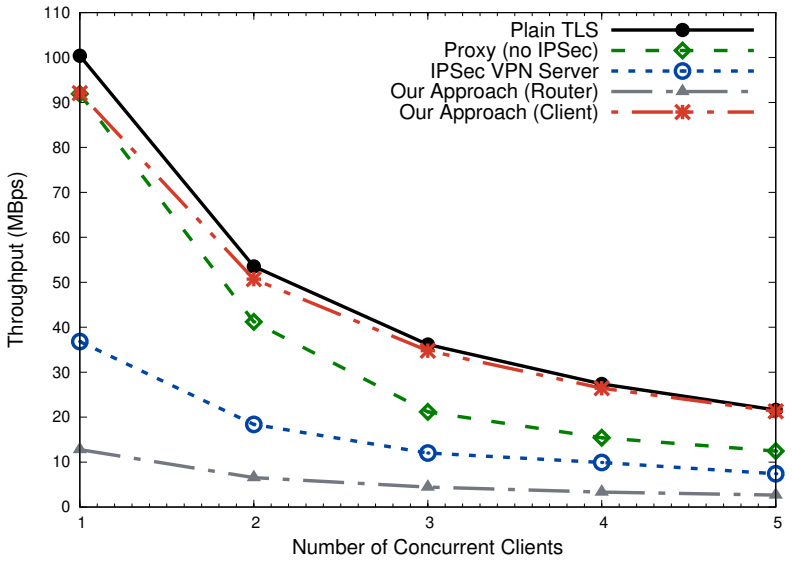

Figure 6.10: Median client throughput in different security tool deployment scenarios.

to send traffic. In the second scenario, `Proxy (no IPSec)`, we forward the network traffic through an Ubuntu server VM that simply proxies traffic (i.e., forwards it using IP addresses) without any additional services (like IPSec). The throughput decreases in this scenario since the gateway host starts to constrain throughput and the CPU on the gateway vary from 35.41% to 95.83% utilization as we increase number of concurrent clients. In the `IPSec VPN Server` scenario, we enable IPSec on the Ubuntu proxy server and clients use the IPSec tunnel to reach the web server. This scenario represents an enterprise configuration in which organizations host their VPN services on a generic server. The CPU use climbs on the VPN server to around 100% usage and the result is a decrease in bandwidth from the baseline by around 65%. Compared to the 21.6 MBps in the baseline, the VPN server is only able to provide 7.4 MBps throughput per client during a five concurrent client scenario.

Next, we explore our approach using SDN support in either a consumer-grade router or in the endpoint itself. We first enable our approach in a consumer-grade router in the `Our Approach (Router)` scenario. This scenario does not require the gateway server from the second or third scenarios and reflects the architecture as described in Section 6.4. The residential router scenario results show the challenges of repurposing hardware with limited computational capabilities. While it can deliver 12.8 MBps for

a single client our tests of a 100 MByte file, it exhausts the router's computational resources (since it uses the general purpose CPU for OpenFlow forwarding lookups). While we explore up to five concurrent clients for a consistent presentation of results, these client-side routers would likely only service a single user at a time and do not act as an aggregation point, unlike the VPN server. With more capable residential routers, the computational limits would be less likely to constrain performance.

In our final scenario, `Our Approach (Client)`, we explore the approach where the SDN agent runs in software on the client, allowing us to characterize the performance implications of running the SDN agent on the residential router. When running on the client, the SDN functions no longer serve as a performance bottleneck (CPU usage at the client does not exceed 31%). This approach yields a performance improvement of roughly 2.5 to 2.9 times the throughput of an IPSec VPN. The performance decrease of the SDN approach verses the baseline ranges from 1% to 8%. Accordingly, with endpoint software, clients can attain far better throughput than VPNs and approximate the baseline.

6.6.4 Packet Header Overhead

VPNs may have to combine multiple protocols together to support some clients. A standardized implementation for this combines IPSec with L2TP [126]. When used with ESP and preshared secrets, the combined packet headers and trailers for the two protocols amounts to around 92 bytes (40 bytes for L2TP with UDP, 20 bytes for an encapsulated IP header, 16 bytes for the ESP header, 2 bytes for padding, and 14 bytes for the ESP trailer and authentication data). This can reduce the maximum transmission unit (MTU) for payload in many networks from 1500 to 1408, which is around a 6.1% reduction in payload per packet. This overhead occurs in each packet in the flow.

In our approach, we use packet encapsulation on the first packet sent from a client to an application server. Our IP-in-IP shim uses 20 bytes for the inner IP header, with an additional 40 bytes for our IP option, for a total of 60 bytes of overhead. Unlike VPN traffic, this overhead only applies to the first packet in a flow. For applications

using TCP, this overhead would apply to the TCP SYN packet. Since those packets do not carry payload, our approach would likely avoid the MTU complications present in VPN protocols.

6.7 Discussion

Our approach focuses on mechanisms that eliminate the need for VPN software by providing application servers with evidence that a client has been successfully authenticated. We now explore how a similar concept could be used with other kinds of services.

For sensitive transactions, enterprises with public-facing services can minimize risk by using multiple sources of evidence. For example, financial institutions may authenticate an end-user in multiple ways to minimize the risk in financial transactions. These forms of validation can include username and password, browser cookies, the use of one-time passcodes via SMS messaging or applications, or answers to secret questions.

Some organizations try to prevent risk by identifying a user's location. They may leverage databases that map IP addresses to geographical location and thereby prevent logins, or they may or require more robust verification when a user's location changes by a configured distance. Unfortunately, such mechanisms may be less effective when CGN is widely deployed.

Our approach is a lightweight, secondary factor that simply indicates if a client is located within a given source network (e.g., inside the LAN serviced by a given residential router) or if it is the same physical device (e.g., for a laptop or mobile device that changes networks). In such circumstances, the SDN controller can effectively act like a password manager by tracking secondary factors for a user across infrastructure.

6.8 Outcomes

This chapter explores the roles that VPNs play in organizational security. With the rise of application-layer encryption and authentication, the secure tunneling features

of VPNs are increasingly redundant. However, VPN tunnels are still useful in simplifying perimeter-based access control by allowing authenticated remote users to bypass perimeter policies and interact with insider infrastructure.

In this chapter, we examined this access control mechanism and proposed an alternative that achieves the same goals while reducing bottlenecks, extra server infrastructure, redundant cryptography, packet header overheads, and complexity. We proposed an SDN-based architecture to facilitate user-controlled persistent identity. We used `iptables` modules to implement the approach and the results show that our method is effective, lightweight, and incrementally deployable.

Chapter 7

SDN for Mobile Devices

7.1 Introduction

Although mobile device security and application security are well studied, the features of mobile devices, such as outdated versions of OS, unpatched firmware, and roaming, can still make mobile devices vulnerable. These devices in a residential network bring uncertainty to residential network security. This chapter provides an on-host SDN based method on Android to fuse user interface (UI) information into network data. With this tool, devices are under protection even there is no in-network sensor deployed when they travel. Further, with this system-level information, we expect to understand more about network flows on Android, thus provide better network management and control.

In enterprise networks, IT system and network managers also need visibility into endpoints to expedite problem diagnosis and reduce resource usage and infrastructure costs [143]. They can use methods like root certificate to enhance the visibility, a firewall server thus can further improve the effectiveness of network monitoring. However, these methods are unrealistic in a residential network, where has no IT expert. Further, even with detailed logging infrastructure, the link between user actions and the resulting behavior of a device may be unclear. Finding out whether a network flow is associated with a user action can largely help understanding the network and device. Some other on-host methods usually have impractical requirements for normal users,

like recompiling the Android kernel or gaining root access to their device, which may incur its own security risks [55].

This exploration leads to two research questions. *Can user interface (UI) interaction and network activity be used to successfully predict and associate network flows with user actions on Android devices? Since mobile phones are usually resource limited [39], what impact would such instrumentation have on CPU, memory, battery, and network latency on mobile devices?*

In exploring these research questions, we make the following contributions:

- **Create a UI-aware Endpoint Network Sensing System:** We create a new Android application, called AppJudicator, which leverages user interface (UI) interaction and software-defined networking (SDN) principles to determine whether network flows are legitimately user-initiated (Section 7.3).

- **Characterize Network Profiling Potential for UI-aware Systems:** We explore AppJudicator's ability to perform real-time network profiling and management on Android devices. With enable dynamic allow-lists that can leverage user actions to permit user-driven events that would otherwise have be denied under default-deny policies. We found that our system can enhance the accuracy of network profiling from 14.42% to 97.93% (Section 7.4).

- **Evaluate the Performance of the AppJudicator Prototype:** The costs of our system include the resource overhead of running the app (e.g., CPU cycles, battery power, memory consumption), additional network latency for access control decisions, and resources in running a controller server. In our experiments AppJudicator adds 11 milliseconds of total end-to-end latency, on average, of intercepted first packets of a flow, and less than 2.7ms for the other 90% non-first packets. We believe this delay will not impact user experience (Section 7.5), despite our current implementation incurs big throughput reduction, from 100 MBps to 8.3 MBps.

7.2 Background and Related Work

Our approach is related to prior work in the following areas: (1) profiling Android system and network characteristics, (2) graphic user interfaces and accessibility services, and (3) end-point software-defined networking (SDN) techniques. We now explore each of these areas.

7.2.1 Profiling Systems and Networks on Android

Prior work has shown the value of profiling communication patterns on Android. ProfileDroid [174] characterized application traffic and privacy concerns. They linked coarse-grained user-level information with network flows to identify traffic associated with advertising from primary application traffic. Netsight [62] helped localize the root causes associated with network failures. Other work demonstrated the effectiveness of endpoint network logging by using browser data to explore the end-to-end network path to distinguish on-path failures from attacks [22].

While powerful, some network instrumentation on Android devices may require root privileges to devices, raising security concerns [184] and possibly complicating deployment. To avoid such challenges, NoRoot [155] and NetGuard [156] configure the built-in Android VPN client to route traffic to their own local applications in order to enable profiling and firewalling services. Meddle [135] enables inspection of network traffic remotely using the VPN interface. Our work likewise uses the VPN interface to obtain network traffic without requiring root privileges; however, unlike prior work, we fuse this information with the user's activities in the UI to distinguish user-driven traffic from automated processes.

Prior work has explored network logging on mobile devices for network troubleshooting [147] and digital forensics [147]. Further, Sipola et al. [146] used deep learning methods on network logs to identify malicious network activities. Our approach augments traditional network logs with user activity to help provide context for operations.

The instrumentation of Android devices can look at internal interactions as well. Prior work has explored system calls and their ability to reconstruct users' intended

actions [67, 87]. While Android maintains a system log, debugger or root access is typically needed to access the full logs. Using static analysis, AppContext [180] found patterns that distinguish malicious apps and benign apps. However, such approaches are not amenable for real-time network profiling.

7.2.2 User Interface (UI) and Accessibility Sensors

End users control programs on mobile devices via the user interface. That interface can provide context for different operations. AppIntent [181] uses static analysis to trace UI activities to data transmission events; however, this technique cannot be used in real-time traffic classification. The GUILeak [172] tool traces UI actions and records user input, which is manually compared with contractual and privacy policies described by the vendor to detect privacy violations. Given the manual steps involved, these processes are not amenable to real-time traffic evaluation. SmartDroid [186] proposes to enforce a sequence of system calls associated with UI elements. However, the method is difficult to implement on average users' mobile phones since it usually requires rooting a device.

Prior efforts have used the UI with traces and logs to help identify application defects and to localize issues [25, 51, 69, 160]. Tools to automate UI interactions have also been developed to help with software testing. The Android `Monkey` [159] tool is an official mechanism that allows developers to send sequential events to the device via its `adb` debugger interface. We use this tool in our experiments to examine hierarchical UI elements. Harbinger et al. [26] propose an UI based network control system based on Windows.

Android provides an accessibility API [157] that allows accessibility services developers to cross the traditional boundary of sandboxed applications. The services can send commands and receive UI events invoked by a human user to other applications. Unfortunately, this capability can be abused by adversaries in a form of click-jacking and overlay attacks [76].

7.3 Approach: UI and Network Sensor Fusion

We combine sensors for the user interface and the network at the mobile device endpoint. Since the Android OS dominates the market share for mobile devices, we use that operating system for our prototype. As we build sensors, we focus on options that can be accomplished using existing APIs and standard application installation procedures rather than requiring steps that would hinder deployability, such as rooting a device or recompiling the operating system.

This section introduces each component in our architecture. We will first talk about utilizing a VPN service (Section 7.3.1) to intercept the packets between applications and remote application servers without rooting a device. The VPN service works with an SDN module (Section 7.3.3). Our SDN agent fuses the captured network flows with UI information provided by an Accessibility Services module (Section 7.3.2). The SDN agent works with a remote SDN controller that analyzes the flows and makes forwarding decisions for the network flows. Figure 7.1 visually depicts the components of our system and how they relate to each other.

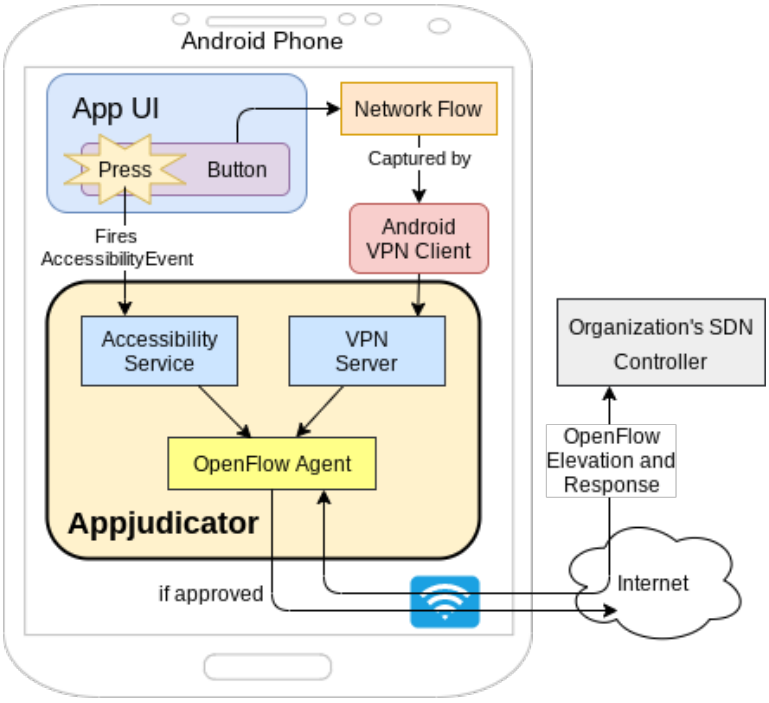

Figure 7.1: System Overview

7.3.1 Instrumenting the Network via the Android VPN API

As described in Section 7.2.1, prior work has established that capturing network traffic through Android's built-in virtual private network (VPN) client allows an application to intercept traffic device-wide without requiring root access to the device. Such functionality has also been used in Apple's IOS for similar purposes. We use this established approach to enable our novel data fusion efforts.

In our configuration, the built-in VPN API provides an interface that can tunnel all traffic to Appjudicator's local VPN server component. We further use the interface to build two stream: 1) a VPNInputStream that intercepts packets from applications and 2) a VPNOutputStream that allows the module to transmit reply packets to nte application. We depict the sequence of the module's actions in Figure 7.2. The VPN service allows Appjudicator to control the intercepted packets and implement an SDN agent. If the SDN agent has a table entry for the flow, it will deliver the packet as directed by that table entry (step m in Figure 7.2). Otherwise, the agent will initiate the OpenFlow elevation process (steps d through i in Figure 7.2).

Since TCP and UDP packets need different processing, Appjudicator divides packets and routes them to either the `TCPVPNService` or the `UDPVPNService`. When a connection is received in the TCPVPNService, it initiates a connection to the remote system through a *protected socket*. The protected socket designation ensure that the communication is not itself intercepted by our VPN service again (and thus creating a loop). Upon establishing the protected channel, the TCPVPNService works as a translation device, stitching together the connections between the local application and the VPN service and the VPN service to the remote server (shown as step n and o in Figure 7.2).

The Android VPN service API avoids the typical cryptographic or tunneling overheads associated with a VPN since the Android OS allows our application to simply read and write packets directly using a `ParcelFileDescriptor` instance. This allows us to use standard packet parsing libraries (e.g., `Pcap4J`).

Finally, we need a mechanism to link packets back to their associated application. We use the Android `ConnectivityManager` API to identify the user ID (UID) as-

sociated with the flow's fields. Since each Android application has a different UID, we can use the Android `PackageManager` and the `QUERY_ALL_PACKAGES` permissions to determine the package name associated with the UID. Accordingly, we can aggregate packets into flows using the standard flow tuple (IP_{src}, IP_{dest}, protocol, $Port_{src}$, $Port_{dest}$). To link the flow with a package, we first use `ConnectivityManager` Android API to look up the UID of the flow owner by (IP_{src}, protocol, $Port_{dest}$) information. The `PackageManager` Android API further allows us to obtain the package name and context for each UID. To support SDN functionality, we use the flow tuple as the key in a hash table combined with a queue data structure for packets awaiting a verdict from the SDN controller.

7.3.2 Instrumenting the UI with Accessibility Services

The accessibility services and APIs on the Android platform provide a way to gather data about UI events. These accessibility services are designed to support alternative user interactions, such as screen readers for individuals with impaired vision. We leverage these services to correlate UI events with networks flows to establish the origin for each network action.

We enable this service in our application by declaring it in our application's `Manifest` [158] file and specifying the `BIND_ACCESSIBILITY_SERVICE` and the `canRetrieveWindowContent` permissions. Using the `intent-filter`, we specify the scope of applications that we monitor. In our prototype, we include all applications in our scope but note that production deployment may tailor the scope to achieve privacy goals. Given the potential privacy and security implications of these permissions, the Android OS itself provides a clear statement about the service's capabilities and the associated risks.

The accessibility services represent UI events as a UI state change. Our application listens for all types of UI events, including button presses, swipes, long presses, and focus changes. For each event, we can obtain a context description from the event object. With the `AccessibilityEvent.getSource()` function, we can extract more information about the layout hierarchy, including information associated with parent

and child widgets. This function allows our system to identify text and labels most closely associated with an action, allowing us to create detailed context for an action. For example, for the `TYPE_VIEW_TEXT_SELECTION_CHANGED` UI event type, we can obtain the contents of an `EditText` UI object, which often contains a user's input. When these inputs contain an IP address, DNS host name, or URL, it can aid correlation of network flows; otherwise, the data can be ignored to preserve privacy. We further use the `FLAG_INCLUDE_NOT_IMPORTANT_VIEWS` flag with accessibility services to obtain full visibility into the application's UI [56].

With the UI information, we can characterize the user interface and provide this data to the SDN controller. However, one challenge is associated with unique identifiers that can be incorporated into policy. Some UI elements have a resource identifier which may be unique. Others do not have such an identifier. Accordingly, we use an established heuristic of combining the resource ID (if it exists), the hierarchical path between the UI element and the root widget in the application, and the properties of the UI element itself (class type and label text, if any).

Appjudicator obtains and stores all UI events in the scope of monitoring on a per-application basis. When queried by the SDN agent, the service provides the most recent events.

7.3.3 Fusing UI and Network Sensor Data via SDN

Appjudicator implements a subset of the OpenFlow v1.0 specification [103]. Since our software only operates on a single device, it does not implement the full set of OpenFlow options. We thus omit data about the data link layer headers, the physical ports, or VLANs. As with traditional OpenFlow agents, we allow wildcards in the rule match fields. Our SDN agent finds the most specific matching rule and returns the associated action to the VPN service to apply to any queued or subsequent packets associated with the flow.

If the OpenFlow agent does not find a table match on the packet received from `VPNService`, it elevates the packet to the SDN controller, allowing the controller to profile the traffic and provide direction on how to handle the packet. In our prototype,

we only implement the *drop* and *forward* actions, which will discard or transmit packets, respectively. Future work may extend the actions the agent can perform.

Unlike the traditional OpenFlow `PACKET_IN` message sent to an SDN controller, our agent provides additional UI context. Our SDN agent queries the accessibility module for UI information associated with the flow (Section 7.3.2). The UI events are attached in reverse-chronological order. However, the agent filters some UI events, like `TYPE_VIEW_CONTENT_CHANGED`, that are very frequent and have limited utility to the controller. Accordingly, Appjudicator elevates the types of UI events that are most likely to reveal user intentions to controller. As reported in Section 7.1, we empirically found that limiting reports to UI events from the last four seconds is most effective.

Our `PACKET_IN` message contains an OpenFlow header, the encapsulation of the original packet, and as much UI information as possible while keeping the entire `PACKET_IN` message within 1500 bytes (to avoid packet segmentation). The OpenFlow agent then sends the message to the remote SDN controller.

7.4 Effectiveness Evaluation: Network Profiling

Our goal is to determine what role our UI context can serve in network profiling and whether it can reveal whether a flow is the direct result of an end-user action or due to an automated process, and how useful it is for network traffic monitoring. In doing so, we compare ourselves against more traditional network-based sensors used in enterprise networks. This results in three broad classes of sensors:

- **Network Header Sensor**: Since the mobile device's IP address and local port typically lack significance, this sensor only considers the remote system's IP addresses and port numbers when profiling a network flow.

- **DNS-aware Sensor**: This sensor fuses recent DNS behavior with the Network Header Sensor data. This sensor groups traffic to the same DNS host name, even if the traffic is to different remote IP addresses.

- **UI-aware Sensor**: The UI sensor considers DNS matches for background traffic.

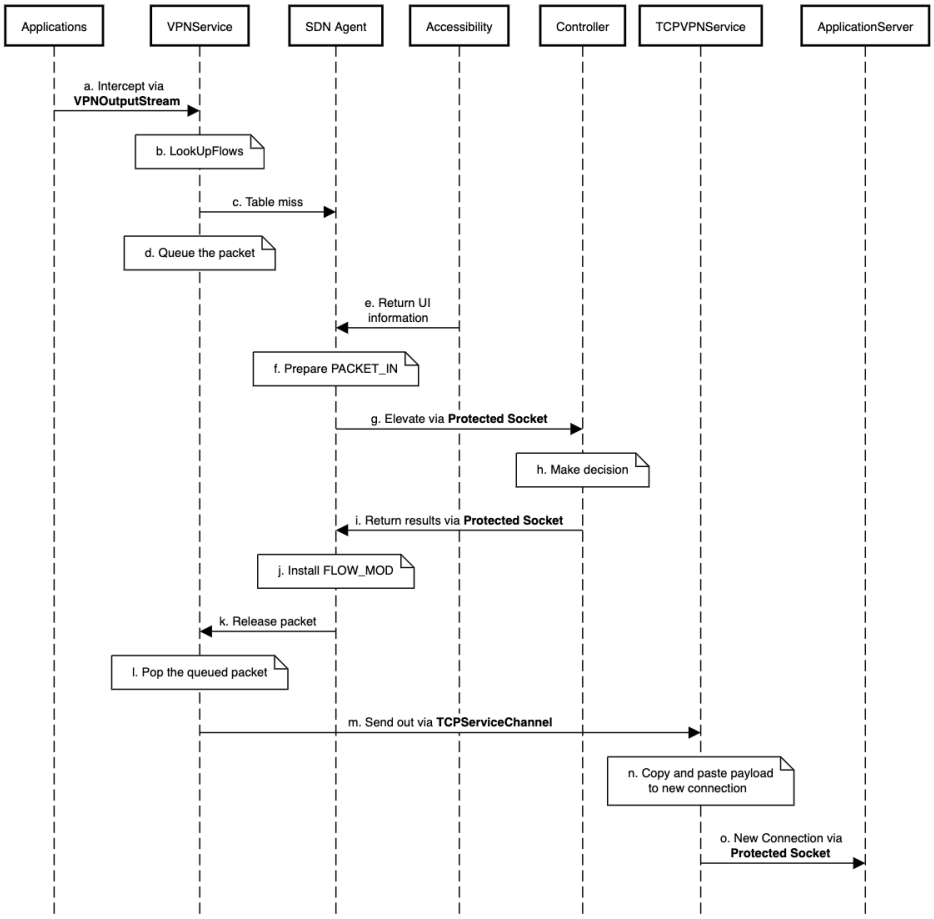

Figure 7.2: The sequence diagram of how an outbound packet is processed in our approach.

For user-initiated traffic, the UI sensor requires a historical match between the destination and the UI actions or requires the remote machine (e.g., host name or IP address) to appears in the UI context data (e.g., in a URL).

We begin our exploration by determining the baseline activity associated with an application without end-user activity. While the each tested application is idle, we record all network activities for three different sensor types. All such network traffic is added to a `Baseline List` allow-list that approves background activities that do not need user intervention.

We next train our approach to construct an association between end-user actions and network activity. We use Appium [47] to automatically invoke a scripted sequence of UI elements. In our test, we did not traverse all possible UI elements; however, prior work shows such enumeration is feasible in a practical implementation by traversing the UI tree [26, 186]. Instead, we focus on different types of UI events for Android [35], such as `TYPE_VIEW_CLICKED`, `TYPE_VIEW_TEXT_SELECTION_CHANGED`, and `TYPE_TOUCH_INTERACTION_START`. These events only occur when a user interacts with the device (or in our case, when Appium uses its advanced privileges to replicate these inputs). In this training phase, we record the UI events and network behaviors observed at our sensors. We label this data as the `Training Data`, which defines what the UI information and network activity appears together in a non-compromised application. During our later testing phase, each sensor can construct allow-list rules using the training data to determine if it can correctly classify traffic.

In our evaluation, we choose applications that are popular on the Google Play Store. The programs cover a wide range of people's daily use. We study Gmail, YouTube, and Chrome. The Gmail and YouTube applications are representative of many smartphone programs because they typically interact with a constrained set of remote systems (e.g., the servers associated with the application developer or advertisers).

Other applications, like Chrome, allow the user to specify arbitrary destinations, providing significant flexibility to the user while complicating profiling efforts. The training phase for such applications may be less effective since it is unlikely that all future user-supplied destination will be captured during the training phase. Since the

Application under Test	Match Rates		
	IP Sensor	DNS Sensor	UI Sensor
Gmail	46.24%	100%	100%
YouTube	63.12%	100%	99.69%
Chrome	14.42%	30.56%	97.93%

Table 7.1: Match rates across sensor types in the testing data set. DNS sensors appear to be more effective than IP sensors, but still have low match rates when destinations are specified by the user. The UI sensor can detect these destinations and leverage them in match rules.

UI sensor can supply UI event information containing URLs, IP addresses, and DNS host names, the sensor may aid profiling efforts. In this experiment, we focus on user inputs in web browsers. Future work can enhance web application traffic analysis, as described in Section 7.6.1.

During the Baseline List training phase, we activate each application and leave the program running without user input for two hours. For the Training Data stage, we develop four to five work flow scripts for Gmail and YouTube. For Chrome, we create four work flows that visit 4 different websites. For both these data sets, we collect the data at the SDN controller. During the testing phase, we execute the same work flow scripts while the controller employs a rule matching module with rules constructed from the previous training data.

In Table 7.1, we show our effectiveness results. We first examine the accuracy of these different sensor classes using the Gmail and Youtube applications. We see that the Network Header Sensor has the lowest accuracy for these applications. The result appears to be influenced by DNS load balancing or the effects of content distribution network (CDN) deployments. The DNS sensor's accuracy is 100% for Gmail and YouTube, demonstrating the value of host name matching rules.

When examining web browser traffic, we see that the Network Header Sensor and the DNS sensor have lower accuracy, matching around 14.42% to 30.56% of traffic. In contrast, our UI sensor enables a match rate of 97.93% at the controller. This highlights the importance of understanding the user-specified destinations when profiling traffic.

The window of traffic to consider for these sensors is important. When we configured the SDN agent to send events from only the last three seconds, our match rate was only 76.75% whereas raising that window to four seconds achieved the higher 97.93% match rate. This demonstrates the importance of tuning the reporting window appropriately.

7.5 Performance Evaluation: Newtork and Resources

We explore the performance implications of AppJudicator via an Android emulator and using a physical device. We evaluate the network end-to-end delay for network packets, throughput, and the impact on CPU, RAM, and battery consumption.

7.5.1 Networking Evaluation

Our local VPN service intercepts all outgoing and incoming packets as packets traverse the VPN service and SDN agent modules. The first packet of each flow further must wait for context from the UI Monitor, the elevation to SDN controller, and for the local Android SDN agent to install the rule. We perform this test on a simulated Google Pixel 4 phone with Android API level 30. We run our SDN controller on a separate machine on the same local network as the emulated phone. The controller is a Python program that responds to each `PACKET_IN` with a `PACKET_OUT` upon making a decision. The controller extracts any included UI information from the `PACKET_IN` messages.

We measure the network latency of the approach by recording two time stamps in AppJudicator: the time when a packet is first intercepted (t_1) and when that packet is about to be transmitted to the remote destination (t_2). The delay between these two time stamps includes any delays resulting from the consultation with the SDN agent, the UI Monitor, and the SDN controller (as described in Section 7.3). Accordingly, the difference between t_2 and t_1 represents the total additional delay added by AppJudicator. We use the `SystemClock.elapsedRealtimeNanos()` Android API [58] to obtain nanosecond-resolution times. To generate network traffic while collect timing data, we visit a sequence of websites by entering URLs in Chrome and clicking links rendered on the browser.

In our evaluation session, we collect timing data for 3,520 packets, including 352 TCP `SYN` packets. These `SYN` packets incur extra delay due to the SDN elevation process. As shown by the blue line in Figure 7.3, Appjudicator adds 26 milliseconds or less of delay to around 90% of all TCP SYN packets. The median delay is 8.67 millisecond, with a standard deviation of 11.89 milliseconds, resulting in an average delay of 11.15 milliseconds. These results show that there is a small portion of `SYN` packets that incur much larger delays than the typical `SYN` packet. Future work may be able to optimize the system by identifying the sources of such delay and eliminating them.

For non-`SYN` TCP packets, the process does not require SDN consultation. As a result, the delays are much smaller. As shown by the green line in Figure 7.3, Appjudicator adds at most 2.76 milliseconds or less to 90% of all non-SYN packets. The median is 0.61 milliseconds with a standard deviation of 3.50 milliseconds, resulting in an average delay of 1.52 milliseconds. As with the `SYN` packets, a minority of connections have much larger delays than the typical packet. These behaviors may be caused by the same phenomenon.

In practice, the location of the SDN controller can have a significant impact on the flow elevation latency. While our results are for an SDN controller in the same LAN as the phone, a remote SDN controller may have higher latency. Taylor et al. measured the connectivity of residential networks and found that for 90% residential networks, a public data center could be found to host an SDN controller with a 50 millisecond round-trip time (RTT). Importantly, only the first packet in each flow is affected by such elevation delays.

Next, we use the same configuration and evaluate the throughput impact of our system by downloading a large text file from an Apache web server hosted on a separate machine in the same LAN. Without our method, the download bandwidth achieves 100 MBytes per second. However, with Appjudicator running, the download time downgrades to 8.3 MBytes per second, which is a big reduction. As part of future work, we need to locate the cause of this throughput downgrade and optimize the performance. More future works regarding the evaluation is discussed in section 8.2.4.

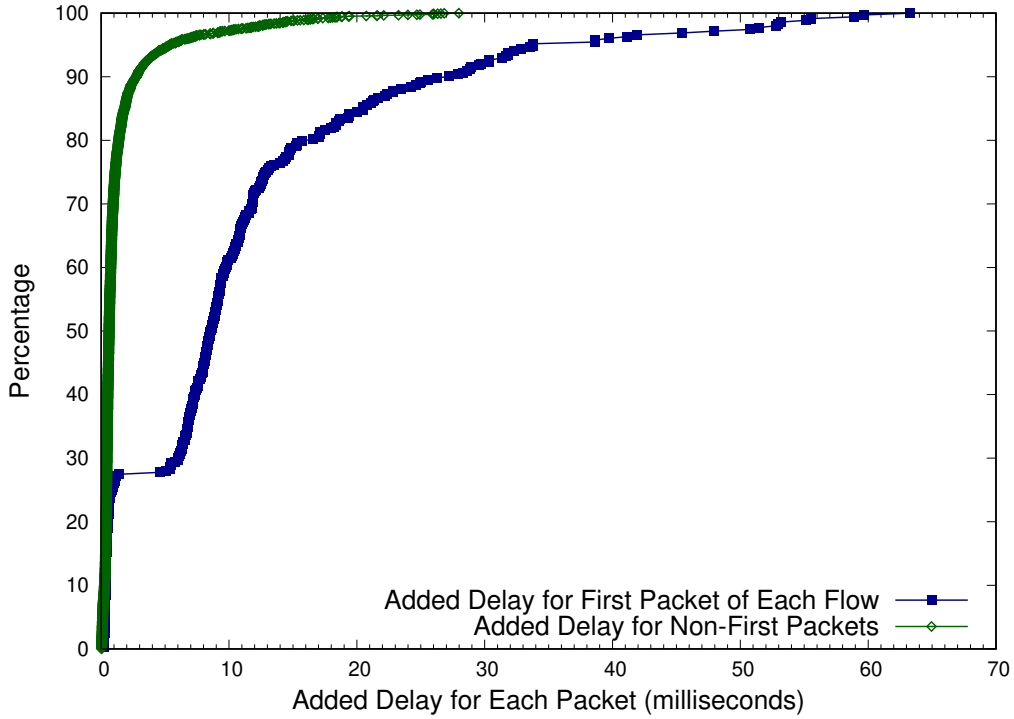

Figure 7.3: The added time delays in ms added by Appjudicator.

7.5.2 Computational Resources

Energy consumption is a significant consideration for mobile devices. Accordingly, we evaluate battery consumption along with the CPU and memory usage of AppJudicator on Android emulator.

Our tests use Android Studio's Profiler [57] to gather data. We run a Google Pixel 4 with an Android API level of 30 on a desktop with 8 cores and 16 GB memory. During the test, we interacted with the virtual device for 30 minutes. We emulate a typical user's usage of the device by visiting websites, checking email, watching Youtube videos.

When AppJudicator runs on the virtual Android device, it consumes a relatively constant amount of memory, with an average of 108.8 MBytes. The CPU consumption averages around 4%, with occasional peaks up to 50%. Android Profiler classifies the energy consumption level of the application as "light." As a result, we do not believe that Appjudicator has a substantial impact on computational resources or energy consumption.

7.6 Discussion

We now describe how our approach relates to two key concepts: web browsers and privacy.

7.6.1 Support for Browsers

Web browsers have different characteristics than most other mobile applications. A website visit often results in connections to multiple servers. Since website often employ end-to-end encryption, a middlebox typically cannot predict what resources are needed to complete a specific page load in advance. As a result, the profiling we describe in Section 7.4 needs additional functionality to achieve its goals.

Some web browsers have symmetric key export functionality [9], which allows external applications (like Wireshark) to decrypt TLS-protected communication. However, this functionality typically exists on traditional desktop operating systems and does not appear widespread on mobile devices. Techniques to hook specific library calls, such as the `libssl.so` library's `SSL_read` and `SSL_write` functions, with tools such as Frida [122] enable access to the plaintext communication associated with the application. However, these tools require root access to the phone's operating system, which introduces challenges for deployment and security.

Alternatively, the use of a trusted device-wide root certificate can enable interception of TLS communication in the VPN service module, enabling examination of unencrypted communication. The use of root certificates are typically discouraged since they break the end-to-end encryption model; however, the positioning of the interception and decryption on the endpoint device itself may offset some concerns. With such an approach, the web browser would no longer be able to see the TLS certificate associated with the remote server, since it would only see the certificate the VPN service module presents. Accordingly, the VPN service module would need to perform appropriate certificate validation of the remote server's certificate, much like the prior work in TLSDeputy [165].

7.6.2 Privacy Implications

The use of an SDN controller with the smartphone grants a third-party service the ability to perform network profiling and management. For corporate-owned devices in which the user agrees to monitoring, privacy may not be an issue. However, for other use-cases, this monitoring may significantly affect the privacy of the device user while running AppJudicator. We now explore these concerns and mechanisms that can help manage these risks.

As engineering efforts with AppJudicator continue, we will explore creating a controller on the phone that synchronizes policy with an organization's main controllers. That phone-based controller can then make access control decisions locally. To grant visibility to the organization, the phone's controller could report connectivity information to the organization controller, indicating what policy it applied but without supplying the UI sensor's detailed records. This would allow an organization controller to know a flow was related to a user's actions, without visibility into the details.

Another option would be to consult different controllers for different applications on a phone. For applications associated with an organization, the phone may consult the organization's controllers. For all other applications, the phone may contact a controller associated with an end-user-designated service provider. This model would allow users to split their phone into roles associated with their profession and with their personal life automatically.

7.6.3 Concluding Remarks

In this work, we propose and evaluate AppJudicator, an SDN architecture for mobile devices that associates UI elements with network flows. With the ability to consult external SDN controllers for assistance, the AppJudicator tool can respond to evolving threats while providing sufficient context for access control decisions. This mechanism helps increase the accuracy of network profiling from 14.42% to 97.93%. In our evaluation, we find that AppJudicator consumes acceptable computation resources while introducing modest network delay.

Chapter 8

Dissertation Conclusion

8.1 Concluding Remarks

To conclude, this dissertation contributes to residential network security from two perspectives: 1) protecting security-neglected devices within a home network, and 2) improving the accountability of residential network flows to combat Internet-wide threats more effectively and efficiently. This dissertation makes progress towards making residential networks function safely for network owners, make them more manageable, and have them improve network hygiene of their portion of the Internet environment.

To ensure a home network can safely serve the network owner, we first improve the security of less-protected devices connected to a home network. Our IoT device protection method adds additional authentication and real-time communication inspection between IoT devices and a smartphone (Chapter 3). With our protection, malicious requests and messages cannot reach IoT devices, thus attackers have less opportunity to control and damage the IoT devices. Our Android-based SDN approach improves the effectiveness of network profiling, which can enhance the security of Android devices (Chapter 7). Home network security can also largely benefit from secure mobile devices as they are less likely to be compromised when they are roaming, then less likely to propagate virus to other devices inside a home network.

The other type of devices that use the Internet service include laptops and desktops, which already have many protection options, such as real-time antivirus programs. As

a result, with our contribution, we avoid neglected devices in a home network. Our methods help network owners securely enjoy their smart home services.

Second, this dissertation utilizes network-level traffic labeling techniques to make residential networks more accountable for their own network traffic. In the view of the whole Internet, our method converts residential network components from unsupervised network areas to well-managed network areas, which can defend against DDoS attacks more effectively since the defense can happen at attack origins (Chapter 5), rather than totally relying on victim networks. Further, traffic labeling at the sources can grant persistent identity to residential networks. Requests from trusted residential networks can be distinguished from malicious traffic, so an organization is able to prevent their employees from suffering attacks like account lockout (Chapter 4). And an organization can avoid using VPN if their only goal is to simplify the authentication (Chapter 6), so residential network users can enjoy better network performance without sacrificing security benefits.

8.2 Limitations and Future Work

Despite the contributions of this dissertation, it is not possible to comprehensively address all security risks in a single document. Here we note some limitations and opportunities for follow-on work.

8.2.1 Authenticating Endpoints and Vetting Connections (Chapter 3)

Our SDN-based method that adds 2FA to the interactions between smartphone and IoT devices requires modifying the IoT applications installed on smartphones. Further, the state machine enforcement method currently requires manually analyzing the protocol and packets patterns of all IoT devices. However, other IoT devices, like smart washing machines, can have more complicated functionalities. There are also many other types of IoT devices. This variety and complexity of devices may affect the applicability of our method.

A third-party authority may help tackle these problems by labeling devices whose manufacturers modify their applications to be compliant with our architecture, and provide the network patterns of the devices via certain APIs, which can be directly imported to our traffic vetting system. Since users may prefer to buy the devices with security assurances, manufacturers have incentives to make their devices compliant with such security architectures.

This dissertation considers only the case that the smartphone and the IoT devices are inside the same LAN. The context can thus be unencrypted and our method is effective. However, IoT devices also accept commands from applications when the mobile phone is out of the network. Such communication is usually encrypted when it crosses the Internet. Further, even when both mobile device and IoT device are in the same LAN, they can still use HTTPS to communicate, which is up to their designs. Our protocol enforcement method does not apply to either situations since it cannot recognize encrypted packets.

8.2.2 Incentivizing Network Hygiene via Distributed Attack Reporting (Chapter 5)

The effectiveness of the distributed reporting architecture is highly related to the scale of the deployment. The ideal situation is that most of the households deploy our architecture and connect to an SDN controller that supports our protocol. In that scenario, most of the network flows sent from residential networks are under management at the sources. Otherwise, we still cannot effectively mitigate a considerable level of DDoS and release the defense burden on the victim networks.

8.2.3 Client Identity with Widespread IP Address Sharing (Chapter 6)

Our flow-level identification method is designed for the systems that use the Kerberos protocol and architecture. We note that 90% of the Top 1000 companies in the USA use Active Directory, which uses Kerberos for their single sign-on (SSO) services. Fo-

cusing on Kerberos-like architectures can enhance the impact of our approach. Another widely used SSO architecture, OAuth, is designed for convenient logins for various applications. It uses a different method and architecture, which is not considered in this dissertation.

Although our flow-level identifier provides a first-pass network-level authentication, there are other constraints to consider when deploying our method. First, for single sign-on (SSO) services, our method requires the application server to deploy the key verification module, even when the application server is managed by a third-party organization. For example, to verify logins to Microsoft Teams with a corporate account, we need to deploy our key verification module on Azure servers so that a user can obtain the token issued by the corporation's on-premise authentication server, and get verified on Azure-hosted services. However, this requirement may affect the deployability of our method.

Systems that have specifics requirements on cryptography, e.g., Federal Information Security Modernization Act (FISMA) compliant systems [102], should exercise care when deploying our method. If TLS cannot meet the cryptographic strength requirement, a VPN could be a better method to meet the standards. Further, a VPN is convenient to support legacy systems that are not upgraded to meet certain cryptographic requirements.

VPN is sometimes used for concealing communications for privacy purposes. In this case, the VPN encryption is not redundant, and our method cannot achieve a similar goal.

8.2.4 SDN for Mobile Devices (Chapter 7)

This dissertation explored an SDN architecture to monitor the networking on Android devices without rooting devices, while our evaluation only covered 3 applications. Our evaluations are a good proof-of-concept. The strongest evidence will be to evaluate our system on more daily application used to further prove the scalability and impact of our method.

Regarding the UI information, obtaining UI from the Android accessibility API

does not work for all types of applications, such as Google Maps, which uses OpenGL. Although we intuitively expect that most applications use Android accessibility API, we need to estimate the percentage of applications that can be incorporated into our UI-based security method. Further, currently we are only able to use *TextEdit* UI object to obtain user input. However, listening for keystrokes on a virtual keyboard can be a better method. However, the Android accessibility events listener does not include system soft keyboard. Exploring whether a third-party soft keyboard can invoke an UI accessibility event could be an interesting exploration.

To further improve security effectiveness, as part of the future works, we would like to explore methods that can analyze plain text network payload on the local VPN server to further understand web applications' network behaviors. For example, can we maintain a root certificate on the VPN server, so that we can observe the external URLs contained in each website? If yes, the controller can thus obtain and link these URLs with user-initiated web requests, and further add them into the allow-list based on this information. This can help us better understand web traffic on Android devices.

Since Android SDN needs to process each packet of a network flow, a bandwidth evaluation is required, as mentioned in section 2.4. We need to evaluate how Appjudicator impacts streaming services. We can evaluate this via video chat, playing YouTube videos, and playing online games. These results could explore that if our mobile SDN method incurs unacceptable usability issues.

Bibliography

[1] POP3 protocol client. `https://docs.python.org/3/library/poplib.html`, 2018.

[2] M. Alsaleh, M. Mannan, and P. C. van Oorschot. Revisiting defenses against large-scale online password guessing attacks. *IEEE Transactions on dependable and secure computing*, 9(1):128–141, 2012.

[3] Amazon, Inc. Amazon SES IP blocklist FAQs. `https://docs.aws.amazon.com/ses/latest/DeveloperGuide/blocklists.html`, 2019.

[4] I. Amazon Web Services. Aws shield managed ddos protection. `https://aws.amazon.com/shield/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc`, 2021.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, M. Damian, C. Seaman, N. Sullivan, K. Tomas, and Y. Zhou. Understanding the mirai botnet. In *USENIX Security Symposium (USENIX Security)*, pages 1093–1110, 2017.

[6] Apache. Access control. `https://httpd.apache.org/docs/2.4/howto/access.html`, 2020.

[7] K. J. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to Denial-of-Service attacks. In *USENIX Annual Technical Conference*, volume 38, 2005.

[8] M. Arlitt and C. Williamson. An analysis of TCP reset behaviour on the Internet. *ACM SIGCOMM Computer Communication Review*, 35(1):37–44, 2005.

[9] AskF5. K50557518: Decrypt ssl traffic with the sslkeylogfile environment variable on firefox or google chrome using wireshark. `https://support.f5.com/csp/article/K50557518`, 2021.

[10] P. Associates. 76% of North American broadband households use Wi-Fi as their primary connection technology. `https://www.parksassociates.com/blog/article/pr-01242018`, 2018.

[11] D. Athow. 7 best business VPN solutions 2020. `https://www.techradar.com/news/best-vpn-for-business-our-5-top-choices`, 2020.

[12] R. Atkinson. Security architecture for the internet protocol. IETF RFC 1825, `https://tools.ietf.org/html/rfc1825`, 1995.

[13] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In *International Workshop on Security Protocols*, pages 170–177. Springer, 2000.

[14] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad. Proposed security model and threat taxonomy for the Internet of Things (IoT). In *International Conference on Network Security and Applications*, pages 420–429. Springer, 2010.

[15] Belkin International, Inc. Wemo switch smart plug. `http://www.belkin.com/us/p/P-F7C027`, 2018.

[16] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *2010 IEEE International Conference on Communications*, pages 1–5. IEEE, 2010.

[17] A. A. Bilal and Umer. Study of abnormal TCP/HTTP connection. *Digitala Vetenskapliga Arkivet*, 2011.

[18] S. Bommareddy, M. Kale, and S. Chaganty. VPN device clustering using a network flow switch and a different mac address for each VPN device in the cluster, Aug. 2004. US Patent 6,772,226.

[19] S. Bradley. How to minimize the risks of split tunnel VPNs. `https://www.csoonline.com/article/3539509/how-to-minimize-the-risks-of-split-tunnel-vpns.html`, 2020.

[20] D. Bukszpan. Working remotely due to the coronavirus? this technology from your employer is key. `https://www.cnbc.com/2020/03/10/working-remotely-due-to-the-coronavirus-this-technology-is-key.html`, 2020.

[21] C. Buragohain and N. Medhi. Flowtrapp: An SDN based architecture for DDoS attack detection and mitigation in data centers. In *International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 519–524. IEEE, 2016.

[22] S. Burnett, L. Chen, D. A. Creager, M. Efimov, I. Grigorik, B. Jones, H. V. Madhyastha, P. Papageorge, B. Rogan, C. Stahl, et al. Network error logging: Client-side measurement of end-to-end web service reliability. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 985–998, 2020.

[23] W. Chang, A. Mohaisen, A. Wang, and S. Chen. Measuring botnets in the wild: Some new trends. In *ACM Symposium on Information, Computer and Communications Security*, pages 645–650. ACM, 2015.

[24] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. `https://tools.ietf.org/html/rfc7413`, 2014.

[25] W. Choi, G. Necula, and K. Sen. Guided gui testing of android apps with minimal restart and approximate learning. *Acm Sigplan Notices*, 48(10):623–640, 2013.

[26] Z. Chuluundorj, C. R. Taylor, R. J. Walls, and C. A. Shue. Can the user help? leveraging user actions for network profiling.

[27] Cloudflare. Understanding the cloudlfare security level. https://support.cloudflare.com/hc/en-us/articles/200170056-Understanding-the-Cloudflare-Security-Level, 2020.

[28] I. Cloudflare. Cloudflare DDoS protection. https://developers.cloudflare.com/ddos-protection/, 2021.

[29] M. Cooney. Coronavirus challenges remote networking. https://www.networkworld.com/article/3532440/coronavirus-challenges-remote-networking.html, 2020.

[30] R. Craven, R. Beverly, and M. Allman. A middlebox-cooperative TCP for an end-to-end internet. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 151–162. ACM, 2014.

[31] C. Cullen. Sandvine releases 2019 global internet phenomena report. https://www.sandvine.com/press-releases/sandvine-releases-2019-global-internet-phenomena-report, 2019.

[32] J. David and C. Thomas. DDoS attack detection using fast entropy approach on flow-based network traffic. *Procedia Computer Science*, 50:30–36, 2015.

[33] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *USENIX Security Symposium*, volume 42, 2001.

[34] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 122–133, 2017.

[35] A. Developers. Accessibilityevent | android developers. https://developer.android.com/reference/android/view/accessibility/AccessibilityEvent, 2021.

[36] J. Dufresne. Python-ldap on github. https://github.com/python-ldap/python-ldap/blob/python-ldap-3.2.0/Doc/index.rst, 2017.

[37] S. Durinovic-Johri and P. E. Wirth. Access control system with lockout, 1997. US Patent 5,699,514.

[38] K. Egevang and P. Francis. The IP network address translator (NAT). Technical report, RFC 1631, May, 1994.

[39] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[40] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A learning-based approach for IP geolocation. In *International Conference on Passive and Active Network Measurement*, pages 171–180. Springer, 2010.

[41] A. C. Estes. The dangers of supercookies. `https://www.theatlantic.com/technology/archive/2011/08/dangers-supercookies/354297/`, 2011.

[42] F5, Inc. Dynamic blocklisting of IP addresses. `https://docs.nginx.com/nginx/admin-guide/security-controls/blocklisting-ip-addresses/`.

[43] F5, Inc. What is a reverse proxy server? `https://www.nginx.com/resources/glossary/reverse-proxy-server/`.

[44] N. Feamster. Outsourcing home network security. In *ACM SIGCOMM Workshop on Home Networks*, pages 37–42. ACM, 2010.

[45] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *DARPA Information Survivability Conference and Exposition*, volume 1, pages 303–314. IEEE, 2003.

[46] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, et al. Architectural guidelines for multipath TCP development. IETF RFC 6182, `https://tools.ietf.org/html/rfc6182`, 2011.

[47] J. Foundation. Appium: Automation for apps. https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide, 2021.

[48] N. George. Upload vs. download speed: what's the difference? https://www.allconnect.com/blog/difference-between-download-upload-internet-speeds/, 2019.

[49] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, and J. Polakis. O single sign-off, where art thou? an empirical analysis of single sign-on account hijacking and session management on the web. In *USENIX Security Symposium*, pages 1475–1492, 2018.

[50] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A framework for IP-based virtual private networks. IETF RFC 2764, https://tools.ietf.org/html/rfc2764, 2000.

[51] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 72–81. IEEE, 2013.

[52] F. Gont, R. Atkinson, and C. Pignataro. Recommendations on filtering of IPv4 packets containing IPv4 options. IETF RFC 7126 https://tools.ietf.org/html/rfc7126, 2014.

[53] Google. Https usage in chrome worldwide. https://transparencyreport.google.com/https/overview?hl=en&time_os_region=chrome-usage:1;series:time;groupby:os&lu=load_os_region&load_os_region=chrome-usage:1;series:page-load;groupby:os, 2020.

[54] Google. Prevent mail to gmail users from being blocked or sent to spam. https://support.google.com/mail/answer/81126, 2020.

[55] Google. Security risks with modified (rooted) Android versions, 2020.

[56] Google Developers. Create your own accessibility service | Android developers, Dec. 2019.

[57] Google Developers. Measure app performance with Android Profiler, Oct. 2020.

[58] Google Developers. Systemclock, Feb. 2021.

[59] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, and M. F. Theofanos. Digital identity guidelines, authentication and lifecycle management. `https://pages.nist.gov/800-63-3/sp800-63b.html#throttle`, 2018.

[60] S. Grover, M. S. Park, S. Sundaresan, S. Burnett, H. Kim, B. Ravi, and N. Feamster. Peeking behind the NAT: an empirical study of home networks. In *Internet Measurement Conference*, pages 377–390. ACM, 2013.

[61] T. Habets. Google Authenticator PAM module. `https://github.com/google/google-authenticator-libpam`, February 2018.

[62] N. A. Handigol. *Using packet histories to troubleshoot networks*. Stanford University, 2013.

[63] O. Haq, Z. Abaid, N. Bhatti, Z. Ahmed, and A. Syed. Sdn-inspired, real-time botnet detection and flow-blocking at isp and enterprise-level. In *2015 IEEE International Conference on Communications (ICC)*, pages 5278–5283. IEEE, 2015.

[64] Harvard University. Registrars of fortune 1000 companies - raw data. `https://cyber.harvard.edu/archived_content/people/edelman/fortune-registrars/fortune-list.html`.

[65] C. Herley and D. Florêncio. Protecting financial institutions from brute-force attacks. In *IFIP International Information Security Conference*, pages 681–685. Springer, 2008.

[66] K. Hill. Cloudflare blocking my IP? `https://community.cloudflare.com/t/cloudflare-blocking-my-ip/65453`, 2019.

[67] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[68] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu. Towards sdn-defined programmable byod (bring your own device) security. In *NDSS*, 2016.

[69] C. Hu and I. Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, pages 77–83, 2011.

[70] ickerwx. tcpproxy on github. `https://github.com/ickerwx/tcpproxy`, 2018.

[71] R. Inayat, R. Aibara, K. Nishimura, T. Fujita, Y. Nomura, and K. Maeda. MAT: An end-to-end mobile communication architecture with seamless IP hand-off support for the next generation Internet. In *International Conference Human Society at Internet*, pages 465–475. Springer, 2003.

[72] B. Ives, K. R. Walsh, and H. Schneider. The domino effect of password reuse. *Communications of the ACM*, 47(4):75–78, 2004.

[73] N. Jamili. xtables-addons. `https://github.com/nawawi/xtables-addons`, 2020.

[74] S. Jin and D. S. Yeung. A covariance analysis model for DDoS attack detection. *IEEE International Conference on Communications*, 4:1882–1886, 2004.

[75] A. G. Johansen. 10 benefits of VPN you might not know about. `https://us.norton.com/internetsecurity-privacy-benefits-of-vpn.html`, 2020.

[76] A. Kalysch, D. Bove, and T. Müller. How android's ui security is undermined by accessibility. In *Proceedings of the 2nd Reversing and Offensive-oriented Trends Symposium*, pages 1–10, 2018.

[77] J. Kastrenakes. FCC fines verizon 1.35 million over 'supercookie' tracking. `https://www.theverge.com/2016/3/7/11173010/verizon-supercookie-fine-1-3-million-fcc`, 2016.

[78] S. Kent and R. Atkinson. IP authentication header. IETF RFC 2402, 1998.

[79] S. Kent and R. Atkinson. IP encapsulating security payload (ESP). IETF RFC 2406, 1998.

[80] J. Y. Koh, J. T. C. Ming, and D. Niyato. Rate limiting client puzzle schemes for denial-of-service mitigation. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1848–1853. IEEE, 2013.

[81] J. Kohl and C. Neuman. The Kerberos network authentication service (V5). IETF RFC 1510, 1993.

[82] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[83] M. Komu, M. Sethi, and N. Beijar. A survey of identifier–locator split addressing architectures. *Computer Science Review*, 17:25–42, 2015.

[84] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: illuminating the edge network. In *ACM SIGCOMM Conference on Internet Measurement*, pages 246–259. ACM, 2010.

[85] O. Kulyk, S. Neumann, J. Budurushi, and M. Volkamer. Nothing comes for free: How much usability can you sacrifice for security? *IEEE Security & Privacy*, 2017.

[86] F. lab. The myth of network address translation as security. `https://www.f5.com/services/resources/white-papers/the-myth-of-network-address-translation-as-security`, 2016.

[87] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda. Access-miner: using system-centric models for malware protection. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 399–412, 2010.

[88] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem. mesdn: Mobile extension of sdn. In *Proceedings of the fifth international workshop on Mobile cloud computing & services*, pages 7–14, 2014.

[89] Y. Lei, J. P. Lanson, R. M. Kaldawy, J. Estrada, and C. A. Shue. Can host-based sdns rival the traffic engineering abilities of switch-based sdns? In *2020 11th International Conference on Network of the Future (NoF)*, pages 91–99. IEEE, 2020.

[90] Y. Lei and C. A. Shue. Detecting root-level endpoint sensor compromises with correlated activity. In *International Conference on Security and Privacy in Communication Systems*, pages 273–286. Springer, 2019.

[91] K. Lister. Latest work-at-home/telecommuting/mobile work/remote work statistics, year = 2020, bdsk-url-1 = https://globalworkplaceanalytics.com/telecommuting-statistics. `https://globalworkplaceanalytics.com/telecommuting-statistics`, March.

[92] Y. Liu, C. R. Taylor, and C. A. Shue. Authenticating endpoints and vetting connections in residential networks. In *International Conference on Computing, Networking and Communications (ICNC)*, 2019.

[93] I. Livadariu, K. Benson, A. Elmokashfi, A. Dhamdhere, and A. Dainotti. Inferring carrier-grade NAT deployment in the wild. In *IEEE INFOCOM*, pages 2249–2257, 2018.

[94] G. LLC. Google cloud armor. `https://cloud.google.com/armor`, 2021.

[95] D. C. MacFarland, C. A. Shue, and A. J. Kalafut. Characterizing optimal DNS amplification attacks and effective mitigation. In *International Conference on Passive and Active Network Measurement*, pages 15–27. Springer, 2015.

[96] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[97] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 90–102. ACM, 2009.

[98] A. M. Mandalari, A. Lutu, A. Dhamdhere, M. Bagnulo, and K. C. Claffy. Tracking the big nat across europe and the us. *arXiv preprint arXiv:1704.01296*, 2017.

[99] A. Margosis. Security baselines for Windows 8.1, Windows server 2012 R2 and Internet Explorer 11. `https://blogs.technet.microsoft.com/secguide/2014/08/13/security-baselines-for-windows-8-1-windows-server-2012-r2-and-internet-explorer-11-final/`, 2014.

[100] A. Margosis. Security baseline for Windows 10. `https://blogs.technet.microsoft.com/secguide/2016/10/17/security-baseline-for-windows-10-v1607-anniversary-edition-and-windows-server-2016/`, 2018.

[101] T. McCue. Benefits of a VPN. `https://www.forbes.com/sites/tjmccue/2019/06/20/benefits-of-a-vpn/#149611632466`, 2019.

[102] K. A. McKay and D. A. Cooper. Guidelines for the selection, configuration, and use of transport layer security (tls) implementations. `https://csrc.nist.gov/News/2019/nist-publishes-sp-800-52-revision-2`, 2019.

[103] 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.

[104] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. In *ACM SIGCOMM Conference on Internet Measurement*, pages 336–341. ACM, 2004.

[105] R. Merchant. New study from dashlane reveals extremes people will go to for online protection, shortcomings of steps they take to secure themselves, 2016. `https://blog.dashlane.com/study-reveals-extremes-people-go-online-protection/`.

[106] Microsoft. Autodiscover for exchange. https://docs.microsoft.com/en-us/exchange/client-developer/exchange-web-services/autodiscover-for-exchange, 2015.

[107] Microsoft Corporation. Troubleshooting. https://sendersupport.olc.protection.outlook.com/pm/troubleshooting.aspx, 2018.

[108] Microsoft support. Changing the default interval for user tokens in IIS. https://support.microsoft.com/en-us/help/152526/changing-the-default-interval-for-user-tokens-in-iis, 2018.

[109] Microsoft support. Office365 login page. login.microsoftonline.com, 2019.

[110] MITRE Corporation. CWE-645: Overly restrictive account lockout mechanism. https://cwe.mitre.org/data/definitions/645.html, 2019.

[111] A. D. Monica, M. Baldwin, S. Cai, and C. Casey. Thousands of apps, one identity. https://docs.microsoft.com/en-us/enterprise-mobility-security/solutions/thousands-apps-one-identity, 2016.

[112] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. on Computer Systems*, 24(2):115–139, 2006.

[113] R. Moskowitz, P. Nikander, P. Jokela, et al. Host identity protocol (HIP) architecture. IETF RFC 4423, 2006.

[114] MurphyMc. The POX network software platform. https://github.com/noxrepo/pox, 2017.

[115] M. E. Najd and C. A. Shue. Deepcontext: An openflow-compatible, host-based sdn for enterprise networks. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 112–119. IEEE, 2017.

[116] S. M. T. Nezhad, M. Nazari, and E. A. Gharavol. A novel DoS and DDoS attacks detection algorithm using ARIMA time series model and chaotic system in computer networks. *IEEE Communications Letters*, 20(4):700–703, 2016.

[117] S. Nichols. Corporate VPN huffing and puffing while everyone works from home over COVID-19? you're not alone, admins. `https://www.theregister.com/2020/03/11/corporate_vpn_coronavirus_crunch/`, 2020.

[118] K. Nishizuka. Carrier-grade-NAT (CGN) deployment considerations. IETF Draft, `https://tools.ietf.org/id/draft-nishizuka-cgn-deployment-considerations-00.html`, 2013.

[119] Q. Niyaz, W. Sun, and A. Y. Javaid. A deep learning based ddos detection system in software-defined networking (sdn). *arXiv preprint arXiv:1611.07400*, 2016.

[120] E. Nordmark and M. Bagnulo. Shim6: Level 3 multihoming shim protocol for IPv6. IETF RFC 5533, 2009.

[121] Norton Security. Hundreds of malicious apps are showing up on the google play store, disguised as legitimate applications. `https://us.norton.com/internetsecurity-emerging-threats-hundreds-of-android-apps-containing-dresscode-malware-hiding-in-google-play-store.html`, 2020.

[122] NowSecure. Frida, a world-class dynamic instrumentation framework. `https://frida.re/docs/home/`, 2021.

[123] nyxgeek. Lyncsmash. `https://github.com/nyxgeek/lyncsmash`.

[124] J. Pan, S. Paul, R. Jain, and M. Bowman. MILSA: a mobility and multihoming supporting identifier locator split architecture for naming in the next generation internet. In *IEEE GLOBECOM*, 2008.

[125] R. Pandrangi. IP prioritization and scoring system for DDoS detection and mitigation, Jan. 13 2015. US Patent 8,935,785.

[126] B. Patel, B. Aboba, W. Dixon, G. Zorn, and S. Booth. Securing L2TP using IPsec. IETF RFC 3193, https://tools.ietf.org/html/rfc3193, 2001.

[127] PCIPolicyPortal. PCI compliance password requirements: Best practices to know. http://pcipolicyportal.com/blog/pci-compliance-password-requirements-best-practices-know/, 2015.

[128] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based IP filtering. In *IEEE International Conference on Communications, 2003.*, volume 1, pages 482–486, 2003.

[129] C. E. Perkins. Mobile IP. *International Journal of Communication Systems*, 11(1):3–20, 1998.

[130] C. Pope and K. Kaur. Is it human or computer? defending e-commerce with captchas. *IT professional*, 7(2):43–49, 2005.

[131] Project Floodlight. Project Floodlight. http://www.projectfloodlight.org/floodlight/, 2018.

[132] L. F. C. Projects. Open vSwitch. https://www.openvswitch.org, 2016.

[133] Pylon Technology News. Active directory in today's regulatory environment. https://pylontechnology.com/active-directory-todays-regulatory-environment/, 2014.

[134] L. Qihoo 360 Technology Co. Insight into global DDoS threat landscape. https://ddosmon.net/insight/, 2019.

[135] A. Rao, J. Sherry, A. Legout, A. Krishnamurthy, W. Dabbous, and D. Choffnes. Meddle: middleboxes for increased transparency and control of mobile traffic. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, pages 65–66, 2012.

[136] F. Rebecchi, J. Boite, P.-A. Nardin, M. Bouet, and V. Conan. Traffic monitoring and DDoS detection using stateful SDN. In *IEEE Conference on Network Softwarization (NetSoft)*, pages 1–2. IEEE, 2017.

[137] Y. Rekhter, B. Moskowitz, D. Karrenberg, and G. de Groot. Address allocation for private internets. IETF RFC 1597 `https://tools.ietf.org/html/rfc1597`, 1995.

[138] P. Richter, F. Wohlfart, N. Vallina-Rodriguez, M. Allman, R. Bush, A. Feldmann, C. Kreibich, N. Weaver, and V. Paxson. A multi-perspective analysis of carrier-grade NAT deployment. In *Internet Measurement Conference*, pages 215–229. ACM, 2016.

[139] F. Rösler, A. Nitze, and A. Schmietendorf. Towards a mobile application performance benchmark. In *International Conference on Internet and Web Applications and Services*, volume 9, pages 55–59, 2014.

[140] A. Rousskov. Feature: Sslbump peek and splice. `https://wiki.squid-cache.org/Features/SslPeekAndSplice`, 2019.

[141] C. Ryan. Computer and Internet use in the United States: 2016. `https://www.census.gov/content/dam/Census/library/publications/2018/acs/ACS-39.pdf`, 2018.

[142] SANS Institute. Top 10 mistakes on windows internal networks. `https://www.sans.org/reading-room/whitepapers/windows/top-10-mistakes-windows-internal-networks-1016`, 2003.

[143] Sematext. Log analysis tutorial: What it is, why, and when devops use it. `https://sematext.com/blog/log-analysis/`, Sep 2020.

[144] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[145] W. Simpson. IP in IP tunneling. IETF RFC 1853, 1995.

[146] T. Sipola, A. Juvonen, and J. Lehtonen. Anomaly detection from network logs using diffusion maps. In *Engineering Applications of Neural Networks*, pages 172–181. Springer, 2011.

[147] A. J. Slagell, Y. Li, and K. Luo. Sharing network logs for computer forensics: A new tool for the anonymization of netflow records. In *Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks, 2005.*, pages 37–42. IEEE, 2005.

[148] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE INFOCOM*, volume 2, pages 878–886, 2001.

[149] K. Srinivasan, A. Mubarakali, A. S. Alqahtani, and A. D. Kumar. A survey on the impact of ddos attacks in cloud computing: Prevention, detection and mitigation techniques. In *Intelligent Communication Technologies and Virtual Mobile Networks*, pages 252–270. Springer, 2019.

[150] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. In *International Conference on Information and Communications Security*, pages 495–506. Springer, 2007.

[151] Standford University. Alphabetic list of us universities and domains. `http://doors.stanford.edu/~sr/universities.html`, 1996.

[152] Statistia. Global market share held by security appliance vendors from 2012 to 2019, by quarter. `https://www.statista.com/statistics/235347/global-security-appliance-revenue-market-share-by-vendors/`, 2019.

[153] J. Steinberger, A. Sperotto, H. Baier, and A. Pras. Collaborative attack mitigation and response: a survey. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 910–913. IEEE, 2015.

[154] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 73–86, 2002.

[155] G. P. Store. Noroot firewall. `https://play.google.com/store/apps/details?id=app.greyshirts.firewall&hl=en_US&gl=US`, 2020.

[156] G. P. Store. Netguard - no-root firewall. `https://play.google.com/store/apps/details?id=eu.faircode.netguard`, 2021.

[157] A. Studio. Create your own accessibility service. `https://developer.android.com/guide/topics/ui/accessibility/service`, 2019.

[158] A. Studio. Create your own accessibility service. `https://developer.android.com/guide/topics/ui/accessibility/service`, 2020.

[159] A. Studio. Ui/application exerciser monkey. `https://developer.android.com/studio/test/monkey`, 2020.

[160] Y. Sui, Y. Zhang, W. Zheng, M. Zhang, and J. Xue. Event trace reduction for effective bug replay of android apps via differential gui state analysis. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1095–1099, 2019.

[161] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: Connection migration for service continuity in the internet. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 469–470. IEEE, 2002.

[162] M. Sweet. CUPS plenary. `https://ftp.pwg.org/pub/pwg/liaison/openprinting/presentations/cups-plenary-may-18.pdf`, 2018.

[163] C. R. Taylor, T. Guo, C. A. Shue, and M. E. Najd. On the feasibility of cloud-based SDN controllers for residential networks. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–6. IEEE, 2017.

[164] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue. Contextual, flow-based access control with scalable host-based SDN techniques. In *IEEE INFOCOM*, 2016.

[165] C. R. Taylor and C. A. Shue. Validating security protocols with cloud-based middleboxes. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 261–269. IEEE, 2016.

[166] C. R. Taylor, C. A. Shue, and M. E. Najd. Whole home proxies: Bringing enterprise-grade security to residential networks. In *IEEE International Conference on Communications (ICC)*, 2016.

[167] The LEDE Project. Welcome to the OpenWrt project. `https://openwrt.org`, February 2018.

[168] Trend Micro Mobile Threat Response Team. Dresscode android malware finds apparent successor in milkydoor. `https://blog.trendmicro.com/trendlabs-security-intelligence/dresscode-android-malware-finds-successor-milkydoor/`, April 2017.

[169] P. Trudeau and S. Laroche. Configurable quality-of-service support per virtual access point (vap) in a wireless lan (wlan) access device, Nov. 11 2014. US Patent 8,885,539.

[170] B. Varettoni. Verizon to launch 5G residential broadband services in up to 5 markets in 2018. `https://www.verizon.com/about/news/verizon-launch-5g-residential-broadband-services-5-markets-2018`, 2017.

[171] A. Wang, W. Chang, S. Chen, and A. Mohaisen. Delving into Internet DDoS attacks by botnets: characterization and analysis. *IEEE/ACM Transactions on Networking (TON)*, 26(6):2843–2855, 2018.

[172] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu. Guileak: Tracing privacy policy claims on user input data for android applications. In

Proceedings of the 40th International Conference on Software Engineering, pages 37–47, 2018.

[173] Y. Wang, Y. Huang, W. Zheng, Z. Zhou, D. Liu, and M. Lu. Combining convolutional neural network and self-adaptive algorithm to defeat synthetic multi-digit text-based CAPTCHA. In *IEEE International Conference on Industrial Technology (ICIT)*, pages 980–985. IEEE, 2017.

[174] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 137–148, 2012.

[175] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *ACM Conference on Computer and Communications Security (CCS)*, pages 162–175. ACM, 2010.

[176] K. Weniger, J. Bachmann, and R. Hakenbert. Method for mitigating denial of service attacks against a home, Dec. 10 2009. US Patent App. 12/514,999.

[177] Wikipedia. Openflow. `https://en.wikipedia.org/wiki/OpenFlow`, 2019.

[178] R. J. Witty and A. Allan. Best practices in user ID formation. `https://www.bus.umich.edu/kresgepublic/journals/gartner/research/117900/117943/117943.html`, 2003.

[179] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security & privacy*, 2(5):25–31, 2004.

[180] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 303–313. IEEE, 2015.

[181] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In

Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 1043–1054, 2013.

[182] S. Yu, W. Zhou, and R. Doss. Information theory based detection against network behavior mimicking DDoS attacks. *IEEE Communications Letters*, 12(4):318–321, 2008.

[183] S. Zanero. Wireless malware propagation: A reality check. *IEEE Security & Privacy*, 7(5):70–74, 2009.

[184] H. Zhang, D. She, and Z. Qian. Android root and its providers: A double-edged sword. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1093–1104, 2015.

[185] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh. IoT security: ongoing challenges and research opportunities. In *IEEE Conference on Service-Oriented Computing and Applications*, 2014.

[186] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou. Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 93–104, 2012.