Inspecting Traffic in Residential Networks with Opportunistically Outsourced Middleboxes

Shuwen Liu, Yu Liu and Craig A. Shue
Worcester Polytechnic Institute
{sliu9, yliu25, cshue}@wpi.edu

***Abstract*—Home networks lack the powerful security tools and trained personnel available in enterprise networks. This complicates efforts to address security risks in residential settings. While prior efforts explore outsourcing network traffic to cloud or cloudlet services, such an approach exposes that network traffic to a third party, which introduces privacy risks, particularly where traffic is decrypted (e.g., using Transport Layer Security Inspection (TLSI)). To enable security screening locally, home networks could introduce new physical hardware, but the capital and deployment costs may impede deployment.**

In this work, we explore a system to leverage existing available devices, such as smartphones, tablets and laptops, already inside a home network to create a platform for traffic inspection. This software-based solution avoids new hardware deployment and allows decryption of traffic without risk of new third parties. Our investigation compares on-router inspection of traffic with an approach using that same router to direct traffic through smartphones in the local network. Our performance evaluation shows that smartphone middleboxes can substantially increase the throughput of communication from around 10 Mbps in the on-router case to around 90 Mbps when smartphones are used. This approach increases CPU usage at the router by around 15%, with a 20% CPU usage increase on a smartphone (with single core processing). The network packet latency increases by about 120 milliseconds.

I. Introduction

Residential networks have grown increasingly complicated. While end-point solutions, such as anti-virus and software firewalls, are effective for some devices, they are not available for others. Embedded devices associated with the Internet of Things may have varying security, but lack mechanisms to install additional security features. Residential users have expressed concern about the risks on their networks. In a study of fifteen people with smart home tools, eleven participants indicated they were worried about physical risks of these devices and five participants were concerned about the associated privacy risks [1].

In-network security controls, such as screening on routers or middleboxes, can help protect these devices from network-borne threats, but current consumer-grade routers do not effectively manage network risk [2]. Such consumer-grade routers have hardware with limited computational capabilities. While prior work has proposed lightweight functionality on residential routers [3], there are inherent limits on the tasks these routers can perform. As an example, efforts to profile and examine encrypted traffic using machine learning [4] would exceed the resources of many such routers.

With the limitations of residential routers, prior work has explore mechanisms to shift the computational tasks associated with network screening to remote servers. Feamster [5] proposed using software-defined networking (SDN) techniques to allow home networks to outsource their security and management functionality to cloud-hosted servers. Likewise, TLSDeputy [6] uses remote servers to validate the TLS certificates and protocol settings associated with home network connections to ensure the authenticity of communicating end-points. However, both techniques allow the operators of cloud infrastructure to have insight into the activities of a home network, introducing new privacy risks. They also expand the trusting computing base (TCB) to include servers and personnel outside the home.

In contrast to prior efforts, we consider mechanisms to deploy home network traffic inspection in an opportunistic fashion. We explore mechanisms to leverage existing devices in a home network when they are available to screen communication. In doing so, we ask the following research questions:

- To what extent can we utilize current resources within a home network to build real-time packet inspection?
- To what extent would such a packet inspection system influence the performance of the home network, in terms of traffic latency, resource consumption, and throughput?

Our approach uses devices such as smartphones, tablets, laptops and desktops to perform traffic analysis. These devices can operate as security proxies when they are available, enabling detailed analysis. In pursuing this direction, our work makes the following contributions:

- **Creation of Router and Middlebox Support:** We introduce a mechanism that forwards network traffic from a router to a middlebox to leverage the spare computational resources. We use open source firmware on a consumer-grade residential router. We use simple IP-address based screening as conservative example of the computational requirements of security tools. We build tools to screen traffic locally on a consumer-grade router to establish a baseline. We then implement a technique to transparently direct traffic through a smartphone middlebox using network address translation (NAT) rules on the router.
- **Performance Evaluation of Deployment Options:** We compare the baseline on-router inspection with diverting traffic through a smartphone that performs inspection. Our evaluation shows that on-router inspection has a

throughput of around 10 Mbps whereas outsourcing the inspection to a smartphone achieves roughly 90 Mbps throughput. The smartphone middlebox approach adds around 15% CPU usage to the router and 20% CPU usage to a smartphone (with single core processing). It introduces 120 milliseconds of round trip time (RTT) delay to network traffic.

II. Background and Related work

In this section we provide a background and describe prior work on residential network computation and security.

A. Computation in Residential Networks

A 2015 survey found that 77% of US households subscribed to broadband Internet service and 78% of homes have a desktop or laptop computer [7]. However, modern home networks face many security challenges. Attackers can gain sensitive information or directly control the devices and launch attack on other devices, such eavesdropping, replay attack, network scanning, and data theft [8], [9].

There are effective ways to detect these attacks, but they require sufficient computational resources. Hafeez *et al.* [8] find that machine learning methods can detect a series of attacks with accuracy as high as 99%. Jan *et al.* [10] propose a method to detect a compromised device that joins a botnet with very limited data through a deep learning algorithm. In this work, we create a platform using existing devices to enable such traffic inspection in home networks.

B. Perimeter Defense for Home Networks

Perimeter defenses can be useful for residential networks. While the basic NAT functionality on residential routers typically prevents unsolicited inbound communication, it is ineffective at detecting or stopping existing compromises within a network or attacks that are launched via a connection initiated from inside the network.

Li *et al.* propose applying deep learning anomaly detection techniques for securing home networks; however, their method runs on equipment with computational resources that may not be available in many home networks [11]. ParaDrop [12] proposes allowing third-party application providers to install lightweight containers to provide a gateway for simple tasks. However, ParaDrop does not have sufficient resources to run resource-consuming tasks like intrusion detection. Another work [13] adds plug-and-play devices to a consumer-grade router, which enables the router to work as an intelligent IoT gateway that can inspect traffic; however, it incurs capital costs and requires hardware modifications inside consumer routers that are likely beyond the technical abilities of many home users.

Shirali-Shahreza *et al.* [14] summarized commercial home network firewall products. Each requires the installation of additional devices in the network with an initial cost of at least $200 and with ongoing monthly service costs. These devices may augment or replace existing home routers. Some use virtual private network (VPN) techniques to tunnel traffic to a remote VPN server that inspects and analyzes home network traffic en route to the destination. These methods introduce additional costs and equipment for users.

To simplify home network management while retaining security capabilities, Feamster proposed to outsource security needs to a remote cloud server using an SDN architecture [5]. This approach allows experts and security professionals to manage the network remotely. Since cloud servers have greater computational resources, they can run controller modules that improve analysis while gaining a cross-network perspective. Other efforts explore firewall modules for such SDN controllers [8], [14]–[18]. Most of these efforts use the router as an OpenFlow switch in the home network. Others propose to use a locally-available device, such as a Raspberry Pi, instead. However, these outsourcing methods require users to trust a third-party provider. This may raise significant privacy concerns, particularly when network traffic must be decrypted to provide security services.

C. Edge Computing in Local Networks

The edge computing paradigm builds decentralized computing pools for processing jobs from clients, bringing the computation closer to the source of data [19]. Cloudlet [20] is a popular edge computing prototype that offloads tasks to nodes that can scale. These nodes can be hosted by ISPs or other providers. Drop computing [21] builds a collaborative computing cloud using mobile devices in which one device can offload tasks to other devices. When there is no available device, the system seeks help from cloud servers. This method is designed for ad hoc networks, which lack reliability since devices may enter and leave the network frequently. Similarly, Verbelen *et al.* [22] split tasks and offload them to a virtualized environment, either on mobile devices or on cloud servers. Gedeon *et al.*propose to use a more reliable device, such as a home network gateway, run a broker to coordinate tasks. The gateway seeks available cloudlet nodes to help with its tasks [23]. This method outsources computation to third-party platforms, which can raise privacy concerns. Their use of a broker on a residential router, and their finding that it does not introduce significant overhead, is inspiration for our own approach.

III. Approach

Recognizing the computational constraints in residential routers, our approach compares on-router inspection with techniques that offload this work to other devices in the local network. In doing so, our hypothesis is that redirecting network traffic to a locally-available device with greater computational resources while limiting the router's work to traffic forwarding may yield better performance than attempting to perform the inspection on the router itself. This led us to develop the approach of on-phone inspection via NAT redirection.

Our research compares two approaches: on-router inspection via `NFQUEUE` and on-phone inspection via NAT redirection. We start by introducing the threat model and scope we assume in this work. Then, we illustrate the process of on-router

inspection. Finally, we describe the functionalities of each component of the phone-based inspection platform and how they work together.

A. Threat Model and Scope

When exploring technologies related to security, a model of the assumed threats can determine the applicability of the work and its scope. In this work, we construct a platform that is designed to enable inspection of traffic that crosses the boundary of a home network. Our platform's goal is to provide computational resources for inspection tools while achieving reasonable performance. We do not seek to develop a new detection algorithm or technique. Instead, we use a particularly lightweight filter, one based on packet addresses, to demonstrate the minimal overhead costs associated with each.

A trusted computing base (TCB) is the set of devices that must operate correctly to achieve the desired security goals. Our TCB includes the residential router and any smartphones hosting middleboxes that proxy traffic. We do not need to trust the communicating endpoint device, the remote machine it is communicating with, or other infrastructure associated with the Internet. Unlike approaches that outsource communication, our TCB does not include third-party cloud servers or the personnel associated with cloud data centers.

B. On-Router Inspection via `NFQUEUE`

We create an approach that is designed to provide efficient on-router traffic inspection. We implement a basic C++ program that we compile to natively run on the router to inspect IP addresses. The program uses the `iptables` packet inspection tool and the `netfilter_queue` library (often referred to as `NFQUEUE`) to inspect traffic. Essentially, the `iptables` tool operates on each packet processed by the Linux stack on the router. This action occurs when packets cross from the LAN interfaces to the WAN interface associated with the Internet. The `iptables` program sets an `NFQUEUE` judgment for all packets, causing them to enter a kernel queue data structure. The C++ program extracts the packets from that queue, inspects the destination network address, and returns the packets to the kernel queue for transmission. This program represents the minimum inspection required for a general-purpose user-space inspection program on the router.

C. On-Phone Inspection via NAT Redirection

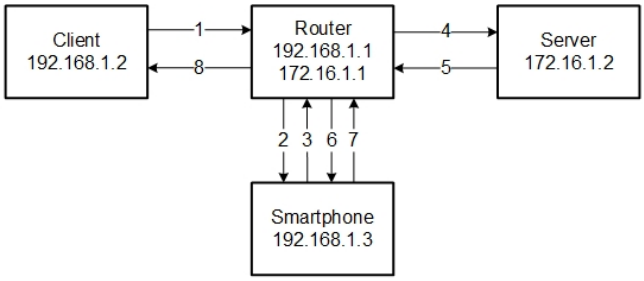
There are two components that support our traffic inspection on a separate smartphone. The first is a set of NAT rules on the router that will appropriately forward the traffic. For this, we use the `iptables` program, which can manage IP packet rules in the Linux kernel. We use the `iptables` NAT table to implement translation rules that transform the original destination IP address of the packets from the server to the IP address of the smartphone. This causes the traffic sent from the client to be redirected to the smartphone. In the example shown in Figure 1, we first apply a DNAT rule as `iptables -t nat -A PREROUTING -p tcp -s 192.168.1.2 -d 172.16.1.2 --dport 6666 -j DNAT --to-destination 192.168.1.3:6666` and an SNAT rule as `iptables -t nat -A POSTROUTING -p tcp -s 192.168.1.2 -d 192.168.1.193 --dport 6666 -j SNAT --to-source 192.168.1.1` to forward traffic to the smartphone. The smartphone can then work as a proxy that receives packets and sends them back to the router after inspection. When these packets return to the router, the router transforms their destination IP address to the original server destination IP address based on another DNAT rule, such as `iptables -t nat -A PREROUTING -p tcp -s 192.168.1.3 -d 192.168.1.1 --sport 7777 -j DNAT --to-destination 172.16.57.216:6666`. Since these NAT rules function bidirectionally, the packets sent from the server will traverse the reverse path through the smartphone. Rather than processing traffic as an arbitrary user space program in the router's Linux stack, our method forwards them using kernel data structures. This feature avoids potentially costly transitions to user space on the router.

The second component in our approach is the proxy software and service that runs on the smartphone. We implement a Java program that uses TCP to accept traffic for inspection on a pre-defined port. Figure 1 shows how the phone accepts traffic from the router using a new TCP connection. Since the smartphone is on the network path between the communicating endpoints, it receives the raw payload of every network packet. While we only apply IP list filtering in our tests, more advanced inspection can be deployed in our method, such as TLS inspection. The following performance evaluations show even this common lightweight operation saturates the router with on-device inspection whereas our NAT approach provides headroom. This approach can support more computationally demanding use cases without requiring new physical hardware deployments.

IV. Implementation

We implement our method in a lab environment. We run the OpenWrt 21.02.2 operating system (OS) on a consumer-grade TP-LINK AC1750 Wireless Dual Band Gigabit Router. We simulate a home network user with a client on a laptop with four cores and 16 GBytes of memory, running the Windows OS. We simulate a server outside of the home network on a laptop with four cores and 16 GBytes of memory, running the Ubuntu 20.04 OS. We use a smartphone with eight 2.0 GHz cores and 4 GBytes of memory, running the Android 11 OS as the proxy device.

For the network configuration, as shown in Figure 2, we create two VLANs: one is on interface `eth0` and the other is on interface `eth1`. We assign the LAN ports and wireless radio to the `eth0` VLAN and assign the WAN port to the `eth1` VLAN. The client connects to a LAN port via a category 6 Ethernet cable that supports full-duplex gigabit throughput. The server also connects to the WAN port using a category 6 cable. For the radio, we build an access point on



Packet #	Source IP	Source Port	Destination IP	Destination Port
1	192.168.1.2	47289	172.16.1.2	6666
2	192.168.1.1	6001	192.168.1.3	6666
3	192.168.1.3	7777	192.168.1.1	6001
4	172.16.1.1	6002	172.16.1.2	6666
5	172.16.1.2	6666	172.16.1.1	6002
6	192.168.1.1	6001	192.168.1.3	7777
7	192.168.1.3	6666	192.168.1.1	6001
8	172.16.1.2	6666	192.168.1.2	47289

Fig. 1. An example of packet forwarding via NAT rules. As the client sends the original packet to the server, the router modifies the packet and forwards it to the smartphone. After the smartphone performs packet inspection, it sends the packet back to the router. Then the router forwards it to the server. Since all of the NAT rules work bidirectionally, the packets sent from the server will follow the reverse path.

5.785 GHz using a Qualcomm Atheros QCA 9880 802.11ac adapter. We connect the smartphone to this access point at a distance of 3 feet with an unobstructed, line-of-sight path.

After configuring the home network, we add three NAT rules to `iptables` in the router, as described in Section III. These rules include SNAT and DNAT rules and have the capability of redirecting traffic between the client and the server to traverse the smartphone. On the smartphone side, we use Android Studio to build a Java application that performs packet inspection based on a malicious IP block list and hosts a proxy service.

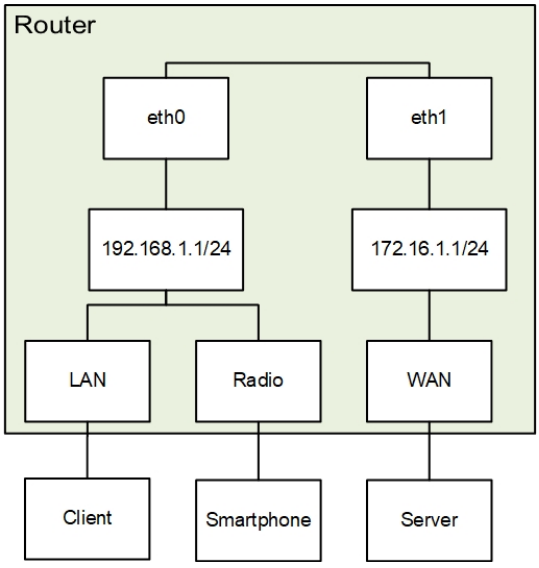


Fig. 2. The network configuration for our experiments

V. Performance Evaluation and Results

An on-router inspection module is straightforward since it uses a device that is already physically on the network path between communicating endpoints. To justify the added complexity of opportunistic middleboxes, we explore the performance implications of using such commodity devices. We first establish a baseline for the performance of the home network. We use a typical network setting, without the use of inspection functionality, to establish the baseline. We then explore on-router inspection using a simple block-listing application on a router. Finally, we examine an inspection method in which NAT rules are used to reroute traffic to a middlebox, using both a smartphone emulator and a physical commodity smartphone for analysis.

In examining these scenarios, we evaluate the performance of each using four metrics: flow throughput, end-to-end round trip time (RTT), the CPU usage at the router, and the CPU usage of the smartphone when it is in use. We note that there is performance improvement in using on-phone inspection after comparing the three scenarios with these four metrics.

A. The Baseline: LAN to WAN traffic

Our baseline scenario connects a client to a server though a residential router. Often, the WAN port is used on the router to connect to upstream networks, such as the Internet, and the servers available through those networks. Therefore, we connect an Ubuntu server to the WAN port of the router using a full-duplex category 6 Ethernet cable. The server uses a gigabit Ethernet card. We statically configure the IP addresses of the server and the router's WAN port within a subnet that is only used by those two devices.

We begin by exploring the case in which the client is connected to a LAN port on the router via a category 6 Ethernet cable. We use the router's built-in DHCP server, which assigns an address to the client in a subnet that the router and client share, yet is disjoint from the subnet used by the server. We use the router's default NAT capabilities to translate across the subnets, which is a common deployment scenario in homes. Using the `iperf3` benchmarking tool [24], we test a TCP connection between the client and the server. We configure `iperf3` to attempt to maximize throughput in the channel and observe it for 1,100 seconds. We conducted 3 trials and measured the throughput for 1,000 seconds after an initial delay of 100 seconds to accommodate TCP's slow-start behavior. As we see in the right-most two lines in Figure 3, the median download throughput is around 440 Mbps and the median upload throughput is around 254 Mbps, with tight distributions (the standard deviation is 4.90 Mbps for download throughput and 3.27 Mbps for upload throughput).

Since the communication throughput via Ethernet appears to be less than the medium's theoretical maximum, we explore whether the router could be causing a bottleneck. In particular, we examine the CPU of the router. While we test the maximum throughput, we use the `top` tool to record the CPU usage of the router for 1000 seconds. As shown in Table I, the CPU

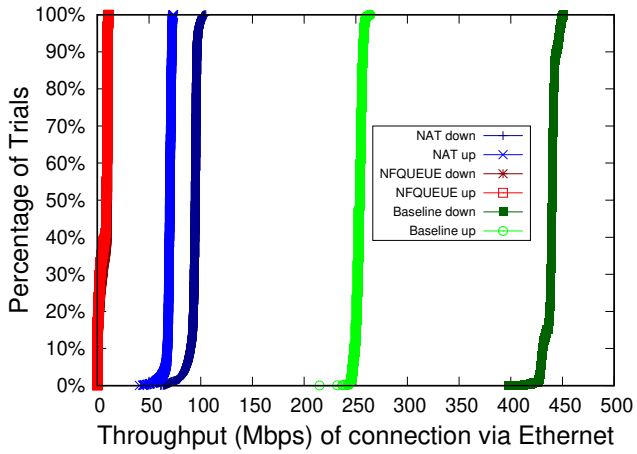


Fig. 3. Results from throughput tests when the client connects to the router via a category 6 Ethernet cable. The green lines show upload and download throughput under a baseline setting. The red lines show the throughput after applying on-router inspection via `NFQUEUE` library. The blue lines show the throughput after applying on-phone inspection using NAT redirection rules.

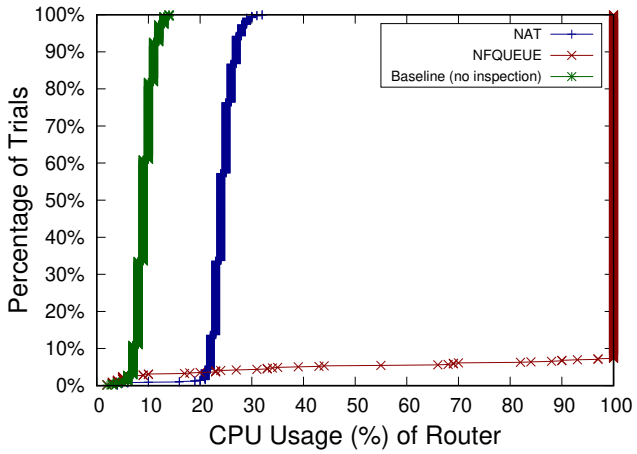


Fig. 4. CPU usage of the router when applying on-phone inspection using NAT redirection rules, applying on-router inspection via `NFQUEUE` library, and a baseline without inspection when throughput is limited to 10 Mbps.

usage of the router is at its limit more than 90% of the time when testing maximum throughput. These results suggest that the CPU of our router acts as a performance bottleneck when throughput is high.

TABLE I
CPU USAGE OF THE ROUTER WHILE TESTING THE MAXIMUM THROUGHPUT IN SIX SCENARIOS.

Percentile of Trials	10th	50th	90th
CPU Usage in Baseline Upload	100%	100%	100%
CPU Usage in Baseline Download	100%	100%	100%
CPU Usage in NAT Upload	98%	100%	100%
CPU Usage in NAT Download	97%	100%	100%
CPU Usage in NFQUEUE Upload	100%	100%	100%
CPU Usage in NFQUEUE Download	100%	100%	100%

To determine the added CPU usage from different traffic inspection methods, we need to measure the router's CPU usage in a moderate throughput scenario, rather than when throughput is maximized. We thus evaluate the scenario in which the TCP connection throughput is reduced to 10 Mbps of randomized payload to the server. We also record the CPU usage of the router for 1,000 seconds. The green line in Figure 4 shows that the median CPU usage of the router is 9.00% with standard deviation of 1.58%.

While throughput is an important metric, the end-to-end round trip time (RTT) is also important for understanding the delay introduced by the network paths and the router. To test this, we construct an echo program on the server and a recording device on the client to measure the time difference between the client sending a specific payload and receiving a reply. Across 1,000 trials, we see that the left-most line in Figure 5 has a median RTT of 1.12 ms with a standard deviation of 0.12 ms.

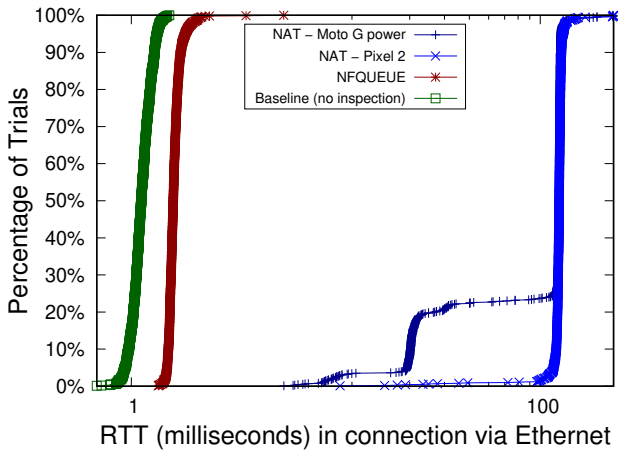


Fig. 5. RTT with a log scale in milliseconds between the client and the server when the client connects to the router via Ethernet. The leftmost green line shows baseline result. The middle red line shows the result after applying on-router inspection via the `NFQUEUE` library. The two rightmost blue lines show the results with two separate phones after applying on-phone inspection using NAT redirection rules.

B. *On-Router Inspection via `NFQUEUE`*

To explore whether the router itself can feasibly inspect traffic, we implement a basic C++ program, that is compiled to run natively on the router, to inspect IP addresses. The program's details are described in Section III-B.

We explore the throughput, RTT, and router CPU metrics of the on-device inspection program using the same tools and settings used in Section V-A. In the two left-most lines of Figure 3, we see the upload and download throughput after applying this inspection approach. We conducted 3 trials and measured the throughput for 1,000 seconds after an initial delay of 100 seconds to accommodate TCP's slow-start behavior. As we seen in the right-most two lines in Figure 3, the median download throughput is 9.62 Mbps, and the median upload

throughput is 8.40 Mbps (standard deviation of 3.91 Mbps for download and 3.93 Mbps for upload). Given this substantially decreased throughput from the baseline, we hypothesize that the change introduces a bottleneck on the router.

When we examine the CPU usage of the router, we confirm that this resource is exhausted. In Figure 4, we see that the baseline CPU usage is around 9% when throughput is limited to 10 Mbps, but is 100% when the router performs packet inspection. The process elevates all traffic to the router's Linux user space environment, which requires significant computational resources on the router. Such routers tend to be manufactured with lower-end CPUs for economic reasons [25] and there appears to be little headroom for this additional operation. However, when the router is not overwhelmed, as in the simple echo server RTT tests, we see that the on-device router introduces minimal RTT increases over the baseline. These results are shown by the red line in Figure 5, which is close to the baseline results.

C. On-Phone Inspection via NAT Redirection

With the CPU limitations of residential routers, we explore the potential of re-routing packets via a smartphone to inspect traffic. As described in Section III, we add three different NAT rules via `iptables` on the router to cause traffic to be sent via the phone. An example of traffic forwarding, after applying NAT rules, is shown in Figure 1.

Using the same settings as in the two prior sections, we explore the throughput when traffic is directed through the Moto G Power smartphone. In the middle two lines of Figure 3, we see that the median download throughput is 94.80 Mbps and the median upload throughput is 70.10 Mbps, with tight distributions (standard deviation of 4.32 Mbps for download and 2.87 Mbps for upload). The throughput is substantially higher than the on-router inspection approach in Figure 3. In effect, the processing of the NAT rules on the router may incur less computational overhead than the full process of inspecting the traffic. Since the router's CPU was the bottleneck in the on-router inspection scenario, this adjustment increases the rate traffic can flow.

In Figure 4, we can confirm that the NAT-based approach yields significantly lower CPU utilization than on-device inspection when throughput is limited to 10 Mbps. The middle line in that graph shows that the NAT approach has a median of 24.0% CPU utilization with a standard deviation of 2.61%.

The insertion of another device on the network path through a loop will necessarily increase the packet's propagation delay and may be observable in the overall end-to-end RTT. This is apparent in Figure 5, with the RTT of the NAT approach represented by the two right-most lines. We see patterns where 20% of traffic has an RTT less than 30.44 ms while 75% of traffic has an RTT over 120.17 ms. This is significantly higher than either the baseline scenario or when on-router inspection occurs. Importantly, this experiment uses a simple echo server approach and does not tax the CPU of the router. The on-router scenario would incur greater RTT delays when the CPU is a bottleneck due to processing delay.

Our last metric explores the energy usage of the proxy application on the phone. We run the application while maximizing throughput transmission from the client to the server, with a music-playing application in the background for comparison. We then record the CPU usage of the proxy application and the music application for 1,000 seconds using the `top` tool in the phone. We monitor the idle percentage of the 8 cores in the proxy device. In Table II, we show the CPU usage of the proxy application and the music application, along with the time for which the CPU core is idle. In this table, 100% represents the full utilization of a single core on the device and 800% represents the full utilization of all eight device cores. The first row in Table II represents the proxy application, which uses only about 21% of a single core verses the roughly 107% CPU usage of the music application in the median case. We see that the majority of the device's computational resources are unused. Even in lower-end smartphones, the CPU impact of the proxy was about 20% of a single core. As a result, we anticipate that the CPU-based energy consumption of the device would be a small fraction of a music application. Since phones are regularly used for music playing without significant power-related disruptions to end-users, it is likely that the proxy application would likewise represent a reasonable workload on phones.

TABLE II
CPU USAGE OF THE SMARTPHONE FOR DIFFERENT APPLICATIONS WHEN MAXIMIZING THROUGHPUT WHILE APPLYING ON-PHONE INSPECTION.

Percentile of Trials	10th	50th	90th
CPU Usage of Proxy App	18%	21%	24%
CPU Usage of Music App	98%	107%	114%
CPU Idle	535%	560%	584%

VI. CONCLUSION

While residential networks need traffic inspection and analysis tools to protect their traffic, existing residential routers lack the computational resources for on-router inspection. Even a straightforward, IP address-based inspection tool on such a router can greatly limit the throughput the router can support. However, with carefully-crafted NAT rules, a router can redirect communication through another device, such as a smartphone, to inspect traffic. This opportunistic outsourcing of inspection to in-network devices avoids the privacy concerns associated with outsourcing such services to cloud providers.

In our experiments, we find that NAT-based diversion through a smartphone can substantially raise the communication throughput from around 10 Mbps in an on-router implementation to around 90 Mbps through a smartphone. The router can periodically examine its ARP and DHCP data structures to detect the availability of a phone in the LAN, contact an application on the phone to configure proxy services, and then divert traffic through the phone to enable outsourced inspection. With such an approach, residential networks can opportunistically use available smartphones as middleboxes to enable higher-throughput traffic inspection.

REFERENCES

[1] E. Zeng, S. Mare, and F. Roesner, "End user security and privacy concerns with smart homes," in *Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017, pp. 65–80.

[2] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2015, pp. 336–341.

[3] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[4] B. Anderson and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *ACM Workshop on Artificial Intelligence and Security*, 2016, pp. 35–46.

[5] N. Feamster, "Outsourcing home network security," in *ACM SIGCOMM Workshop on Home Networks*. ACM, 2010, pp. 37–42.

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

[7] C. Ryan and J. M. Lewis, "Computer and internet use in the united states: 2015," https://www.census.gov/content/dam/Census/library/publications/2017/acs/acs-37.pdf.

[8] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "Iotkeeper: Securing iot communications in edge networks," *arXiv preprint arXiv:1810.08415*, 2018.

[9] Z. A. Almusaylim and N. Zaman, "A review on smart home present state and challenges: linked to context-awareness internet of things (iot)," *Wireless networks*, vol. 25, no. 6, pp. 3193–3204, 2019.

[10] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *IEEE Symposium on Security and Privacy (IEEE SP)*, 2020.

[11] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[12] D. Willis, A. Dasgupta, and S. Banerjee, "Paradrop: a multi-tenant platform to dynamically install third party services on wireless gateways," in *ACM workshop on Mobility in the evolving internet architecture*, 2014, pp. 43–48.

[13] A. Wieczorek and B. Markowski, "Intelligent iot gateway on openwrt," https://elinux.org/images/4/41/Intelligent_IoT_Gateway_on_OpenWrt.pdf, 2015.

[14] S. Shirali-Shahreza and Y. Ganjali, "Protecting home user devices with an sdn-based firewall," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 1, pp. 92–100, 2018.

[15] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home iot using openflow," in *International conference on availability, reliability and security (ARES)*. IEEE, 2016, pp. 147–156.

[16] R. F. Moyano, D. F. Cambronero, and L. B. Triana, "A user-centric sdn management architecture for nfv-based residential networks," *Computer Standards & Interfaces*, vol. 54, pp. 279–292, 2017.

[17] K. Xu, F. Wang, and X. Jia, "Secure the internet, one home at a time," *Security and Communication Networks*, vol. 9, no. 16, pp. 3821–3832, 2016.

[18] M. Boussard, D. Thai Bui, R. Douville, P. Justen, N. Le Sauze, P. Peloso, F. Vandeputte, and V. Verdot, "Future spaces: Reinventing the home network for better security and automation in the iot era," *Sensors*, vol. 18, no. 9, p. 2986, 2018.

[19] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things*, vol. 3, no. 5, pp. 637–646, 2016.

[20] M. Satyanarayanan, "Cloudlet-based edge computing," http://elijah.cs.cmu.edu.

[21] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, and G. Mastorakis, "Drop computing: Ad-hoc dynamic collaborative computing," *Future Generation Computer Systems*, vol. 92, pp. 889–899, 2019.

[22] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *ACM Workshop on Mobile Cloud Computing and Services*, 2012, pp. 29–36.

[23] J. Gedeon, C. Meurisch, D. Bhat, M. Stein, L. Wang, and M. Mühlhäuser, "Router-based brokering for surrogate discovery in edge computing," in *International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 145–150.

[24] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iperf - the ultimate speed test tool for tcp, udp and sctp," https://iperf.fr/, 2020.

[25] Hall, Michael, and R. Jain, "Performance analysis of openvpn on a consumer grade router," *cse. wustl. edu*, 2008.