Spring 6-1-2022

TorSH: Obfuscating consumer Internet-of-Things traffic with a collaborative smart-home router network

Adam Vandenbussche
Adam.C.Vandenbussche.22@Dartmouth.edu

TorSH: Obfuscating consumer Internet-of-Things traffic with a collaborative smart-home router network

A thesis

submitted to the faculty

in partial fulfillment of the requirements for the degree of

Bachelor of Arts

in

Computer Science modified with Engineering Sciences

by

Adam Vandenbussche

Dartmouth College
Hanover, NH

June 1, 2022

Advised by

Professor David Kotz
Professor Timothy J. Pierson

Department of Computer Science

Abstract

When consumers install Internet-connected "smart devices" in their homes, metadata arising from the communications between these devices and their cloud-based service providers enables adversaries privy to this traffic to profile users, even when adequate encryption is used. Internet service providers (ISPs) are one potential adversary privy to users' incoming and outgoing Internet traffic and either currently use this insight to assemble and sell consumer advertising profiles or may in the future do so. With existing defenses against such profiling falling short of meeting user preferences and abilities, there is a need for a novel solution that empowers consumers to defend themselves against profiling by ISP-like actors and that is more in tune with their wishes. In this thesis, we present The Onion Router for Smart Homes (TorSH), a network of smart-home routers working collaboratively to defend smart-device traffic from analysis by ISP-like adversaries. We demonstrate that TorSH succeeds in deterring such profiling while preserving smart-device experiences and without encumbering latency-sensitive, non-smart-device experiences like web browsing.

Acknowledgements

I would like to thank Professors David Kotz and Timothy Pierson for their expertise and guidance while writing this thesis, in addition to Professor Sean Smith, for agreeing to serve as the third committee member at my final presentation. I owe further thanks to Beatrice Perez for her valuable mentorship, her encouragement, and her (attempts at) holding me accountable while writing, to the rest of Dartmouth's SPLICE team – Chixiang Wang, Matthew Wallace, Namya Malik, and Nurzaman Ahmed – for their collaboration and companionship in research, and to Tina Pavlovich, for her administrative support throughout the project.

I owe a special thank you (and apology) to my friends and family, who supported me continuously throughout this endeavor and who were always willing to lend an eager ear when all I talked about over the past six months was *the thesis*.

This work results from the SPLICE research program, supported by a collaborative award from the National Science Foundation (NSF) SaTC Frontiers program under award number 1955805. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of NSF. Any mention of specific companies or products does not imply any endorsement by the authors, by their employers, or by the NSF.

An die Opfer des Konzentrationslagers Sachsenhausens, deren Blut durch den unheimlich entworfen Bodenablauf strömte, die Vorstellung davon die Schrift dieser Diplomarbeit gegen menschenwürdelose Innovation motivierte.

An die Opfer des Stasi-Gefängnisses Hohenschönhausen, die wegen ihres Widerstandes gegen eine privatsphärelose Welt litten.

Mögen ihre Geschichten nie vergessen werden.

To the victims of the Sachsenhausen concentration camp whose blood flowed through the specially designed floor drain, the image of which motivated the writing of this thesis on the consequences of innovation without regard for humanity.

To the victims of the Hohenschönhausen political prison, who suffered as a result of their opposition to a world without privacy.

May their stories never be forgotten.

Contents

Chapter 1

Introduction

As consumers increasingly deploy Internet-connected *smart devices* in their homes, residences are rapidly becoming novel troves of personal data [1]. When users interact with these devices, they produce data that divulges insight into their activities and behaviors [2]. In the age of "surveillance capitalism," the idea that those with access to this data would attempt to profit from it is far from unthinkable [3, 4].

The decision about whom to entrust with their data is guided by users' *privacy preferences*, which can be thought of as a spectrum: while most users care about privacy to some extent, the trade-offs they are each willing to make in the name of increased privacy may differ significantly. To some users, privacy risks may be worth the convenience of using a particular gadget or service. Particularly in consumer Internet-of-Things (IoT) environments, smart devices are advertised as convenience-enhancing products: smart assistants make it easier to check the weather, enjoy music, or otherwise manage the home, smart medical devices enable patients to receive medical treatments or services from the comfort of their home, smart cameras and smart sensors provide peace-of-mind while away, etc. [5, 6].

While consumers are free to choose smart devices from vendors that respect their privacy preferences, they may not realize that their smart devices inherently divulge data to intermediaries that assist in these devices' communications, including but not limited to their Internet service providers (ISPs), who can in turn use this data to profile their customers. Although most devices use encryption to safeguard their communications with their *cloud-based service providers* (although some disappointingly fail to do so), Landau explains how access to mere communications *metadata* can still divulge significant insight [7, 8]. For example, Apthorpe et al. demonstrated how an adversary with access to the metadata arising from encrypted communications between a sleep monitor, a security camera, a smart assistant, and each of their respective cloud-based service providers could infer detailed information about a smart-home resident's sleep habits, their movement through the home, and their daily routines. Although ISPs can profile non-smart-device users from their web-browsing traffic alone, Apthorpe et al. emphasize that the "single-purpose IoT nature" of smart devices can significantly enhance the meaning embedded in these profiles [2].

Rich user data such as that collected from a smart home could prove extremely valuable for service improvement, health or home insurance, advertising, or even political conformance. As such, *ISP-like adversaries* who have assembled consumer profiles may be tempted to use

them to sell targeted advertising space on their networks or to sell the profiles themselves to third parties. Beyond ISPs, such adversaries could include apartment-building network managers, VPN providers, or even governments. These actors are essential to providing Internet connectivity and are thus difficult to bypass, enjoying monopolistic or oligopolistic protection by virtue of market conditions or their location in network infrastructure. As such, consumers need strategies to defend themselves against profiling by these powerful intermediaries.

Unfortunately, existing solutions seeking to improve privacy against ISP-like adversaries fail to fully embrace particularities of IoT traffic while respecting user privacy preferences. While solutions such as Tor [9], Virtual Private Networks (VPNs) [10, 11], and other similar technologies offer promising solutions, all fall short of consumers' needs: a robust, low-cost, and low-complexity tool that leaves smart-home experiences unencumbered.

Mass user profiling by ISP-like adversaries – whether for power or for profit – is a threat best mitigated, especially when said profiling is enriched by smart-device traffic. With existing solutions inadequate in offering users the best of both worlds – that is, the autonomy to entrust smart-device vendors of their choosing with their smart-home data without divulging this data to intermediaries – we present **The Onion Router for Smart Homes (TorSH)**, a collaborative, private Tor network of smart-home routers working together to deter ISP-like adversaries from profiling users from their smart devices' traffic.

In this thesis, we make the following contributions:

- we motivate the need for a novel solution to the issue of user profiling from smart-home traffic;
- we present TorSH, we empirically and qualitatively evaluate its performance and effectiveness, and we outline directions for future work to further improve our solution.

The rest of this thesis is organized as follows. Chapter 2 provides background information relevant to this work. Chapter 3 presents TorSH's design goals and implementation details. Chapter 4 explains the methodology behind our evaluation of TorSH, while Chapter 5 presents the result of this evaluation. Chapter 6 summarizes and concludes our work.

Chapter 2

Context

This chapter motivates the need for a novel solution to combat smart-home user profiling by ISP-like adversaries and provides background information that is relevant to the remainder of this thesis.

2.1 Background information

In this section, we provide background information on user privacy preferences, smart devices and their traffic, and known analysis attacks on Internet traffic.

2.1.1 Privacy

Privacy is a deeply complex and often ineffable concept [12]. In the context of smart homes, it may be articulated as the total freedom from surveillance or as an individual's agency over their personal information [4, 12]. We conceptualize privacy as a spectrum of user preferences that may be informed by personal identities and experiences, situational context, and social norms; a practice that conforms to the privacy preferences of one user may violate those of another [13]. With privacy concerns being one of the leading barriers to global smart-home adoption, acknowledging this spectrum of preferences is essential to ensure the adoption and effectiveness of privacy-enhancing solutions for the smart-home environment [14].

Lederer et al. found that users considered the role of a data-collecting actor to be more important than the context of the data collection when contemplating whether a particular data-collection situation conformed to their privacy preferences [15]. Furthermore, Dolin et al. found that users were more likely to be comfortable with their data being used for targeted advertising if they deemed the collection of that data as fair [16]. These findings are particularly important in the context of our work because we are concerned with an intermediary collecting data based on intercepted communications, whereas studies like those of Lederer et al. and Dolin et al. usually consider more "obvious" actors such as smart-device vendors or a home's co-occupants. An intermediary whose privileged network

visibility empowers them to intercept and collect user data, possibly without meaningful notice and consent, is an actor scarcely considered in studies of consumer privacy preferences, and we believe that users are likely to deem such data collection as unfair.

Consider a user who has installed an Amazon Echo equipped with the Alexa smart assistant in their home [17, 18]. That user made the (hopefully informed) decision to introduce the product into their home knowing that Amazon could use the data produced by their interactions with Alexa to serve them more relevant ads while shopping on Amazon's e-commerce platform. While such data collection and use may fall within that user's privacy preferences (the trade-off is worth it to them), their ISP analyzing the traffic originating from the very same device to serve them relevant ads while they shop elsewhere on the Internet may not.

Alternatively, let us conceive of a business model whereby a user pays a firm a small monthly fee to use their smart assistant, and in return the firm explicitly promises *not* to use or resell the user's data for advertising purposes. Such a business model may better respect this user's privacy preferences than Amazon's model. However, the user is ultimately no better off if their ISP still analyzes the traffic originating from their interactions with the smart assistant to better target them online.

Many users find comfort in the idea of privacy by *anonymity*. That is, they feel more comfortable with the collection of their data as long as it cannot be tied back to them [13]. However, assuring users that an actor is taking adequate steps to anonymize their data is challenging, as users are rarely able to inspect this process for themselves.

2.1.2 Smart devices, smart homes, and their traffic

We paraphrase Kotz and Peters' definition of a *smart thing* for our definition of a *smart device* [19]:

> **Definition 1.** *Smart devices include digital appliances, small or large, that may communicate with each other, servers hosted on the Internet, or both, via network interfaces. These devices may interact with the environment via sensors or actuators, may be stationary or mobile, and may or may not be equipped with a user interface for manual configuration or interaction.*

Examples of smart devices include smart speakers, smart televisions, smart video cameras, smart mattresses, smart refrigerators, etc. We define a *smart home* as an individual residential dwelling, such as a detached home, a semi-detached home, or an apartment, in which one or more smart devices are deployed. We contrast smart homes with *not-so-smart homes*, which do not contain smart devices but may contain other Internet-connected devices such as personal computers or smartphones.

Smart devices can communicate with each other over the *local area network* (LAN) or with their cloud-based service providers over the *wide area network* (WAN). These cloud-based service providers may be maintained by the device's vendor or by other third parties. Communications with cloud-based service providers may be conducted using either vendors' custom protocols or existing protocols such as Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), Message Queueing Telemetry Transport (MQTT),

or Extensible Messaging and Presence Protocol (XMPP), which are in turn based on either the connection-based Transmission Control Protocol (TCP) or the connectionless User Datagram Protocol (UDP) for packet transmission [20]. Connection-based protocols' packet payloads can be protected from snooping using encryption protocols like Transport Layer Security (TLS) or HTTP-over-TLS (HTTPS).

Cloud-based service providers can be contacted by *endpoints*, which we follow Thomasset in defining as a tuple composed of an address, being either an Internet Protocol (IP) address (e.g., `93.184.216.34`) or a *fully qualified domain name* (FQDN; e.g., `www.example.com.`),[1] a 16-bit port number, and a packet transmission protocol (i.e., TCP or UDP) [21, 22]:

Definition 2. *Endpoint* := (*Address*, *Port* $\in [1, 65535]$, *Protocol* $\in \{ TCP, UDP \}$)

While devices can communicate with other devices on the same LAN using their local IP address, devices trying to communicate with servers across the WAN typically do so using their human-readable FQDNs. Because there may be many servers on the WAN that are capable of handling requests for one FQDN, clients must specify which one they would like to communicate with using its machine-readable, public IP address. Because IP addresses may be reassigned to various FQDNs over time, and there may be dozens or even hundreds of IP addresses for a given FQDN worldwide at any point in time, software developers typically encode FQDNs instead of IP addresses into their programs. However, this means that when devices want to communicate with a server, they first need to *resolve* its FQDN into one of the current IP addresses associated with it. This resolution process uses the hierarchical, distributed Domain Name System (DNS), whereby universities, content-delivery companies like Google and Cloudflare, and ISPs maintain databases relating FQDNs to the IP addresses of nearby servers ready to handle traffic destined for each domain. DNS requests are typically unencrypted; while encrypted alternatives such as DNS-over-TLS and DNS-over-HTTPS exist, traditional, unencrypted DNS dominates in practice [23–25]. Once resolved, devices, routers, or both may cache the DNS response, and the requesting device can begin to communicate with its target endpoint.

Most (but not all) smart-device communications are encrypted, meaning the payload attached to each packet is decipherable only by its intended recipient. However, some metadata, such as the IP addresses of the packet's source and destination as well as its size, is directly attached to the encrypted payload in an unencrypted *packet header*. Further metadata, such as the time and rate of successive communications, can be easily recorded by an observer [6, 8, 26]. We henceforth define *metadata* as follows:

Definition 3. *Metadata includes the IP addresses of a packet's source and destination, the packet's transmission protocol (i.e., TCP or UDP), the ports used for communication, the size of each packet, and the rate of communications, which may change over time.*

Typically, smart devices communicate more frequently with their endpoints while in active use (e.g., as a user is actively speaking to a smart assistant, as a user is standing on a smart scale) [2]. In fact, these communications may be necessary to ensure proper device functionality; blocking them could inhibit certain features [14]. Thomasset notes how

[1] FQDNs are typically suffixed by a period (`.`), which denotes the universal DNS *root zone*.

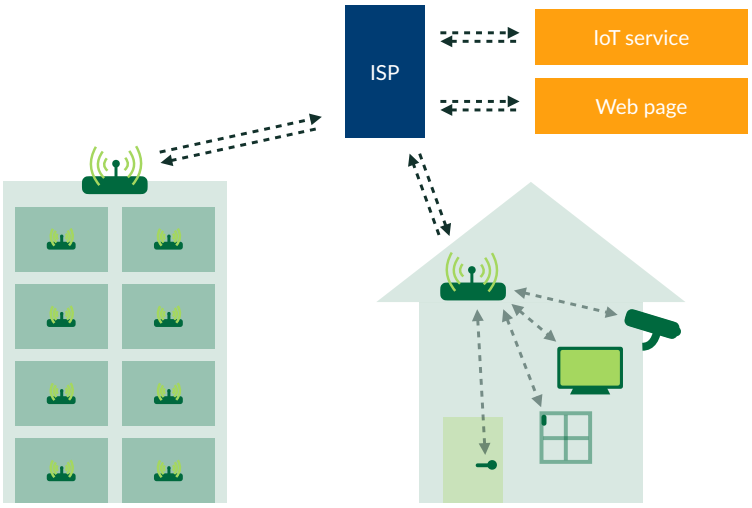


Figure 2.1: Topology of typical smart-home network. Single-family homes typically connect to the Internet via their ISP directly, whereas some apartment buildings may offer Internet connections to each of their units instead of having each privately contract with an ISP of their choosing [27].

(uncompromised) smart devices tend to communicate with fewer endpoints than personal computers or smartphones, whose web browsers and vast availability of third-party applications enable users to communicate with a far wider subset of devices and services hosted on the LAN or the WAN [21, 22].

Ardito et al. further highlight how endpoints may be either within the home's *cyberperimeter*, such as other smart devices within the home, or beyond it, such as cloud-based service providers [14]. The home's *router*, which facilitates communications between devices and services, delineates the home's cyberperimeter. When a smart device communicates with another device within the cyberperimeter, the router redirects packets to the proper device per the destination's local IP address. To communicate with external endpoints, smart devices require additional infrastructure, such as gateways (e.g., operated by an apartment building), ISPs, and DNS resolvers, all of which are privy to communications leaving the home's cyberperimeter. Figure 2.1 illustrates this topology. When sending packets to the WAN, routers typically replace the originating device's local IP address with the home's public IP address as part of a process known as *network address translation* (NAT); the destination IP address of any inbound response packets is then restored from that of the home to that of the original device [2].

2.1.3 Traffic-analysis and website-fingerprinting attacks

When adversaries are privy to network traffic – even when encrypted – they can analyze the metadata associated with that traffic to infer characteristics of or interactions with the originating devices [28]. Such attacks are known as *traffic-analysis attacks*, or *website-fingerprinting attacks* when their goal is to profile users based on their web-browsing habits.

We note that traffic analysis can also be performed to benevolent ends, such as to identify and isolate compromised devices (e.g., [21, 22, 29]) or to help network administrators identify which devices are connected to their networks (e.g., [30–33]).

Traffic analysis can be performed on both wireless and wired communications, albeit to different ends. Analyzing wireless communications requires an adversary to be in physical proximity to the victim, but analysis is not limited to only Internet traffic; adversaries can also analyze Bluetooth and Zigbee packets, for example. In the wired medium, the adversary lacks access to certain metadata only found in the wireless medium (e.g., signal strength), but the adversary no longer needs to be in physical proximity to the victim. Instead, they can observe traffic at any point in the wired network, although different points in the wired network will provide different insight. Namely, adversaries with access to only traffic outside the home's cyberperimeter cannot observe inter-device communications within the cyberperimeter.

While traffic-analysis attacks on smart devices in the wireless medium are well documented in literature (e.g., [5, 6, 28, 34, 35]), in this thesis we focus particularly on traffic-analysis attacks by an ISP-like adversary on smart-device traffic from beyond the home's cyberperimeter.

Inferring user–smart-device interactions from traffic metadata

Apthorpe et al. offer a three-step, high-level strategy outlining how an adversary could use IoT traffic metadata to infer smart-device activities [2].

1. **Separate traffic streams.** Because packet streams may be concurrent, the adversary must begin by grouping packets belonging to the same stream. They can do this by looking at the destination IP address (being that of the cloud-based service provider), as well as, in the case where two distinct devices communicate with the same IP address simultaneously, the destination port.

2. **Categorize traffic streams.** Once they have separated the streams, the adversary can try to infer the device that originated them. They can do this by again examining destination IP addresses and performing reverse DNS requests to infer their FQDNs, which may reveal the exact device or its vendor.

3. **Examine traffic rates.** Considering that many smart devices communicate with their endpoints more frequently while actively interacting with a user, simply looking at peaks in traffic rates for a particular destination IP–port combination may reveal insight into device interactions. The adversary could also develop more sophisticated fingerprinting models in a laboratory setting that they then deploy on live traffic.

2.2 Threat model

Ardito et al. emphasize that while the mere presence of any smart device in a home may present privacy risks, these risks are heightened for devices whose communications leave the home's cyberperimeter [14]. Given the unique vantage point of infrastructure providers beyond the home's cyberperimeter, they may be tempted to profit off their insight: even

with access to only the metadata associated with clients' network traffic, they can build rich user profiles that they can in turn use to improve services or sell to interested third parties such as advertisers, insurance companies, or any other actor seeking to *nudge* individuals to consumerist, political or other ends [4–6, 8, 14, 23, 26, 36–38]. These adversaries can further enrich these profiles by incorporating any demographic data they may have about their customers, such as their gender identity and residential address [36, 38].

We refer to infrastructure providers privy to smart-home communications metadata as *ISP-like adversaries*. These could include but are not limited to ISPs, commercial VPN providers, gateway operators (e.g., the management of an apartment building offering Internet to all their units [27]), a government offering ISP services, or even a government tapping into private ISP services (whether legally or illegally).

The absence of smart devices from a home does not preclude user profiling from network traffic; web-browser traffic alone still empowers ISP-like adversaries to build advertising profiles [36]. However, the "single-purpose IoT nature"[2] of smart devices can significantly enrich these profiles. ISP-like adversaries can identify traffic from smart devices by virtue of the small set of endpoints they may contact during normal operation, which may be unique to a particular device or brand of devices.

Case study

To illustrate the extent to which smart-device traffic can enrich user profiles, consider the following scenario:

> *Jane is responding to emails from her computer in the evening. At 9:00 p.m., she brushes her teeth and gets into bed, from where she watches television before eventually turning out the lights. At 10:30 p.m., she lays down and falls asleep. A few hours later, at 1:15 a.m., Jane wakes up and, craving a late-night snack, goes to the kitchen, opens the refrigerator, grabs a treat, and returns to bed at 1:30 a.m. At 7:30 a.m. the next day, she wakes up and gets dressed. At 8:00 a.m. she heads to the kitchen, opens the refrigerator to get some milk for her cereal and by 8:45 a.m. is sitting down at her desk again to start working remotely for the day.*

Let us begin by assuming Jane lives in a not-so-smart home – that is, she has a traditional cable television, pillow, and refrigerator. From her network traffic, her ISP will observe that she stopped sending emails around 9:00 p.m. and produced little-to-no Internet traffic until 8:45 a.m. the next day.[3] Judging by the time of day, Jane's ISP could infer that she went to bed sometime after 9:00 p.m. and woke up before 8:45 a.m., at which point she started working from her desk again.

Now let us assume Jane has a Wi-Fi-connected smart television from which she streams her television program, a Wi-Fi-connected pillow that logs her sleep data to the cloud so she can review her night's sleep in the morning, and a Wi-Fi-connected smart refrigerator. Now, using her network traffic, her ISP could see the time at which she stopped working, that she

[2]Credit for this expression belongs to Apthrope et al. [2].

[3]Some Internet traffic can be expected overnight even while Jane is not actively using any Internet-connected devices; her computer may perform software updates in the background, for example.

watched television for approximately an hour and a half before going to bed, the time at which she went to bed, that she awoke in the middle of the night and that she opened her refrigerator at that time (suggesting her craving for a midnight snack), the time at which she returned to her bed, the time at which she got out of bed at the next morning, the time at which she ate breakfast and that her breakfast likely composed of a refrigerated food, and the time at which she started to work again the next morning.

The difference in detail between these two profiles is stark. With access to such insight about Jane, advertisers would be in a much better position to target her with career coaching services (as a professional working long days), sleep aids, remedies (or tasty options) for late-night food cravings, and refrigerated breakfast foods. As such, Jane's smart-device-enriched traffic profile would be of great interest to ISP-like adversaries, as they could profit from its collection and sale to interested third parties.

Assumptions about adversary

We assume an ISP-like adversary that is interested in mass profiling their customer base and that has sufficient computational resources to do so at scale. We assert this in contrast with an adversary that is willing to dedicate significant computational resources towards profiling a single customer (e.g., a government spying on a suspected criminal), as the financial reward for doing so may not be worth the costs (e.g., in the case of a profit-seeking firm).

Assumptions about user

Furthermore, we assume a user that has average technical prowess and network-administration abilities. At a basic level, they can independently configure a new smart device in their home with the help of a manufacturer's setup guide but may lack confidence performing network administration beyond that.

2.2.1 IoT-traffic-analysis attacks in literature

The feasibility of user profiling by ISP-like adversaries has been demonstrated in literature.

Several works have successfully shown how merely analyzing smart-device traffic metadata can reveal insight into user behavior. Namely, Ma et al. demonstrated an ISP's ability to accurately identify both the type and quantity of a particular smart device hidden behind NAT, and to infer user–smart-device interactions from observed traffic [39]. Wan et al. proposed IoT Athena, a system that can identify user activities with their smart devices by matching network traffic against known interaction-specific traffic signatures [40]. Apthorpe et al. performed traffic analysis on a selection of commercially available Internet-connected smart devices, namely a sleep monitor, an indoor security camera, a light switch, and a smart assistant. They monitored outgoing Internet traffic from the home and observed clear spikes in DNS requests and TLS traffic whenever they interacted with each device [2].

As one recent example of a website-fingerprinting attack in literature, Gonzalez et al. profiled web-browser users based on their browsing habits. They collected the domain names of the web pages users visited, and with the help of existing databases mapping domain names to categorical interests (e.g., `booking.com` to *travel*, `espn.com` to *sports*), they were able to

serve more relevant ads to users, leading to higher click-through rates compared to the ads served by traditional advertising networks [36].

2.2.2 Profiling by ISPs currently

ISPs currently study their traffic to manage their networks and their bandwidth, to route packets, and to filter malicious traffic (e.g., spam, viruses) [3]. This analysis is deemed necessary to ensure a high quality of service and is widely accepted by users and governments alike. As Fuchs et al. warn, however, the risk of *surveillance creep*, or the gradual implementation of a practice (traffic analysis) once justified for one reason (network management) to another (user profiling) is ever present, especially when said surveillance can begin furtively [3].

Beyond being technically feasible as shown in literature, the collection and sale of customer data by ISPs have been demonstrated in practice as well. In 2007, David Cancel, the CEO of web-traffic analytics firm Compete Inc., admitted that American ISPs were selling customers' click streams to advertisers [41]. In 2008, British contextual advertising firm Phorm signed agreements to purchase customers' browsing habits with British mobile ISPs BT Group, Virgin Media, and Carphone Warehouse [3]. In 2019, the Mozilla Foundation accused American ISPs of using DNS requests to profile customers, tracking them with "supercookies," and extorting them with additional fees (on the order of $29 per month) to avoid "the collection and monetization of their browsing history for targeted ads" [42]. American ISPs – with many also offering cable and satellite television services – have since internally rebranded themselves as advertising networks instead of mere telecommunications companies, with some even launching marketing subsidiaries such as AT&T AdWorks and Verizon's Precision Market Insights that sell targeted ad space both online and on their mainstream television platforms using customers' data profiles [38, 43].

In the United States (US), the sale of customer data by ISPs without meaningful consent was legal until October 2016, when the Obama-era Federal Communications Commission (FCC) introduced new Internet privacy rules outlawing the practice [37]. In March 2017 – mere months later – Congress under the new Trump administration reversed the Obama-era rules before they came into effect [37, 44].

While it remains unclear whether ISPs' profiling of their customers is limited to analyzing their DNS requests or now includes richer data such as smart-device traffic, these firms have demonstrated an active interest in monetizing consumer data. The risk of surveillance creep where the law condones it is ever present, particularly when the unique network visibility of ISP-based advertisers, which empowers them to include the traffic from *all* of a home's smart devices in addition to its occupants' web-browsing habits, could provide them a competitive advantage over edge-based advertisers like Facebook and Google, who can only profile users to the extent they actively engage on their platforms [3, 37, 38]. By contrast, a smart-device vendor would only have this level of insight into smart-device traffic if they supported all of the smart devices in a consumer's home; emerging smart-device interoperability projects like Matter decrease the likelihood that this would occur because consumers no longer need to commit to any single brand's smart-home solutions to fully integrate their smart devices [45].

While some jurisdictions such as California, Canada, the European Union, and the United Kingdom (UK) have passed laws forbidding ISPs from collecting and selling customer data without meaningful consent, the case of the US shows that such laws are not permanent

but rather can be repealed with changing political landscapes [3, 37, 46, 47]. In practice, even where legal restrictions exist, firms may choose to knowingly violate laws when the penalty for doing so is less than the promise of profits the violation would allow or when enforcement lacks [48]. Furthermore, many jurisdictions worldwide lack privacy legislation altogether. Thus, while legal protections are certainly advantageous when safeguarding user privacy, empowering users to take matters into their own hands with a technical solution offers more robust protection.

2.3 Existing solutions

There have been many proposed defenses against IoT-user profiling by ISP-like adversaries – none of them perfect. In this section, we offer an overview and evaluation of various approaches proposed in literature and in open-source projects.

2.3.1 Traffic blocking and minimization

If traffic leaving the home's cyberperimeter poses a particularly great privacy risk, as suggested by Ardito et al., then one way to reduce that risk could be to minimize outgoing smart-device traffic or even block it entirely [14]. However, many devices require outgoing traffic to offer their full functionality (or any functionality at all) and blocking outgoing traffic would thus impede operation [14]. As Apthorpe et al. demonstrated, even certain smart devices whose traffic a user could reasonably expect to stay within the home's cyberperimeter (e.g., turning on an outlet from a smartphone on the same LAN as the outlet) may not function properly without WAN access [23].

As an alternative to traffic blocking, *edge computing*, or processing smart-device data within the home's cyberperimeter instead of on cloud servers when possible, would also greatly reduce the home's outgoing IoT traffic.

Apple HomeKit

Apple's HomeKit platform empowers users to restrict the network access of their home's smart devices from the *Home* app. On the most secure setting, devices are not permitted to access the Internet at all, which Apple warns could hinder device functionality and block software updates [49].

IoT Databox

Crabtree et al. proposed IoT Databox as an edge-computing solution for consumer smart devices [50]. Under the IoT Databox model, smart-device data is kept in encrypted *datastores* on a gateway device on the home's network. Users can then grant access to datastores to other devices on their LAN or to apps installed on the gateway that process the data locally and optionally export processed data to new datastores. With user permission, apps can also export their data to cloud-based services.

While promising, the IoT Databox solution is incompatible with existing smart devices, as it requires many vendors to collaborate and implement a novel solution that users in turn have to install on their routers, which may lack the computing resources to store and process data on device. Furthermore, some smart devices may require outgoing traffic to achieve their intended functionality. For example, when a user asks their smart assistant whether it will rain on a given day, the assistant requires Internet access to fetch the latest weather forecast, and this traffic could be studied by an ISP-like adversary.

2.3.2 Single-hop routing

Another technique to avoid profiling by ISP-like adversaries could be to have another device route traffic on a user's behalf.

Virtual private networks

A virtual private network (VPN) increases user privacy by routing Internet traffic via a trusted server on the WAN that places requests on the user's behalf and relays responses back to them [11]. They defend against traffic analysis by making all of a home's outgoing packets appear to be intended for the same destination IP address (that of the VPN server). While VPN usage among consumers is rapidly increasing, particularly in light of the recent commercialization of VPN services, using commercial VPN services to defend against profiling by ISPs merely shifts the burden of trust from the ISP to the VPN provider. A recent survey of commercial VPN providers by Khan et al. exposed widespread poor privacy and marketing practices, including misrepresentations of their servers' physical locations and failure to disclose terms of service and privacy policies – with only 23% of surveyed providers explicitly claiming that they do not log their customers' traffic [10].

As an alternative to untrustworthy commercial VPNs, users could configure their own private VPN servers. However, doing so would merely shift the apparent source of a smart home's traffic on the Internet, which could still enable their ISP – whether the same ISP as their home or a new ISP, possibly in a different jurisdiction than their home – to profile their traffic and target them with ads. By contrast, commercial VPNs offer the advantage of mixing multiple users' traffic together, such that an ISP would struggle to distinguish the traffic belonging to any given user. Finally, most consumers, including the type we assume in our threat model, lack the technical prowess to set up and maintain their own VPN server.

Some research has shown that even encrypted VPN traffic is not immune to effective traffic-analysis attacks [51]. For example, Herrmann et al. were able to identify the websites visited by users of popular VPN implementations (e.g., OpenVPN) using a simple Multinomial Naïve Bayes classifier, although their technique was only tested in laboratory conditions and would (by the authors' own admission) perform less accurately in real-world conditions [52]. We further discuss the risks of traffic-analysis attacks on single-hop routing solutions in Section 2.3.4.

VPN-Zero

Varvello et al. present VPN-Zero, a distributed VPN (dVPN) with strong privacy guarantees [11]. They envisioned a decentralized network of collaborative nodes acting as VPN servers for each other, whereby a user seeking to increase their privacy could randomly select another participating node through which it could route its communications *without the intermediary node being able to see where they were routing traffic to* in order to protect the privacy of the originating node. The authors proactively assume the possibility of malicious dVPN users seeking to route illegal traffic, such as child pornography or darknet markets, through other nodes who would in turn appear to be the originator of said traffic to their ISPs. To counteract these adversaries, each node would be able to set their own allowlist of the traffic they would be willing to route. The authors propose a zero-knowledge proof that would empower intermediary nodes to verify that a particular packet is destined for an allowlisted server without them being able to see its exact destination.

Such a system sounds promising for a collaborative network of smart-home routers that route traffic for each other. However, to offer their security and privacy guarantees, Varvello et al. can only support TLS v1.3 traffic with the proposed Encrypted Client Hello feature – two relatively new standards rarely implemented by smart devices – which disadvantages this solution for consumer IoT contexts [53].

2.3.3 Traffic shaping

Whereas strategies like outgoing traffic minimization and routing traffic through VPNs can defend against analysis of categorical packet metadata such as domain names, IP addresses, and ports, *traffic shaping* attempts to obfuscate packet size and packet timing. Reshaping traffic (e.g., fragmenting or padding packets), altering the timing of transmissions, or injecting random noise into outgoing traffic streams (i.e., packets appearing to originate from a particular device but do not originate from an actual user interaction) would confuse an ISP-like adversary attempting to profile users based on their outgoing traffic [54].

Two specific implementations of traffic shaping include *independent link padding* (ILP), a technique proposed by Datta et al. that pads or fragments packets as required to ensure transmissions match a pre-specified rate or schedule, and *stochastic traffic padding* (STP), an algorithm proposed by Apthorpe et al. that shapes upload and download traffic equivalently for activity resulting from different user interactions with smart devices and randomly injects additional upload and download traffic that is indistinguishable from legitimate traffic [23, 54]. While both techniques were shown to be effective, ILP could hinder users' smart-home experiences if devices normally transmit at rates higher than those specified in the ILP implementation. Other techniques include those of Alsheri et al., who proposed adding uniform noise to packets, Xiong et al., who proposed Differentially Private Shaper (DPS), a tunable differential privacy-based traffic-shaping technique, and Yu et al., who proposed PrivacyGuard, a deep-learning-based technique that learns traffic patterns from devices in the home and injects realistic-appearing outgoing noise [51, 55, 56]. While many more such techniques are published in literature, we chose these for their particular focus on consumer IoT traffic and their recency.

Traffic shaping that pads packets may consume additional bandwidth overhead, perhaps to an unrealistic extent. For example, Apthorpe et al. argue that ILP is too expensive for

real-world use: assuming ILP at a constant rate of approximately 40 KB/s, a home would consume an overhead of 104 GB per month. In contrast, STP would result in an overhead of only tens of megabytes in overhead traffic for a typical smart home [23].

Traffic-shaping techniques can be implemented in either smart devices or the home's router. However, the device-level approach taken by Datta et al., for example, requires implementation at development time, which precludes devices whose vendors no longer actively maintain them or choose not to implement traffic shaping. Implementing traffic shaping at the router level is more extendable, as the router could reshape outgoing packets from devices that do not themselves implement traffic shaping.

One significant drawback of traffic shaping as a defense against profiling is the need of additional infrastructure to avoid breaking communications with cloud-based service providers whose devices do not implement traffic shaping themselves. DPS, PrivacyGuard, STP, and the uniform noise technique of Alsheri et al. all require a specially configured intermediary, akin to a VPN, to reconstruct reshaped packets and ignore decoy traffic while forwarding legitimate traffic to its intended destination [23, 51, 55, 56]. The need for an intermediary server raises the question of who would manage it and whether they should in turn be trusted with the data to which the adversary would otherwise have access.

Finally, traffic shaping is not infallible to analysis. A recent work by Engelberg and Wool demonstrated that an ISP-like adversary who is aware that traffic shaping is occurring on IoT traffic can accurately identify the originating devices, even if they do not know the shaping parameters [57].

2.3.4 Multi-hop routing

Although single-hop routing provides some privacy protection against profiling by ISP-like adversaries, the solution has shortcomings as discussed in Section 2.3.2. An alternative to single-hop routing is multi-hop routing, whereby packets are routed through more than one intermediary before being forwarded to their intended recipient.

EPIC

Liu et al. present EPIC, a collaborative network of Wi-Fi routers within wireless range of each other that obfuscates the originator of a network request by selecting a different participant via a random walk to place the request on its behalf [58]. This selection process happens wirelessly before the packet reaches the Internet, thereby preventing its analysis by an ISP-like adversary and ensures that the intermediary node cannot determine for certain the identity of the true originator.

Although a promising solution when contemplating a collaborative network of smart-home routers, the requirement that participants be within wireless range of each other acts as a barrier to adoption; users living in rural or even some suburban areas could not benefit from such a network, seeing that their wireless router would likely be out of range from other routers also supporting the feature. Moreover, the protection offered to an EPIC user varies based on the number of participants in their vicinity. For example, assuming each unit is equipped with an EPIC-compatible router, a user living in a 6-unit building enjoys less privacy than a user living in a 100-unit building. Finally, the geographic proximity of the

nodes enables ISP-like adversaries to profile users on a neighborhood scale, which could still provide insight into a user's daily routines, demographic data, socioeconomic status, etc., based on aggregated data of similar users living nearby.

Tor

Tor is an open-source tool that empowers users to access the Internet anonymously by routing their traffic through a network of volunteer-run servers before sending packets to their intended destination [9]. Although many anonymization protocols have been proposed since the advent of computer networking, Tor has consistently been hailed as one of the most reliable and secure. The protocol gained infamy in 2013 when ex-National Security Agency (NSA) employee Edward Snowden used it to leak classified documents. Landau names Tor as one of the only general solutions to the issue of communication metadata protection in the context of web accesses [8].

Tor is based on *onion routing*, an anonymous-communication technique that was developed in the mid-1990s by the United States Naval Research Laboratory.[4] Under onion routing, packets are encapsulated into fixed-size *cells* before being routed through three (or more; Tor uses three by default) intermediary servers, collectively forming a *circuit*, before being forwarded to their intended destination. Each packet is enveloped under multiple layers of encryption – like the layers of an onion – with each only decryptable by the intermediary predetermined by the client. As a result, each intermediary is privy to some metadata pertaining to the user's web access, but not enough to piece together their entire interaction. Namely, intermediaries are known as *guard nodes* (also known as *entry nodes*), *middle nodes*, and *exit nodes*.

- **Guard nodes** are the first servers contacted in a circuit. They are privy to the client's identity (i.e., their IP address) but do not know its intended destination. Upon decrypting their layer, they are instructed to forward packets' contents to the middle node predetermined by the client.
- **Middle nodes** know neither the identity of the client nor of the intended recipient. Upon decrypting their layer, they forward packets to the exit node predetermined by the client or to another middle node if the client wishes to use more than three hops.
- **Exit nodes** are the last of the intermediaries. They receive packets from middle nodes, and upon decrypting their encryption layer, the intended destination is revealed. The identity of the client, however, is unknown to the exit node. Exit nodes assume the responsibility the traffic they route: from an ISP's point of view, exit nodes appear to originate the traffic they route, not the clients who originally produced it. As such, the Tor Project discourages everyday Internet users from hosting exit nodes. Instead, they encourage reputable institutions such as universities to do so and to declare their hosting of a Tor exit node to their ISP to mitigate legal responsibilities for their outgoing traffic [59].

Any responses from the destination server are rerouted back to the client via the same circuit, albeit with intermediaries traversed in reverse order, each rewrapping the response in a layer of encryption.

[4] Indeed, Tor is an acronym for **T**he **O**nion **R**outer.

Tor clients build circuits with the help of *directory authorities*, or special servers that advertise willing intermediary Tor nodes. As such, Tor is not fully decentralized. Tor circuits are ephemeral: a client will use the same circuit for a maximum of 10 minutes (as a recommended lifetime, clients are free to change this value in their configuration) after which point they will build a new circuit with different nodes. This prevents the need to build a circuit for each network request while maintaining a high level of anonymity.

As of mid-May 2022, Tor's approximately 3 million daily users were dependent on 7000 volunteer-run relays to route their traffic, of which only approximately 1500 offered themselves as exit nodes (2000 users per relay) and approximately 4400 offered themselves as guard nodes (682 users per relay) [60]. To balance this traffic, relays must throttle their outgoing traffic, which further increases the latency that traffic experience on top of onion routing across the Internet.

Although widely hailed as an infallible anonymity-enhancing technology, even Tor is susceptible to certain types of attacks. First, as a centralized network, access to Tor can be prevented by blocking the IP addresses of the directory authorities [61]. However, the Tor Project encourages users where Tor is not censored to run *bridges* (essentially secret Tor proxies) through which users in censored jurisdictions can connect to the Tor network; censored users can securely correspond with volunteers to obtain the IP address of a Tor bridge so as to avoid publishing a list of all available bridges only for those to be blocked as well [62]. Only China has successfully blocked all Tor connections outright, including the use of Tor via bridges, by analyzing traffic patterns and blocking traffic appearing to be Tor-related [63]. To combat such fingerprint-based blocking, users can install *pluggable transports*, which reshape Tor traffic to evade such detection [64]. Second, like VPNs, Tor has also been demonstrated to be susceptible to fingerprinting attacks in laboratory settings using machine learning techniques [51, 65]. Wang and Goldberg outline potential challenges that adversaries could encounter trying to apply attacks developed under laboratory conditions to real-world traffic, such as noise from concurrent traffic streams (i.e., downloading a file while browsing web pages) and identifying when one request ends and another begins, and they offer strategies to improve existing attacks models in light of these challenges [65]. Despite these improvements, noise and request separation pose significant barriers to an adversary.

Furthermore, Tor only supports TCP traffic. Considering that some smart devices may need to send data over UDP to their cloud-based service providers, routing smart-device traffic through Tor could break functionality if outgoing UDP traffic is blocked, and routing UDP traffic normally could leave users vulnerable to profiling [66].

Aranea

Bahalul Haque et al. propose Aranea, a portable Wi-Fi router that redirects traffic from connected devices into the mainstream Tor network [67]. They motivate the need for their work by explaining how, without Aranea, users with multiple devices would need to configure Tor on each of them to benefit from increased privacy, but they do not elaborate on the types of devices they seek to support. While the authors note increased latency and decreased bandwidth while using Aranea, they do not elaborate on their testing conditions. Furthermore, they do not explain how to ensure compatibility with devices dependent on UDP traffic for proper functionality.

TorBox

Similar to Aranea, TorBox is an open-source project offering privacy-enhancing router firmware to be installed on a Raspberry Pi [68]. TorBox routes all TCP traffic originating from connected devices into the mainstream Tor network, but offers no solutions for devices dependent on UDP traffic to ensure functionality.

Tails

In the work most closely related to ours, Hoang and Pishva present The Amnesic Incognito Live System (Tails), an operating system for a personal computer also acting as a wireless router that automatically redirects all browser traffic and traffic from connected devices to the mainstream Tor network [69].[5] Because Tor is only compatible with TCP, while network traffic originating from smart devices may be UDP, the authors encapsulated UDP packets in TCP packets before routing then into Tor. Then, the Tor exit node (supposedly) decapsulated the UDP packets and forwarded them to their intended destination. This being said, it is unclear how Hoang and Pishva achieved this functionality seeing that they used the mainstream Tor network, which does not natively support UDP-over-TCP, as opposed to a private Tor network whose exit nodes they could instruct to perform this decapsulation. The authors tested their solution with a smart television and a Voice-over-IP (VoIP) system, and found that the television's main features (streaming video and playing games) were mostly unencumbered (albeit with slower buffering speeds than without Tails but no interruptions once playback began), but that the short time-to-live (TTL) of the VoIP UDP packets crashed the call once it had been placed.

Tails demonstrates the plausibility of using Tor to improve privacy against IoT-traffic profiling. Despite having to make three intermediary hops across the Tor network, video streaming – a highly latency-sensitive smart-device experience – was not markedly hindered by the solution.

Multihoming

Another multi-hop routing implementation is *multihoming*, or connecting the home to the Internet via multiple networks (e.g., using multiple ISPs, or connecting devices simultaneously to Wi-Fi and cellular networks) [71]. This way, when a router receives a packet destined for an external endpoint, it has the choice of two or more networks through which to route it, with only the one ultimately selected able to analyze it.

As one example, Henri et al. proposed HyWF, a non-deterministic scheduler for multihoming that improves privacy compared to a deterministic scheduler and does not incur costs in terms of overhead bandwidth. Their approach first assumes that users to have access to two different ISPs (whether wired or cellular). They then route packets using multipath TCP (MPTCP) over the Tor network via an MPTCP-enabled Tor bridge. Because each ISP would only see a subset of the user's traffic, it would be harder for them to reconstruct the

[5] Hoang and Pishva's Tails is not to be confused with the Tails project, which offers a full-fledged operating system designed to safeguard user privacy. The Tails project offers no support for connecting other devices over a wireless network [70].

entirety of a user's activity; doing so would require both ISPs to collaborate by realigning the traffic data they respectively collected [71].

While an effective solution, we find the requirement that a user have multiple ISPs to be unrealistic for the average consumer, as this would require more maintenance and incur higher monthly costs for their Internet access. Furthermore, multihoming is unusable for users who cannot choose their ISPs, such as those living in an apartment building where Internet is provided to all units by the same ISP [27].

2.3.5 Lessons from existing solutions

While the multitude of solutions presented in the preceding sections each offer promising approaches to the issue of mass IoT-user profiling by ISP-like adversaries, all fall short of providing an ideal solution for our model of the average consumer.

While blocking all outgoing traffic could significantly improve privacy from ISP-like adversaries, doing so would hinder device functionality. Implementing edge computing, while effective, would require a significant coordinated effort among device vendors and is unlikely to come to fruition in the near future. Commercial VPN providers have proven to be unreliable, and private VPNs require technical expertise to set up and maintain beyond the abilities of the average consumer. Most traffic-shaping strategies depend on additional infrastructure like VPNs, shifting the burden of trust from the ISP-like adversary to the entity hosting that infrastructure, and may still ultimately be susceptible to timing analysis. The multihoming approach's dependence on two or more ISPs imposes high financial costs on consumers, and may even be impossible for users that are unable to control their own ISPs.

Of all considered options, private-dVPN- and Tor-based solutions appear to be the most promising so far: they do not prevent outgoing traffic, they require no coordination between device vendors to implement, most required infrastructure is already in place and requires little-to-no maintenance on the user's part, they alleviate rather than shifting the burden of trust, and they do not require any changes to the user's ISP. However, current privacy-preserving dVPN implementations like VPN-Zero depend on experimental technical standards, and increased latencies incurred by Tor may hinder users' smart-home experiences [8, 72, 73]. An improved private-dVPN- or Tor-based solution would have to avoid hindering latency-sensitive smart-home experiences, support widely implemented technical standards, and be simple enough to set up such that even the average consumer with limited technical prowess could enjoy increased privacy with little maintenance overhead.

Chapter 3

TorSH Overview

In this section, we present The Onion Router for Smart Homes (TorSH), a collaborative, private Tor network designed to defend smart-device traffic from profiling by ISP-like adversaries without hindering smart-home experiences.

3.1 Network architecture

In light of the lessons from our survey of existing solutions (Section 2.3.5) we aimed to design a privacy-enhancing network, akin to dVPN or Tor, that would obfuscate smart-home traffic.

While Tails stood out as a promising solution, the privacy gained by routing all of a home's traffic through the Tor network may not be worth the hindrance in smart-home experiences (e.g., noticeably higher web-page load times). Although a single-hop routing solution like VPN-Zero, which would incur lower latency than onion routing, would be ideal, we could not conceive of a mechanism to defend against profiling by malicious or honest-but-curious intermediaries without the use of scarcely implemented standards like TLS v1.3 with Encrypted Client Hello [11].

In contemplating how to improve privacy without hindering smart-home experiences, we returned to the "single-purpose IoT nature" of smart devices discussed in Section 2.2: if ISP-like adversaries can significantly enrich user profiles by analyzing their smart-device traffic, perhaps users could improve their privacy by only protecting smart-device traffic from analysis while leaving other, perhaps less-granular traffic, such as that originating from web browsers and smartphones, susceptible to analysis. While some users may be uncomfortable with leaving even only some traffic vulnerable according to their privacy preferences, others may find such a compromise quite palatable. With this in mind, we envisioned a router application that would selectively route smart-device traffic, such as that pertaining to smart mattresses or smart cameras, through our privacy-preserving network while leaving other traffic unencumbered. This idea gives rise to the first design challenge:

Challenge 1. *How will network participants agree on which traffic pertains to smart devices and should therefore be routed into our network?*

Inspired by our collaboration with the SPLICE Project,[1] which is currently articulating and prototyping the security and privacy features of the ideal smart-home router, we envisioned a network composed of such routers working together to improve privacy. This vision was further motivated by Thomasset's decentralized, collaborative smart-device intrusion detection system that runs on the home's router [21, 22]. Drawing inspiration from his approach, we conceived of an application that a user could install on their router to benefit from this collaborative feature. A second design challenge thus presents itself:

Challenge 2. *How will network participants discover each other?*

While contemplating solutions to Challenges 1 and 2, we identified decentralized and centralized decision-making as two possible approaches.

Decentralized model

A decentralized network would (by definition) alleviate any dependence on a central authority to help coordinate the discovery of participants; peer discovery in peer-to-peer networks is well studied, and Challenge 2 is thus solved [74].

However, a decentralized architecture would leave Challenge 1 (endpoint consensus) unresolved. Thomasset used a blockchain-based approach whereby network participants possessing a particular smart device contribute to a chain documenting the endpoints with which that particular device communicates [21, 22]. While the endpoints contacted by that device may evolve over time (e.g., with firmware updates), the majority or participants will reject rogue endpoint contributions that significantly deviate from the established consensus (i.e., contributions from a malicious participant). If a user's device begins communicating with different endpoints than those logged in the blockchain, that user will know their device has been compromised. While a promising solution in the context of intrusion detection, blockchain-based endpoint consensus would fail in the context of collaborative smart-device traffic routing because, unlike in Thomasset's threat model, not all smart homes will possess the smart devices whose traffic is being proposed in order to verify for themselves the validity of the proposition.

To illustrate, if a malicious participant in our network claims that `illegalwebsite.com` is a smart-device-related server, other participants cannot validate for themselves whether that is indeed the case because we do not assume that all participants possess the same devices. If we assume that traffic from one home will only be routed by participants also possessing that same device so they can verify the relevance of the traffic, then the ISP-like adversary, aware of this fact, could infer the types of smart devices each smart home possesses. For homes possessing a smart medical device associated with a particular medical issue, for example, this could lead to a particularly serious privacy violation, although admittedly not worse than if such a network did not exist at all.

The decentralized model further leaves the network prone to known vulnerabilities such as *Sybil attacks*, whereby a single malicious user could spawn large numbers of virtualized participants in order to gain influence and route irrelevant traffic into the network [75]. While there are mechanisms to increase the cost of such attacks, they too come with their

[1] SPLICE is an acronym for **S**ecurity and **P**rivacy in the **L**ifecycle of **I**oT for **C**onsumer **E**nvironments. The project's website can be found at `splice-project.org`.

own disadvantages. For example, blockchain, which facilitates decentralized decision making and trust building, consumes significant amounts of energy [75]. Proof-of-unique-identity mechanisms to prevent Sybil attacks would still ultimately require a centralized authority to validate identities.

Centralized model

Centralizing the network would provide simple solutions to Challenges 1 and 2: a trusted central authority would dictate (possibly with participant feedback) which traffic is allowed to be routed into the network and would help participants discover each other. Multiple central authorities could collaborate in fulfilling these responsibilities. However, this raises yet another challenge:

> **Challenge 3.** *Who could act as a trusted central authority to a centralized, privacy-preserving network?*

Tor is the most prominent example of a centralized, privacy-preserving network. Ten long-time Tor Project contributors, who have demonstrated their trustworthiness through their dedication, each host one of the network's directory authorities.

Although our central authorities would not have access to sensitive participant data, they would be taking on a long-term responsibility to host and maintain an up-to-date list of smart-device endpoints as well as a participant-discovery service.

Our choice for TorSH

We ultimately decided to pursue a centralized model for our privacy-preserving network. While a decentralized architecture would offer lower barriers to adoption, alleviating dependence on a centralized authority to maintain a list of valid smart-device-related servers, we found there to be too many peripheral challenges to designing and deploying a robust smart-home-privacy-enhancing peer-to-peer network, many of which could constitute theses of their own.

With a centralized architecture, the question becomes whom to entrust as a central authority, particularly considering there seems to be no single authority on matters concerning consumer smart devices. Yet, seeing that multiple organizations could collaboratively assume the responsibilities of a central authority, it is not necessary to choose just one. As such, we believe organizations and alliances who have already dedicated themselves to the consumer IoT space, such as Consumer Reports, the Connectivity Standards Alliance (behind the Matter project), the ioXt Alliance, and universities would be prime candidates to host such a service.

We further evaluate our choice of a centralized architecture in Section 5.2.5.

3.2 Design criteria

Having decided on a centralized architecture, we identified the following set of criteria for our network and for its routing software, respectively, that would ensure an effective collab-

orative solution to the issue of user profiling via smart-device traffic analysis. We label each criterion with a unique *design criterion* (DC) label for ease of reference when we evaluate each in Chapters 4 and 5.

Network

We envisioned a network of smart-home routers connected to each other across the WAN that would:

- **DC1: offer strong privacy guarantees** to avoid leaving users vulnerable to other types of adversaries;
- **DC2: maintain low-latency smart-device experiences** to avoid undermining users' desires to enjoy the conveniences afforded by their smart devices;
- **DC3: grow sustainably** to avoid performance bottlenecks as the number of participants increases;
- **DC4: not be geographically restricted** to minimize barriers to adoption (unlike EPIC).

Router application

The smart-home routers that make up this network would need to:

- **DC5: leave non-smart-device traffic unencumbered** to ensure other latency-sensitive smart-home experiences, such as web browsing, are still enjoyable;
- **DC6: run on widely available hardware** to minimize barriers to adoption;
- **DC7: support both TCP and UDP** to maximize compatibility with smart devices, which may depend on both protocols [66];
- **DC8: support legacy standards** (e.g., TLS v1.2), as smart-device vendors may eventually stop updating their products;
- **DC9: prevent abuse** to protect users from malicious participants seeking to route irrelevant or unethical traffic through the collaborative network.

3.3 Implementation details

In this section, we summarize the main features and implementation details of our privacy-preserving network.

3.3.1 Single-hop vs. multi-hop routing

As discussed in Section 3.1, we could not conceive of a privacy-assuring routing mechanism requiring fewer than three hops without assuming the availability of widely unimplemented standards. As such, we envisioned a network that would route smart-device traffic through multiple intermediary nodes before being placed, just as under onion routing or EPIC.

We ultimately use Tor as the underlying routing tool for our centralized network, because Tor has proven to be stable, scalable, and customizable for our needs. However, instead

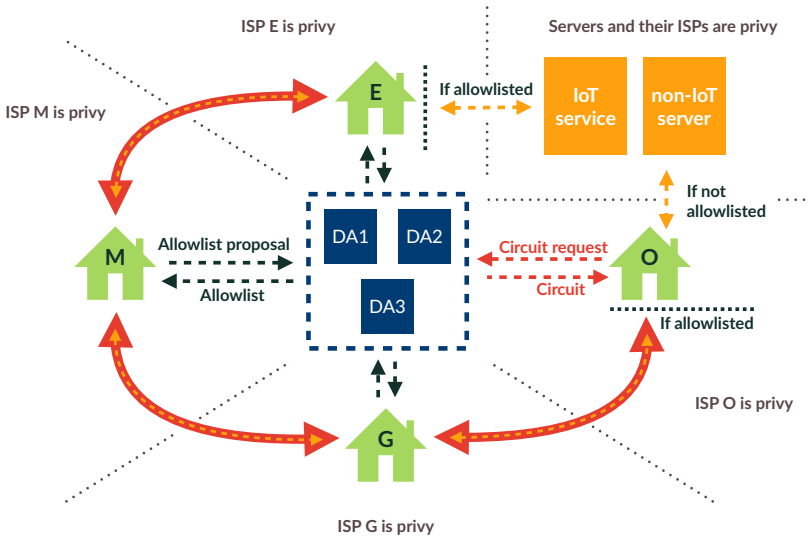


Figure 3.1: TorSH network architecture overview. Participant **O** is the **o**riginator router. Participant **G** is the **g**uard router. Participant **M** is the **m**iddle router. Participant **E** is the **e**xit router. Servers **DAx** act as **d**irectory **a**uthorities and dictate the allowlist to participants. Dashed arrows indicate traffic flows that are not protected by TorSH (but may be otherwise encrypted), whereas solid arrows indicate traffic that is protected by TorSH.

of routing traffic into the mainstream Tor network, we envision that users of our privacy-preserving network will participate in a private Tor network solely composed of other homes' routers equipped with our software, which will offer additional features more apt for smart-device traffic compared to mainstream Tor. This is the network we call *TorSH*.

3.3.2 Features overview

Figure 3.1 illustrates TorSH's routing mechanism. The principal components that distinguish TorSH from typical Tor include the **allowlist**, **selective packet routing**, **UDP-over-TCP**, and an **allowlist-proposal** mechanism. We discuss each of these features in this section.

Henceforth, we use the term *originator router* to designate a router that sends a packet into TorSH (analogous to a Tor client). In contrast, we use the term *exit router* to designate a router that forwards packets to a cloud-based service on behalf of an originator (analogous to a Tor exit node). We also distinguish between the TorSH network and the `torsh` application running on TorSH-equipped routers, as we do the Tor network and the `tor` application.

Allowlist

TorSH's *allowlist* contains the FQDNs that are known to be contacted by consumer smart devices. Each router in the network has access to the same allowlist. The allowlist serves two purposes: (1) to tell help originator routers decide which packets to route into TorSH

and which to route directly to their intended destination, and (2) to help exit routers ensure that the traffic they are routing is related to smart devices, as opposed to unethical or malicious traffic that could get them in legal trouble.

The allowlist is collaboratively maintained by the central authorities, who may push periodic updates to the network participants no less often than some universal update interval t_u. These updates are transmitted to participants over HTTPS as an *allowlist database*, being an allowlist encoded in a convenient data-exchange format such as JavaScript Object Notation (JSON).

Each FQDN in an allowlist database also contains an *effective date*. To avoid some participants receiving new FQDNs via allowlist updates before others due to misaligned update intervals, the effective date of each new entry in an allowlist database is set to the time of the central authorities' decision to amend the allowlist plus at least one update interval t_u. This way, all participants begin to accept traffic for that FQDN at the same time. Note that we assume participants who fail to download an updated allowlist database within the interval t_u are offline and therefore unable to participate in traffic routing altogether; they will receive the latest allowlist database once they return online.

Selective packet routing

TorSH protects smart-device traffic from profiling without encumbering non-smart-device traffic by selectively redirecting smart-device-related packets into the TorSH network. To do so, `torsh` uses the native `iptables` Linux utility to insert packet-redirection rules at specific points (denoted by *tables* and *chains*) in the kernel-level packet-routing pipeline that redirect outgoing packets into the TorSH network if they are destined for IP addresses related to allowlisted servers and that block outgoing packets from the local `tor` process on exit routers if they are not [76]. Figure 3.2 illustrates the locations of these tables and chains relative to a packet's flow through the kernel. TorSH also depends on two `iptables` extensions: the `libnetfilter_queue` library that enables packet-redirection logic to be handled by a custom process (in our case, `torsh`) instead of by `iptables`, as well as `ipset`, which allows the same `iptables` rule to apply to multiple IP addresses that are stored in a hash set to ensure $O(1)$ lookup times (without `ipset`, `torsh` would have to insert one `iptables` rule per IP address, which would significantly slow all packet routing) [77, 78].

By virtue of the respective particularities of these protocols, `torsh` handles the redirection of DNS, TCP, and non-DNS UDP packets differently.

The `torsh` process inserts a rules for an originator router's outgoing unencrypted DNS requests (identified as UDP packets destined for port 53) in the `PREROUTING` chain of `iptables`'s `nat` table. This rule brings the packets into userspace, where `torsh` extracts the FQDN the request seeks to resolve (because the DNS packets are unencrypted, this is simple to do). If the FQDN is in the allowlist – a rapid verification by virtue of the fact that `torsh` maintains a hash set of allowlisted FQDNs to ensure $O(1)$ lookup times – the packet is redirected to the local `tor` process's DNS-resolution port. Otherwise, the packet is routed normally, and the the user's configured DNS resolver handles the request.

While our allowlists and our DNS-packet-routing logic are based on FQDNs, Linux kernel-level routing of TCP and non-DNS UDP packets requires rules based on the endpoints being contacted, which consider combinations of IP addresses, port numbers, and protocols

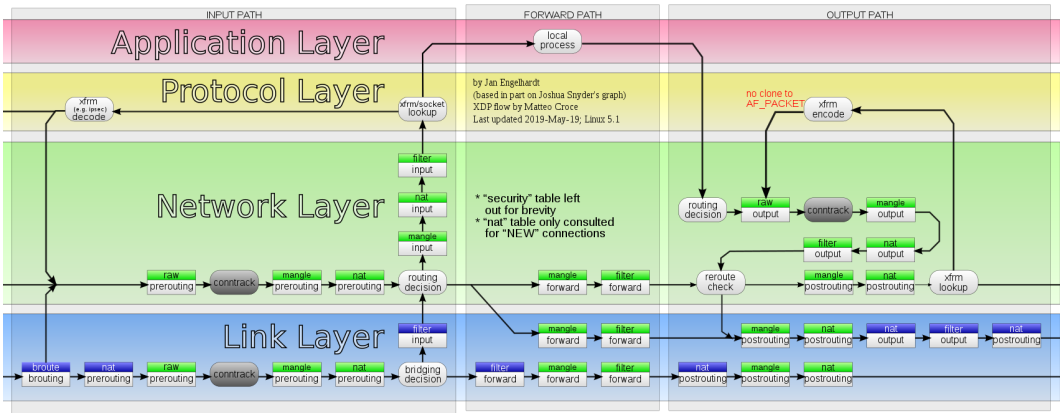


Figure 3.2: Packet flow through the Linux kernel. Figure adapted from Engelhardt [79].

(Definition 2); the FQDN of a packet's destination is not among its associated metadata. As discussed in Section 2.1.2, determining all of the IP addresses associated with a particular FQDN is practically impossible under DNS: these IP addresses change regularly and no central authority maintains a list of all the IP addresses associated with a particular FQDN worldwide at a given point in time. This poses a novel challenge for TorSH:

> **Challenge 4.** *How can TorSH participants agree on the set of IP addresses associated with a particular FQDN, when their respective DNS servers are likely geographically disperse and therefore likely to return different IP addresses for the same FQDN?*

Our solution to this challenge is to decentralize the FQDN–IP-address association process by assuming that the exit router of the circuit used to resolve an originator router's DNS request is the same as that used to open the TCP connection with the associated IP address. We present this solution in the following list and explain the corresponding `iptables` rules we use to achieve each step in the respective sub-bullets.

1. A smart device attempts to contact a cloud-based service provider by first placing a DNS request for an allowlisted FQDN. The originator router redirects this request into the TorSH network to be resolved by an exit router.

   - This rule is that aforementioned, which is inserted in the `PREROUTING` chain of the `nat` table.

2. The exit router indeed confirms that the DNS request is for an allowlisted FQDN. It is therefore to be trusted and the router places the request on behalf of the originator. If the exit router does not recognize the FQDNfrom the allowlist, it refuses to place the DNS request.

   - This rule, inserted in the `OUTPUT` chain of the `mangle` table, redirects DNS requests originated by the `tor` process (such as those an exit router places on behalf of an originator router) into userspace for processing by `torsh`. If `torsh` finds the requested FQDN in the allowlist hash set, it allows the request to be placed and adds the FQDN to a hash map relating IP addresses to FQDNs (the entry

for this FQDN remains empty until the router receives the corresponding DNS response).

3. The exit router intercepts the response from its DNS server, which returns some set $IP := \{ip_0, ip_1, \ldots, ip_n\}$ of IP addresses associated with that FQDN. The exit node saves these IP addresses locally. The DNS response is then forwarded back to the originator router via the TorSH network.

    - This rule, inserted in the `PREROUTING` chain of the `mangle` table, intercepts *all* incoming DNS responses and redirects them into userspace for processing by `torsh`. If `torsh` finds the response's FQDN in the hash map relating FQDNs to IP addresses (which would only happen if the FQDN is in the allowlist), it inserts the returned IP addresses into the hash map under the key of that FQDN. The `torsh` process also appends the returned IP addresses to the local *torsh-allowlist* `ipset` collection. The packet is then returned to kernelspace to be routed to its destination (whether a device on the LAN or to the `tor` process, to be returned to the originator router).

4. The originator router receives the DNS response, saves the set IP locally, and forwards the response to the smart device that originated it.

    - This rule, inserted in the `OUTPUT` chain of the `mangle` table, redirects DNS responses returned from an exit router into userspace for processing by `torsh`. As with the exit router, the originator's `torsh` process also appends the returned IP addresses to the local *torsh-allowlist* `ipset` collection. The packet is then returned to kernelspace to be routed to the smart device that originated it.

5. The originating smart device randomly selects some $ip_k : 0 \leq k \leq n$ with which to initiate the TCP connection (or to which it will send a UDP packet, the mechanics of which we outline in the next subsection). The router will henceforth redirect packets destined for ip_k into the TorSH network.

    - This rule, inserted in the `PREROUTING` chain of the `nat` table, redirects TCP connections destined for IP addresses associated with allowlisted FQDNs into the TorSH network. It does so without redirecting the packet to userspace for processing, but rather by comparing the packet's destination IP address with those in the *torsh-allowlist* `ipset` collection. Packets destined for IP addresses that are not in *torsh-allowlist* are routed normally.

6. The exit router receives a packet destined for $ip_k \in IP$ and therefore concludes that the traffic is destined for an allowlisted cloud-based service provider. It accepts to open the connection on behalf of the originator router. If the exit router receives a packet destined for some $ip_j \notin IP$, it can refuse to place the traffic on behalf of the originator node.

    - This rule, inserted in the `OUTPUT` chain of the `mangle` table, ensures that TCP connections originating from the exit router's `tor` process are only instantiated with servers whose IP addresses are included in the *torsh-allowlist* `ipset` collection. Attempted connections on behalf of originator nodes that are not with IP addresses in *torsh-allowlist* are blocked, thereby preventing exit routers from forwarding non-smart-device-related traffic on behalf of malicious originator routers.

And thus, the issue of FQDN–IP-address consensus is solved. This solution is also extensible to an exit node opening connections on behalf of several originator routers simultaneously.

The assumption that an exit node for a TCP connection be the same as that which resolved the associated DNS request is practical: `tor` allows specifying one or more exit routers from within its configuration files, and by periodically rotating between TorSH participants as a given router's exit router, anonymity can still be preserved. This solution also assumes that DNS requests are unencrypted. We believe this assumption to be practical considering that unencrypted DNS resolution is still the norm among smart devices [23–25].

To ensure that routers' Random Access Memory (RAM) is not exhausted by an ever-growing list of valid IP addresses, we assign each cached IP address a TTL that is longer than the typical gap between smart devices' DNS requests for the same domain name (e.g., 24 hours). When an IP address expires, it is erased from memory, but a fresh DNS request for the same FQDN could reset its TTL. In case two FQDNs point to the same IP address, exit routers count the number of references to each IP address per FQDN so that IP address will only be deleted upon TTL expiry if no other originator nodes are still counting on it.

We explain how we reroute UDP packets in the following subsection.

UDP-over-TCP

A drawback of Tor for smart-device traffic is its restriction to TCP packets (other than for DNS resolution, which is a UDP-based protocol). However, some smart devices depend on UDP to offer full functionality [66]. To support these devices, we follow Hoang and Pishva in encapsulating UDP packets in TCP before routing them through the TorSH network [69]. Because TCP is a connection-based protocol whereas UDP is connectionless, such encapsulation entails having originator routers open a new TCP connection with a special IP address that is known to be recognized by TorSH exit routers as the UDP-over-TCP address. Upon receiving TCP packets destined for this address, exit routers will not actually route them to the special address but rather decapsulate them and send the UDP packet to its intended endpoint. This system is admittedly quite inefficient, as a TCP connection must be opened, including a full handshake, before the UDP packet can be sent. We discuss strategies to improve the implementation of this feature in Section 5.2.6.

As with the redirection of DNS and TCP packets, we implement UDP-packet redirection using a series of `iptables` hooks:

- We insert a rule into the `PREROUTING` chain of the `nat` table that intercepts outgoing UDP packets destined for an IP address included in the *torsh-allowlist* `ipset` collection and redirects them into userspace for processing by the `torsh` process. The `torsh` process encapsulates the UDP packet in a TCP packet that includes a custom packet header specifying the packet's intended recipient and sends it to the "dummy" UDP-over-TCP IP address.
- We insert another rule into the `OUTPUT` chain of the `mangle` table that intercepts non-`SYN`-flagged TCP packets (which contain the encapsulated UDP packets) destined for the dummy UDP-over-TCP IP address and redirects them into userspace for processing by the `torsh` process. The `torsh` process in turn decapsulates the TCP packet, revealing the original UDP packet and custom header containing the IP address of

the intended recipient, which would also be included in the exit router's local *torsh-allowlist* `ipset` collection (assuming this IP address were resolved by the FQDN–IP-address consensus mechanism detailed in the previous subsection). The exit router then forwards the UDP packet to its intended destination on behalf of the originator router.

We note that our implementation of UDP-over-TCP does not support forwarding servers' UDP responses to a smart device's UDP traffic back to the device that originated it. While many UDP applications may not depend on the reliable delivery of UDP responses from servers (with UDP being a connectionless protocol), one notable exception is the Network Time Protocol (NTP), which synchronizes clocks over networks using UDP. To avoid breaking device functionalities by preventing smart devices from synchronizing their clocks over NTP, we do not route NTP traffic into the TorSH network.[2] We discuss the ramifications of this decision in Section 5.2.6.

Allowlist proposal

The responsibility of maintaining an up-to-date list of the servers contacted by (ideally) all commercially available smart devices is daunting: the allowlist would require frequent updates as these devices receive firmware updates and as new devices come to market.

To aid in this process, we implemented an opt-in feature whereby users can consent to having the list of FQDNs contacted by *some* of their devices anonymously shared with the central authorities. As Thomasset highlighted, smart devices typically contact fewer endpoints than devices like smartphones or personal computers [21, 22]. Figure 3.3 illustrates the relative number endpoints contacted by different types of devices potentially found in a smart home. TorSH allows users to specify the maximum "complexity" of the device whose FQDNs they are willing to share with the central authority, using the number of servers contacted as a proxy for complexity. While this data could reveal the presence of a device in the home, it could not reveal usage patterns because timing data is not included. We anticipate some users will not mind sharing this data, provided only data from smart devices like smart refrigerators, smart switches, and smart cameras, for example, are shared, and not from personal computers or web browsers. If many users independently begin reporting a new FQDN for a particular device, say following a software update, the central authorities could add that FQDN to the allowlist.

In practice, we implemented this feature by keeping track of the set of FQDNs contacted by each device on a router's network in a local hash map, with the hardware address of the device used as the key to the map. As soon as a device contacts more than the user's specified maximum number of FQDNs, profiling for that device ceases and its FQDNs are no longer shared with the central authorities during periodic updates. When participants share their device profiles with the central authorities, the FQDNs are grouped by originating device, but no identifying information alluding to the nature of that device is included.

[2] NTP traffic is easily identifiable because it uses port 123.

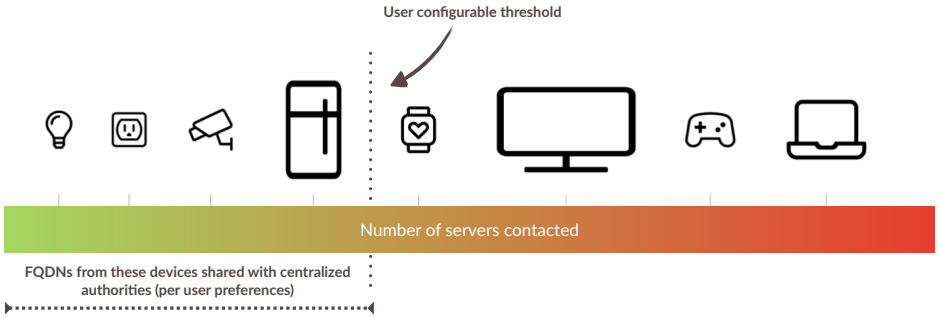


Figure 3.3: The number of servers contacted by different types of devices potentially found in a smart home. Figure adapted from Figure 3.2 of Thomasset [22].

Distribution and configuration

We compiled `torsh` as an OpenWrt package to ensure compatibility with a large number of common router models, and we use the existing `tor` OpenWrt package as a dependency. Upon installation, `torsh` configures the local `tor` installation to use TorSH's private Tor network instead of the mainstream Tor network for traffic routing. The `torsh` service is added to the list of start-up services. When the binary is run, `torsh` automatically adds any required `iptables` rules, which it also clears when `torsh` is shut down.

Source code and development

We wrote `torsh` in Rust, a compiled language designed with performance and security in mind [80]. The `torsh` *crate* (Rust package) containing the routing application, the test server, and their common library resides in one repository, and the tools used to build and test TorSH locally resides in another. We adapted the latter repository and testing tools from those developed by Heijligers for testing his implementation of Tor over QUIC in a cluster of containerized relays [81]. Our source code is available upon request.

Chapter 4

Methodology

We planned to evaluate TorSH according to the design criteria we outlined in Section 3.2. We summarize these criteria and whether we evaluated them empirically or qualitatively in Table 4.1.

Table 4.1: Summary of design criteria outlined in Section 3.2 and evaluation method, where "Emp." designates an empirical evaluation and "Qual." designates a qualitative evaluation. Note that we evaluated certain criteria both empirically and qualitatively.

		Design criterion	Emp.	Qual.
Network	**DC1**	offers strong privacy guarantees		•
	DC2	maintains low-latency smart-device experiences	•	•
	DC3	grows sustainably		•
	DC4	not be geographically restricted		•
Router	**DC5**	leaves non-smart-device traffic unencumbered	•	
	DC6	runs on widely available hardware	•	•
	DC7	supports both TCP and UDP	•	
	DC8	supports legacy standards		•
	DC9	prevents abuse		•

The remainder of this chapter presents the experiments we designed to evaluate TorSH empirically. We defer the evaluation of TorSH's qualitatively measurable design criteria to Chapter 5.

4.1 Experimental design

We designed a set of experiments to evaluate TorSH's empirically measurable design criteria (DC2, DC5, DC6, and DC7).

Throughout the rest of this thesis, we use the terms *normal routing* or *regular routing* to refer to packet routing in the absence of TorSH or Tor. When we refer to routing *under TorSH*, we mean routing with TorSH enabled on the router such that smart-device traffic

is routed into the TorSH network while non-smart-device traffic is routed normally. When we refer to routing *under Tor*, we mean routing all traffic into the mainstream Tor network. Collectively, we refer to normal routing, routing under TorSH, and routing under Tor as the three *routing strategies*.

4.1.1 Evaluating latency of smart-device experiences (DC2)

To evaluate TorSH's impact on the latency of smart-device experiences, we designed a series of experiments measuring the delay between common user–smart-device interactions and the expected response (henceforth, *interaction–response delays* or *interaction–response latencies*) when the device's traffic was routed under each routing strategy. Because the consumer smart-device landscape is heterogeneous, we aimed to perform these experiments using devices representing the spectrum of those that may be present in a smart home in terms of the nature of each device's primary traffic. Table 4.2 summarizes the smart devices we used to perform these experiments.

Table 4.2: Smart devices used to evaluate TorSH's impact on the latency of smart-device experiences.

Type of device	**Anticipated nature of primary traffic**	**Device name**	**Model number**
Smart sensor	Periodic TCP	Govee Thermo Hygrometer	`H5179`
Smart lamp	Simple TCP exchange	Govee Aura Table Lamp	`H6052`
Smart assistant	TCP stream	Amazon Echo Dot (Alexa)	`B7W64E`
Smart camera	TCP or UDP stream	Blink Mini	`BCM00300U`

We summarize the device interactions we conducted during these experiments in Table 4.3.

Table 4.3: Principal user–smart-device interactions performed to measure interaction–response latencies.

Type of device	**Interaction**
Smart sensor	*None* (periodic reporting)
Smart lamp	Turning lamp on from accompanying smartphone app
	Turning lamp off from accompanying smartphone app
	Changing lamp color from accompanying smartphone app
Smart assistant	Prompting (with wake word) and asking question
Smart camera	Watching live stream of feed from accompanying smartphone app

During each device interaction, we used packet capture software to gain insight into the type of traffic produced by each device and to validate that packets were being routed as expected. Furthermore, we began each experiment by unplugging and re-plugging the device being studied to ensure consistency across each trial.

To measure interaction–response delays, we used several simple but effective techniques depending on the device. To measure the delay between triggering a change in the lamp's state from its accompanying smartphone app and the change becoming visually effective, we filmed each interaction with both the testing smartphone and the lamp in frame and

counted the number of frames required for the change to become effective. To measure delays during smart-assistant interactions, we recorded the audio of the interaction and measured the delay between when we finished asking Alexa our question and the beginning of her response. To measure the delay between requesting a live stream of the smart camera's feed from its accompanying smartphone app and the live feed starting, we used the screen recording feature of the testing smartphone.[1]

The smart sensor differs from the other devices in that it offers no manual interactions that are latency-sensitive. Instead, it uploads temperature and humidity data to the cloud periodically (configurable to either 10-minute, 30-minute, or 1-hour intervals; we configured it to report every 10 minutes). Then, the user can view this data from the accompanying app on their smartphone. As such, we did not evaluate TorSH's impact on the latency of this device but rather on whether the device continued to successfully upload its data under each routing strategy.

Thus, to compare the interaction–response delays for the smart lamp, smart assistant, and smart camera under each routing strategy, we performed the following experimental protocol:

1. we unplugged the device being tested;
2. we began packet capture;
3. we began the latency-capturing tool (i.e., camera, audio recording, screen recording);
4. we plugged in the device being tested;
5. we performed device-specific interactions (detailed below);
6. we stopped the latency-capturing tool;
7. we stopped packet capture.

We repeated this protocol three times per routing strategy.

Device-specific interactions

To measure the delay in the latency-sensitive interactions outlined in Table 4.3, we performed the following protocols for the smart lamp, smart assistant, and smart camera.

- **Smart lamp**
  We successively performed each of the following interactions from the lamp's accompanying smartphone app, with a five second delay between each:

  1. we turned on the lamp;
  2. we changed its color to orange;
  3. we changed its color to yellow;
  4. we changed its color to green;
  5. we changed its color to blue;
  6. we changed its color to cyan;
  7. we changed its color to purple;
  8. we changed its color to white;
  9. we changed its color to red;
  10. we turned the lamp off.

- **Smart assistant**
  We successively asked the smart assistant each of the following questions, waiting five seconds after the end of the assistant's response before asking the next question:

[1] The Blink Mini does not upload a constant video stream to the cloud. It only live streams upon motion detection (if enabled) or if the user manually requests the feed from the accompanying smartphone app.

1. "Alexa, what is the weather today?" (henceforth *Q1*);
2. "Alexa, what is the weather tomorrow?" (henceforth *Q2*);
3. "Alexa, where is Dartmouth College?" (henceforth *Q3*).

We chose these questions because they are location-specific,[2] which would reveal whether traffic appearing to originate from a different IP address than that of the actual user would affect the assistant's response. Furthermore the answers to the weather-related questions were relatively long, which would provide insight into whether increased latency from onion routing would hinder playback quality.

- **Smart camera**
  We requested the live stream of the smart camera's feed by successively performing the following actions:
  1. we opened the accompanying smartphone app;
  2. we clicked on the button that requests the live camera feed;
  3. we waited for the live stream to load and begin playback;
  4. we exited the accompanying smartphone app and quit its background process (to ensure a fresh launch during the next trial).

4.1.2 Evaluating impact on non-smart-device traffic (DC5)

To ensure TorSH does not encumber a home's non-smart-device experiences (e.g., web-page load times), we conducted an experiment comparing web-page load times across the three routing strategies.

To this end, we wrote a Python script that repeatedly launches a fresh browser session (with a cleared browser cache), sequentially downloads a variety of web pages (summarized in Table 4.4), notes the time to render each according to the World Wide Web Consortium's Navigation Timing Level 2 specification, and clears the client's DNS cache [82]. We executed the script, which we share in Appendix B, from a laptop connected via Wi-Fi to the TorSH-enabled router.

Table 4.4: Web pages used to evaluate TorSH's impact on web page load times.

Web address and description	**Nature of contents**
`https://www.gutenberg.org/files/1342/1342-h/1342-h.htm` Project Gutenberg edition of Jane Austen's *Pride and Prejudice*	Long text No images or ads
`https://www.google.com` Home page of the Google search engine	Small images (icons) No ads
`https://en.wikipedia.org` Home page of Wikipedia, the Free Encyclopedia	Some images No ads
`https://www.bbc.com` Home page of the British Broadcasting Corporation (BBC)	Many images Some ads

We measured the render time of each web page a total of 100 times under each routing strategy.

[2] Q3 is location-specific because Alexa responded with how far away Dartmouth College was from the Echo Dot we used for testing.

4.1.3 Evaluating hardware resource consumption (DC6)

To evaluate the `torsh` process's hardware resource consumption, we designed a "stress test" whereby we measured Central Processing Unit (CPU) and RAM utilization as we placed many DNS requests from a client connected to the router via Wi-Fi in rapid succession. Specifically, we measured the process's CPU and RAM utilization as it resolved the 1000 most visited domain names in the Open PageRank dataset [83]. To ensure reproducibility, we used Google's `8.8.8.8` DNS server to resolve each request.

4.1.4 Evaluating support for both TCP and UDP (DC7)

Tor natively supports routing TCP packets. As such, we will inherently confirm TorSH's ability to support TCP packets while evaluating its impact on smart-device experiences. Evaluating TorSH's implementation of UDP-over-TCP may prove more difficult. It is not immediately obvious whether the devices listed in Table 4.2 depend on non-DNS, non-NTP UDP traffic to offer their full functionality (although we guessed that of all devices, the smart camera would be the most likely to because of the high bandwidth demands of video live streams and tolerance for dropped frames). Furthermore, devices that depend on UDP may have different tolerances for latency depending on their respective applications. For example, while Hoang and Pishva's Tails supposedly supported UDP-over-TCP, the additional incurred latency inhibited proper functionality for VoIP applications, whose packets tend to have shorter TTLs [69]. With this in mind, our evaluation of TorSH's UDP support will depend on the observed traffic from our evaluation of its impact on other smart-device experiences.

4.2 Testing tools

In this section, we document the tools we used to perform our empirical evaluations.

Client

We used a mid-2014 MacBook Pro running macOS `11.6.3` equipped with a 2.6 GHz dual-core Intel Core i5 processor and 8 GB of RAM as the client while evaluating TorSH's impact on web-page render times and hardware resource utilization. The client's web browser was version `101.0.4951.54` of Google Chrome.

Router

The wireless router we used for testing was a Raspberry Pi 4B equipped with a quad-core 1.5 GHz ARM processor and 8 GB of RAM running OpenWrt version `21.02.1` [84, 85]. We configured the router to act as a Wi-Fi access point. We connected it to the WAN via an Ethernet port beyond Dartmouth's firewall so as to best emulate a typical smart-home router's WAN connection. We also disabled the router's DNS cache to ensure all DNS requests were freshly resolved by the ISP's DNS server.

Syrupy

We used commit `4062918` of Jeet Sukumaran's `Syrupy` utility to profile the `torsh` process during our evaluation of TorSH's hardware resource consumption [86].

Smart devices

We summarize the smart devices we used for testing in Table 4.2.

Smartphone and accompanying apps

We used an iPhone 12 running iOS `15.4.1` to interact with the smart devices via their accompanying apps, which we summarize in Table 4.5.

Table 4.5: Accompanying apps to the smart devices listed in Table 4.2.

Smart device	**Accompanying app**	**App version**
Govee Thermo Hygrometer Govee Aura Table Lamp	Govee Home	`5.0.1`
Amazon Echo Dot	Amazon Alexa	`2.2.473272.0`
Blink Mini	Blink Home Monitor	`6.11.0`

We note that the smartphone was not connected to the testing router's Wi-Fi network; rather, it was connected to the WAN via Dartmouth's main Wi-Fi network, from which the Raspberry Pi testing router was isolated. We leave evaluating interaction–response latencies with smartphone traffic also being routed under TorSH to future work.

tcpdump

We captured outgoing traffic using version `4.9.3` of the Linux `tcpdump` utility, a popular command-line packet capture tool [87].

Tor

We use version `0.4.6.10` of the `tor` application for testing.

TorSH network

Because TorSH is not yet widely deployed, we configured a test TorSH network using virtual "routers" hosted on Google Cloud Platform's *Compute Engine*. Each instance ran Ubuntu `20.04` (as OpenWrt is not a supported operating system on the Compute Engine platform) and was configured to have 2 virtual CPU cores (each capable of one simultaneous thread) and 1 GB of RAM, which together represented the most hardware-restrained configuration possible.[3] We instantiated each node in geographically dispersed data centers

[3] At the time of writing, this configuration was known as the `e2-micro` *machine type* in Compute Engine.

(one in Oregon, one in Japan, and one in Finland) so as to subject TorSH-routed packets over realistic distances while evaluating a potentially worldwide network. We configured the Oregon node to act as the test router's exit node.

We similarly set up a directory authority using Compute Engine, which was hosted in a data center in South Carolina.

Wireshark

We analyzed the packets captured with `tcpdump` using version `3.6.1` of Wireshark, a popular packet capture analysis tool [88].

Chapter 5

Evaluation

In this section, we discuss the findings of the experiments presented in Chapter 4 and we evaluate the extent to which TorSH succeeds in implementing the design criteria we outlined in Section 3.2. We also present opportunities for future work.

5.1 Return to design criteria

The following subsections each present our evaluation of one of the design criteria according to the evaluation methodology summarized in Table 4.1.

5.1.1 Privacy guarantees (DC1)

In this subsection, we evaluate the extent to which TorSH succeeds in improving user privacy against ISP-like adversaries.

Privacy afforded by Tor

TorSH benefits from Tor's well-studied privacy benefits, which Landau claims are some of the most effective at obfuscating Internet traffic [8]. While not perfectly infallible, Tor significantly hinders WAN observers' ability to study users' Internet communications. In October 2013, Edward Snowden leaked a then-classified slide show entitled "Tor Stinks" from the year prior in which the NSA admitted even it struggled to analyze Tor users' traffic [89, 90].

At first glance, TorSH's effectiveness lies in its use of three-hop onion routing, which obfuscates the rich categorical metadata associated with smart-device traffic (i.e., source and destination port and IP address, packet size, and underlying protocol, seeing that UDP packets would be encapsulated in TCP). By reshaping packets into cells destined for randomly selected TorSH participants, an ISP-like adversary privy to the originator's traffic would find it difficult to fingerprint traffic using these properties.

Basyoni et al. recently surveyed proposed attacks on Tor, evaluated the feasibility of launching them on the live mainstream Tor network, and identified significant weaknesses with each when deployed under realistic as opposed to laboratory conditions [91]. Furthermore, the attacks developed by Wang and Goldberg (presented in Section 2.3.4) assume a victim whose traffic originated from the Tor browser, where a user typically sequentially browses web pages while perhaps downloading at most a couple files simultaneously in the background. It is unclear how the efficacy of such attacks would be affected if multiple smart devices in a home simultaneously routed their own traffic streams into the Tor network.

Dong et al. recently developed HomeMole, a novel smart-device-traffic fingerprinting attack to be carried out by an ISP-like adversary that can identify the smart devices behind NAT when the traffic is routed through a VPN with relatively high accuracy [66]. Because VPN obfuscates a packet's true destination port and IP address and distorts packet sizes, their attack primarily depends on the relative timing of packets in a sequence to identify the originating device. Furthermore, they developed their attack considering both single-device environments (in which a sole smart device originates all of the router's outgoing traffic) as well as noisy mixed-device environments (in which multiple smart and non-smart devices originate outgoing traffic simultaneously). They found that identifying smart devices based on VPN-encrypted traffic is hindered but not impossible when non-smart-device traffic is simultaneously routed over the same VPN tunnel.

At first glance, the findings of Dong et al. appear to put TorSH's privacy assurances into question, especially considering that TorSH does not simultaneously route non-smart-device traffic into its network but rather clearly distinguishes smart-device traffic. In practice, several factors render that conclusion less obvious. First, VPN and Tor differ in that Tor reshapes all outgoing packets into fixed-size cells whereas VPN merely appends metadata to packets (or might combine two or more outgoing packets together). Tor's approach thus further restricts the metadata with which HomeMole could classify traffic. Second, HomeMole was developed using UDP-based VPN implementations whereas Tor is based on encrypted TCP connections. Dong et al. acknowledge that further work is required to evaluate the effectiveness of their attack using TCP-based VPN implementations. Finally, it is unclear how Tor's overhead traffic (e.g., from circuit construction) would affect the accuracy of the HomeMole attack.

We hypothesize that these sources of doubt notwithstanding, certain types of smart devices may be more susceptible to fingerprinting attacks under TorSH than others. Namely, an ISP-like adversary could identify smart devices with distinct traffic patterns, such as smart sensors that upload data to their cloud-based service providers at consistent intervals (e.g., the Govee Thermo-Hygrometer we used for testing). While they could possibly identify such devices as some type of smart sensor, the exact type of smart sensor could be difficult to determine unless it reports data at peculiar intervals characteristic of a specific model of device (e.g., every 27 minutes as opposed to every 30 or 60 minutes). Furthermore, smart devices that generate highly consistent traffic patterns when interacted with (e.g., a smart refrigerator that consistently leads to two Tor cells being successively originated whenever opened) could be more easily fingerprinted than devices with less consistent patterns. Finally, users who have few smart devices in their homes may be more susceptible to fingerprinting attacks than users with many smart devices that may generate traffic simultaneously, as suggested by the findings of Dong et al.

One possible defense against the profiling of such devices could be the use of Tor pluggable

transports to randomize or otherwise reshape outgoing Tor traffic patterns. Another defense primarily benefiting users with few smart devices could be for TorSH-enabled routers to generate smart-device-traffic-like noise, which exit routers would know to ignore, to confound ISP-like adversaries. Further work is required to evaluate TorSH's susceptibility to fingerprinting attacks, possible defenses against them, and the impact of these defenses on smart-device experiences.

In essence, Tor (and TorSH by extension) achieve anonymity by reducing the traffic any one ISP-like adversary is privy to. For example, although the ISP of an exit router would be able to infer the identity of a circuit's middle router (by virtue of the fact that outgoing TorSH traffic placed by the exit router would necessarily be preceded by incoming Tor cells from the middle router), they would struggle to identify the originator router, whose identity is hidden by the buffer of the middle router. This being said, this buffer can be undermined if both the originator and exit nodes (or possibly the middle node as well) are served by the same ISP: this would empower the ISP, who would be privy to both incoming and outgoing traffic from the circuit, to perform timing attacks against their TorSH-using customers. Because many ISPs serve customers nationwide, this is not an impossible scenario. Fortunately, Tor currently allows users to prefer exit nodes from a configurable list of countries, which Tor can deliver on by looking up IP addresses using *GeoIP* searches. Because TorSH requires that routers periodically rotate between explicitly configured exit nodes (to ensure DNS requests are resolved by the same exit nodes as the subsequent TCP traffic, as explained in Section 3.3.2), TorSH could similarly enforce that exit nodes are hosted in a different country than the router being configured, which would minimize the likelihood of the originator and the exit routers having the same ISP.

While TorSH conceals a home's own outgoing smart-device traffic from ISP-like adversaries, TorSH-enabled routers still place outgoing, unconcealed smart-device traffic on behalf of other TorSH users by virtue of the fact that all participants are exit routers. At first glance, this appears to offer the added benefit that attempted profiling by ISP-like adversaries would result in nonsensical profiles: users' daily routines and their homes' smart devices would appear to change often. In practice, we anticipate that ISP-like adversaries would be able to identify TorSH-enabled routers because they would regularly communicate with other TorSH participants, whose IP addresses are published in a publicly accessible directory, and their own IP address would appear in the same directory. Ultimately, we anticipate that the ISP-like adversaries would realize the futility of attempting to profile TorSH users and give up.

Non-smart-device traffic

By design, TorSH explicitly excludes non-smart-device traffic from its network. As a result, TorSH provides less-comprehensive privacy assurances than Tor or other solutions which apply to all traffic. However, other strategies could help users to protect their non-smart-device traffic from profiling with minimal impact on related experiences. Namely, users could install extensions on their web browsers or on their routers to replace unencrypted DNS requests with encrypted DNS-over-HTTPS or DNS-over-TLS requests. While this traffic could still be susceptible to fingerprinting attacks, revoking access to cleartext FQDNs would significantly hinder the ability for ISP-like adversaries to profile users based on their DNS requests, as Gonzalez et al. did by studying DNS requests from users' web-browsing sessions [36]. Many popular web browsers recently added support for DNS-over-HTTPS,

although not all enable it by default [92]. Furthermore, some routers, including those running OpenWrt, can be configured to replace traditional DNS requests with encrypted ones [93].

5.1.2 Impact on smart-device experiences (DC2)

We separate our evaluation of TorSH's impact on smart-device experiences by testing device, each of which we presented in Table 4.2.

Before experimenting, we used TorSH's allowlist-proposal mechanism to assemble an allowlist of all the servers contacted by each smart device. Unless otherwise noted, we ensured each IP address contacted by each device was indeed being routed into the TorSH network by verifying the originator and exit routers' `ipset` lists.

Smart sensor

We observed via packet capture that the Govee Thermo Hygrometer uploads data to its cloud-based service providers over a TCP connection, which was consistently successfully established under each routing strategy. Because the sensor is a non-latency-sensitive device, we did not think it meaningful to compare the times to establish this TCP connection between routing strategies; as long as the connection was consistently, successfully established well within the 10-minute reporting interval (which it was), the user would not perceive any difference in performance between each routing strategy.

We noted that the device uploads its data to Govee servers over unencrypted HTTP, which poses a privacy risk. The plaintext JSON object that the device uploaded contained a device ID, its model number, a list of any warnings reported by the device, the latest temperature and humidity data, the device's battery level, and its Wi-Fi signal strength. While none of this information is inherently personal in nature, the device's ID is likely unique and a malicious exit router could use it to track the device's readings over time. This being said, the exit router used by an originator router would change regularly, so it would be unlikely that any one exit router could track the device's activity over extended periods of time. TorSH can do little to prevent snooping by exit routers on unencrypted traffic; the onus to protect consumer data falls on smart-device vendors.

Smart lamp

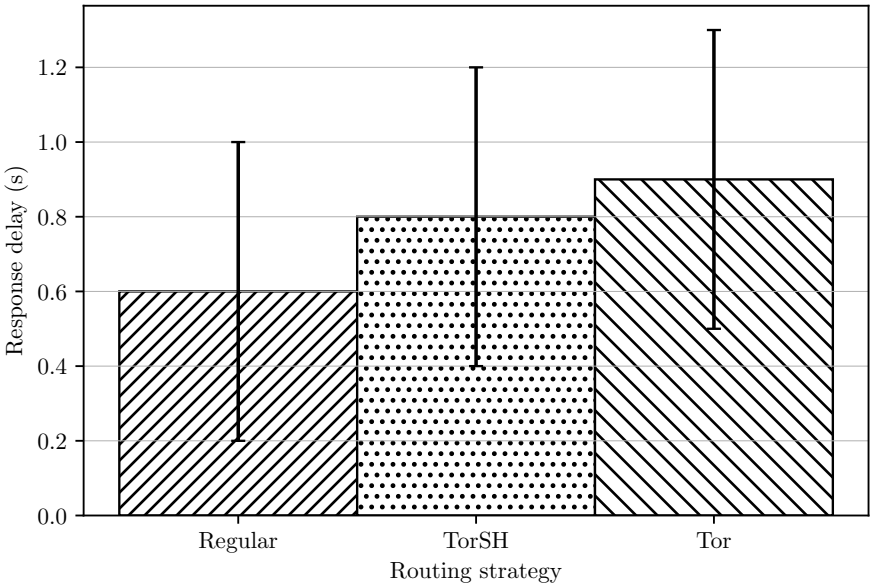
By contrast, smart lamps, like the Govee Aura Table Lamp we used for testing, are latency-sensitive smart devices: when a user changes the lamp's state (i.e., turning it on or off, changing its color), they expect the device to visibly respond swiftly. We present a summary of the delays between requests to change the lamp's state from the Govee Home app and the change visibly coming into effect under each routing strategy in Table 5.1.

Table 5.1: Delay between requesting Govee Aura Table Lamp state change (either turning lamp on, turning lamp off, or changing its color) and the state change visibly taking effect when routing normally, under TorSH, and under Tor. All values are in seconds.

	Regular				**TorSH**				**Tor**			
	min	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$	**min**	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$	**min**	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$
Trial 1	0.3	0.6	1.3	0.4	0.4	0.7	1.5	0.4	0.4	0.6	1.2	0.3
Trial 2	0.2	0.6	1.1	0.3	0.4	0.9	1.6	0.4	0.4	1.1	2.1	0.6
Trial 3	0.2	0.7	1.3	0.5	0.4	0.9	1.5	0.4	0.4	1.0	1.6	0.4
Average	**0.2**	**0.6**	**1.2**	**0.4**	**0.4**	**0.8**	**1.5**	**0.4**	**0.4**	**0.9**	**1.6**	**0.4**

We visualize the averaged data from Table 5.1 in Figure 5.1.

Figure 5.1: Delay between requesting Govee Aura Table Lamp state change (either turning lamp on, turning lamp off, or changing its color) and the state change visibly taking effect when routing normally, under TorSH, and under Tor. Error bars show one standard deviation.



On average, routing the lamp's traffic through TorSH resulted in a latency increase by a factor of 1.3, while routing under Tor resulted in an increase by a factor of 1.5. While this increase is substantial, many TorSH users may not notice it or care because of the delay's relatively small time scale: we believe a 0.2 s delay on a relatively infrequent smart-home experience is a small price to pay for increased privacy.

We note that the lamp opened an encrypted TLS v1.2 connection with a Govee server at startup, and we observed one incoming data packet from the server whenever we changed the device's state from the app. Once per trial, we also noted the lamp uploading its model number, Wi-Fi and Bluetooth versions, and device ID in a plaintext JSON object over an unencrypted HTTP connection to a cloud-based service provider. We believe this poses a

similar privacy risk to the Govee smart sensor, as no personal information is uploaded in plaintext, but a unique device ID that could be used to track the device over time is.

Smart assistant

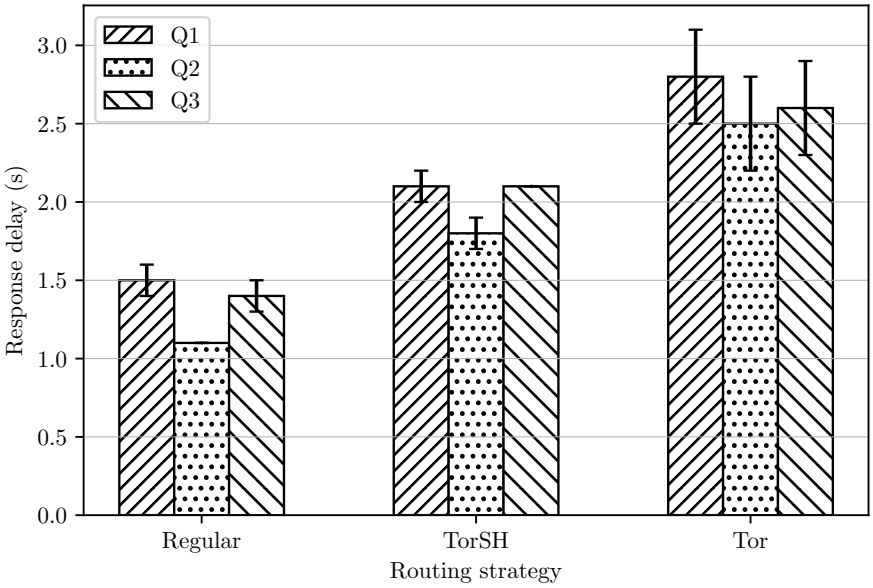
Interactions with smart assistants like Alexa are also latency-sensitive experiences: lengthy delays between prompts and responses feel unnatural. We present a summary of the delays between the end of our prompt to Alexa and the beginning of her response under each routing strategy in Table 5.2.

Table 5.2: Delay between the end of our question to Alexa and the beginning of her response when routing normally, under TorSH, and under Tor. All values are in seconds.

	Regular				**TorSH**				**Tor**			
	min	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$	**min**	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$	**min**	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$
Q1	1.4	1.5	1.5	0.1	2.0	2.1	2.1	0.1	2.7	2.8	3.0	0.1
Q2	1.1	1.1	1.1	0.0	1.7	1.8	1.9	0.1	2.1	2.5	2.7	0.3
Q3	1.3	1.4	1.5	0.1	2.0	2.1	2.1	0.0	2.3	2.6	3.0	0.3
Average	**1.3**	**1.3**	**1.4**	**0.0**	**1.9**	**2.0**	**2.0**	**0.1**	**2.4**	**2.6**	**2.9**	**0.3**

We visualize the averaged data from Table 5.2 in Figure 5.2.

Figure 5.2: Delay between the end of our question to Alexa and the beginning of her response when routing normally, under TorSH, and under Tor. Error bars show one standard deviation.



On average, routing Alexa's traffic through TorSH resulted in an interaction–response delay increase by a factor of 1.5, while routing under Tor resulted in an increase by a factor of 2.0.

Like with the smart lamp, this increase is substantial, yet users may be willing to accept an additional 0.7 s delay in getting a response from Alexa in exchange for increased privacy under TorSH.

We noted that the Echo Dot established both TLS v1.2 and TLS v1.3 connections with several Amazon servers upon startup, but that both upstream voice data and downstream responses appeared to be transmitted over a TLS v1.2 stream.

Smart camera

Smart cameras, like the Blink Mini we used for testing, are perhaps just as jitter-sensitive devices as they are latency-sensitive: users will maximize their utility from a smart video camera if its live playback is smooth and readily accessible. To this end, we present a summary of the delays between requesting the camera's live stream from the Blink Home Monitor app and the live stream commencing under normal routing and routing through TorSH in Table 5.3. While we attempted to route the camera's traffic through Tor, an error occurred in the app while doing so; we discuss this error in more detail at the end of this subsection.

Table 5.3: Delay between requesting the Blink Mini's live camera feed in the Blink Home Monitor app and playback commencement when routing normally and under TorSH. We do not report values for routing through Tor as an error occurred when requesting playback. All values are in seconds.

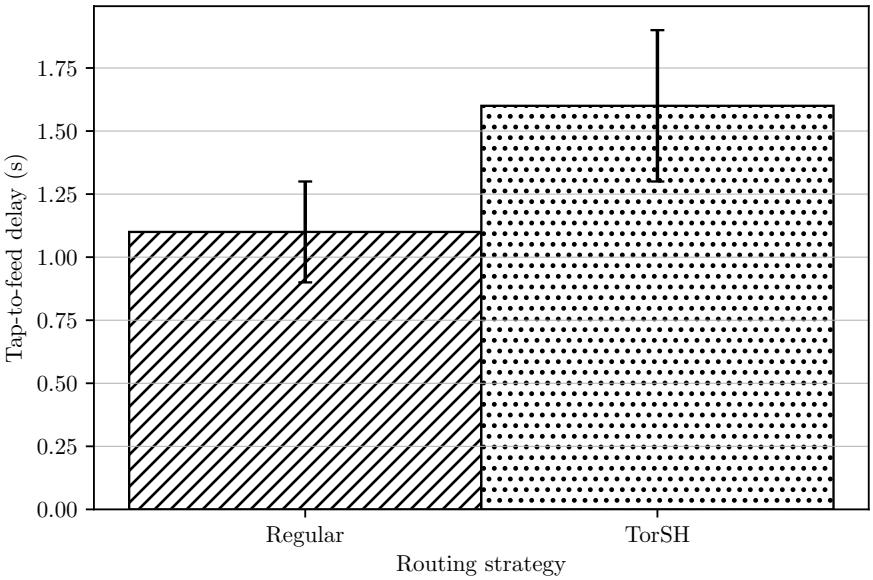
	min	$\boldsymbol{\mu}$	**max**	$\boldsymbol{\sigma}$
Regular	0.9	1.1	1.5	0.2
TorSH	1.2	1.6	2.0	0.3
Tor	-	-	-	-

We visualize the mean data from Table 5.3 in Figure 5.3.

On average, it took 1.5 times as long for playback to commence in the Blink Home Monitor app when routing under TorSH compared to normal routing. Again, while a significant increase, many users may be willing to wait an additional 0.5 s for playback in exchange for increased privacy.

Furthermore, we noted that playback under TorSH was just as smooth as under regular routing. However, this observation should not be interpreted as a testament to TorSH's low latency but rather results from the Blink Mini being implemented differently than we expected. Although we ensured all FQDNs contacted by the device were included by the allowlist during testing, the camera consistently uploaded its live video to an IP address that was not returned in any unencrypted DNS response to the device, and this IP address consistently changed across trials. As such, the packets containing the camera's live stream were not actually subjected to onion routing across the WAN; only those leading up to the live stream were. We assume that the Blink Mini's cloud-based service providers perform their own internal DNS-over-TLS with load balancing. This mechanism prevents the TorSH application from knowing whether to route the live stream traffic into the TorSH network, and even if the originator router knew to do so, the exit router would not know to trust traffic destined for that server because it did not see a DNS response for an allowlisted

Figure 5.3: Delay between requesting the Blink Mini's live camera feed in the Blink Home Monitor app and playback commencement when routing regularly and under Tor. We do not report values for routing through Tor as an error occurred when requesting playback. Error bars show one standard deviation.



FQDN including that IP address. As a result of the live stream's traffic being routed normally, an ISP-like adversary could likely infer from traffic patterns that a device in the home is uploading a video stream to some server, but it may not be obvious whether the live stream is originating from a smart camera or from a social media live stream (assuming no unencrypted DNS requests preceded the stream upload, as this would divulge the FQDN of the server receiving the stream). Ultimately, this experiment revealed that smart devices that use encrypted DNS cannot benefit from TorSH's privacy assurances (perhaps paradoxically, given the privacy advantages of such protocols over unencrypted DNS). This is a weakness of TorSH's design, and we leave it to future work to adapt out FQDN–IP-address consensus mechanism to support devices that use encrypted DNS-resolution protocols.

The normal routing of the live stream under TorSH means our experiment fails to capture TorSH's true impact on the Blink Mini's smart-camera experience, which makes it hard to generalize to all smart cameras. While our measurement of the delay between requesting a live stream and the stream beginning is meaningful, our observation that the ensuing playback is just as smooth with TorSH enabled as it is under normal routing is meaningless, as both are ultimately routed the same way (normally).

Finally, the error that occurred while routing the live stream under Tor merits discussion. While the generic error message displayed by the Blink Home Monitor app did not provide any insight into the nature of the issue, we have several hypotheses as to what went wrong.

- At some point shortly after power-up, the Blink Mini reported its IP public address to the cloud-based service provider accepting the video upload. The server then refused

to accept the video stream appearing to originate from the Tor exit node, whose public IP address differed from that of the camera.

- The delay created by onion routing across the WAN exceeded some internal TTL value specified by the Blink Mini. The server thus rejected the expired data.
- The Tor exit node was in a country that blocked traffic to the IP address of the server, or the server rejects traffic originating from the country of the Tor exit node.

All of these possibilities are cause for concern because each is just as likely to happen under TorSH as they are under Tor; if the aforementioned routing anomaly could be resolved, routing under TorSH might too fail. We leave the evaluation of TorSH's impact on smart-camera video streams whose traffic is actually routed into the TorSH network to future evaluation.

Consequences of traffic egress

As evidenced by the Blink Mini's inability to upload its live stream under Tor, smart devices may behave unexpectedly when their traffic appears to originate from a TorSH exit router rather than from the originator router. For example, this could be because a cloud-based service provider may internally identify its clients by their reported IP address and not by the IP address of their incoming connection, or because only the instance of the server that is geographically closest to the smart device is prepared to serve it, whether for efficiency or regulatory reasons. Either way, smart devices whose experiences would otherwise be unhindered by increased latency may cease to function as expected under TorSH.

While we only noted such hindered functionality while experimenting with the Blink Mini, Ren et al. studied the impact of smart-device traffic egress in more depth [94]. They purchased 81 Internet-connected smart devices, some from the US and some from the UK, and studied the effect of routing traffic from US-based devices through a VPN server located in the UK, and vice versa. They observed that while US-based devices contacted fewer third-party cloud-based service providers when their traffic was routed through the UK, likely due to more relaxed privacy laws in the US than the UK, no devices appeared to offer reduced functionality as a result of their traffic egress. This bodes well for TorSH.

However, the authors noted that content-based devices such as smart televisions offered different content depending on where their traffic exited (i.e., US-purchased smart televisions offered popular British programs on various streaming platforms when they routed the device's traffic through the UK-based VPN, and vice versa). While not ultimately resulting in *hindered* functionality in the case of Ren et al., some users may find it inconvenient to be presented with less relevant content upon first opening such apps, particularly if content they are expecting to find by virtue of their physical jurisdiction is unavailable in that through which their traffic is ultimately routed (i.e., due to differences in licensing rights). This would primarily impact services who serve content by traffic origin as opposed to by the jurisdiction of the user's billing address. We further evaluate the impact of the decision regarding whether to route content from streaming services into TorSH in Section 5.2.1.

We also note that routing non-smart-device traffic into TorSH may result in similarly mismatched localization, which users may find annoying. For example, while measuring the time to load Google's home page under Tor as part of our evaluation of DC5, we noticed

that we were presented with cookie-use notices in Dutch and German (likely reflecting the locations of the exit routers our Tor circuit used).

Impact of experimental conditions

We recognize that testing TorSH under our experimental conditions may result in lower interaction–response latencies compared to what consumers may experience in real-world conditions. Despite deploying virtual cloud-based TorSH nodes on restricted hardware, as detailed in Section 4.2, Google's data centers likely have access to faster Internet connections than most residential homes. While the maximum allowed egress bandwidth of the virtual machine tier we used was 1 GB/s at the time of experimentation (this metric being the most meaningful we could find in Google's documentation), the average worldwide broadband download and upload speeds were 105 MB/s and 56 MB/s, respectively, as of May 2021 (although "fiber" residential Internet subscriptions offering bandwidth above 1 GB/s are becoming increasingly commonplace) [95, 96]. We leave it to future work to evaluate TorSH's performance under more real-world conditions.

5.1.3 Network scalability (DC3)

One of the mainstream Tor network's biggest challenges is its dependence on volunteers to operate Tor relays. After all, operating a relay is a demanding task: it entails maintaining a server with a relatively reliable network connection and high bandwidth limits, as well as bearing legal responsibility for traffic the operator did not truly originate (which may be especially nerve-wracking considering some Tor users perform illegal or unethical activity online). As a result, most Tor users are *not* relay operators, which forces relay operators to throttle their bandwidth as discussed in Section 2.3.4.

By contrast, TorSH's requirement that all users reciprocate as relays alleviates resource constraints. With one relay per user, the traffic each relay is individually responsible for routing is greatly reduced, and the use of the allowlist to prevent abuse builds trust such that all nodes can feel comfortable acting as exit nodes.

5.1.4 Geographic restrictions (DC4)

Unlike EPIC, TorSH does not require participating nodes to be in geographic proximity (i.e., within wireless range) of each other [58]. In fact, having geographically dispersed nodes is advantageous to TorSH because the likelihood that all nodes in a circuit have the same ISP decreases.

This being said, as is the case with Tor, TorSH could be blocked by adverse governments or ISPs. This could be achieved by blocking the IP addresses of the central authorities, which would inhibit participant discovery and allowlist updates, or by blocking Tor-like traffic patterns (i.e., characterized by a fixed cell size). Even then, TorSH could adopt Tor techniques such as bridges and pluggable transports. Ultimately, the issues of Tor (and by extension, TorSH) censorship and Internet censorship more broadly are beyond the scope of this thesis and are already widely discussed in literature; we defer to these works for further strategies to combat such censorship.

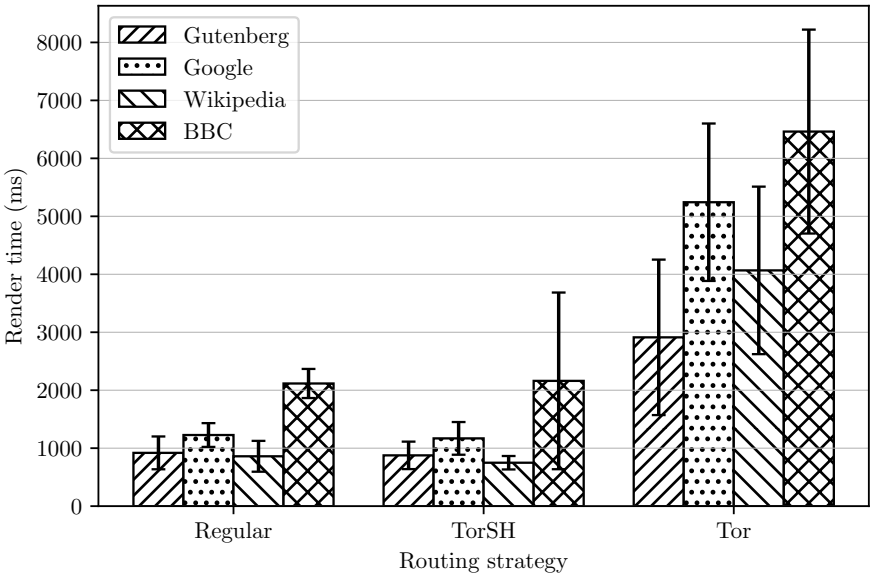
5.1.5 Impact on non-smart-device experiences (DC5)

Table 5.4 summarizes the results of our experiment comparing the times to load the web pages listed in Table 4.4 under each routing mechanism. Figure 5.4 visualizes these results.

Table 5.4: Times to render the web pages listed in Table 4.4 when routing normally, under TorSH, and under Tor. We rendered each web page 100 times. All values are in milliseconds.

	min	**μ**	**max**	**σ**	**min**	**μ**	**max**	**σ**
	Gutenberg				**Google**			
Regular	701	919	3029	283	1037	1227	2347	205
TorSH	599	876	2799	237	928	1169	2781	282
Tor	2101	2912	13426	1341	2976	5243	8246	1358
	Wikipedia				**BBC**			
Regular	565	860	2635	266	1892	2116	3703	251
TorSH	574	749	1060	116	1799	2011	4212	296
Tor	2146	4067	10736	1445	3957	6462	9730	1759

Figure 5.4: Mean time to render each web page listed in Table 4.4 when routing normally, under TorSH, and under Tor according to the Navigation Timing Level 2 specification after 100 renderings. Error bars show one standard deviation.

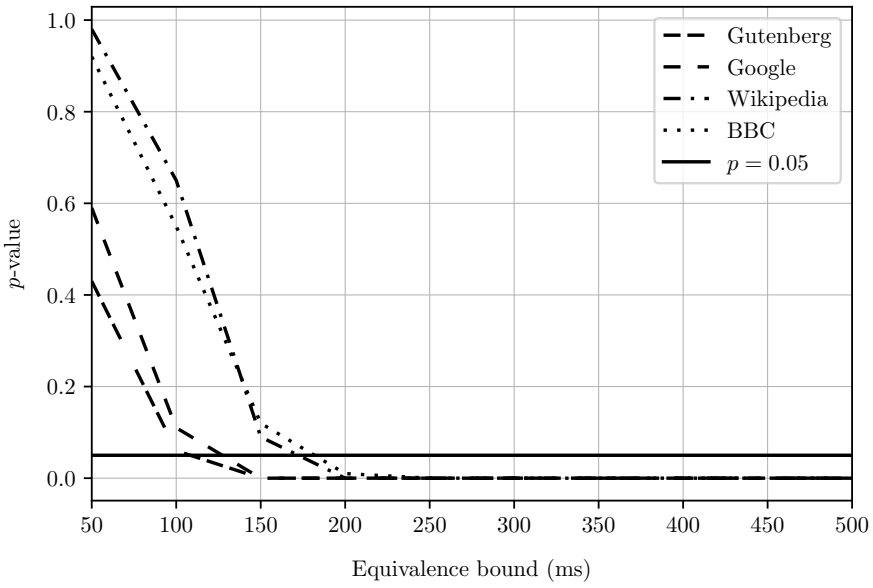


At first glance, the distributions for normal routing and routing under TorSH appear to be similar, while routing under Tor increases mean web-page render times by a factor of 3.81 compared to under regular routing – a latency hit many users may find unbearable despite the increased privacy benefit.

By comparison, routing under TorSH *decreases* mean render times by a factor of 0.93 compared to normal routing. Because we find little reason to believe routing through TorSH

would meaningfully and consistently result in decreased web-page render times, we believe this finding to be coincidental. Ultimately, to demonstrate statistically significant equivalence between web-page render times under regular routing and under TorSH, we conducted a *two one-sided tests* (TOST) procedure, which Lakens explains can be used "to statistically reject the presence of effects large enough to be considered worthwhile" [97]. The TOST procedure requires that we define an *equivalence bound*, within which we compare two distributions for equivalence. To determine appropriate values for this parameter in the context of web-page load times, we turned to Arapakis et al., who studied the effect of response latency on users' search engine experiences by adding artificial delays to the results' load time [98]. They found that artificial response-time delay increases of up to 500 ms were imperceptible to users. We thus decided to perform the TOST procedure considering delay bounds at every 50 ms between 50 ms and 500 ms, inclusive, to better understand the nature of the statistical equivalence between web-page render times under normal routing and under TorSH. We use $p < 0.05$ to determine statistically significant equivalence. Figure 5.5 illustrates our findings.

Figure 5.5: Results of TOST procedures examining statistical significance of equivalence between web page render times with and without TorSH enabled, considering delay bounds at every 50 ms between 50 ms and 500 ms, inclusive. We use $p < 0.05$ to conclude statistically significant equivalence.



Our TOST procedures results in $p < 0.05$ for all web page render times both with and without TorSH enabled below bounds of 200 ms, well below the 500 ms threshold for user perceptibility determined by Arapakis et al. As such, we conclude that TorSH does not meaningfully encumber non-smart-device traffic whose FQDNs are not in the allowlist.

FQDNs contacted by both smart and non-smart devices

We caution that servers likely to be contacted by smart devices as well as during non-smart-device experiences, such as web browsing, should not be added to the allowlist. For example, while certain Google Home smart devices may place DNS requests to resolve `www.google.com`, consumers may not ultimately be happy if Google's home page suddenly takes significantly longer to load than they are accustomed to given the frequency with which they may visit that page. Even domains that users are unlikely to intentionally visit in a browser, such as `ad2.doubleclick.net`, which smart devices may contact for advertising purposes, should not be added to the allowlist as doing so may slow every web page that serves ads using the DoubleClick advertising service.[1] Fortunately, their exclusion from the allowlist is unlikely to reveal the presence of a particular smart device in the home because these domain names are likely to be resolved during web browsing anyways and are thus likely to be cached by either the client or the router.

5.1.6 Router resource requirements (DC6)

Each trial of our DNS request "stress test" lasted about 8 seconds. In Table 5.5, we present the rates of outgoing DNS requests during this experiment. We observed a 100% success rate in resolving the requests without need for retries.

Table 5.5: Rates of outgoing DNS requests during hardware resource consumption evaluation "stress test." All values are in outgoing DNS requests per second.

	Regular			**TorSH**		
	Trial 1	**Trial 2**	**Trial 3**	**Trial 1**	**Trial 2**	**Trial 3**
max	146	143	154	158	170	152
$\boldsymbol{\mu}$	125	116	124	138	91	139

We present the average CPU and RAM utilization of the `torsh` process across each trial in Table 5.6. This data results from sampling the process's consumption 20 times per second (we could not identify a technique to continuously monitor a process's resource consumption over time).

Table 5.6: Hardware resource utilization of the `torsh` process during DNS request bombardment.

		Trial 1	**Trial 2**	**Trial 3**
CPU (%)	**max**	0.9	1.1	1.2
	$\boldsymbol{\mu}$	0.9	1.1	1.1
	min	0.9	1.0	1.1
RAM (KB)	**max**	7092	8696	7512
	$\boldsymbol{\mu}$	7089	8080	7337
	min	7088	7240	7328

[1] We recognize the irony in tying to prevent ISP-like adversaries from collecting smart-device data from consumers for advertising purposes while condoning similar data collection by advertising services, but per our model of the consumer, they have made the informed decision to use their smart devices knowing that cloud-based service providers may use the data they produce to profile them.

We note that we began profiling the `torsh` process before we started the DNS request bombardment, and we similarly ended the profiling after the bombardment ended. This allowed us to measure differences in resource utilization while the process was idle compared to while under heavy use.

Ultimately, we observed virtually no increased hardware resource consumption that correlated with the bombardment. This suggests that TorSH's average of 1.0% of CPU utilization and approximately 7500 KB RAM usage remains relatively constant over time (although RAM usage is likely dependent on the size of the allowlist and the number of cached IP addresses).

We further note that the installation size of the `torsh` OpenWrt package compiled for our test router's architecture was 6.9 MB in size, which includes all libraries required for the `torsh` binary.[2] The `tor` OpenWrt package, which is a dependency of `torsh`, requires an additional 3.4 MB when installed.

Tor hardware requirements

Our evaluation of TorSH's resource consumption so far only considers the hardware consumption of the `torsh` process. TorSH's onion routing functionality is handled by a separate `tor` process, which too has its own hardware requirements. On their website, the Tor Project outlines these requirements for relays, which we summarize here [99]:

- the ability to handle at least 7000 concurrent connections, and more for exit relays;
- a minimum of 10 Mbit/s upload and download bandwidth;
- a sufficiently generous outgoing traffic allowance;
- between 512 MB and 1.5 GB of RAM;
- approximately 200 MB of storage for Tor-related data;
- any modern CPU.

These requirements are demanding enough so as to preclude some existing consumer router models from participating in the TorSH network. However, as explained in Section 5.1.3, TorSH's requirement that each TorSH user also reciprocate in offering their router as a relay spreads the traffic routing burden such that each individual router can expect to handle far fewer connections and far less traffic than a regular Tor relay. As such, we believe that in practice, TorSH-enabled routers would be able to make do with fewer hardware resources than the Tor Project prescribes.

In fact, we estimate that participating in the TorSH network would cost users approximately seven times the total bandwidth (i.e., upload and download combined) consumed by the *average* TorSH user's smart-device traffic (i.e., not including web-browsing or other general traffic). This results from TorSH's mandatory participation in three-hop onion routing, which we illustrate in Figure 5.6. This estimate includes neither the overhead incurred during Tor processes such as participant discovery and circuit construction nor during TorSH's allowlist update and proposal processes; we assume these to be negligible relative to the burden of routing actual smart-device traffic. We predict that this amount of bandwidth, in real terms, is significantly less than that of the average Tor relay. This being said,

[2] We compiled `torsh` using Rust's `release` build optimization level. More fine-tuned compilation optimization settings may achieve a smaller installation size for minimal performance trade-off, but we leave this investigation to future work.

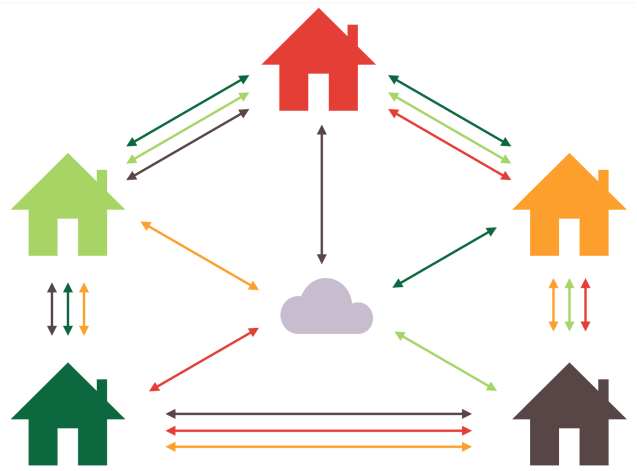


Figure 5.6: Sample smart-device traffic distribution among TorSH participants in a three-hop onion routing topology. Each arrow represents one unit of the average TorSH participant's smart-device traffic bandwidth. The color of each arrow reflects the participant originating that traffic. Each Tor circuit is composed of three connections between participants as well as one connection from the exit router to the cloud-based service provider. In practice, TorSH participants would divide their smart-device traffic across multiple circuits, which would distribute their bandwidth.

we emphasize that our estimate is crude and further work is required to more accurately predict the amount of additional bandwidth a TorSH participant can expect to route, as well as to investigate realistic minimum hardware requirements for TorSH-enabled routers. It is also unclear how ISPs would react to TorSH-using customers' increased bandwidth consumption, which they may argue is redundant (although, to our knowledge, ISPs do not currently demand that their customers be frugal while using their services).

5.1.7 Support for both TCP and UDP (DC7)

It is clear from our evaluations with smart-device traffic that TorSH can reliably establish TCP connections and send packets over them.

We did not observe any non-DNS, non-NTP UDP traffic originating from any of the smart devices we used for testing. As such, we could not effectively evaluate TorSH's UDP-over-TCP feature, nor could we understand the impact that increased latency would have on smart-device experiences with short UDP packet TTLs (e.g., VoIP). We leave the evaluation of TorSH's impact on UDP-based smart-device experiences to future work.

5.1.8 Support for legacy standards (DC8)

Unlike VPN-Zero, TorSH is not dependent on novel technologies or standards [11]. Tor, as TorSH's underlying routing mechanism, is indifferent as to the version of TLS that clients use.

5.1.9 Ability to prevent abuse (DC9)

As detailed in Section 3.3.2, TorSH's allowlist-based routing serves two purposes: it ensures that honest originator nodes only route relevant smart-device traffic into the TorSH network, and it protects honest exit nodes from routing irrelevant traffic placed by malicious originator nodes. The latter is important because malicious originator nodes can bypass TorSH's filtering and route any traffic into the TorSH network by directing it to `tor`'s SOCKS proxy port, into which the `torsh` process ultimately routes smart-device traffic.

Defending against malicious originator nodes

One inevitable weakness of an allowlist-based exit-node defense is that maliciously routed irrelevant traffic is still inevitably routed through the TorSH network to the exit node. This leaves TorSH susceptible to denial-of-service (DoS) attacks, whereby a malicious originator node repeatedly routes irrelevant traffic into the network in an effort to overwhelm intermediary routers.

One possible extension of TorSH could be to defend against DoS attacks by sanctioning participants who repeatedly perform them, but we leave the designing of such a defense to future work.

Defending against malicious guard and middle nodes

We anticipate few attacks from malicious guard and middle nodes. All traffic they are privy to is encrypted, and it is difficult for them to appear as honest nodes so as to be included in the Tor consensus while failing to forward traffic to the next relay: doing the latter could only be accomplished by blocking traffic exiting Tor's onion routing port, which would preclude the node from being reachable and therefore included in the directory in the first place.

Defending against malicious exit nodes

One possible attack by malicious exit nodes could be to refuse to forward allowlisted traffic to its intended destination. While such an attack could be unbeknownst to originator nodes with UDP-over-TCP, because connectionless protocols like UDP preclude assurance of message delivery, the originator node would know if relaying data over TCP to its intended destination failed because they would not receive a valid TCP `ACK` packet in return. In this case, the originator node's `tor` process would try to establish a connection with the same intended destination over a circuit with a different exit node. Tor assigns exit nodes that consistently fail to route outgoing traffic a `BadExit` label to warn clients to avoid using these relays as exit nodes [100].

Similarly, Tor's configuration options empower nodes seeking to "free ride" off TorSH to pose as participating relays while configuring their local Tor setup to refuse to route traffic to certain (or all) port numbers: without being labeled as a `BadExit`, Tor exit relays are allowed to declare that they will only route traffic to certain port numbers (e.g., 80 and 443). Under TorSH, exit routers could simply refuse to route outgoing traffic to any port, and the directory authorities would not offer these nodes as exit routers as a result (but

would still expect them act as guard or middle relays). This would disproportionately burden honest TorSH participants with exit traffic. Fortunately, Tor also publishes the range of port numbers a node in the consensus has agreed to route traffic to. While we have yet to implement this feature, we believe TorSH could easily be extended to enforce that participating nodes route traffic to all ports deemed valid by the directory authorities. We leave implementing this feature to future work.

5.2 Future work

Throughout this thesis, we have identified several directions for future work to further evaluate and improve TorSH. In summary, these aforementioned areas for expansion include:

- implementing periodic exit-router rotation (and using exit routers from countries other than that of the originator router);
- evaluating interaction–response latencies when traffic of the smartphone on which interactions are occurring is also routed into TorSH;
- devising defenses against fingerprinting attacks, including noise generation for TorSH users with few smart devices;
- improving support for smart devices that use encrypted DNS protocols;
- evaluating TorSH's impact on the quality of smart-camera video streams when these streams are actually routed through other TorSH routers;
- rigorously estimating the amount of additional bandwidth a router could expect to route by participating in the TorSH network;
- evaluating TorSH's latency performance under real-world as opposed to experimental conditions;
- evaluating the impact of UDP-over-TCP on UDP-based smart-device experiences;
- conceptualizing and implementing defenses against DoS attacks on the TorSH network;
- defending against "free riders" by enforcing a minimum subset of accepted ports for outgoing connections.

In the rest of this section, we present new directions for future work not yet presented.

5.2.1 Consider burden of streaming-service traffic

We recognize that traffic destined for streaming services may disproportionately burden the TorSH network. For example, routing domain names such as `*.spotify.com`, which smart speakers may contact to stream music as might a laptop from the Spotify application, would result in a significant increase in the amount of bandwidth routed into the TorSH network. Even if the incurred additional latency was not enough to noticeably decrease the quality of Spotify streams, the extra traffic could stress lower-end routers or incur extra latency for

other smart-device experiences that would become noticeable. The same could be said for domains such as `*.youtube.com` or `*.netflix.com`, which a smart television may contact. Although not allowlisting these domains would leave such streams vulnerable to profiling, users are ultimately no worse off than had they not used TorSH at all. There may be some middle ground with regards to the type of streams TorSH can accommodate (e.g., perhaps smart-camera live streams would not overwhelm the network). We leave the evaluation of the TorSH network's ability to accommodate different types of streaming-service traffic to future work.

We further note that, as discussed in Section 5.1.2, routing streaming services into the TorSH network may impact the availability of certain licensed content. Further work is required to identify the platforms which may be affected by such content-licensing issues.

5.2.2 Consider Tor-over-QUIC implementation

Because TorSH is a private Tor network, it is possible to enhance the network by modifying `tor`'s source code and asking all participants to use this modified version of `tor` rather than the normal distribution. One such modification could be replacing TCP with QUIC, a performance-oriented, UDP-based protocol, for inter-relay connections. Heijligers found that doing so could result in up to 50% improvement in time-to-last-byte performance [81].

5.2.3 Distinguish between DNS-only and full-traffic routing

We believe one way to extend TorSH to further defend against profiling by ISP-like adversaries could be to extend the allowlist to specify whether an exit node should resolve a DNS request for a particular FQDN and route subsequent packets destined for any related IP addresses **or** should simply resolve the DNS request without routing subsequent packets destined for related IP addresses. We believe such a feature could be useful in the context of the streaming issue presented in Section 5.2.1, whereby a user may want to protect smart-device-related DNS requests from profiling when the TorSH network cannot handle the related traffic.

5.2.4 Evaluate with more devices

Despite mindfully attempting to include a variety of smart devices in our evaluation, our selection falls short of encompassing the true breadth of the consumer smart-device market. As such, more work is required to evaluate TorSH's impact on smart-device experiences (considering factors such as latency and functionality hindrance) using more types of devices by a wider selection of manufacturers.

5.2.5 Explore improved allowlist-consensus mechanisms

TorSH's dependence on one or more trusted organizations to maintain an allowlist of valid, smart-device-related FQDNs could act as a barrier to adoption: until an organization agrees to do so, users cannot use TorSH. If the organization eventually stops updating the allowlist, privacy protections may wane with time.

Ideally, TorSH's architecture could be adapted to empower users to decide amongst themselves which FQDNs are indeed smart-device-related. As explained in Section 3.1, in the context of a collaborative intrusion-detection system, Thomasset proposed that participants log the endpoints contacted by each of their smart devices to device-specific blockchains, to which users subscribe for each smart device they own. If a user observes that their device contacts substantially different endpoints than those reported by the majority of owners of the same device, their device may be compromised [21, 22]. While an elegant solution, such a system fails when nodes cannot verify for themselves the validity of a proposal: TorSH exit routers are asked to route traffic for smart devices they may themselves not own, and allowing them to only route traffic for devices they do own reveals the presence of a smart device in the home to ISP-like adversaries.

We also note that many users may be hesitant to make use of TorSH's configurable, opt-in allowlist-proposal mechanism; here too exists the possibility of "free riders." To combat this, consumer IoT organizations, including smart-device vendors themselves, could be asked to share the FQDNs of the cloud-based service providers contacted by their devices.

Ultimately, we leave the exploration of an alternative, possibly decentralized allowlist consensus mechanism for TorSH to future work.

5.2.6 Improve UDP support

As explained in Section 3.3.2, TorSH's current UDP-over-TCP implementation, which must open a new TCP connection (requiring a three-way handshake) once per UDP packet, is unnecessarily inefficient. It could be improved by periodically opening one TCP connection that is used to transmit multiple UDP packets and by using the TCP *keepalive* feature to prevent the connection from collapsing. The connection would ideally be reinstantiated periodically over a different circuit to ensure sufficient privacy.

Furthermore, TorSH's inability to properly support NTP packets could pose privacy weaknesses. On the one hand, smart devices may use vendor-specific NTP servers to synchronize their clocks (for example, we noted that the Amazon Echo Dot we used for testing contacted the NTP server `ntp-g7g.amazon.com`). An ISP observing a home trying to contact this server could infer that an Amazon smart device is installed in the home, but the exact type of device would likely remain unclear to them. On the other hand, smart devices may use public, generic NTP servers (such as `pool.ntp.org`) as opposed to vendor-hosted servers to synchronize their clocks, which would provide no such insight to ISP-like adversaries [101]. Furthermore, NTP synchronization requests are infrequent and unlikely to correlate with smart-device interactions (we observed no such correlations during testing) [34]. As such, there is likely little an ISP-like adversary could infer from a home's NTP traffic, beyond perhaps the presence of a smart device of a certain brand in the home if the device uses a vendor-specific NTP server. While our implementation of UDP-over-TCP does not currently enable exit routers to forward UDP responses to originator routers, we could consider implementing a special NTP-over-TorSH protocol that would allow exit routers to respond to originator routers' NTP requests (as every NTP request would merit a response, unlike vendors' custom UDP protocols, which may be unpredictable). We note, however, that it is unclear how the delays incurred during onion routing would affect device behavior, seeing that NTP synchronization is a time-sensitive mechanism. We leave exploring strategies to obfuscate smart-device NTP requests to future work.

Chapter 6

Summary and conclusion

In this thesis, we present TorSH, a private Tor network of smart-home routers working together to improve privacy against ISP-like adversaries by obfuscating smart-device network traffic.

We began this work by motivating the need to protect users against profiling by ISP-like adversaries, who stand to gain by monetizing consumer smart-device traffic. ISP-based advertisers' unique perspective allows them to profile consumers based on all of a home's traffic, which provides them a competitive advantage over edge-based advertisers like Facebook and Google who can only use data collected while users engage with their platforms. Given the "single-purpose IoT nature" of smart devices, access to the metadata associated with all of their traffic can provide detailed insight into users' at-home activities, such as their daily routines or any smart medical devices they may use. While some jurisdictions have barred ISPs from selling customers' traffic data, legal protections are neither permanent (as the case of the US demonstrates) nor universal.

We then outlined how existing technical solutions to the issue of user profiling by ISP-liked adversaries fall short of meeting the abilities and desires of many consumers, who may lack technical prowess or may enjoy the prospect of increased privacy but not at the expense of smart-home experiences. Solutions that block or minimize outgoing traffic may hinder smart-device functionalities. Using commercial VPNs merely shifts the burden of trust from ISPs to other profit-seeking entities, many of which have been shown to be dishonest towards consumers, and maintaining a personal VPN may be beyond the price range or technical abilities of the average consumer. Using traffic shaping to disguise smart-device traffic may incur significant bandwidth overhead and either requires intermediate infrastructure like a VPN to "de-shape" traffic before forwarding it to cloud-based service providers or requires that developers use traffic-shaping libraries while building their smart products. Existing multi-hop solutions either require users to maintain multiple ISPs (multihoming), require participating nodes to be in wireless proximity of each other (EPIC), depend on experimental technologies (VPN-Zero), or incur substantial latency increases that many users may find unbearable (e.g., mainstream Tor).

Next, we presented TorSH, our novel solution to the issue of user profiling by ISP-like adversaries. By selectively routing smart-device traffic into a collaborative, private Tor network composed of smart-home routers while leaving other traffic unencumbered, we are

able to reduce the amount of traffic metadata to which ISP-like adversaries are privy, thereby limiting the granularity of the consumer advertising profiles they are able to compile.

We also evaluated TorSH both empirically and qualitatively to ensure it met a suite of predetermined security, privacy, and performance goals. We measured TorSH's impact on interaction–response latency using a series of smart devices and found that the latency incurred while routing smart-device traffic through TorSH is unlikely to unbearably hinder smart-home experiences. We further demonstrated that TorSH has no statistically significant impact on non-smart-device experiences and that TorSH can likely be installed on widely available smart-home router hardware. Finally, we discussed the qualities of the TorSH network that make it secure and widely scalable, and we outlined directions for future work

The most effective way to defend against profiling by ISP-like adversaries where legal protections lack is to not use the Internet at all. It goes without saying that very few users would see this solution as practical in today's world. While routing all of a home's traffic through a VPN or the mainstream Tor network may come at too high a cost – whether financial or experiential – for some users, many may appreciate the ability to at least *somewhat* improve their privacy at little such expense. Ultimately, TorSH succeeds in offering this middle-of-the-road solution to the issue of protecting consumers against profiling by ISP-like adversaries: consumers who are satisfied with existing tools are free to use them, but those whose preferences are best met with TorSH's compromise now have a more suitable option.

Those who would eviscerate sanctuary are keen to take the offensive, putting us off guard with the guilt-inducing question "What have you got to hide?" But as we have seen, the crucial developmental challenges of the self-other balance cannot be negotiated adequately without the sanctity of "disconnected" time and space for the ripening of inward awareness and the possibility of reflexivity: reflection on and by oneself. The real psychological truth is this: If you've got nothing to hide, you are nothing.

— Shoshana Zuboff, *The Age of Surveillance Capitalism* [4]

Appendix A

Peripheral code contributions

The `torsh` program depends on GitHub user *rustn00b*'s fork of Guo's `nfq-rs` crate, which offers a Rust interface to the Linux `libnetfilter_queue` library [77, 102, 103]. The *rustn00b* fork offers an asynchronous implementation of Guo's library. We extended the *rustn00b* fork by implementing a call to *unbind* a queue as well as a test to ensure its proper functionality. We opened a pull request to incorporate our contributions into the *rustn00b* fork, which had yet to be approved as of June 1, 2022 [104].

Appendix B

Script to measure web-page render times

We used the following Python script to measure the render times of the web pages listed in Table 4.4.

```
from selenium import webdriver
import os
import time

# Based on https://stackoverflow.com/a/63196705
def time_url(driver, url):
    driver.get(url)
    navigation_start = driver.execute_script("return
        window.performance.timing.navigationStart")
    dom_complete = driver.execute_script("return
        window.performance.timing.domComplete")
    total_time = dom_complete - navigation_start
    print(f"Took {total_time}ms to render {url}")
    return total_time

def load_pages_sequentially(driver):
    times = []
    times.append(time_url(driver,
        "https://www.gutenberg.org/files/1342/1342-h/1342-h.htm"))
    times.append(time_url(driver, "https://www.google.com"))
    times.append(time_url(driver, "https://en.wikipedia.org"))
    times.append(time_url(driver, "https://www.bbc.com"))
    return times
```

```
def run_sequential_test():
    all_times = []
    num_times = 100
    for n in range(num_times):
        print(f"Iteration {n+1} of {num_times}")
        driver = webdriver.Chrome()
        try:
            os.system("sudo dscacheutil -flushcache; sudo killall -HUP
                mDNSResponder")
            time.sleep(1)
            all_times.append(load_pages_sequentially(driver))
            print( )
        finally:
            driver.close()
    print(f"All times for sequential requests are {list(zip(*all_times[::-1]))}")
    print("Sequential load times as CSV:")
    num_cols = len(all_times[0])
    for row in all_times:
        for col in range(num_cols):
            print(f"{row[col]}", end = "")
            if col != num_cols-1:
                print(",", end = "")
        print("")

run_sequential_test()
```

Bibliography

[1] J. Bugeja, A. Jacobsson, and P. Davidsson, "The Ethical Smart Home: Perspectives and Guidelines," *IEEE Security Privacy*, vol. 20, no. 1, pp. 72–80, Jan. 2022, ISSN: 1558-4046. DOI: 10.1109/MSEC.2021.3111668.

[2] N. Apthorpe, D. Reisman, and N. Feamster, "A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic," May 18, 2017. arXiv: 1705.06805 [cs].

[3] C. Fuchs, "Societal and Ideological Impacts of Deep Packet Inspection Internet Surveillance," *Information, Communication & Society*, vol. 16, no. 8, pp. 1328–1359, Oct. 1, 2013, ISSN: 1369-118X. DOI: 10.1080/1369118X.2013.770544.

[4] S. Zuboff, *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*, First edition. New York: PublicAffairs, 2019, 691 pp., ISBN: 978-1-61039-569-4.

[5] F. Möllers, S. Seitz, A. Hellmann, and C. Sorge, "Short paper: Extrapolation and prediction of user behaviour from wireless home automation communication," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks - WiSec '14*, Oxford, United Kingdom: ACM Press, 2014, pp. 195–200, ISBN: 978-1-4503-2972-9. DOI: 10.1145/2627393.2627407.

[6] X. Yu, Y. Zhang, X.-Y. Li, and X. Guo, "The Truman Show: Attack On The Privacy Of Smart Homes Through Traffic Analysis," in *2021 7th International Conference on Big Data Computing and Communications (BigCom)*, Aug. 2021, pp. 121–128. DOI: 10.1109/BigCom53800.2021.00033.

[7] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1362–1380. DOI: 10.1109/SP.2019.00013.

[8] S. Landau, "Categorizing Uses of Communications Metadata: Systematizing Knowledge and Presenting a Path for Privacy," in *New Security Paradigms Workshop 2020*, ser. NSPW '20, New York, NY, USA: Association for Computing Machinery, Oct. 26, 2020, pp. 1–19, ISBN: 978-1-4503-8995-2. DOI: 10.1145/3442167.3442171.

[9] "Tor," Tor Project. (Feb. 16, 2022), [Online]. Available: https://www.torproject.org.

[10] M. T. Khan, J. DeBlasio, G. M. Voelker, A. C. Snoeren, C. Kanich, and N. Vallina-Rodriguez, "An Empirical Analysis of the Commercial VPN Ecosystem," in *Proceedings of the Internet Measurement Conference 2018*, Boston MA USA: ACM, Oct. 31, 2018, pp. 443–456, ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278570.

[11] M. Varvello, I. Q. Azurmendi, A. Nappa, P. Papadopoulos, G. Pestana, and B. Livshits, "VPN-Zero: A Privacy-Preserving Decentralized Virtual Private Network," in *2021 IFIP Networking Conference (IFIP Networking)*, Jun. 2021, pp. 1–6. DOI: 10.23919/IFIPNetworking52078.2021.9472843.

[12] D. J. Solove, *Understanding Privacy*. Cambridge: Harvard University Press, May 2008.

[13] P. Emami-Naeini *et al.*, "Privacy expectations and preferences in an IoT world," in *Proceedings of the thirteenth USENIX conference on usable privacy and security*, ser. SOUPS '17, USA: USENIX Association, 2017, pp. 399–412, ISBN: 978-1-931971-39-3.

[14] L. Ardito, L. Barbato, P. Mori, and A. Saracino, "Preserving Privacy in the Globalized Smart Home: The SIFIS-Home Project," *IEEE Security Privacy*, vol. 20, no. 1, pp. 33–44, Jan. 2022, ISSN: 1558-4046. DOI: 10.1109/MSEC.2021.3118561.

[15] S. Lederer, J. Mankoff, and A. K. Dey, "Who wants to know what when? privacy preference determinants in ubiquitous computing," in *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*, Ft. Lauderdale, Florida, USA: ACM Press, 2003, p. 724, ISBN: 978-1-58113-637-1. DOI: 10.1145/765891.765952.

[16] C. Dolin *et al.*, "Unpacking Perceptions of Data-Driven Inferences Underlying Online Targeting and Personalization," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, Montreal QC Canada: ACM, Apr. 21, 2018, pp. 1–12, ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3174067.

[17] "Amazon Alexa," Wikipedia. (Jan. 11, 2021), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Amazon_Alexa&oldid=999682782 (visited on 01/14/2021).

[18] "Amazon Echo," Wikipedia. (Jan. 11, 2021), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Amazon_Echo&oldid=999628117 (visited on 01/19/2021).

[19] D. Kotz and T. Peters, "Challenges to ensuring human safety throughout the lifecycle of Smart Environments," in *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, ser. SafeThings'17, New York, NY, USA: Association for Computing Machinery, Nov. 5, 2017, pp. 1–7, ISBN: 978-1-4503-5545-2. DOI: 10.1145/3137003.3137012.

[20] B. H. Çorak, F. Y. Okay, M. Güzel, Ş. Murt, and S. Ozdemir, "Comparative Analysis of IoT Communication Protocols," in *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, Jun. 2018, pp. 1–6. DOI: 10.1109/ISNCC.2018.8530963.

[21] C. Thomasset and D. Barrera, "SERENIoT: Distributed Network Security Policy Management and Enforcement for Smart Homes," in *Annual Computer Security Applications Conference*, Austin USA: ACM, Dec. 7, 2020, pp. 542–555, ISBN: 978-1-4503-8858-0. DOI: 10.1145/3427228.3427235.

[22] C. Thomasset, "SERENIoT : politiques de sécurité collaboratives pour maisons connectées," M.S. thesis, Polytechnique Montréal, Jul. 2020.

[23] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the Smart Home Private with Smart(er) IoT Traffic Shaping," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 128–148, Jul. 1, 2019, ISSN: 2299-0984. DOI: 10.2478/popets-2019-0040.

[24] S. Bradshaw and L. DeNardis, "Privacy by Infrastructure: The Unresolved Case of the Domain Name System: Privacy by Infrastructure," *Policy & Internet*, vol. 11, no. 1, pp. 16–36, Mar. 2019, ISSN: 19442866. DOI: 10.1002/poi3.195.

[25] S. García, K. Hynek, D. Vekshin, T. Čejka, and A. Wasicek, "Large Scale Measurement on the Adoption of Encrypted DNS," Jul. 9, 2021. arXiv: 2107.04436 [cs].

[26] S. Pai *et al.*, "Transactional Confidentiality in Sensor Networks," *IEEE Security Privacy*, vol. 6, no. 4, pp. 28–35, Jul. 2008, ISSN: 1558-4046. DOI: 10.1109/MSP.2008.107.

[27] J. Sauter. "Property Owners Can Be Internet Providers," Broadband Communities Magazine. (Jul. 2017), [Online]. Available: https://www.bbcmag.com/multifamily-broadband/property-owners-can-be-internet-providers.

[28] A. Acar *et al.*, "Peek-a-Boo: I see your smart home activities, even encrypted!" In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, Linz Austria: ACM, Jul. 8, 2020, pp. 207–218, ISBN: 978-1-4503-8006-5. DOI: 10.1145/3395351.3399421.

[29] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2017, pp. 2177–2184. DOI: 10.1109/ICDCS.2017.283.

[30] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "IoTSense: Behavioral Fingerprinting of IoT Devices," Apr. 11, 2018. arXiv: 1804.03852 [cs].

[31] Y. Meidan *et al.*, "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*, Marrakech Morocco: ACM, Apr. 3, 2017, pp. 506–509, ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019878.

[32] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, Aug. 1, 2019, ISSN: 1536-1233, 1558-0660, 2161-9875. DOI: 10.1109/TMC.2018.2866249.

[33] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Computer Networks*, vol. 148, pp. 318–327, Jan. 2019, ISSN: 13891286. DOI: 10.1016/j.comnet.2018.11.013.

[34] B. Copos, K. Levitt, M. Bishop, and J. Rowe, "Is Anybody Home? Inferring Activity From Smart Home Network Traffic," in *2016 IEEE Security and Privacy Workshops (SPW)*, San Jose, CA: IEEE, May 2016, pp. 245–251, ISBN: 978-1-5090-3690-5. DOI: 10.1109/SPW.2016.48.

[35] V. Srinivasan, J. Stankovic, and K. Whitehouse, "Protecting your daily in-home activity information from a wireless snooping attack," in *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp '08*, Seoul, Korea: ACM Press, 2008, p. 202, ISBN: 978-1-60558-136-1. DOI: 10.1145/1409635.1409663.

[36] R. Gonzalez, C. Soriente, J. M. Carrascosa, A. Garcia-Duran, C. Iordanou, and M. Niepert, "User profiling by network observers," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '21, New York, NY, USA: Association for Computing Machinery, Dec. 2, 2021, pp. 212–222, ISBN: 978-1-4503-9098-9. DOI: 10.1145/3485983.3494859.

[37] I. Srivastava, "Uncertainty Surrounding the Repeal of the Internet Privacy Rules," *Intellectual Property and Technology Law Journal*, vol. 23, no. 2, pp. 99–116, Spr. 2019.

[38] "A Look at What ISPs Know About You: Examining the Privacy Practices of Six Major Internet Service Providers," Federal Trade Commission, Oct. 21, 2021.

[39] X. Ma, J. Qu, J. Li, J. C. Lui, Z. Li, and X. Guan, "Pinpointing Hidden IoT Devices via Spatial-temporal Traffic Fingerprinting," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Toronto, ON, Canada: IEEE, Jul. 2020, pp. 894–903, ISBN: 978-1-72816-412-0. DOI: 10.1109/INFOCOM41043.2020.9155346.

[40] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTAthena: Unveiling IoT Device Activities From Network Traffic," *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 651–664, Jan. 2022, ISSN: 1558-2248. DOI: 10.1109/TWC.2021.3098608.

[41] H. Blodget. "Compete CEO: ISPs Sell Clickstreams For $5 A Month," Seeking Alpha. (Mar. 13, 2007), [Online]. Available: https://seekingalpha.com/article/29449-compete-ceo-isps-sell-clickstreams-for-5-a-month (visited on 04/08/2022).

[42] J. Brodkin. "ISPs lied to Congress to spread confusion about encrypted DNS, Mozilla says," Ars Technica. (Nov. 4, 2019), [Online]. Available: https://arstechnica.com/tech-policy/2019/11/isps-lied-to-congress-to-spread-confusion-about-encrypted-dns-mozilla-says/ (visited on 04/08/2022).

[43] T. Brewster. "Now Those Privacy Rules Are Gone, This Is How ISPs Will Actually Sell Your Personal Data," Forbes. (Mar. 30, 2017), [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/03/30/fcc-privacy-rules-how-isps-will-actually-sell-your-data/.

[44] B. Naylor. "Congress Overturns Internet Privacy Regulation," NPR. (Mar. 28, 2017), [Online]. Available: https://www.npr.org/2017/03/28/521831393/congress-overturns-internet-privacy-regulation (visited on 04/09/2022).

[45] C. S. Alliance. "Homepage," Matter. (2021), [Online]. Available: https://buildwithmatter.com/ (visited on 06/01/2021).

[46] F. O'Sullivan. "Do ISPs Track and Sell Your Browsing Data?" How-To Geek. (May 4, 2021), [Online]. Available: https://www.howtogeek.com/724472/do-isps-track-and-sell-your-browsing-data/ (visited on 02/27/2022).

[47] "A Practical Guide to Data Privacy Laws by Country," i-Sight Software. (Mar. 5, 2021), [Online]. Available: https://www.i-sight.com/resources/a-practical-guide-to-data-privacy-laws-by-country/ (visited on 05/16/2022).

[48] J. Wolff and N. Atallah, "Early GDPR Penalties: Analysis of Implementation and Fines Through May 2020," *Journal of Information Policy*, vol. 11, no. 1, pp. 63–103, Jan. 1, 2021, ISSN: 2381-5892, 2158-3897. DOI: 10.5325/jinfopoli.11.2021.0063.

[49] "Use routers secured with HomeKit," Apple Support. (Nov. 12, 2021), [Online]. Available: https://support.apple.com/en-us/HT210544 (visited on 04/22/2022).

[50] A. Crabtree *et al.*, "Building accountability into the Internet of Things: The IoT Databox model," *Journal of Reliable Intelligent Environments*, vol. 4, no. 1, pp. 39–55, Apr. 1, 2018, ISSN: 2199-4676. DOI: 10.1007/s40860-018-0054-5.

[51] A. Alshehri, J. Granley, and C. Yue, "Attacking and Protecting Tunneled Traffic of Smart Home Devices," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, New Orleans LA USA: ACM, Mar. 16, 2020, pp. 259–270, ISBN: 978-1-4503-7107-0. DOI: 10.1145/3374664.3375723.

[52] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security - CCSW '09*, Chicago, Illinois, USA: ACM Press, 2009, p. 31, ISBN: 978-1-60558-784-4. DOI: 10.1145/1655008.1655013.

[53] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes, "IoTLS: Understanding TLS usage in consumer IoT devices," in *Proceedings of the 21st ACM Internet Measurement Conference*, Virtual Event: ACM, Nov. 2, 2021, pp. 165–178, ISBN: 978-1-4503-9129-0. DOI: 10.1145/3487552.3487830.

[54] T. Datta, N. Apthorpe, and N. Feamster, "A Developer-Friendly Library for Smart Home IoT Privacy-Preserving Traffic Obfuscation," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, Budapest Hungary: ACM, Aug. 7, 2018, pp. 43–48, ISBN: 978-1-4503-5905-4. DOI: 10.1145/3229565.3229567.

[55] S. Xiong, A. D. Sarwate, and N. B. Mandayam, "Network Traffic Shaping for Enhancing Privacy in IoT Systems," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2022, ISSN: 1063-6692, 1558-2566. DOI: 10.1109/TNET.2021.3140174.

[56] K. Yu, Q. Li, D. Chen, M. Rahman, and S. Wang, "PrivacyGuard: Enhancing Smart Home User Privacy," in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, Nashville TN USA: ACM, May 18, 2021, pp. 62–76, ISBN: 978-1-4503-8098-0. DOI: 10.1145/3412382.3458257.

[57] A. Engelberg and A. Wool, *Classification of Encrypted IoT Traffic Despite Padding and Shaping*, Oct. 21, 2021.

[58] J. Liu, C. Zhang, and Y. Fang, "EPIC: A Differential Privacy Framework to Defend Smart Homes Against Internet Traffic Analysis," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1206–1217, Apr. 2018, ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2799820.

[59] "The Legal FAQ for Tor Relay Operators," Tor Project. (Feb. 16, 2022), [Online]. Available: https://community.torproject.org/relay/community-resources/eff-tor-legal-faq/ (visited on 02/16/2022).

[60] "Tor Metrics," Tor Project. (Feb. 16, 2022), [Online]. Available: https://metrics.torproject.org.

[61] L. Lee, D. Fifield, N. Malkin, G. Iyer, S. Egelman, and D. Wagner, "Tor's Usability for Censorship Circumvention," in *Proceedings on Privacy Enhancing Technologies*, vol. 2, De Gruyter, 2017, pp. 1–20.

[62] "Bridges," Tor Project. (2022), [Online]. Available: https://tb-manual.torproject.org/bridges/ (visited on 05/09/2022).

[63] P. Winter and S. Lindskog, "How China Is Blocking Tor," Apr. 2, 2012. arXiv: 1204.0447 [cs].

[64] "Circumvention," Tor Project. (2022), [Online]. Available: https://tb-manual.torproject.org/circumvention/ (visited on 05/09/2022).

[65] T. Wang and I. Goldberg, "On Realistically Attacking Tor with Website Fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 21–36, Oct. 1, 2016, ISSN: 2299-0984. DOI: 10.1515/popets-2016-0027.

[66] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your Smart Home Can't Keep a Secret: Towards Automated Fingerprinting of IoT Traffic," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, Taipei Taiwan: ACM, Oct. 5, 2020, pp. 47–59, ISBN: 978-1-4503-6750-9. DOI: 10.1145/3320269.3384732.

[67] A. Bahalul Haque, S. Tahura Nisa, M. Amdadul Bari, and A. Nusreen Anika, "Anonymity Network Tor and Performance Analysis of ARANEA; an IOT Based Privacy-Preserving Router," Jun. 1, 2019.

[68] radio24, *TorBox*, version 0.5.0, 2022.

[69] N. P. Hoang and D. Pishva, "A TOR-based anonymous communication approach to secure smart home appliances," in *2015 17th International Conference on Advanced Communication Technology (ICACT)*, Jul. 2015, pp. 517–525. DOI: 10.1109/ICACT.2015.7224918.

[70] Tails, *Tails*, version 4.29, Center for Cultivation of Technology gGmbH, 2022.

[71] S. Henri, G. Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran, "Protecting against Website Fingerprinting with Multihoming," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 2, pp. 89–110, Mar. 31, 2020. DOI: 10.2478/popets-2020-0019.

[72] M. Basalla, J. Schneider, M. Luksik, R. Jaakonmäki, and J. Vom Brocke, "On Latency of E-Commerce Platforms," *Journal of Organizational Computing and Electronic Commerce*, vol. 31, no. 1, pp. 1–17, Jan. 2, 2021, ISSN: 1091-9392, 1532-7744. DOI: 10.1080/10919392.2021.1882240.

[73] J. Li, W. Liang, W. Xu, Z. Xu, and J. Zhao, "Maximizing the Quality of User Experience of Using Services in Edge Computing for Delay-Sensitive IoT Applications," in *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Alicante Spain: ACM, Nov. 16, 2020, pp. 113–121, ISBN: 978-1-4503-8117-8. DOI: 10.1145/3416010.3423234.

[74] N. Jafari Navimipour and F. Sharifi Milani, "A comprehensive study of the resource discovery techniques in Peer-to-Peer networks," *Peer-to-Peer Networking and Applications*, vol. 8, no. 3, pp. 474–492, May 2015, ISSN: 1936-6442, 1936-6450. DOI: 10.1007/s12083-014-0271-5.

[75] A. A. Monrat, O. Schelen, and K. Andersson, "A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2936094.

[76] R. Russell, *Iptables*, version 1.8.7, Netfilter Core Team, 1998.

[77] H. Welte, *Libnetfilter_queue*, version 1.0.5, Jun. 12, 2022.

[78] J. Kadlecsik, *Ipset*, version 7.15.

[79] J. Engelhardt, *Schematic for the packet flow paths through Linux networking and Xtables*, May 19, 2019.

[80] G. Hoare, *Rust*, version 1.60.0, The Rust Foundation, Apr. 7, 2022.

[81] J. Heijligers, "Tor over QUIC," Technische Universiteit Delft, Oct. 14, 2021.

[82] "Navigation Timing Level 2," The World Wide Web Consortium, Mar. 10, 2022.

[83] "Top 10 million domains based on Open PageRank data," DomCop. (Feb. 13, 2022), [Online]. Available: `https://www.domcop.com/top-10-million-domains` (visited on 05/03/2022).

[84] R. P. Foundation, "Raspberry Pi 4 Tech Specs," Raspberry Pi Foundation, 2022.

[85] O. Project, *OpenWrt*, version 21.02.1, Oct. 25, 2021.

[86] J. Sukumaran, *Syrupy*, version 4062918, Apr. 16, 2020.

[87] tcpdump Team, *Tcpdump*, version 4.99.1, Jun. 9, 2021.

[88] W. Team, *Wireshark*, version 3.6.3, Mar. 23, 2022.

[89] J. Ball, B. Schneier, and G. Greenwald, "NSA and GCHQ target Tor network that protects anonymity of web users," *The Guardian*, Oct. 4, 2013.

[90] "Tor Stinks," American Civil Liberties Union. (Jun. 2012), [Online]. Available: `https://www.aclu.org/sites/default/files/assets/tor_stinks.pdf`.

[91] L. Basyoni, N. Fetais, A. Erbad, A. Mohamed, and M. Guizani, "Traffic Analysis Attacks on Tor: A Survey," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, Doha, Qatar: IEEE, Feb. 2020, pp. 183–188, ISBN: 978-1-72814-821-2. DOI: `10.1109/ICIoT48696.2020.9089497`.

[92] C. Cimpanu. "Here's how to enable DoH in each browser, ISPs be damned," ZDNet. (Feb. 26, 2022), [Online]. Available: `https://www.zdnet.com/article/dns-over-https-will-eventually-roll-out-in-all-major-browsers-despite-isp-opposition/`.

[93] "DoT with Dnsmasq and Stubby," OpenWrt. (Mar. 23, 2022), [Online]. Available: `https://openwrt.org/docs/guide-user/services/dns/dot_dnsmasq_stubby`.

[94] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach," in *Proceedings of the Internet Measurement Conference*, Amsterdam Netherlands: ACM, Oct. 21, 2019, pp. 267–279, ISBN: 978-1-4503-6948-0. DOI: `10.1145/3355369.3355577`.

[95] "General-purpose machine family," Google Cloud. (May 30, 2022), [Online]. Available: `https://cloud.google.com/compute/docs/general-purpose-machines#e2-shared-core`.

[96] S. O'Dea, "Average mobile and fixed broadband download and upload speeds worldwide as of May 2021," Speedtest, Jul. 13, 2021.

[97] D. Lakens, "Equivalence Tests: A Practical Primer for t Tests, Correlations, and Meta-Analyses," *Social Psychological and Personality Science*, vol. 8, no. 4, pp. 355–362, May 2017, ISSN: 1948-5506, 1948-5514. DOI: `10.1177/1948550617697177`.

[98] I. Arapakis, X. Bai, and B. B. Cambazoglu, "Impact of response latency on user behavior in web search," in *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, Gold Coast Queensland Australia: ACM, Jul. 3, 2014, pp. 103–112, ISBN: 978-1-4503-2257-7. DOI: 10.1145/2600428.2609627.

[99] "Relay Requirements." (2022), [Online]. Available: https://community.torproject.org/relay/relays-requirements/.

[100] P. Winter *et al.*, "Spoiled Onions: Exposing Malicious Tor Exit Relays," in *Privacy Enhancing Technologies*, E. De Cristofaro and S. J. Murdoch, Eds., vol. 8555, Cham: Springer International Publishing, 2014, pp. 304–331, ISBN: 978-3-319-08505-0 978-3-319-08506-7. DOI: 10.1007/978-3-319-08506-7_16.

[101] "Homepage," NTP Pool Project. (May 18, 2022), [Online]. Available: https://www.ntppool.org/en/.

[102] "Nbdd0121/nfq-rs." in collab. with G. Guo, GitHub. (Jan. 26, 2022), [Online]. Available: https://github.com/nbdd0121/nfq-rs.

[103] "Rustn00b/nfq-rs." in collab. with rustn00b and G. Guo. (Jan. 30, 2022), [Online]. Available: https://github.com/rustn00b/nfq-rs.

[104] "Implemented socket unbind for async nfqueues #1." in collab. with A. Vandenbussche, rustn00b, and G. Guo, GitHub. (Apr. 24, 2022), [Online]. Available: https://github.com/rustn00b/nfq-rs/pull/1.