If you cite this paper, please use the PETS reference: Anna Maria Mandalari, Daniel J. Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, David Choffnes. Blocking without Breaking: Identification and Mitigation of Non-Essential IoT Traffic. In Privacy Enhancing Technologies Symposium (PETS) 2021.

Anna Maria Mandalari*, Daniel J. Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David Choffnes

Blocking without Breaking: Identification and Mitigation of Non-Essential IoT Traffic

arXiv:2105.05162v1 [cs.NI] 11 May 2021

Abstract:
Despite the prevalence of Internet of Things (IoT) devices, there is little information about the purpose and risks of the Internet traffic these devices generate, and consumers have limited options for controlling those risks. A key open question is whether one can mitigate these risks by automatically blocking some of the Internet connections from IoT devices, without rendering the devices inoperable.

In this paper, we address this question by developing a rigorous methodology that relies on automated IoT-device experimentation to reveal which network connections (and the information they expose) are essential, and which are not. We further develop strategies to *automatically* classify network traffic destinations as either required (*i.e.*, their traffic is *essential* for devices to work properly) or not, hence allowing firewall rules to block traffic sent to non-required destinations without breaking the functionality of the device. We find that indeed 16 among the 31 devices we tested have at least one blockable non-required destination, with the maximum number of blockable destinations for a device being 11. We further analyze the destination of network traffic and find that all third parties observed in our experiments are blockable, while first and support parties are neither uniformly required or non-required. Finally, we demonstrate the limitations of existing blocklists on IoT traffic, propose a set of guidelines for automatically limiting non-essential IoT traffic, and we develop a prototype system that implements these guidelines.

Keywords: IoT, privacy, firewall, filtering, blocking

***Corresponding Author: Anna Maria Mandalari:** Imperial College London, E-mail: anna-maria.mandalari@imperial.ac.uk
Daniel J. Dubois: Northeastern University, E-mail: d.dubois@northeastern.edu
Roman Kolcun: Imperial College London, E-mail: roman.kolcun@imperial.ac.uk
Muhammad Talha Paracha: Northeastern University, E-mail: paracha.m@husky.neu.edu
Hamed Haddadi: Imperial College London, E-mail: h.haddadi@imperial.ac.uk
David Choffnes: Northeastern University, E-mail: choffnes@ccs.neu.edu

1 Introduction

Consumer Internet of Things (IoT) devices (*e.g.*, smart TVs, speakers, surveillance cameras, appliances, *etc.*) are rapidly gaining presence in homes, offices, and public spaces [1]. While these devices often come with convenient services, they open the door to numerous privacy and security risks [2–4]. These devices often expose information to a large number of destinations [2, 5], including third party advertising and tracking services.

A fundamental approach for mitigating such risks would be to automatically block any connections that are not essential for the essential functionality of a device. For this approach to work, we need a systematic approach to identify and block traffic that is not essential for a device to work, with little-to-no user configuration, and without causing any device malfunction. Unfortunately, existing solutions are not sufficient for this purpose. Approaches such as Pi-hole [6] block DNS requests for advertising and tracking services using blocklists, but destinations on those blocklists are often based on web tracking, thus missing blockable destinations for our IoT devices. While standard IoT security solutions might be able to arbitrarily block connections, they are unable to determine the consequences of any blocking on device functionality.

In this paper, we design and validate a methodology for automatically determining the necessity of the destinations contacted by an IoT device for the correct execution of its primary functionality. The intuition behind our approach is that IoT device functions can be invoked using interfaces amenable to automation (*e.g.*, using a voice synthesizer or scripting companion app interactions). Further, one can automatically determine whether the execution of such functions has been successful, by observing the IoT device signals (*e.g.*, the screenshot of a companion app, or the network traffic patterns generated). Based on these intuitions, our methodology can be used for a target device and se-

lected functionality, to build a list of non-required destinations that can be automatically blocked, without breaking such functionality. Similarly, we can build a list of required destinations that can be automatically allowed (to preserve functionality), while blocking the rest of the traffic.

The key building blocks of our system are: automatically interacting with devices to exercise their functions; systematically blocking one or more observed connections; and automatically determining whether each interaction was successful after blocking a connection. We use an extensive testbed and large number of trials to find that 28 out of 31 devices (across five categories) are amenable to fully automated blocking analysis.

We then turn to analyzing the blockable destinations. We find that 16 of our 31 devices contact at least one non-required destination (and as many as 11 destinations) to execute their main functions. Across all devices, we find that 62 non-required destinations are contacted. We further analyze the destinations of network traffic and find that all third parties observed in our experiments are blockable, while first and support parties are neither uniformly required or non-required. Additionally, we show that uniformly blocking all 62 non-required destinations for all devices can lead to breaking device functionality: three devices exhibit a required destination that is a non-required destination for a different device. We find that non-required/required destinations do not change over time for all the devices, and that, for 90.32% of the devices, such destinations tend to be the same across different device functions. Finally, we propose a set of guidelines for automatically limiting non-essential IoT traffic, and we develop a prototype system that implements these guidelines.
To summarize, our key contributions include:

– A methodology for determining required and non-required destinations by automatically executing IoT device functions and determining the execution outcome while blocking selected destinations.
– An analysis of required/non-required destinations contacted by a diverse set of consumer IoT devices.
– The design of a testing system (*IoTrigger*) and a blocking system (*IoTrimmer*) that use our method for building the required and non-required destinations list (*IoTrim* list), and use it to block non-essential traffic. *IoTrigger*, *IoTrimmer*, and the *IoTrim* list are publicly available at `http://iotrim.net/`.

2 Assumptions and Goals

In this section, we set the assumptions, the definitions, the goals, and the non-goals for this work.

2.1 Assumptions and Definitions

Threat Model. We assume a system composed of three entities: (*i*) an off-the-shelf *IoT device*, with the ability to communicate to any destinations over the Internet; (*ii*) the *network traffic destinations*, which include any Internet destinations that the IoT device creates a connection to; and (*iii*) the *user*, who has access to the IoT device functionality. In our threat model, we consider an IoT device and the network traffic destinations as potential adversaries, since the IoT device can potentially expose information about its users (the victim) to any destination. Since most IoT traffic is encrypted or encoded [2] and the vast majority of IoT systems are closed, it is infeasible to perfectly infer what information is exposed through network connections using blackbox techniques. Instead, we question whether a given connection is necessary for supporting a device's function (*e.g.*, ringing a doorbell), and if not, we consider the connection to be a threat for unnecessary data exposure, in line with GDPR data minimization [7] and purpose limitation principles [8]. Hence, blocking such traffic can potentially reduce information exposure for users without affecting the device functionality.
Essential Traffic Definition. We define *essential traffic*, with respect to a given IoT device function, the network traffic that is essential to fulfill such function.
Required Destination Definition. We define as *required* all the network traffic destinations that are contacted as part of essential traffic.

2.2 Goals

Fig. 1 illustrates the three main goals of this work. More specifically, we want to answer the following questions.
RQ1. How can we automatically identify non-essential IoT traffic? We seek to understand which destinations are not required for device functionality, so that we can block them to mitigate their potential risks. To address this, we propose a methodology for automatically detecting whether a network traffic destination is required or not for a given function of an IoT device (*e.g.*, in the case of a smart bulb and its switch on/off function, destinations that are not necessary for switch-

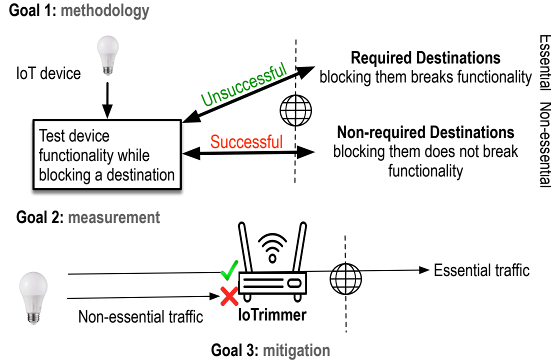

Fig. 1. The three main goals of this paper.

ing the light on/off). The presence of a non-required destination means that all the traffic sent to such destination is non-essential, and therefore an avoidable case of information exposure.

RQ2. What is the nature of non-essential IoT traffic? Armed with a measurement methodology to detect non-essential traffic, we apply it to identify and study the non-essential traffic produced by our set of 31 popular IoT devices spanning five categories. As part of this research question, we are interested in the type of destinations contacted (*e.g.*, if they belong to the device vendor), if any required destinations for a device are non-required for another device, if different devices have non-required destinations in common, if different device functions have different required and non-required destinations, and if any of those destinations are present in existing blocklists. Characterizing non-essential traffic for existing devices is important to find correlations that can assist in detecting such traffic in real-time for future devices, without relying on the methodology proposed to answer RQ1.

RQ3. Can we automatically mitigate non-essential IoT traffic? The knowledge of what destinations can be blocked for every device allows us to make automatic run-time decisions on what traffic to allow or not in a typical IoT deployment. To answer this research question, we first determine the feasibility of a blocking solution by analyzing how much essential and non-essential traffic changes over time, as a way to assess the risk of allowing non-essential traffic or breaking the device functionality. Then, we describe our prototype software for automatically generating destination lists, and to transparently block as much non-essential IoT traffic as possible, thus reducing information exposure without affecting devices' main functions.

2.3 Non-Goals

In this work, we do not consider the following as goals, and leave them for future work.

No control over how an IoT device works internally. We consider the IoT devices as off-the-shelf consumer items that provide a finite set of functions and that communicate over the Internet. For these devices, we have no control over their internal functions, but we can still interact with them using their user interface and we can measure their network activity.

No content interception and inference. While we consider the visibility of the content out of scope, we are able to see the destinations of such traffic. We make this assumption because the vast majority of the traffic is encrypted and the devices are assumed as black-boxes, where there is no possibility to install custom self-signed certificates to use man-in-the-middle techniques to intercept encrypted traffic. We also do not try to infer the content of encrypted flows as means to measure privacy exposure since this has been studied in previous works, but we still use traffic patterns to test if blocking/allowing a destination prevents a given function from working.

We do not test *all* functions. IoT devices typically offer several functions; however, for this work, we apply our methodology by selecting only a subset of them for every IoT device under test so that we can have more coverage by devices rather than by functionality. We consider this limitation reasonable since our analysis of multiple functions in §5.6 shows that the vast majority of the devices we tested use the same destinations for different functions. In this work, we assert that a given function is either executed correctly or not. We do not consider the case of a function partially working.

One trigger per function. Some IoT device functions can be triggered in several ways (*e.g.*, through a companion app, IFTTT [9], Samsung hub [10], *etc.*). In this work we only focus on one trigger per function.

3 Methodology

We answer our first research question by proposing a methodology to detect non-essential IoT network traffic by classifying destinations as either required or not.

3.1 Testbed

Our classification method relies on a testbed that provides a controlled environment for testing IoT devices. Our testbed consists of: (*i*) a *router* that offers IP connectivity to the IoT devices under test, and the ability to capture and control network traffic for each device; and (*ii*) a set of *support scripts* to turn on and off an IoT device, trigger a function, and determine whether a function is successfully executed.

3.1.1 Router

The router is configured using a standard NAT setup, with one network interface connected to the Internet and another one bridged to the IoT devices under test. As part of the router's DHCP support, IoT devices are assigned a DNS server that we control (and that serves as a proxy for the ISP's DNS server). Together with traffic redirection rules and a *dnsmasq* instance, our testbed intercepts all DNS requests, even if an IoT device uses a DNS server other than the DHCP-advertised one (*e.g.*, by using a public resolver). We collect all network traffic traversing the testbed using *tcpdump*. The router can block IoT traffic destinations by IP address (including IP masks) and by altering DNS responses whose request matches a given pattern. When a DNS destination is blocked, it is resolved as localhost (127.0.0.1).

3.1.2 Support Scripts

We use support scripts to power on/off the devices, to *trigger* their functions, and to *probe* them to find out if a function execution is successful or not. The invocation of these scripts is fully automated, as part of our automated experiments methodology. Please note that while every step of every experiment (including the invocation of support scripts) is fully automated to allow our approach to scale, the creation of support scripts requires programming effort, which is a manual process that is device-dependent and functionality-dependent. However, once support scripts are written, they can be reused across experiments and only need to be rewritten after major changes in the device interaction interface.

Power on/off Scripts. The IoT devices are plugged into programmable smart plugs, and we use scripts to turn these smart plugs on and off so that we can reset the IoT devices by power-cycling them after every test.

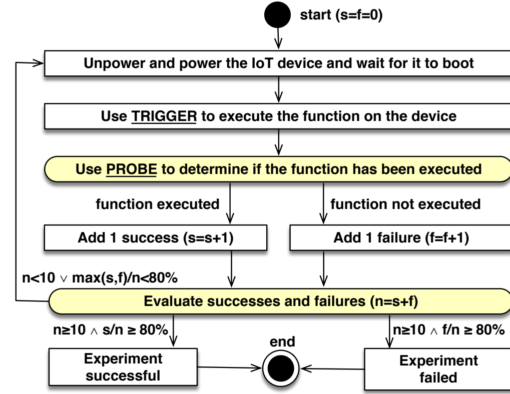

Fig. 2. Functionality experiment. The algorithm iterates the execution of a function at least 10 times: s, f, n are counters for successful, failed, and total iterations. When an 80% consensus is achieved the algorithm terminates with a success or fail result.

Trigger Scripts. To scale our analysis to many devices, we automate interactions with IoT devices by triggering their functions programmatically. We call the different automation strategies *device triggers*, which are function and device-specific. An example of trigger for turning on a smart bulb is to programmatically provide input to its companion app in such a way to turn it on.

Probe Scripts. To verify that a trigger correctly executes a function, our methodology relies on some additional scripts, called *device probes*, which programmatically query the status of a device, or analyze any signals it produces. The scripts then compare this information against ground truth, to check whether the execution of a function was successful or not. Probe scripts are also device-specific. An example of probe to determine if a light bulb is on is to retrieve the screenshot of its companion app and compare it to a previously retrieved screenshot where we know that the bulb was on.

3.2 Functionality Experiments

The basic unit of our methodology is the *functionality experiment* (see Fig. 2). We define it as the fully automated process to verify if a function of an IoT device can be executed or not. During our preliminary experiments we found that several IoT devices are less than 100% reliable in terms of correctly executing their functionality in normal operating conditions (see our evaluation in §4.4). The reasons are various, and span from random reboots, to the temporary disruptions in connectivity to cloud services. To cope with such events that prevent 100% accurate probe scripts, we choose to use probe scripts as long as they have at least 80% accuracy (*i.e.*, they can be incorrect at most 20% of the time due to

whatever reason, including the device not behaving correctly). To ensure that we can reliably use information from probe scripts that may be inaccurate, we test the function *multiple times* (at least 10), and we consider the result correct if a strong majority (at least 80%) of the test results are consistent. This ensures that the whole functionality experiment has a negligible probability of an incorrect result: while the odds of any one test failing can be significant, the odds that all of the multiple tests failing is substantially lower.

Specifically, each functionality experiment iterates at least 10 times the following three steps.

Step 1. We power on the device using the testbed's power script (to turn on the smart plug powering the device) and then wait for it to finish booting. The wait time is determined empirically using probe scripts, and we have found that a two-minute delay is enough for all the devices we tested.

Step 2. We trigger the function of the experiment by invoking the proper device trigger.

Step 3. We use the device probe to verify that the function has been actually executed: if it is, we report the iteration as successful, otherwise as failure. If, after all the iterations, at least 80% of them is successful, we report the experiment as successful; if at least 80% of the iterations fails, we report the experiment as a failure.

If 80% of the iterations is neither a success or a failure, we run an additional iteration and evaluate this test again. The algorithm keeps performing iterations until the 80% threshold is reached. If the threshold is never reached the algorithm would not terminate (*i.e.*, it keeps iterating with no success or fail result); however since we assume probes that are at least 80% accurate (as it will be shown in §4.4), having a threshold of 80% ensures that we achieve convergence in the long run.

Given that the probes have a maximum 20% probability to produce an incorrect result during an iteration, there is a chance that a functionality experiment terminates with an incorrect result; however, such a chance is negligible (less than 0.0078%[1]) since the incorrect result must happen in at least 80% of the iterations. During our experiments the algorithm always converges, meaning that the probes fulfilled their accuracy requirement.

3.3 Building the List of Destinations

To determine what destinations are required or not for a given IoT function, we need to first obtain the list of destinations during a preliminary destination-observing experiment, which consists of running a functionality experiment (which is composed of minimum 10 iterations of function invocations) without blocking any network traffic, and collecting the list of destinations contacted by the device. All destination-observing experiments under normal circumstances are successful since we do not block any traffic. We primarily identify all contacted destinations by hostname rather than by IP address. To do this, for each IP destination, we look at all the DNS traffic for the device to find the DNS hostname that resolved to the IP address. If the hostname cannot be determined using this method, we simply use the IP address as the destination. We exclude from this process DNS (TCP/UDP port 53) and NTP (network time protocol, UDP port 123) destinations. We always allow these protocols since they are needed to resolve hostnames (DNS) and synchronize device clocks (NTP) for checking TLS certificate validity.

Many cloud services for IoT devices use replicated servers that provide the same functionality, and they sometimes use different (but similar) DNS names and IP addresses for each replica. To facilitate analysis and streamline blocklists, we *group* destinations that are *ephemeral*, *i.e.*, they appear in less than 80% of the iterations of the destination-observing experiments. For example, if `a.zz.com` is an ephemeral destination contacted in half of the iterations, and `b.zz.com` is contacted in the other half, they are both replaced by a single destination group `*.zz.com`, which appears in 100% of the iterations. All ephemeral destinations encountered in our experiments were successfully replaced with second-level wildcard domains. For more details on this process, see Appendix A.

3.4 Determining Required Destinations

The algorithm for creating the list of required and non-required destinations is reported in Fig 3.

Step 1. *Building the list of destinations.* See §3.3.

Step 2. *Marking each contacted destination.* Iteratively test all contacted destinations by running a functionality experiment for each of them. In each iteration, the considered destination is blocked. If the experiment succeeds, such destination is marked as non-required, and will stay blocked. If the experiment fails, such destination is marked as required, and will be unblocked. This

---

[1] This upper bound for the probability of an incorrect result for our algorithm has been calculated by considering that the number of incorrect results of a probe (over n iterations) follows a binomial distribution with parameters n and $p = 0.2$.

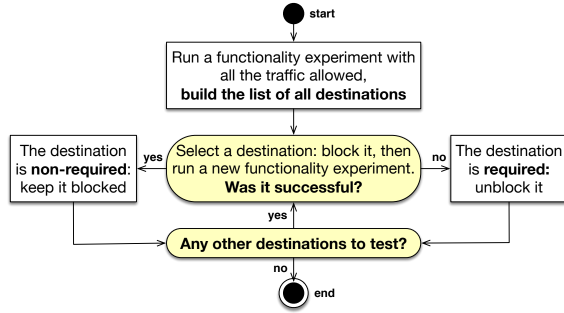

Fig. 3. Methodology for detecting required destinations.

process repeats until all destinations collected during the first step are classified.

4 IoT Devices

In this section we describe the IoT devices we use in our experiments, and all specialized device triggers and probes we use to apply the methodology described in §3.

4.1 List of Devices and Tested Functions

The devices we consider are consumer IoT devices typically deployed in a smart home. We have chosen devices under these categories (see the first column of Table 1):

- *Camera*. Devices equipped with a camera sensor, such us smart camera systems and smart doorbells. The function we test is to watch a live stream.
- *Home automation and appliances*. Devices that offer home automation capabilities such as smart lights, kitchen appliances. The function we test is switching the device on and off.
- *Smart hubs*. Devices coordinating other non-IP IoT devices (*i.e.*, Zigbee). The function we test is switching the devices on and off.
- *Smart speakers*. Speakers that offer a voice assistant. We test responses to the voice command, "What is the capital of Italy?"
- *Video*. Devices designed to stream video on a TV. We test streaming from YouTube.

The criteria we use for choosing the function to test are: (*i*) it must be a function that is characteristic of the device category; (*ii*) it must be intended for user-initiated interactions and not initiated by the device itself; (*iii*) it must be amenable to triggers and probes.

To better represent how IoT devices behave in the wild, we try to keep their default configuration and privacy settings unaltered and we do not perform user-initiated firmware updates. Devices are still allowed to perform automated firmware updates when such a feature is enabled in the default configuration.

Device	Trigger	Probe	Success (May)	Success (July)	Success (October)
Camera (Watching live)					
Blink	App	Screen	93.3%	96.7%	100%
Bosiwo	App	Screen	100%	100%	100%
iCSee	App	~~Screen~~	80%	76.7%	70%
Reolink	App	~~Screen~~	70%	70%	60%
Wansview	App	Screen	90%	93.3%	100%
Yi	App	Screen	100%	100%	100%
Home-automation (Switching on/off)					
App Kettle	App	Screen	100%	100%	100%
Honeywell thermostat	App	Screen	100%	100%	100%
Magichome	App	Screen	100%	100%	100%
Meross dooropener	App	Screen	100%	100%	100%
Nest thermostat	App	Screen	100%	100%	100%
Netatmo weather	App	Screen	100%	100%	100%
Smarter coffee machine	App	Screen	100%	100%	100%
Smartlife bulb	App	Screen	100%	100%	100%
Smartlife remote control	App	Screen	100%	100%	100%
Sousvide cooker	App	Screen	100%	100%	100%
Switchbot	App	Screen	100%	100%	100%
TP-Link bulb	App	Screen	100%	100%	100%
TP-Link plug	App	Screen	100%	100%	100%
Wemo plug	App	Screen	100%	100%	100%
Xiaomi rice-cooker	App	~~Screen~~	46.7%	53.3%	40%
Smart-hub (Switching on/off)					
Insteon	App	Screen	100%	100%	100%
Lightify	App	Screen	100%	100%	100%
Philips	App	Screen	100%	100%	100%
Samsung	App	Screen	96.7%	100%	100%
Sengled	App	Screen	100%	100%	100%
Smart-Speaker (Asking questions)					
Allure	Voice	Traffic	100%	100%	100%
Echo Dot	Voice	Traffic	100%	100%	100%
Google Home	Voice	Traffic	100%	100%	100%
Video (Watching YouTube)					
Fire TV	App	Traffic	100%	100%	100%
Roku TV	App	Traffic	100%	100%	100%

Table 1. List of our devices by category. For each of them: triggering and probing strategy we used, and probe success rate evaluation in three different point in time (May, July, and October 2020). Crossed out probing strategies are the ones we could not use programmatically due to insufficient success rate (see §4.4).

4.2 Specialized Device Triggers

As discussed in §3.1.2, we use device-dependent trigger scripts to execute functionality. The triggering strategies we use for each device are reported in the second column of Table 1 and described as follows.

Companion app. This triggering strategy is possible for IoT devices that can be controlled via a companion app compatible with Android. We install this app on an Android phone that is *not on the same LAN* as the IoT device (to force the communication to happen over the Internet rather than directly), and then trigger each function by emulating user interactions programmatically using the Android Debug Bridge.

Voice assistant. This strategy is used for smart speakers. We use the Google voice synthesizer connected to

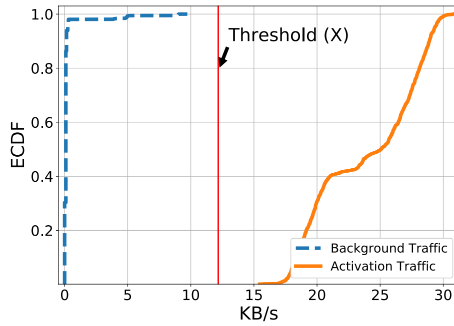

Fig. 4. ECDF of the data peaks $p_{i,j}$ over 135 experiments for the Echo Dot: the plot shows a clear distinction between the data peaks when its main function is executed (activation data peaks) and when it is not (background data peaks).

Device (i)	Destination	B_i^{max} [KB/s]	A_i^{min} [KB/s]	Threshold (X_i) [KB/s]
Allure	bob-dispatch-* *.amazon.com	6.763	13.342	10.052
Echo Dot (3rd gen.)	bob-dispatch-* *.amazon.com	8.889	15.455	12.172
Fire TV	youtube.com	0	109.38	54.69
Google Home	google.com	41.69	50.483	46.086
Roku TV	youtube.com	0	140.364	70.182

Table 2. Network traffic probe thresholds for data peaks. B_i^{max} is the maximum peak in background traffic (*i.e.*, no functionality execution), A_i^{min} is the minimum peak in activation traffic (*i.e.*, with functionality execution), X_i is the data peak threshold, *i.e.*, the minimum peak required for detecting device activation.

a set of regular speakers (placed next to the smart speaker) to programmatically issue voice commands.

4.3 Specialized Device Probes

We use several device-dependent strategies also for probing the devices; the probing strategies we use for each device are reported in the third column of Table 1 and described as follows.

Companion app screenshot. For the majority of IoT devices, a companion app is available as a method to obtain their state. For this method, we use a preliminary experiment during which we take sample screenshots of the app for each device, and we use this as the ground truth for the correct state after executing a function. For subsequent experiments, we take screen shots after using a trigger, and compute the similarity of each screenshot to the ground truth to infer the state of the device.

To quantify this similarity, we simply check how many pixels differ in the two screenshots by more than a particular threshold. We use the same parameters for all companion apps, which we tuned through an analysis on a few sample screenshots.

Network traffic patterns. This probe analyzes the patterns of the network traffic generated by an IoT device in two situations: when the function has been executed (*activation traffic*) and when the function was not executed (*background traffic*). During preliminary experiments we observed that when the main function is executed for some devices (typically streaming devices such as smart speakers), they significantly increase the amount of data transmitted to certain destinations compared to when the function is not executed (see Echo Dot example in Fig. 4). Based on this observation, we automatically detect traffic bursts corresponding to the traffic pattern for the main function of a device.

Specifically, for device i and an experiment j, we consider the data peak $p_{i,j}$, defined as the maximum amount of traffic sent by the device to such destinations among all 20-second window samples over the full duration of the experiment. From a series of preliminary experiments where we know as ground truth that the main function of device i is executed and not executed, we calculate the constants A_i^{min} and B_i^{max}, where A_i^{min} (minimum activation peak) is the minimum data peak $p_{i,j}$ over all experiments j with execution, and B_i^{max} (maximum background peak) is the maximum data peak $d_{i,j}$ over all experiments j *without* execution. We then define the data peak activation threshold X_i, as the average between A_i^{min} and B_i^{max}: any data peak that is larger than this threshold signals the presence of activation traffic.

The probe then uses X_i to determine whether device i had its function executed or not during a new experiment k: if $p_{i,k} > X_i$ (*i.e.*, the experiment has a data peak that is larger than the peak activation threshold), the probe returns success for k, otherwise it returns failure. Table 2 shows the destinations and parameters for the network traffic probes, calculated over a minimum of 135 preliminary experiments for each device.

4.4 Probes Evaluation

Probes Evaluation Method. Our method for classifying required destinations relies on probes that are at least 80% accurate on average. To identify whether this property holds, we run 70 *probe evaluation experiments* per device in three points in time (10 in May, 30 in July, and 30 in October 2020). Each probe evaluation experiment is a set of functionality experiments run in the following three situations: (i) with all the destinations allowed, where we know *a priori* that the function execution succeeds (*i.e.*, testing to see if the probe cor-

rectly detects successful experiments); (*ii*) with all the destinations blocked, where we know *a priori* that the function fails (to ensure that the probe detects experiment failures); (*iii*) with all the destinations allowed, but without executing the trigger, to test whether the probe detects that the function is not executed.

Once the experiments are complete, we calculate the success (failure) rate of the probe, defined as the number of correct (incorrect) probe results over the total number of experiments for that probe. We consider the minimum between the success rate and the failure rate as a conservative metric to measure the accuracy of a probe, and use it as the expected probability to provide a correct result.

Detecting successes. The results of our probes evaluation method for detecting the success of a function execution are reported in the last three columns of Table 1, where each column represents the evaluation at a different point in time. For 28 of 31 devices, our probes correctly and consistently recognize the execution of a function in at least 80% of the cases, which satisfies the requirement of our method for classifying destinations. For the remaining three devices (iCSee Doorbell, Reolink Camera, and Xiaomi Rice Cooker), we could not find probes that are at least 80% accurate during all the three points in times.

Detecting failures. We find that our probes correctly recognize a function execution failure both for cases where all traffic is blocked, and when the function is (intentionally) not triggered. As a result, it is very unlikely that a probe will report as successful an experiment where the execution of a function fails, since this kind of error never happened during our 4,340 probe evaluation experiments.

Dealing with inaccurate probes. For the three devices whose probes are not accurate enough, we cannot use our fully automated analysis approach because we do not have an automated way to detect if a trigger is successful. To still include them in our study, we probe their status manually, while keeping all the remaining steps of our approach automated.

5 Identifying Non-essential Traffic

We answer our second research question by applying the methodology described in §3 to identify and characterize non-essential traffic produced by our IoT devices (§4).

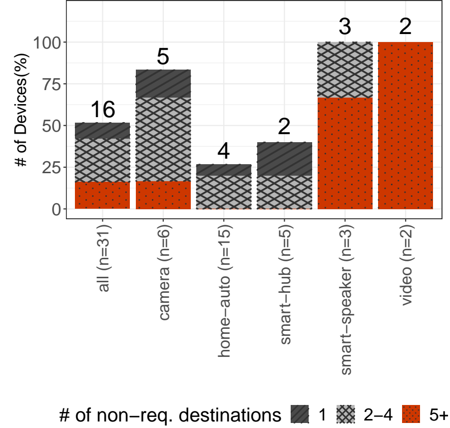

Fig. 5. Percentage of devices with at least one non-required destination. Sub-bars show how the number of non-required destinations is distributed among the devices for each category.

5.1 Impact of Device Category

We now characterize the destinations of the traffic of each device in terms of traffic sent to required destinations (essential traffic) and traffic sent to non-required destinations (non-essential traffic). Fig. 5 shows that 52% of the IoT devices we tested produce non-essential traffic, with 25% and 16% of them contacting respectively at least 2 and 5 non-required destinations. The figure shows that all devices tested in the smart speakers and video categories produce non-essential traffic.

We now consider these devices and non-required destinations in more detail (Table 3). Of these devices, the Amazon Fire TV (11 destinations) and the Roku TV (8 destinations) contact the largest number of non-required destinations. Notably, devices in the camera category also contact up to six non-required destinations—a surprising result since such devices do not have third-party apps or UIs that can include non-required traffic such as advertising. In most of these notable cases the number of non-required destinations tend to be larger than the number of required ones. Finally, we observe several non-required destinations also on simple devices, such as the TP-Link plug, which sends non-essential traffic to three non-required destinations while only having a single required destination.

On the other hand, 15 of 31 devices contact required destinations only, 22 in total. Table 8 in Appendix B lists all the required and non-required destinations.

Takeaways. Our results show that traffic to non-required destinations is present across all categories of IoT devices, and these devices often contact more non-required destinations than required ones. We further find that devices with a richer set of functions—such

	Device	Dest. #	Req. #	Non-Req. #	List of Non-Required Destinations
Camera	Bosiwo	4	2	2	*54.157.82.107*, 210.72.145.44
	iCSee	6	2	4	*47.52.222.172, 47.52.32.118, api.gdxp.com, oss-us-west-1.aliyuncs.com*
	Reolink	2	1	1	pushx.reolink.com
	Wansview	9	3	6	*159.65.95.225, 3.122.229.130 , ajcloud.net, htpdate.ajcloud.net, sdc-isc.ajcloud.net, *.backblaze.com*
	Yi	5	3	2	api.eu.xiaoyi.com, log.eu.xiaoyi.com
Home-auto	Nest thermostat	3	2	1	frontdoor.nest.com
	TP-Link bulb	4	1	3	euw1-api.tplinkra.com, n-deventry.tplinkcloud.com, use1-api.tplinkra.com
	TP-Link plug	4	1	3	euw1-api.tplinkra.com, n-deventry.tplinkcloud.com, use1-api.tplinkra.com
	Xiaomi rice-cooker	7	3	4	*183.84.5.203, 58.83.160.36, 123.125.102.215, 110.43.0.83*
Hub	Philips	4	2	2	diagnostics.meethue.com, ecdinterface.philips.com
	Samsung	3	2	1	fw-update2.smartthings.com
Speaker	Allure	3	1	2	*api.amazon.com, d1enchupjctwud.cloudfront.net*
	Echo Dot	10	3	7	arcus-uswest.amazon.com, *.cloudfront.net, device-metrics-us.amazon.com, dp-gw.amazon.com, fireoscaptiveportal.com, *prod.amcs-tachyon.com*, s3-1-w.amazonaws.com
	Google Home	9	4	5	youtube-ui.l.google.com, clientservices.googleapis.com, fcm.googleapis.com, *.googlevideo.com, storage.googleapis.com
Video	Fire TV	14	3	11	aax-eu.amazon-adsystem.com, arcus-uswest.amazon.com, bob-dispatch-prod-eu.amazon.com, *.cloudfront.net, device-metrics-us.amazon.com, api.amazon.com, ktpx-eu.amazon.com, api-global.eu-west-1.prodaa.netflix.com, mas-ext-eu.amazon.com, mas-sdk.amazon.com, msh.amazon.com
	Roku TV	10	2	8	api-global.eu-west-1.prodaa.netflix.com, configsvc.cs.roku.com, cooper.logs.roku.com, customerevents.eu-west-1.prodaa.netflix.com, ichnaea.eu-west-1.prodaa.netflix.com, partnerad.l.doubleclick.net, scribe.logs.roku.com, uiboot.eu-west-1.prodaa.netflix.com
	Other devices (15)	22	22	0	
Total	*31*	*119*	*57*	*62*	

Table 3. Non-required destinations. We report, for each device (having at least one non-required destination), the total number of destinations, the number of required destinations, the number of non-required destinations, and the list of non-required destinations. Colors identify the destination party type (see §5.2): first party, *support party*, and third party. For a version of this table, which includes all our IoT devices and the list of required destinations as well, see Table 8 in Appendix B.

as smart speakers and video devices—are more likely to have such non-required traffic, followed by smart cameras. For the case of video devices, some of the non-required destinations are advertisers while others are related to video recommendations for pre-installed apps, a topic we discuss in §7.2. Note that these destinations are not from background app activity, since it is disallowed for the Roku TV [4, 11], and we disabled background app activity on the Fire TV [12]. Regarding the cameras and other simpler devices, it is unclear why they produce non-essential traffic without the internal details of the devices and their software.

5.2 Impact of Destination Party Type

In this section, we determine trends relating to whether a destination's party type (first party, support, third) are indicative of whether the destination is required or not for device functionality. We use the same party type definitions and classification approach proposed in [2] and we consider any advertising domain as third-party: a *First party* is a destination related to the device manufacturer or a related company responsible for fulfilling device functionality; a *Support party* is any company providing outsourced computing resources such as CDN and cloud providers, which is not also a first party; a *Third party* is a destination that is not a First party or a Support party. Third-party companies include advertising and analytics companies. Table 3 uses text decoration in the rightmost column to indicate the party type for each observed device-destination pair.

To show aggregate findings, we group results by category in Fig. 6, with the left figure analyzing the number of destinations and the right figure analyzing traffic volumes. We begin with the left plot, which plots how many required and non-required destinations are contacted for each destination party type and device category. We find that third-party destinations are *never* required, meaning that all their traffic is non-essential, while first and support parties are sometimes non-required and sometimes required. Overall, there are slightly more non-required first- and support-party destinations among the majority of devices and categories.

The right plot in Fig. 6 shows the average amount of data (payload only, no headers) that is sent to required and non-required destinations during successful function invocations. We observe that the vast majority of the data is sent to required destinations (*i.e.*, essential traffic) that are either first or support parties, while the volume of non-essential traffic is relatively small (1,931 bytes in total), and a mix of all the three party types.

For the devices we tested, non-essential traffic sent to third parties only occurs for the camera and video category, while all the non-essential traffic produced by

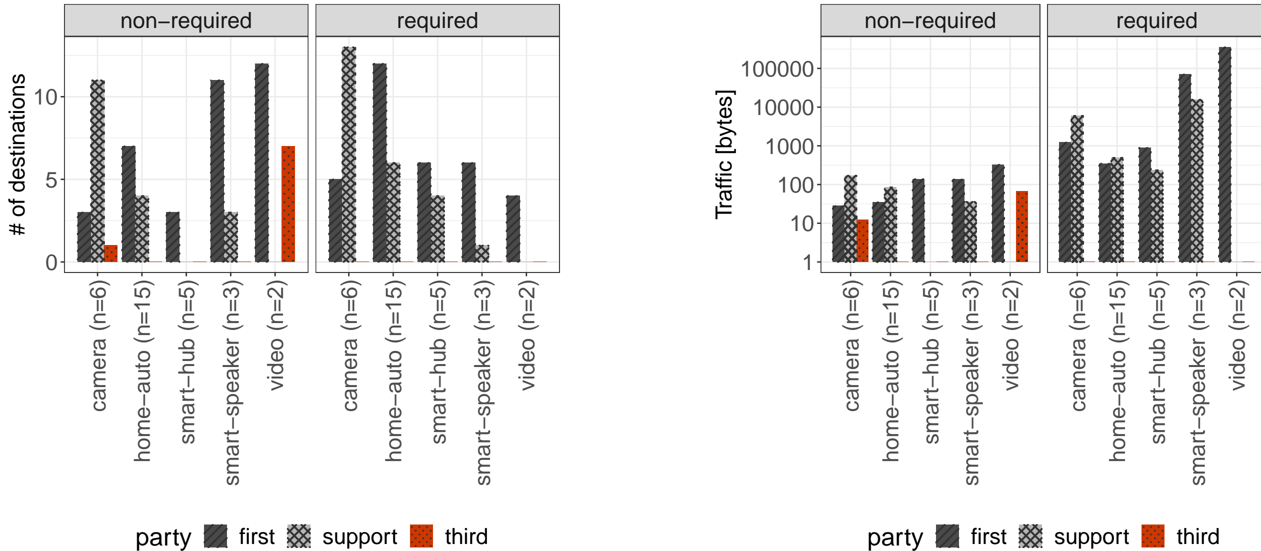

Fig. 6. Number (left) and total traffic (right) of required/non-required destinations. The total traffic is the average payload (without headers) produced by the devices during successful function invocations.

IoT devices in the smart-hub category is sent to the first party. In more detail, the third-party non-essential traffic is sent to advertisers such as `doubleclick.net` and `adsystem.com`, and other services such as `netflix.com`, which are contacted by video devices, even if we do not use the Netflix app during our experiments.

Most non-required destinations are first parties, with domain names suggesting that they are mainly used for logs, diagnostics and device configuration (*i.e.*, `diagnostics.meethue.com`, `device-metrics-us.amazon.com`, `logs.roku.com`; see Table 3 for more detail).

Takeaways. A key finding is that—for the devices we tested—third-party destinations are always non-required. This suggests that a simple blocking approach for such devices is simply to block all third-party communication. We also found that the video category has the largest number of third parties, some of which is explained by the menu screen loading previews of content from third-party apps.

The fact that some non-required destinations are first or support party suggests that the manufacturer includes device activity that is unrelated to the main function. This could occur for good reasons such as firmware updates, or for more concerning reasons such as collection of device/user data. Fortunately, only a small amount of payload is sent to non-required destinations, suggesting that the device is not exposing much information over these connections. On the other hand, we observe a significant number of non-required destinations contacted that are not first parties. This is concerning because recent work shows that in such a small payload it is still possible to signal the device presence, its status, and basic data from its sensors [5, 13, 14], thus constituting a privacy and potential security risk.

Destination	Device (Non-required)	Device (Required)
api.amazon.com	Allure	Echo Dot, FireTV
bob-dispatch-prod-eu.amazon.com	Fire TV	Echo Dot, Allure

Table 4. Device-dependant destinations. Destinations that are both required and non-required for different devices.

5.3 Device-dependent Non-required Destinations

In this analysis we check whether any destinations that are non-required for a device are required for another device. We define these destinations as *device-dependent*. Knowing if there are any destinations that are device-dependent (under our definition) is important since their existence means that a blocking approach to prevent non-essential traffic cannot rely only on a flat list of destinations; rather blocking of destinations must be device-specific (requiring accurate device detection) for at least some devices.

Table 4 shows the list of device-dependent destinations (first column), with the list of devices for which they are non-required (second column) and required (third column). We find two destinations that are non-required for some devices, but required for others, even for devices from the same manufacturer. The

Destination	Device (Category)
*.cloudfront.net	Echo Dot (Smart-speaker), FireTV (Video)
api-global.eu-west-1.prodaa.netflix.com	Fire TV (Video), Roku TV (Video)
arcus-uswest.amazon.com	Echo Dot (Smart-speaker), Fire TV (Video)
device-metrics-us.amazon.com	Echo Dot (Smart-speaker), Fire TV (Video)
euw1-api.tplinkra.com	TP-Link bulb, plug (Home-automation)
n-deventry.tplinkcloud.com	TP-Link bulb, plug (Home-automation)
use1-api.tplinkra.com	TP-Link bulb, plug (Home-automation)

Table 5. Non-required destinations contacted by multiple devices.

first case is `api.amazon.com`, which is mandatory for Amazon devices to function, but not required by the Allure speaker, although it is powered by the same voice assistant of Amazon (Alexa). The second case is `bob-dispatch.prod-eu.amazon.com`, which is required by all Amazon-powered smart speakers to process voice commands, but not required to watch YouTube on an Amazon Fire TV.

Takeaways. On one hand, device-dependent destinations do exist, motivating blocklists that associate destinations to the actual device. However, the function tested is also relevant in determining if a destination is required or not. For example, Amazon Fire TV is primarily designed to stream TV through apps, but it also offers voice assistant functionality: for this reason the presence of a non-required destination typically used by Amazon-enabled smart speakers is not surprising.

5.4 Common Non-required Destinations

We now analyze non-required destinations that are in *common* for the devices we tested, *i.e.*, non-required destinations contacted by more than one device. The reason for this analysis is that, if multiple devices have the same non-required destination, such destination may have the same non-required purpose for other devices as well, which can help generalize our blocking approach.

Table 5 reports the list of common non-required destinations (first column), and the devices/categories contacting them (second column). We observe that the same manufacturers (*e.g.*, Amazon and TP-Link) have an overlap for non-required destinations. In the case of TP-Link, the non-required destinations contacted by a bulb and a plug coincide. Note that there is overlap from devices from different manufacturers in the video category: both Fire TV and Roku TV contact the same third-party service that is non-required (`api-global.eu-west-1.prodaa.netflix.com`).

Takeaways. Our experiments demonstrate that devices from the same vendors tend to behave similarly, probably due to sharing some code among them and integrating them in the same IoT ecosystem. This enables the extension of a blocking approach based on our destination lists to other or future devices from the same manufacturer. Our analysis does not show notable situations in which devices from different vendors contact the same non-required destinations, except for the case of devices in the video category, where both devices show Netflix video recommendations in their menu screen.

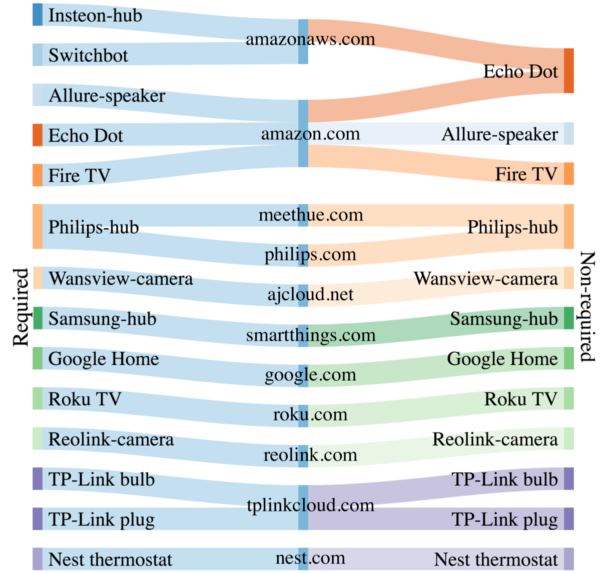

Fig. 7. Second-level domain destinations that are both required and non-required. The width of the flow is proportional to the number of individual destinations contacted by each device.

5.5 Impact of SLDs, Protocols, and Ports

Using only second-level domains (SLDs). We now investigate whether SLDs are sufficient for identifying non-required destinations. Fig. 7 shows the list of SLDs that are simultaneously required and non-required, and the list of devices contacting them as required (on the left) and non-required (on the right): for 12 devices contacting the 11 SLDs in the figure, SLDs are not specific enough since both essential and non-essential traffic use the same SLDs. For the remaining cases (19/31 devices), using only SLDs is sufficient. While this simplification of using SLDs is effective for identifying non-essential traffic for the majority of the devices we tested, it nonetheless would lead to mislabeling traffic for a significant fraction of devices (12/31).

Using protocols and ports. We determine whether the IP protocol and port are alone sufficient to detect traffic to non-required destinations. The answer is generally no: we find that most non-required destinations consistently use HTTPS (TCP/443),

with just the following exceptions: two domains using HTTP (TCP/80) (`fireoscaptiveportal.com` and `diagnostics.meethue.com`), and ICMP packets sent to two IP addresses by the *Bosiwo camera*.

5.6 Experiments with Additional Functions

In the previous analyses, we consider, for each device, only the main function. We now investigate if the list of required/non-required destinations changes when we consider *additional functions*. To this end, we increase functionality coverage by testing, in addition to the main function, at least three additional functions per device, listed per category in Table 6. We then recreate the lists of (non)-required destinations as follows: required destinations are the ones needed by *at least* one tested function, while non-required destinations are the ones that are not needed by *any* tested functions.

After testing additional functions, we see no changes in the list of non-required destinations for devices in all categories except for smart speakers: all of them contact several additional non-required destinations when asked for streaming music on YouTube/Amazon. For example, the Google Home device contacts 4 additional non-required destinations, two of which are third parties (`googleadservices.com`, `googleads.g.doubleclick.net`). With respect to required destinations, only five devices require new destinations to fulfill one of the additional functions: Yi camera (enable motion), Honeywell thermostat (adjust the temperature based on the weather), Google Home (stream music on YouTube), and Echo Dot/Allure (stream music on Amazon). The lists of destinations contacted by additional functions in our tests are reported (in parenthesis) in Table 8 in Appendix B.

Takeaways. While testing additional functions, non-required destinations are unchanged for 90.32% of the devices, meaning that even if we only test the main function, we cover the vast majority or non-required destinations. Only five devices have one additional required destination, which is required and used only for one of the additional functions, suggesting that it is common for required destinations to fulfill more than one function. While streaming content, smart speakers contact up to 4 non-required destinations and blocking those destinations does not break any additional functions.[2]

Category	Additional Functions	Req. #	Non-Req. #
Camera	Recording, get clip recordings, enable motion	1	0
Home-automation	Schedule, timer, set status, set temperature, check water level	1	0
Hub	Schedule, timer, set status	0	0
Speaker	Wikipedia search, google search, play music on YouTube/Amazon	5	6*
Video	Sleeping mode, timer, add to watch list	0	0

Table 6. List of additional tested functions per category and number of additional required/non-required destinations. Only the additional function "*playing music on YouTube/Amazon" triggers additional non-required destinations for smart speakers.

5.7 Similarities with Existing Blocklists

To conclude this section, we determine whether any of the observed required or non-required destinations appear on blocklists from prior work. This can help clarify if any of such lists can be effective in also blocking non-essential IoT traffic, or if they are likely to break some IoT functionality. In this analysis we use the blocklists considered by Varmarken et. al [4], who evaluated the effectiveness of DNS-based blocklists to prevent smart TVs from accessing advertising and tracking service domains. In particular, they consider the most relevant blocklists to smart TVs (actively managed), which are Pi-hole Default [15], the Firebog [16], Mother of all AdBlocking (MoaAB) [17], and StopAd [18].

Table 7 shows the devices having at least one non-required destination. The table shows that existing blocklists contain very few of such destinations only for the most popular devices. Out of the 62 non-required destinations, the most (*i.e.*, six) are obtained in the Firebog list. This is not surprising as the Firebog merges many popular blocklists into a single one. The second most successful blocklist is the Pi-hole, which blocks four non-required destinations, the third is MoaAB blocking two, and the last is StopAd, which does not contain any of the non-required destinations.

Regarding the presence of required destinations in existing blocklists, we have not found any, which means that current popular blocklists should not break the functionality of the devices we tested.

Takeaways. Most (91%) destinations we have identified as non-required do not appear on any of the existing blocklist, making them inadequate for mitigating non-essential traffic in the consumer IoT context. This occurs because existing blocklists primarily target websites and smart TVs, while we consider a broader range of IoT device categories that use different destinations.

2 Despite blocking advertisement destinations, there is no change in advertising behavior.

	Device	Non-required #	Pi-hole	Firebog	MoAB	StopAd
Camera	Bosiwo	2	0	0	0	0
	iCSee	4	0	0	0	0
	Reolink	1	0	0	0	0
	Wansview	6	0	0	0	0
	Yi	2	0	0	0	0
Home-auto	Nest thermostat	1	0	0	0	0
	TP-Link Bulb	3	0	0	0	0
	TP-Link Plug	3	0	0	0	0
	Xiaomi rice	4	0	0	0	0
Hub	Philips	2	0	0	0	0
	Samsung	1	0	0	0	0
Speaker	Allure	2	0	0	0	0
	Echo Dot	7	1	1	0	0
	Google Home	5	0	0	0	0
Video	Fire TV	11	2	3	1	0
	Roku TV	8	1	2	1	0
	Total	*62*	*4*	*6*	*2*	*0*

Table 7. Similarity to existing blocklists. Comparison, by device, of the total number of non-required destinations with the number of such destinations that are present in various blocklists.

6 Mitigating Non-essential Traffic

We answer our last research question by discussing how to limit IoT information exposure in practice.

6.1 Blocking Strategies

Deny-listing: *blocking non-required destinations and allowing the rest of the traffic.* This strategy only works under the assumption that non-required destinations are *stable*, *i.e.*, they do not change over time. To verify this, we measured the non-required destinations at several points in time over six months: May, July, and October 2020. We then compared the three lists of non-required destinations and verified that there are no differences. This means that the destinations that were non-required during our first set of experiments were *still* contacted and non-required six months later.

For this reason we consider all non-required destinations we encountered so far as stable. Having a vast majority of stable non-required destinations means that a *deny-listing* blocking strategy is feasible because it does not need frequent updates on its blocklists, with low risk of allowing non-essential traffic and/or breaking the device functionality. The drawback of this approach is that the possible appearance of new non-required destinations would not be mitigated.

Allow-listing: *allowing required destinations and blocking the rest.* The assumption of this strategy is that required destinations do not change over time. We also verified this assumption on the same three sets of experiments over six months, noticing that required destinations also do not change.

The stability of required destinations makes an *allow-listing* approach also feasible, without breaking the device functionality. The advantage of this approach is that it has the highest mitigation potential, since existing and future non-required destinations will be blocked, but it also carries the highest risk of breaking the functionality of the device since if in the future a function requires a new destination, it will be blocked until the list of required destinations is updated.

Choosing a blocking strategy. Based on the considerations above, choosing between a *deny-listing* and *allow-listing* blocking strategy depends on the priority between functionality and mitigation. We believe that for the typical home IoT scenario a *deny-listing* strategy may be more appropriate, since maintaining functionality is a high priority (and mitigation of newly blockable destinations can be addressed through periodic blocklist updates). In critical scenarios where privacy and security is a priority over functionality (*e.g.*, enterprise deployments), *allow-listing* may be the more appropriate.

6.2 Maintenance of Blocklists

IoT systems may change the set of destinations they contact over time (*e.g.*, via firmware updates or server-side changes), potentially requiring updating the blocklists so they remain effective. While we did not observe such a change in six months, this may occur over longer periods. Since all the steps of our approach are automated (except for the creation of probe and trigger scripts, which is manual, but only needed once), the measurement of (non-)required destinations can be easily iterated to keep the blocklists updated. To minimize the risk of triggers/probes failing (*e.g.*, changes in the device interaction interface), we rely on our probe evaluation algorithm (see §4.4), which is run before measuring the destinations. Specifically, if a probe becomes inaccurate for a function or device, experiments are disabled and the problem is reported so that a human maintainer knows to update the affected trigger/probe scripts. We anticipate that blocklists and the library of trigger/probe scripts will be maintained via options like crowdsourcing or via organizations that conduct our automated measurements on a regular basis (and share the outcomes), similar to what happens for blocklists for web/mobile-app trackers and advertisers.

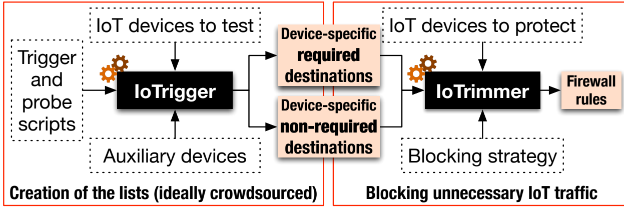

Fig. 8. Design of a blocking system. *IoTrigger* measures required/non-required destinations, while *IoTrimmer* uses them to block non-essential IoT traffic by defining firewall rules.

6.3 Design of a Blocking System

To mitigate non-essential IoT traffic, we propose a blocking system composed of two components: *IoTrigger* and *IoTrimmer* (see Fig. 8). The former runs the methodology in §3 to produce (non-)required destination lists, and the latter uses such lists with a blocking strategy to generate firewall traffic-blocking rules.

***IoTrigger*.** This component runs on a router providing connectivity to a set of IoT devices to test. It manages the lifecycle of functionality experiments for each device, including the invocation of user-provided trigger and probe scripts, and to finally produce (non-)required destination lists. To work, *IoTrigger* needs the IoT devices connected to the same router, the list of their IP addresses, the scripts to trigger and probe their functions, and any other auxiliary devices (*e.g.*, devices used by trigger/probes scripts). Given this, *IoTrigger* will run the experiment and generate the destinations lists without any human interaction. We implemented a command-line prototype of *IoTrigger*, which includes a library of probes and triggers scripts that support the IoT devices we tested. Anyone owning the same IoT devices, and the proper trigger devices (*e.g.*, Android phones) can use the *IoTrigger* prototype to reproduce our results. For new devices and new functions, new trigger/probe scripts must be added.

***IoTrimmer*.** This component runs on a router and uses the destination lists produced by *IoTrigger* to determine which destinations to block. *IoTrimmer* takes as input these lists, the list of IoT devices (and their IP addresses) to be protected, and the blocking strategy (for generating firewall rules). These rules, the final output of *IoTrimmer*, are installed in the router to block non-essential IoT traffic. We implemented a prototype of *IoTrimmer* (Appendix C). It comes preconfigured with the *deny-listing* blocking strategy and uses the blocklist of 62 non-required destinations we found for our set of 31 IoT devices. This *IoTrimmer* prototype automatically detects devices connected to the IoT network: the user is provided with a web interface to associate the detected device with the ones we have analyzed, so the prototype can automatically apply proper firewall rules.

6.4 Effectiveness of a Blocking System

Effectiveness Evaluation. To measure the effectiveness of our blocking system in terms of preserving desired functionality, we run the following test for 7 days on our 31 IoT devices: we first protect the devices using *IoTrimmer* with a *deny-listing* blocking strategy, then we use trigger and probe scripts to run their main functions once per day at different hours and check whether they work. We found all of the 217 function invocations were successful and thus *IoTrimmer* is effective at blocking without breaking for the devices we tested.

Risk of allowing non-essential traffic. Independently from the blocking strategy used, our approach tries to block non-essential traffic that is produced by non-required destinations. However, it is possible that some devices use (or may use in the future, to elude our blocking strategy, see §7.2) the same destination for both essential and non-essential traffic. In this case a future improvement of our approach is to look not just at the destination, but also at other traffic characteristics to find more distinguishing features, and then filter the traffic based on such features.

Risk of breaking device functionality. Our work is motivated by the fact that most consumer IoT devices are relatively simple, offering functions that are easy to test such as changing the state of a light, or asking a question to a smart speaker. Since we test functionality similarly to how a generic algorithm is tested, in the case of complex functionality, we cannot prove that it works for every possible input (the total correctness of an algorithm is not decidable). Due to this limitation we expect cases where complex functionality that has not been fully tested may break. Unfortunately this is a limitation on most blocking systems (*e.g.*, ad-blockers [19] preventing a website from loading non-ad content correctly), which can be partially addressed by increasing the coverage of the functionality tested to the maximum possible extent, and refining blocklists accordingly.

7 Discussion

We now discuss implications of our findings and limitations of our approach.

7.1 Implications

Purpose of non-required destinations. We know that non-required destinations are not essential for the main function of the device, but their intended purpose remains an open question, particularly whether the purpose is benign or malicious. One (optimistic) hypothesis is that they are required for other (*i.e.*, *non-main*) functions that we did not test (*e.g.*, for syncing with a cloud service, checking for a firmware update). A less optimistic hypothesis is that they are used for tracking purposes, given that some destinations (*e.g.*, `partnerad.l.doubleclick.net` and `netflix.com`) are third parties not related to the device manufacturer.

Privacy considerations. In this work we have seen that many contacted destinations are not required for the device to operate. The good news is that the *quantity* (number of bytes) of data we have seen sent to the non-required destinations is very small compared to the rest of the traffic, as anything otherwise would be extremely suspicious. However, even if the amount of data is small, it is still a concern from a privacy perspective, since it is still enough to signal the presence of a device and the functions in use, as shown in previous work [5, 13, 14]. Further, such non-required connections potentially violate the data minimization by design principle of some privacy regulations such as the GDPR [7].

A related question is whether device manufacturers reveal the purpose of these connections in their privacy policies. Unfortunately, many devices' privacy policies provide little information about how they use the data from their customers' devices [20]. In many cases it is unclear whether a destination is used by the IoT device or the mobile app controlling it, and the behavior of some devices is not consistent with what is stated [21].

7.2 Limitations

The experiments of this work have been executed on a fixed set of devices, and limited to a subset of their functions. We do not know if our results extend to other devices, after future firmware updates to existing devices, or for any additional functions. However, our initial results are promising, suggesting that our methodology covers popular unmodified devices across different functional categories. We expect that the non-essential traffic reported in this study represents a subset of all such traffic that our IoT devices generate. As such, our findings represent a lower-bound of such traffic, using an approach that can be automated, *i.e.*, automatically detecting non-required destinations.

Non-observable functionality. Our approach only works for device functions that can be tested using trigger and probe scripts. Some functions cannot be triggered (*e.g.*, device maintenance or synchronization tasks); to allow such functions as needed, one can periodically restart the device and unblock previously flagged non-required destinations temporarily to allow the maintenance connections to proceed.

Firmware updates. While firmware updates are important for adding features and security patches to IoT devices, they may also introduce unwanted behavior [22]. We believe it should be up to the user to decide whether to allow or block these updates. By default our approach can block firmware updates if the corresponding destination(s) are not used for any essential function. If a user chooses to allow firmware updates while blocking non-essential traffic, the following strategies may be used. For unattended updates, we can use the "non-observable functionality" approach (*e.g.*, restart the device while keeping destinations unblocked for a set period of time). For user-initiated updates, we can treat them as a device function, and use a dedicated set of trigger/probe scripts to detect what destinations are used by the firmware update function. Another approach is to allow traffic matching patterns that reveal the firmware update intention (*e.g.*, destinations containing strings such as "fwupdate").

Scalability. Every step of our approach is fully automated, including the execution of probe and trigger scripts, with each function to be tested taking an average of 4 minutes and easily repeatable to allow frequent crowdsourced updates. However, the *creation* of such scripts is a manual process that has to be repeated once for every function tested for each device. A mitigating factor is that devices belonging to similar categories may reuse existing scripts with little modifications (*e.g.*, a simple change of tap coordinates for companion app triggers, and of screenshots for companion app probes). Although not observed in the six months of our study, it is possible that trigger and probe scripts stop working and need to be manually modified when device functionality changes substantially (*e.g.*, via firmware updates). We can identify such cases by periodically running our automated probe evaluation algorithm (see §4.4).

Blocking granularity. Our approach focuses on destination-based blocking to reduce information exposure; however, other factors may be used to identify traffic that should be blocked, *e.g.*, time of day, traffic volumes, device-to-device communication. One advantage of only considering destinations is that it is easy to automatically measure, and easy to block using simple

firewall rules, without the need of fine-grained enforcement mechanisms that may not be readily deployable and may incur heavy overhead at runtime.

Evading blocklists. To evade the blocklists a device can disable its functionality when any of its destinations are unreachable. This limitation also exists in anti-tracking browser plugins, where a website is not allowed to load until anti-tracking software is disabled [23]. There is no simple defense against this evasion technique, but our approach can still block any non-required destinations where a device does not try to evade blocking, *e.g.*, destinations used by third-party apps.

Third-party apps. Some devices include pre-installed third-party apps (*e.g.*, Netflix on video devices). In such cases, background traffic or content previewed on a menu screen may be considered required or non-required depending on whether the device owner wishes to use those apps. A limitation of our work is that we cannot know whether to block or allow the traffic for third-party apps without user input about which apps are required to work. As an example, we found that `netflix.com` was identified as non-required by default in our approach because it is not necessary for the menu screen to work. For users that subscribe to Netflix, we can include results from testing the Netflix app on the device and treat corresponding destinations as required.

Working with MUD profiles. Manufacturer Usage Description (MUD) profiles [24, 25] allow manufacturers to declare the behavior of their devices (including the destinations contacted). None of the devices we tested implements MUD profiles. However, even if a destination is declared in the future, a MUD profile does not help to determine if such destination is used for essential traffic only. Hence, our approach is orthogonal to MUD profile enforcement and can work side-by-side with it.

8 Related Work

Recent research has produced a number of tools to protect against undesirable IoT traffic. Haar and Buchmann presented FANE [26], a firewall that isolates IoT devices into a separate network. FANE allows communication only with the learned set of IPs. If a device contacts a new IP address, the user is alarmed. FANE does not support blocking destinations based on domain names. Simpson *et al.* [27] focus on protecting IoT devices against known vulnerabilities and automatically blocking traffic when a threat is identified. Gupta *et al.* [28] propose a firewall based on a Raspberry Pi with simple *iptables* rules to protect the devices from potential attacks. Heimdall [29] focuses on protecting devices against hacks from the Internet using a pre-learned *allow-list*. Lastdrager *et al.* [30] describe SPIN, a software tool for visualizing and blocking traffic from IoT devices. None of these solutions focus on mitigating information exposure nor blocking connections without breaking device functionality.

Numerous commercial tools provide solutions to protect networks with IoT devices, *e.g.*, ShieldIOT [31], Fing [32], and Bitdefender [33]. These approaches either rely on cloud-based analysis of network traffic, target device manufacturers rather than device users, block or allow the device as a whole, monitor the overall amount of IoT traffic generated, or protect against known vulnerabilities and attacks from the Internet. *IoTrimmer* allows fine-grained control over destinations contacted by the devices and protects users privacy by blocking the unnecessary traffic generated by IoT devices.

There are a number of existing tools for IoT privacy risk analysis. For example, IoT Inspector [34] collects smart home traffic using ARP spoofing. However, this tool focuses on the collection of data, rather than its analysis or the blocking of non-essential traffic. A recent study [2] of 81 consumer IoT devices shows that many IoT devices expose information to first, support, and third parties. Additional research uses traffic generated by the IoT devices to identify devices or device activities [5, 13, 14, 35–39]. Because *IoTrimmer* can reduce the number of destinations contacted by IoT devices, it reduces the attack surface and can prevent an eavesdropper to identify users device or activity.

Two recent IoT papers focus on strategies for defending user privacy against potential eavesdroppers (*e.g.*, ISP). Apthorpe *et al.* [40] propose generating additional dummy network traffic that hides genuine IoT device network traffic patterns from an observer, and Alshehri *et al.* [41] proposes a similar approach using uniform random noise. In contrast, our approach focuses on protecting users' privacy from legitimate destinations that the IoT devices communicate with, not against potential eavesdroppers. However, our approach can be integrated with the above work to enhance user privacy against potential eavesdroppers.

9 Conclusion

This paper demonstrated that it is feasible and effective to block non-essential network traffic from IoT devices,

thus limiting the information they expose to other parties without breaking device functionality. We developed the first comprehensive method to automatically identify non-required destinations from network traffic, and analyzed the results of the corresponding experiments.

We found that 16 of the 31 consumer IoT devices in our study contact destinations that are not required to fulfill their main functions. Most destinations (62 out of 119) are responsible for non-essential traffic, and such non-required destinations are relatively long-lasting in our study— they did not change at all over six months. The vast majority (91%) of destinations responsible for non-essential traffic are not listed in any other general blocklist, demonstrating the benefits of our device-dependent approach. Finally, we produced a set of guidelines and a prototype of a blocking system to mitigate non-essential IoT traffic.

To support further research, all software and data we produced as part of this work are publicly available at http://iotrim.net/.

10 Acknowledgments

We thank the anonymous reviewers for their constructive feedback. The research in this paper was partially supported by the EPSRC (Databox EP/N028260/1, DADA EP/R03351X/1, HDI EP/R045178/1, and Impact Acceleration Account (IAA)), NSF (BehavIoT CNS-1909020, ProperData SaTC-1955227) and Consumer Reports (Digital Lab Fellowship for Daniel J. Dubois).

References

[1] IoT Analytics. IoT 2019 in review: The 10 most relevant IoT developments of the year. https://iot-analytics.com/iot-2019-in-review/. [Online; accessed Nov. 2020].

[2] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*, 2019.

[3] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W. Felten, Prateek Mittal, and Arvind Narayanan. Watching you watch: The tracking ecosystem of over-the-top TV streaming devices. In *CCS'19*, 2019.

[4] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. The TV is smart and full of trackers: Measuring smart TV advertising and tracking. *PETS'20*, 2020(2):129–154, 2020.

[5] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. A haystack full of needles: Scalable detection of IoT devices in the wild. In *IMC'20*, pages 87–100, 2020.

[6] Pi-Hole: A black hole for Internet advertisements. https://pi-hole.net/. [Online; accessed Nov. 2020].

[7] ico. Principle (c): Data minimisation. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/data-minimisation/. [Online; accessed Nov. 2020].

[8] ico. Principle (b): Purpose limitation. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/purpose-limitation/. [Online; accessed Mar. 2021].

[9] IFTTT, Inc. IFTTT helps every thing work better together. https://ifttt.com. [Online; accessed Mar. 2021].

[10] SmartThings, Inc. SmartThings: One simple home system. A world of possibilities. https://www.smartthings.com. [Online; accessed Mar. 2021].

[11] Roku Inc. Roku Developer Documentation: Development Environment Overview. https://sdkdocs.roku.com/display/sdkdoc/Development+Environment+Overview. [Online; accessed Feb. 2021].

[12] Amazon.com Inc. Developer Tools Menu (Fire TV). https://developer.amazon.com/docs/fire-tv/developer-tools.html. [Online; accessed Feb. 2021].

[13] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-level signatures for smart home device events. In *NDSS'20*, 2020.

[14] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. Peek-a-Boo: I see your smart home activities, even encrypted! In *WiSec'20*, page 207–218, 2020.

[15] Pi-Hole LLC Blocking Mode. https://docs.pi-hole.net/ftldns/blockingmode. [Online; accessed Nov. 2020].

[16] WaLLy3K. The big blocklist collection. https://firebog.net. [Online; accessed Nov. 2020].

[17] Mother of All AD-BLOCKING. The big blocklist collection. https://forum.xda-developers.com/showthread.php?t=1916098. [Online; accessed Nov. 2020].

[18] Kromtech Alliance Corp. Stopad for TV. https://stopad.io/. [Online; accessed Nov. 2020].

[19] Ashish Kumar Singh and V. Potdar. Blocking online advertising - a state of the art. In *2009 IEEE International Conference on Industrial Technology*, pages 1–10, Feb 2009.

[20] Consumer Reports. Home security cameras from top brands lack basic digital security measures. https://www.consumerreports.org/wireless-security-cameras/home-security-cameras-from-top-brands-lack-basic-digital-security-measures/. [Online; accessed Nov. 2020].

[21] Alanoud Subahi and George Theodorakopoulos. Ensuring compliance of IoT devices with their privacy policy agreement. In *FiCloud'18*, pages 100–107. IEEE, 2018.

[22] Chris Welch. I guess I have to watch ads everywhere on my $1,500 LG TV now. https://www.theverge.com/tldr/

2021/3/10/22323790/lg-oled-tv-commercials-content-store. [Online; accessed Mar. 2021].
[23] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J Murdoch. Adblocking and counter blocking: A slice of the arms race. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*, 2016.
[24] E. Lear, R. Droms, and D. Romascanu. RFC 8520: Manufacturer usage description specification, 2019.
[25] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman. Clear as MUD: Generating, validating and applying IoT behavioral profiles. In *SIGCOMM '18 Workshop on IoT S&P*, 2018.
[26] C. Haar and E. Buchmann. FANE: A firewall appliance for the smart home. In *FedCSIS '19*, pages 449–458, 2019.
[27] A. K. Simpson, F. Roesner, and T. Kohno. Securing vulnerable home IoT devices with an in-hub security manager. In *PerCom '17 Workshops*, pages 551–556, 2017.
[28] N. Gupta, V. Naik, and S. Sengupta. A firewall for internet of things. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, pages 411–412, 2017.
[29] J. Habibi, D. Midi, A. Mudgerikar, and E. Bertino. Heimdall: Mitigating the internet of insecure things. *IEEE Internet of Things Journal*, 4(4):968–978, 2017.
[30] E. Lastdrager, C. Hesselman, J. Jansen, and M. Davids. Protecting home networks from insecure IoT devices. In *NOMS 2020*, pages 1–6, 2020.
[31] ShieldIOT. https://shieldiot.io/. [Online; accessed Nov. 2020].
[32] Fingbox. https://www.fing.com/. [Online; accessed Nov. 2020].
[33] Bitdefender. https://www.bitdefender.com/iot/. [Online; accessed Nov. 2020].
[34] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. IoT inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(2), June 2020.
[35] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic. *DAT'16*, 2016.
[36] Hamid Tahaei, Firdaus Afifi, Adeleh Asemi, Faiz Zaki, and Nor Badrul Anuar. The rise of traffic classification in IoT networks: A survey. *Journal of Network and Computer Applications*, 154:102538, 2020.
[37] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis. In *SAC '17*, pages 506–509, 2017.
[38] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. IoT SENTINEL: Automated device-type identification for security enforcement in IoT. In *ICDCS'17*, pages 2177–2184, 2017.
[39] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma. IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge. *IEEE Transactions on Network and Service Management*, 17(1):45–59, 2020.
[40] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the smart home private with smart(er) IoT traffic shaping. *PETS*, 2019(3):128 – 148, 2019.
[41] Ahmed Alshehri, Jacob Granley, and Chuan Yue. Attacking and protecting tunneled traffic of smart home devices. In *CODASPY '20*, page 259–270, 2020.

A Grouping Ephemeral Destinations

During our destination-observing experiments (see §3.3), some devices contact destinations that appear in less than 80% of the experiment iterations. We refer to such destinations as *ephemeral destinations*. To facilitate analysis and streamline blocklists, we developed two algorithms to automatically group ephemeral destinations into specific groups that cover ephemeral destinations in at least 80% of the iterations. One algorithm is for ephemeral hostname destinations and the other for ephemeral IP destinations.

Grouping hostname destinations. When a hostname is ephemeral (*i.e.*, it appears in less than 80% of the iterations), we remove the first character from the domain name and replace it with a wildcard matching any number of characters (zero or more). If the resulting group matches domains in at least 80% of the iterations, we consider such group as a new hostname destination (and remove all the matching hostnames from the list of destinations). If not, we repeat the process recursively by replacing additional characters with the wildcard up to the entire second-level domain. For example, ephemeral domains `1.yy.com` and `2.yy.com` are replaced by the group `*.yy.com`, which also happens to be the entire second-level domain.

Note that our algorithm is also capable of finding groups that are *more specific* than second-level domains. For example, ephemeral domains `a-b-c.ww.com` and `b-b-c.ww.com` are replaced by the group `*-b-c.ww.com`, which is more specific than a second-level domain.

Across all our destination-observing experiments, we have found three groups matching ephemeral hostname destinations (`*.backblaze.com`, `*.cloudfront.net`, `*.googlevideo.com`), which also happen to be second-level domain names, since there were no more specific alternatives. We have found no cases of ephemeral hostname destinations that could not be matches by one of the three groups above.

Grouping IP destinations. When an IP address is ephemeral (*i.e.*, it appears in less than 80% of the iterations), we perform a WHOIS query to get the IP mask that includes such IP address. If such IP mask matches an ephemeral IP in at least 80% of the iterations, we consider such IP mask as a destination (and remove all the matching IPs from the list of destinations). For example, if we have two ephemeral IPs 1.2.3.4 and 1.2.4.5, and for both of them we obtain a matching mask from WHOIS that is 1.2.0.0/16, we would use 1.2.0.0/16 as the grouped destination.

Across all our destination-observing experiments, we have found no cases of ephemeral IP addresses, and therefore we have no IP destination groups. Still, should that happen in the future, this algorithm would be able to deal with such cases.

B List of Required and Non-required Destinations

In this appendix we report, for each device, the list of required and non-required destinations. This is the data we used to produce part of the analyses in §5.

From Table 8, we can confirm that 16 out of 31 devices contact at least one non-required destination. In general, the number of non-required destinations tend to be larger than the number of required ones.

We believe this information, together with the *IoTrigger* and the *IoTrimmer* software available at http://iotrim.net/, to be valuable for researchers, device manufacturers, and regulators to support and reproduce our findings.

	Device	Dest. #	List of Required Destinations	List of Non-Required Destinations
Camera	Blink	2	rest-hw-prde.immedia-semi.com, cs-prde.immedia-semi.com	
	Bosiwo	4	*145.239.253.48, 37.187.159.39*	*54.157.82.107,* 210.72.145.44
	iCSee	6	*47.91.198.64, 47.91.207.52*	*47.52.222.172, 47.52.32.118, api.gdxp.com, oss-us-west-1.aliyuncs.com*
	Reolink	2	p2p.reolink.com	pushx.reolink.com
	Wansview	9	*cam-gw-isc-eu02.ajcloud.net, cam-tunnel-isc-eu02.ajcloud.net, fw-isc.ajcloud.net*	*159.65.95.225, 3.122.229.130 , ajcloud.net, htpdate.ajcloud.net, sdc-isc.ajcloud.net, *.backblaze.com*
	Yi	5 (1)	*47.74.255.9, 47.88.59.209, 47.90.240.160,* (motiondetection-eu.oss-eu-central-1.aliyuncs.com)	api.eu.xiaoyi.com, log.eu.xiaoyi.com
Home-automation	App Kettle	2	ak.myappkettle.com, query.jingxuncloud.com	
	Honeywell therm.	2 (1)	*ihsu-prod-bl-003.cloudapp.net,lcc-prodsf-fwu.eastus.cloudapp.azure.com, (weather.clouddevice.io)*	
	Magichome	1	ra8816us.magichue.net	
	Meross opener	1	iot.meross.com	
	Nest thermostat	3	transport.home.nest.com, logsink.devices.nest.com	frontdoor.nest.com
	Netatmo weather	1	netcom.netatmo.net	
	Smarter coffee	1	prd19a.boxen.electricimp.com	
	Smartlife bulb	1	a.tuyaeu.com	
	Smartlife remote	1	a.tuyaeu.com	
	Sousvide	1	pc.anovaculinary.com	
	Switchbot	1	*a2alhn2dfztqv9.iot.us-east-1.amazonaws.com*	
	TP-Link bulb	4	n-devs.tplinkcloud.com	euw1-api.tplinkra.com, n-deventry.tplinkcloud.com, use1-api.tplinkra.com
	TP-Link plug	4	n-devs.tplinkcloud.com	euw1-api.tplinkra.com, n-deventry.tplinkcloud.com, use1-api.tplinkra.com
	Wemo plug	2	*api.xbcs.net, nat.xbcs.net*	
	Xiaomi rice-cooker	7	mi.com, ot.io.mi.com, *120.92.65.243*	*183.84.5.203, 58.83.160.36, 123.125.102.215, 110.43.0.83*
Hub	Insteon	1	*lb-connect-insteon-com-503033429.us-east-1.elb.amazonaws.com*	
	Lightify	3	*srm-emea-p01-lb02.arrayent.com, 35.157.95.104, 35.159.20.196*	
	Philips	4	dcp.dc1.philips.com, ws.meethue.com	diagnostics.meethue.com, ecdinterface.philips.com
	Samsung	3	api.smartthings.com, dc.CoNnect.SMaRTThInGs.cOm	fw-update2.smartthings.com
	Sengled	2	eu.cloud.sengled.com, *18.195.119.104*	
Speaker	Allure	3 (3)	bob-dispatch-prod-eu.amazon.com, (m.media-amazon.com, tinytts.amazon.com)	*api.amazon.com, d1enchupjctwud.cloudfront.net, (msh.amazon.com)*
	Echo Dot	10 (3)	api.amazon.com, bob-dispatch-prod-eu.amazon.com, unagi.amazon.com, (m.media-amazon.com, tinytts.amazon.com)	arcus-uswest.amazon.com, *.cloudfront.net, device-metrics-us.amazon.com, dp-gw.amazon.com, fireoscaptiveportal.com, *prod.amcs-tachyon.com* , s3-1-w.amazonaws.com, (msh.amazon.com)
	Google Home	9 (5)	connectivitycheck.gstatic.com, home-devices.googleapis.com, play.googleapis.com, www.google.com, (*knez.googlevideo.com)	youtube-ui.l.google.com, clientservices.googleapis.com, fcm.googleapis.com, *.googlevideo.com, storage.googleapis.com, (tools.google.com, www.youtube.com, www.googleadservices.com, googleads.g.doubleclick.net)
Video	Fire TV	14	api.amazon.com, unagi-eu.amazon.com, *youtube.com*	aax-eu.amazon-adsystem.com, arcus-uswest.amazon.com, bob-dispatch-prod-eu.amazon.com, *.cloudfront.net, device-metrics-us.amazon.com, api.amazon.com, ktpx-eu.amazon.com, api-global.eu-west-1.prodaa.netflix.com, mas-ext-eu.amazon.com, mas-sdk.amazon.com, msh.amazon.com
	Roku TV	10	api.sr.roku.com, *youtube.com*	api-global.eu-west-1.prodaa.netflix.com, configsvc.cs.roku.com, cooper.logs.roku.com, customerevents.eu-west-1.prodaa.netflix.com, ichnaea.eu-west-1.prodaa.netflix.com, partnerad.l.doubleclick.net, scribe.logs.roku.com, uiboot.eu-west-1.prodaa.netflix.com
Total	*31*	*119 (13)*	*57 (7)*	*62 (6)*

Table 8. Required and non-required destinations per device. Colors identify the destination party type (see §5.2): first party, *support party*, and third party. In parenthesis the additional destinations for the additional functions. Only "playing music on YouTube/Amazon" triggers additional non-required destinations for smart speakers.

C *IoTrimmer* Prototype

We have implemented a prototype version of *IoTrimmer*. Fig. 9 shows its web interface. When a new device is connected to *IoTrimmer* its MAC address appears on the list.

The user then chooses which device is connected to *IoTrimmer*. The blocklist (*IoTrim*) is regularly updated from the Internet and automatically applied to all connected devices. Users can click on a device to display the list of blocked destinations.

192.168.0.52

http://iotrimmer

Welcome to IoTrimmer

Your current WiFi password is shown below. You can edit it by clicking on it. Please note, that you have to restart IoTrimmer in order for the changes to take place. The password must be at least 8 characters long and must not contain a space

iotrimmer

List of devices (monitored devices are displayed in green, not monitored devices are displayed in red). Click on the MAC address of the device to display the list of blocked domains.
MAC address is a unique identified of the device generated by the manufacturer.
Vendor is the manufacturer of the Wi-Fi chipset and may help to identify the device.
IoT Device - please, choose the device model, e.g. Ring Doorbell or TP Link Smart Plug HS110

MAC Address	Vendor	Model Name
cc:f7:35:25:af:4d	Amazon Technologies Inc.	Fire TV
cc:f7:35:49:f4:5	Amazon Technologies Inc.	Echo Dot
ec:71:db:49:af:ee	Shenzhen Baichuan Digital Technology Co., Ltd.	Reolink Camera
58:b3:fc:5e:ca:74	SHENZHEN RF-LINK TECHNOLOGY CO.,LTD.	ICSee Doorbell
50:c7:bf:ca:3f:9d	TP-LINK TECHNOLOGIES CO.,LTD.	TP Link Bulb
['euw1-api.tplinkra.com', 'n-deventry.tplinkcloud.com', 'use1-api.tplinkra.com']		
d0:52:a8:a4:e6:46	Physical Graph Corporation	Smart Things Hub
b0:f1:ec:d4:26:ae	AMPAK Technology, Inc.	Allure Speaker (Alexa)
40:31:3c:e6:77:c2	XIAOMI Electronics,CO.,LTD	Xiami Rice Cooker
b0:be:76:be:f2:aa	TP-LINK TECHNOLOGIES CO.,LTD.	TP Link Plug
c:8c:24:b:be:fb	SHENZHEN BILIAN ELECTRONIC CO.ï¼ŒLTD	YI Camera
c8:3a:6b:fa:1c:0	Roku, Inc	Roku TV
54:60:9:6f:32:84	Google, Inc.	Google Home
ec:b5:fa:0:98:da	Philips Lighting BV	Philips Hub
64:16:66:2a:98:62	Nest Labs Inc.	Nest Thermostat
ae:ca:6:e:ec:89	Unknown	Bosiwo Camera

Enable IoTrimmer:

Fig. 9. Prototype implementation of *IoTrimmer*.