
Proceedings Paper:
Feraudo, Angelo, Popescu, Diana Andreea, Yadav, Poonam et al. (2024) Mitigating IoT Botnet DDoS Attacks through MUD and eBPF based Traffic Filtering. In: ACM International Conference Proceeding Series. ACM, pp. 164-173.

https://doi.org/10.1145/3631461.3631549

Mitigating IoT Botnet DDoS Attacks through MUD and eBPF based Traffic Filtering

Angelo Feraudo
University of Bologna
Bologna, Italy
angelo.feraudo@unibo.it

Diana Andreea Popescu
University of Cambridge
Cambridge, UK
diana.popescu@cl.cam.ac.uk

Poonam Yadav
University of York
York, UK
poonam.yadav@york.ac.uk

Richard Mortier
University of Cambridge
Cambridge, UK
richard.mortier@cl.cam.ac.uk

Paolo Bellavista
University of Bologna
Bologna, Italy
paolo.bellavista@unibo.it

ABSTRACT

As the prevalence of Internet-of-Things (IoT) devices becomes more and more dominant, so too do the associated management and security challenges. One such challenge is the exploitation of vulnerable devices for recruitment into botnets, which can be used to carry out Distributed Denial-of-Service (DDoS) attacks. The recent Manufacturer Usage Description (MUD) standard has been proposed as a way to mitigate this problem, by allowing manufacturers to define communication patterns that are permitted for their IoT devices, with enforcement at the gateway home router. In this paper, we present a novel integrated system implementation that uses a MUD manager (osMUD) to parse an extended set of MUD rules, which also allow for rate-limiting of traffic and for setting appropriate thresholds. Additionally, we present two new backends for MUD rule enforcement, one based on *eBPF* and the other based on the Linux standard *iptables*. The evaluation results reported show that these techniques are feasible and effective in protecting against attacks, with minimal impact on legitimate traffic and on the home gateway.

CCS CONCEPTS

• **Security and privacy** → **Denial-of-service attacks**; **Firewalls**;
• **Networks** → *Network monitoring*.

ACM Reference Format:
Angelo Feraudo, Diana Andreea Popescu, Poonam Yadav, Richard Mortier, and Paolo Bellavista. 2024. Mitigating IoT Botnet DDoS Attacks through MUD and eBPF based Traffic Filtering. In *25th International Conference on Distributed Computing and Networking (ICDCN '24), January 04–07, 2024, Chennai, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3631461.3631549

ICDCN '24, January 04–07, 2024, Chennai, India

ACM ISBN 979-8-4007-1673-7/24/01.
https://doi.org/10.1145/3631461.3631549

1 INTRODUCTION

As the use of Internet-of-Things (IoT) devices, in particular in homes and in open deployment environments, continues to increase, numerous associated security challenges have emerged [27, 35]. Our focus lies in addressing the specific issue of IoT device hijacking for botnet recruitment [4]. Such devices are Internet-connected by design, typically through high-bandwidth home broadband connections, and are often not directly and actively used by residents. In the hijacking process of IoT devices, attackers exploit vulnerabilities in the security of the device, gain unauthorized access, and take control of the device. Once the device is under the attacker's control, it can be used to perform a range of malicious activities, such as generating spam and stealing sensitive information.

Bots commonly target IoT devices using brute-force attacks on default or weak login credentials. This enables attackers to gain access to the device and take control of it. Furthermore, IoT devices that are not regularly updated may have known vulnerabilities that attackers can exploit to gain control. As a consequence, once an IoT device is hacked and recruited into a botnet, its intended function will continue to work, but it can also generate massive amounts of network traffic as part of a Distributed Denial-of-Service (DDoS) attack.

Ideally, IoT devices would not be as susceptible to hacking, but as long as they remain vulnerable, one approach to detecting and mitigating the effects of hacked devices is to monitor and control the traffic they generate. The IETF has attempted to address this issue through RFC 8520, which is the "Manufacturer Usage Description Specification" [23, 24]. This specification allows IoT device manufacturers or service providers to provide a machine-readable description of the network interaction in which the device should engage. This information can be used by the home router to enforce restrictions on devices' network activities. Hence, a hacked device cannot be effectively used as part of a botnet because it will not be allowed to generate arbitrary Internet traffic.

Specifically, these MUD files allow manufacturers or service providers to specify to the local network all the permitted incoming and outgoing source/destination addresses, enabling fine-grained traffic filtering of each IoT device. However, it does not allow any further constraint, for example on the rate or mix of traffic generation. Further, implementing the specified constraints on relatively resource-limited home routers requires efficient means to intercept and control network traffic as the number of constraints rises in

proportion to the number of IoT device types in the home. Thus, the MUD's purpose is to restrict traffic while enabling IoT manufacturers to provide long-term support and updates for their devices, facilitating the scalable deployment and management of IoT ecosystems.

This paper advances the state-of-the-art of the related work in the field by presenting the following original contributions:

(1) we propose extensions to the existing MUD standard that enable fine-grained rate-limiting of traffic from controlled devices, alongside means to estimate the relevant parameter values in realistic deployments §2;
(2) we extend the existing osMUD [12] implementation so that it can be deployed within a virtual machine setup for development and testing or within a real-world setup §3; and
(3) we develop and integrate new backends for osMUD by using Linux standard *iptables* firewalls and eBPF [11] that support more efficient implementation of these constraints as expressed in MUD files §4.
(4) we report quantitative performance results on how our design and implementation choices limit high traffic surges caused by a Botnet attack [22], demonstrating their effectiveness §5.

2 EXTENDING MUD

According to the MUD standard [24], a MUD deployment consists of three architectural building blocks: (*i*) the device behaviour description (MUD file), (*ii*) a uniform resource locator (MUD URL) and (*iii*) a mechanism for local management systems that uses the URLs to request description files. In addition, the standard defines two main components that guarantee deployment and use of MUD files: the MUD file server that makes description files available, and the MUD manager, which requests and receives description files to and from the MUD file server. The workflow between these blocks requires the thing or device to emit the MUD URL indicating where the corresponding MUD file is hosted. For this purpose, three protocol extensions have been defined by the standard: (*i*) in DHCP, a reserved option in request packets is used; (*ii*) in X.509 through a certificate extension; and (*iii*) in Link Layer Discovery Protocol (LLDP) by exploiting a subtype defined in RFC 7042. Once a MUD file has been retrieved, the MUD manager validates and enforces the Access Control Lists (ACLs) produced on the corresponding firewall.

2.1 The MUD data model

The MUD data model consists of a YANG-based file serialized in JSON [26]. The YANG language models different types of data, such as configuration data and notifications for network management protocols. There are three YANG schema components that are serialized in a MUD file: (*i*) *ietf-mud* allows to verify MUD file validity as well as the policy to and from the device; (*ii*) *ietf-access-control-list* [20] defines Access Control Lists by using a YANG data model; (*iii*) *ietf-acldns* allows the DNS matching criteria.

The ACL model incorporates essential network and transport layer protocols (e.g., *IPv4* and *TCP*) as its key features. Additionally, it includes actions such as *ACCEPT*, *DROP*, or *REJECT* to specify the router's behavior for the traffic flow. For osMUD, *REJECT* is the default action to deny device communications with domains not in its MUD file. The inclusion of this model in the MUD file allows manufacturers to define whitelists for their devices' required services.

Table 1: Devices' outgoing traffic in a 60-Second window

Category	TCP	TCP Max	UDP	UDP Max
appliances (pkts)	36.7	223.2	5.033	140.28
smart-hubs (pkts)	21.3	1716.8	9.63	152.38
cameras (pkts)	62.2	1471.34	94.40	7863.0
audio (pkts)	52.01	6687.5	293.6	1837.63
home-aut (pkts)	5.20	702.3	14.51	127.5
tv (pkts)	128.9	7560	33.33	729.3
appliances (KB)	2.35	36.76	1.45	24.39
smart-hubs (KB)	2.38	177.39	1.97	38.51
cameras (KB)	75.07	1257.73	81.2	6727.37
audio (KB)	14.62	3430,51	18.76	141291.4
home-aut (KB)	1.08	56.6	3.22	24.96

2.2 A rate-limiting extension

ACLs adhere to the structure defined in [20]. They consist of a series of rules called Access Control Entries (ACEs), each encompassing a set of match criteria and corresponding actions. According to the standard, a rate-limit operation belongs to the action group of an ACE. Therefore, we add rate-limits defined by *packet-rate* and *byte-rate* along with the *forwarding* field in the *actions* object.

Manufacturers may use these fields to define rate-limits using different metrics, such as second, minute, hour, and day. As these fields belong to the action group, manufacturers can set them for each allowed communication enabling a more precise and targeted approach. Furthermore, packet- and byte-rates can vary independently. The implementation of this extension is discussed in Section 3.

2.3 Learning thresholds

To tune our proposed extension for real deployment environments, we must extract information to produce valid upper bounds for outgoing traffic. We have performed this analysis on the widely accepted data gathered by Ren et al. [35]. This includes different PCAP files containing network traffic generated by 81 IoT devices of six different categories: *appliances*, *smart-hubs*, *cameras*, *audio* *home-automation*, and *televisions*. Those PCAP files were processed to compute the amounts of packets and bytes sent and received by each device in a given time window, defined as follows: the first packet timestamp defines the window start time (wst); if the packet falls within the window ($wst + windowSize$), the packet/byte counters corresponding to this window are incremented; otherwise, while that packet cannot be included within a window, the next window start time is updated with the end time of the previous one ($wst_t \leftarrow wst_{t-1} + window\_size$). We analyzed the data using a window of 60 seconds.

Table 1 summarizes the results of our analysis in terms of devices' outgoing traffic in a window of 60 seconds, aligning with our goal of preventing Botnet attacks from IoT devices. The *TCP* and *UDP*

columns refer to the average of TCP and UDP packets/bytes outgoing traffic during device activities, excluding idle periods (e.g., during the night). Similarly, the other columns indicate the network traffic peaks average for each category. For the sake of brevity, we focus on two device categories: *appliances* (devices aiding home activities like cooking and cleaning) and *smart-hubs* (access points or controllers providing Internet access to IoT devices using proprietary or low-range protocols). As highlighted in Table 1, both categories require TCP to work properly.

For each device category analyzed, we establish two MUD files that describe allowed reliable communications and including two volumetric policies, namely "peaks" and "averages". Table 1 reveals that MUD files using "peaks" as outgoing traffic threshold set 250 packets per minute and 40 KB per minute for devices belonging to *appliances* category, and 1720 packets per minute and 180 KB per minute for *smart-hubs* devices. On the other hand, MUD files using "average" outgoing traffic during device activities define 40 packets per minute and 3 KB per minute for *appliances*, and 22 packets per minute and 3 KB per minute for *smart-hubs*. It is worth noting that selected limits concern all the devices in the categories analyzed. However, as described in section 2.2, the proposed MUD data model allows manufacturers to define multiple limits for each device, i.e., one for each allowed destination. Defining a limit per destination will create fine-grained policies for each device, preventing DoS attacks originating from devices whose category aggregate rate limit is higher than their peak rate.

We leverage this analysis to establish the rate limits that we use to validate the effectiveness of our MUD model proposal for both normal and abnormal IoT network traffic.

3 LIBERATING MUD

Open Source MUD (osMUD) [12] is an open-source implementation of MUD manager, developed by a consortium of device manufacturing and network security companies. As shown in Figure 1, the MUD manager is designed to be easily built, deployed, and run on Open Wireless Router (OpenWRT) platform. The implementation requires a customized version of dnsmasq to enable MUD URL extraction, provide network infrastructure services, and minimize resource usage. From the MUD data model perspective, the current implementation does not support MUD file rules for lateral movement attacks (e.g., same-manufacturer, controller, my-controller) [24]. Thus, after compromising a device inside the MUD compliant network, adversaries can progressively move through other systems, searching for targeted key data and assets, which are exploited to gain access to other hosts or applications within the network.

Figure 2 shows the deployment of our prototype: a Linux Virtual Machine (VM) emulating a Linux-based router and running an osMUD version using the extended MUD parser described in section 2.2 (thereafter VMMUD), an external MUD file server, a VM that represents the IoT device (called IoT-1), and a VM that represents traffic from or to the Internet (called SERVER). This design prioritizes easy replication of configurations, settings, and functionalities onto physical hardware.

After finalizing the configurations of this environment, we proceeded to install the osMUD dependencies on the VMMUD environment and deployed it in its generic form. However, although the

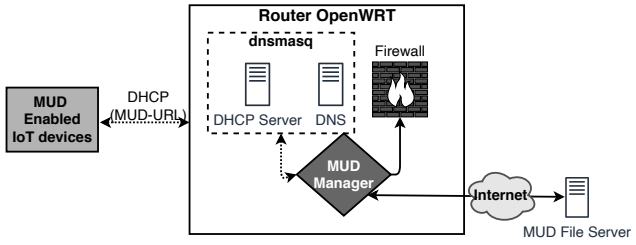


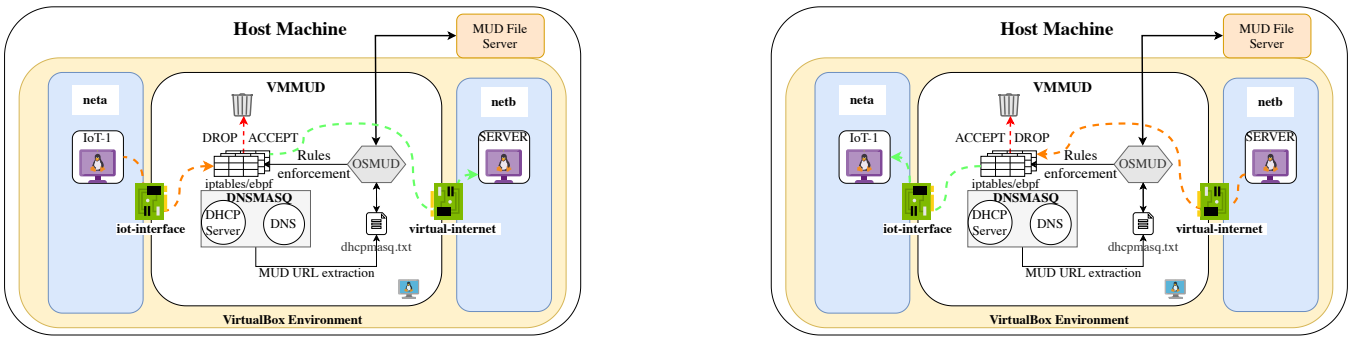
Figure 1: MUD-enabled IoT devices in a home network send and receive traffic to the Internet through the OpenWRT router. The MUD File Server supplies MUD files parsed by the osMUD manager, obtaining rules. The router firewall enforces these rules for IoT devices' traffic.

osMUD designers provided all necessary tools to build this version, there is still no support for a firewall other than OpenWRT. As a result, it becomes imperative to investigate rule enforcement methods for widely used firewalls. To this end, the osMUD Manager structure allows developers to define additional rule enforcement methods by exploiting two main folders containing firewall integration code. The first includes several scripts that focus on rules enforcement and removal in an OpenWRT firewall, while the second folder is provided as a sample code modelling for new firewall integration. Hence, we consider these folders as containers of an *adapter* that enables the MUD manager's independence from the underlying firewall (marked with red dotted line in Figure 3). We develop two new *adapters* which are described in Section 4. The first one, *eBPF-IoT-MUD* leverages eBPF for rule enforcement, and the second adapter leverages *iptables*. Using either of these two new adapters, our system can be deployed on a standard Linux-based router or other constrained devices such as RaspberryPi.

To store and enforce rules produced from the new rate limit fields introduced in Section 2, the osMUD manager parsing procedure needs to be updated accordingly. To achieve this, the MUD file parser (green block in Figure 3) analyses a new string of symbols in the group of action. Moreover, in order to properly store features enabling rate-limit operations, the parser uses an extended version of data types modelling MUD rules. Once the MUD manager receives these customised data types from the parser, it enforces the extended rules on the selected underlying firewall mechanism. It is worth noting that while the emulated environment illustrated in Figure 2 currently focuses on a single compromised IoT device, manufacturers typically define MUD files for each device they produce. Therefore, the considerations that apply to our environment can easily be extended to the DDoS scenario, wherein multiple IoT devices serve as sources of attack.

4 ADAPTING MUD

MUD enforcement is carried out at the router using a backend that enables control over traffic. After introducing eBPF, XDP (eXpress Data Path) and *tc* §4.1, we describe in more detail the structure of an eBPF program §4.2, and then describe how we use eBPF and XDP in one backend implementation §4.3. For comparison purposes, we also describe a second adapter that uses the Linux *iptables* firewall support §4.4.



(a) Output flow (traffic originating from the IoT device)

(b) Input flow (traffic originating from the Internet

Figure 2: Network deployment flows

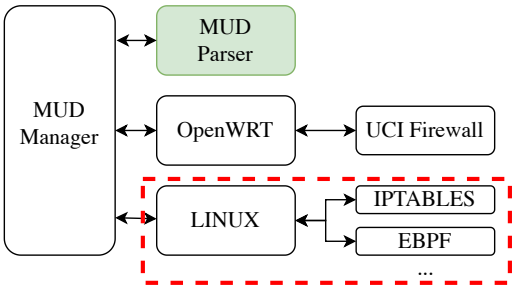


Figure 3: osMUD architectural blocks

4.1 eBPF, XDP, and *tc*

The extended Berkeley Packet Filter (eBPF) is a set of instructions and a virtual machine (VM) for executing programs written in restricted C-language [10, 39]. An eBPF program is "attached" to a specific code path in the kernel. When the code path is traversed, any attached eBPF programs are executed. They can be installed into the Linux kernel without modifying the kernel source code.

Thus, eBPF enables a variety of applications. For instance, an eBPF program can be attached to a network socket to perform tasks such as traffic classification or packet filtering. Furthermore, eBPF is useful for debugging the kernel and carrying out performance analysis, since eBPF programs can access kernel data structures.

The eXpress Data Path (XDP) [19] uses eBPF to enable fast packet processing at the lowest layer of the Linux network stack, immediately after a packet is received. XDP is the lowest layer of the Linux kernel network stack. It is present on the RX path, inside a device's network driver, meaning that it can process only the incoming packets. It allows packet processing at the earliest stage in the network stack, making it suitable for applications such as DDoS mitigation. The context received by an XDP program is defined by the type struct `xdp_md`. The action returned by an XDP program is one of the following actions: the packet is dropped and raise an exception (`XDP_ABORTED`), dropped silently (`XDP_DROP`), passed along to the kernel stack (`XDP_PASS`), retransmit on the same interface (`XDP_TX`) or redirect to another target (for example, to another interface) (`XDP_REDIRECT`).

Finally, the TC layer allows processing both egress traffic (transmitting packets) and ingress traffic (receiving packets). Traffic control policies on Linux are applied at this layer, with different queuing disciplines (`qdisc`) being configured for the different packet queues available in the system. Additionally, there is the possibility to add filters to drop or modify packets.

4.2 eBPF program structure

eBPF programs can be loaded during runtime inside the Linux kernel, and they can interact with different kernel elements, such as kprobes, perf events, sockets and routing tables. An eBPF program can be attached to different *network hooks*, eXpress Data Path (XDP) and Traffic Control (TC). It is executed whenever an event appears on the interface it is attached to. For example, in the case of an eBPF program that does custom packet processing, it will be executed whenever a packet is sent or received.

eBPF programs have a *program type* which defines which layer or subsystem of the Linux kernel the eBPF program is attached to. The type provides information about: (i) what is the input passed to it (its context), (ii) which helper functions it is allowed to use, and (iii) to which kernel hook it will be attached to. For example, `BPF_PROG_TYPE_SOCKET_FILTER` is a program that does socket filtering, while `BPF_PROG_TYPE_XDP` is program attached to the eXpress Data Path hook. A different category of programs are those for kernel tracing and monitoring.

eBPF *maps* are key-value stores, where the keys and values can be user-defined data structures and types. Maps can be accessed from userspace as well as from eBPF programs loaded in the kernel, which makes them a powerful tool for communication between the two. Two common examples are `BPF_MAP_TYPE_HASH` (which is similar to a hash table) and `BPF_MAP_TYPE_ARRAY` (where entries are indexed by a number similar to an array). eBPF programs can use helper functions, such as functions for interacting with maps, for processing packet headers and others.

An eBPF program returns a *code*, which depends on the program type. For example, an XDP program returns a code indicating what action regarding the packet after processing (pass packet, drop packet, redirect etc.). Similarly, TC returns different codes (deliver the packet in the TC queue, drop packet, reclassify packet etc.).

The eBPF in-kernel verifier performs a number of checks. The first check ensures that the eBPF program terminates and does not contain any loops. In the second check, the verifier simulates the execution of the eBPF program one instruction at a time to ensure that register and stack state are valid. Also, if pointer arithmetic

is allowed, all pointer access are checked for type, alignment, and bounds violations. Uninitialized registers cannot be read. Certain registers are marked as unreadable, and checks are carried out to ensure that the read-only frame-pointer is not being written to. Lastly, the verifier restricts which kernel functions can be called from the eBPF programs, and which data structures can be accessed depending on their type.

4.3 eBPF-IoT-MUD adapter

The osMUD manager can use the *eBPF-IoT-MUD adapter* to enforce MUD rules in the lowest layer of the Linux kernel network stack. We implemented *eBPF-IoT-MUD* as an XDP program, attached to the XDP hook. Specifically, there are two programs: `xdpfw_from_device` and `xdpfw_to_device`. The former is inserted on the LAN interface of the home router (Figure 2a), while the latter is injected on the WAN interface of the home router (Figure 2b). The `xdpfw_from_device` is the program used to filter and/or rate-limit the connections made by the IoT devices to the Internet, while the `xdpfw_to_device` is the program used to filter and/or rate-limit the connections from the Internet to the IoT devices. In this paper, we are focusing only on `xdpfw_from_device`, as our aim is to stop DDoS attacks from IoT botnets. We note that `xdpfw_to_device` program functions in the same manner, enforcing MUD rules to prevent unauthorized access from the Internet to the IoT device in this case, thus minimizing the possibility of compromising an IoT device by an outside attacker [32].

We implemented a userspace program which attaches or detaches the programs from the specified network interface, and that can insert the appropriate MUD rules in *allowlists*. There are two allowlists per protocol (*IPv4 v4_allowlist* and *IPv6 v6_allowlist*). The XDP programs use the allowlists to determine whether the packets will be allowed or dropped. The allowlists are implemented using eBPF maps (`BPF_MAP_TYPE_HASH`).

The key and value for the hash map can be seen in Figure 4. The key in the hash map is represented by a structure `flow_ipv4_key` (and correspondingly `flow_ipv6_key`).

The key comprises the destination and source IP addresses for the *from_device* program, or the source and destination IP address for the *to_device* program, the destination port for *from_device*, or the source port for *to_device*, the protocol type and the type of rule (whether it is a *from_device* or a *to_device* rule). The type of rule is implicitly specified in the command line when specifying the type of port (source or destination port). The value in the hash map is represented by a structure `counters_rate`, which records the number of bytes and the number of packets seen a rule. Additionally, the structure has a static maximum byte rate and maximum packet rate extracted from the extended MUD file model when the rule was created and inserted in the map. A default value of zero means that there is no rate-limit.

Whenever a packet arrives, the headers are parsed (layer 2, layer 3 and layer 4) and a key for the maps is built using the header information. The first header parsed is the Ethernet header, the second one is the IP header and, lastly, the transport header. The key is built using information from layer 3 and layer 4, according to the MUD standard. This key (`struct flow_key_ipv4` or `struct flow_key_ipv6`) is searched for in the allowlists. If the key is not found in the allowlists, the packet is dropped (using action

```
struct flow_key_ipv4 {
  __u8 ip_address_src[4];
  __u8 ip_address_dst[4];
  enum addr_type type;
  enum port_protocol proto;
  __u16 port;
};
struct flow_key_ipv6 {
  __u8 ip_address_src[16];
  __u8 ip_address_dst[16];
  enum addr_type type;
  enum port_protocol proto;
  __u16 port;
};
struct counters_rate {
  __u64 packets;
  __u64 bytes;
  __u64 max_pkt_rate;
  __u64 max_bytes_rate;
};
```

Figure 4: eBPF-IoT-MUD maps

`XDP_DROP`). If it is found, the following actions will take place. First, the flow statistics for the matching entry (MUD rule) are updated according to the time window they fall in. If the current time window has expired, the statistics are first reset, and only then are updated. If the current time window has not expired, the flow statistics are updated directly. Next, the conditions for the maximum packet rate and/or maximum byte rate are checked. If these are not met (the current flow statistics for the current window are above the maximum thresholds), the packet is dropped (action `XDP_DROP`), else it is passed along (action `XDP_PASS`).

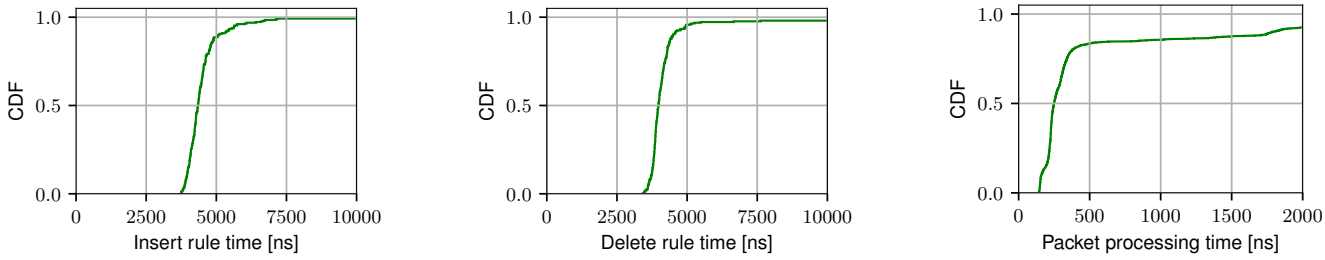
Moreover, an eBPF map of type `BPF_MAP_TYPE_ARRAY` is used to store the current time to determine whether the packet falls in the current time window or whether the time window should be updated. Also, another eBPF map of type `BPF_MAP_TYPE_HASH` is used to set the window size for updating statistics (by default one minute) from the userspace program.

4.4 *iptables* firewall adapter

Referring to Figure 3, in this section we describe the *iptables adapter* that we implemented by analyzing the firewall integration with osMUD. The *iptables* firewall relies on two main concepts: tables and chains, where tables are made of chains, while chains are made of rules. From our perspective, the rules included in a MUD file are enforced in the FORWARD chain, which handles filtering procedures on packets passing through the firewall.

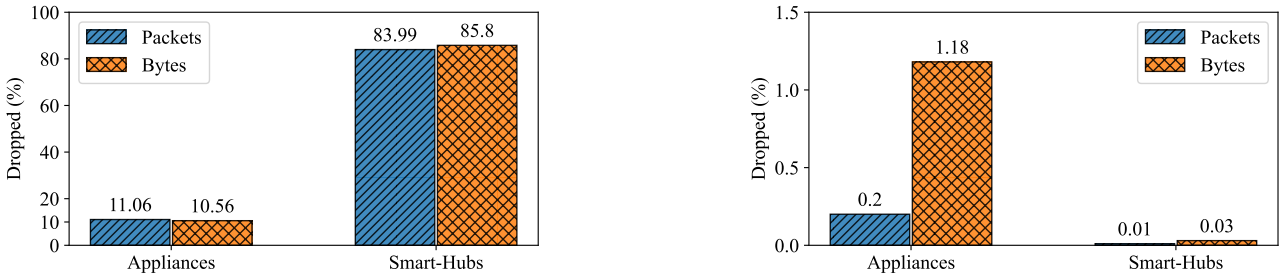
Thus, after parsing a MUD file, the osMUD manager produces two types of rules: ACCEPT and DROP-ALL. The former represents the allowed communications described by device manufacturers, while the latter is added by the osMUD manager to blacklist all the other domains not described in the MUD file. We employ the custom chain concept provided by the *iptables* tool to improve rules management. It allows us to define programs acting only on MUD-related policies, thus without interfering with those system-related.

For everyday IoT device categories, rate-limiting can further improve the protection against DoS attacks. To support these policies in the MUD model, we propose an extension allowing manufacturers to define device behaviours based on the traffic volume. Towards this end, our *iptables adapter* uses the *hashlimit* module enabling the rate-limit match for a group of connections.



(a) Rules insertion time in the IPv4 allowlist (b) Rules deletion time in the IPv4 allowlist (c) Packet processing datapath

Figure 5: eBPF-IoT-MUD evaluation



(a) Average limit (b) Peak limit

Figure 6: Packet and byte drop rates under normal conditions

5 EVALUATION

We have extensively evaluated our system implementation by using the setup in Figure 2. The IoT device and the osMUD manager each run in a separate virtual machine, hosted on a MacBook Pro Intel Core i5 with 8GB RAM. Both *iptables* and *eBPF-MUD-IoT* firewalls are configured on VMMUD, which acts as a bridge between two networks: *neta* and *netb*. The former contains an IoT device named IoT-1, i.e. VM that emulates a resource-constrained device, while the latter contains a server interacting with VMs of *neta*.

5.1 Rule management via eBPF

In this section, we present the results for micro-benchmarks we run on eBPF-IoT-MUD in Table 2. While we expect that most of the MUD files parsing and rule enforcement to happen when the router is first initialised or rebooted, new IoT devices may become part of the IoT network dynamically. Thus, we evaluate the time it takes to insert and to delete rules in the eBPF maps, by inserting 255 rules, and then deleting the 255 rules inserted. For these experiments, we used a Lubuntu 20.04 virtual machine running in VirtualBox 6.1.14 on a laptop with an Intel quad-core i7 processor and 16GiB RAM.

Figure 5a presents the CDF for rules insertion time, with an average of 4533.43 ns, while Figure 5b presents the CDF for rules deletion time, with an average of 4576.58 ns. We also evaluated the packet processing datapath with one rule inserted for a ssh connection from outside the VM to the VM, and we sent different Linux commands on the ssh connection. Figure 5c presents the CDF for packet processing datapath times, with an average of 556.15 ns.

We measure the packet latency using the experimental setup in Figure 2, which runs on a laptop with an Intel quad-core i7 processor and 16 GiB RAM. The experiment runs `netperf` [18] with `TCP_RR` on the client and sending traffic to the server across the router. On the router we inserted the MUD rules that allow the traffic to flow from the client to the server. We run netperf in three experiments: (*i*) the baseline (no firewall); (*ii*) iptables firewall; (*iii*) eBPF-IoT-MUD firewall, and we compare the packet latency and transaction rate. [18] runs five consecutive tests of 100 seconds for each experiment. The results are presented in Table 3. There is no noticeable impact to packet latency and transaction rate compared to the baseline (no firewall) when using a firewall (iptables or eBPF-IoT-MUD), with the minimum packet latency being 186 us for baseline, 187 us with iptables and 188 us with eBPF-IoT-MUD. The transaction rate is 2714 transactions/s for baseline, 2892.51 transactions/s with iptables, and 2915.91 transactions/s with eBPF-IoT-MUD. These measurements show that using either firewalls does not add any additional overhead to packet processing.

5.2 Rate limiting impact on normal traffic

Based on the analysis we carried out in the previous section, we defined two MUD files, i.e., using "peaks" and "averages" policies, for each category considered, i.e. *appliances* and *smart-hubs*. We validate these MUD files using the dataset provided in [37] to emulate IoT device network traffic in normal conditions. The dataset includes traffic traces of 28 different IoT devices over a period of 6 months, of which only two weeks are openly available. We selected two devices for each category analyzed, i.e. a Wi-Fi printer

Table 2: eBPF-MUD-IoT performance (*ns*)

Test	Min	Med	Avg	90th	Max	Std.dev.
Insert	3740	4347.4	4533.4	5086.6	15037	1019.61
Delete	3426	3978	4576.6	4504.2	49581	4145.33
Datapath	140	249	556.2	1787.2	24915	1089.96

Table 3: Packet Latency (μs) and Transactions per Second

Experiment	Min	Avg	Max	Std.dev.	Txns/s
No firewall	186.4	368.21	247286	1223.44	2714.01
iptables	187	345.5	203594	606.81	2892.51
eBPF-IoT-MUD	188	342.72	258549	580.61	2915.91

as *appliance* and an Amazon Echo as *smart-hub*. To make the traffic traces conform to our environment (Figure 2), we used *tcprewrite* tool from the *tcpreplay* suite [38].

Once we processed these traces, we first generated and enforced MUD rules for *iptables* firewall on the VMMUD machine using the osMUD manager. Secondly, we started the collection of iptables statistics in terms of total packets and bytes sent/dropped originating from the IoT device. Finally, to replay the devices' network traffic, we used `tcpreplay` on IoT-1 VM as shown in Figure 2a. We replayed three days of the TCP network traffic from each device selected for each MUD file defined, aiming to understand whether the policies generated allow the corresponding IoT device to function normally.

Figure 6 illustrates the percentage of packets and bytes dropped after applying these rate-limits. As shown in Figure 6a, the *smart-hubs* MUD file using averages as rate-limits is hardly usable in normal device conditions, as it blocks most of the outgoing traffic (over 80%). Similarly, average rate-limits affect the *appliances* with a dropping rate of around 11%, which might be a problem, especially during device updates. Conversely, as shown in Figure 6b enforcing MUD rules using peaks as rate-limits does not affect the overall IoT traffic. In such a scenario, the packet drop rate remains below 1.5% and 0.05% in *appliances* and *smart-hubs*, respectively. Hence, to limit traffic volume for both categories, we decided to use MUD files defining peaks-based rate-limits. Furthermore, this choice is motivated by the fact that devices adopting TCP in output communications tend to reach the peak rapidly. On the one hand, it might be related to user activities, which may cause new iterations of the three-way handshake procedure. On the other hand, other key attributes that make the TCP protocol reliable, e.g., packet retransmission and congestion control, might represent the trigger of traffic peaks.

5.3 Rate limiting impact on abnormal traffic

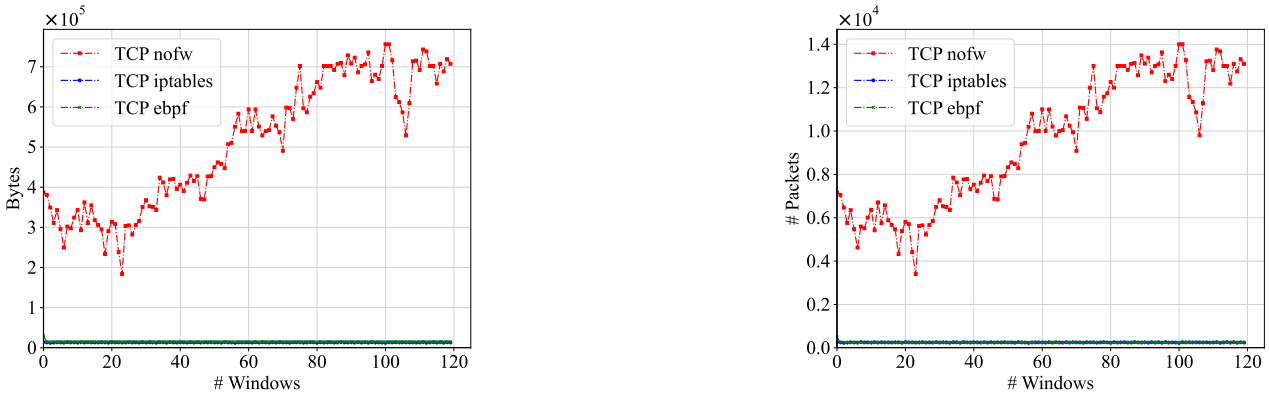
To emulate abnormal IoT behaviors, we used the Network TON_IoT dataset [31] providing network traces of several offensive systems conducting multiple attack scenarios, such as DoS, Ransomware, and injections attacks. We selected those referring to a DoS attack and merged them into a single trace. Next, we used the `tcprewrite` tool to rewrite the IP addresses in the trace to correspond to those of our test setup (Figure 2). As described previously, to replay the traces we used `tcpreplay` on IoT-1, thus becoming the originator of the SYN FLOOD attack, a form of DoS attack based on multiple SYN Request iterations. The traffic flow has been highlighted in Figure 2a. Once we processed the abnormal network traces, we enforced the policies comprised in the MUD files previously selected. To test the effectiveness of the rate-limits against this attack, service ports do not appear in generated MUD files. Figure 7 shows the results for the *appliances* group while those *smart hubs* related are illustrated in Figure 8. The red line shows the network traffic without rate-limiting, while the blue and green line represents the traffic after our shaping (using the rate-limits from the MUD files for the respective IoT device categories).

We run the experiment using the two firewall backends we implemented, based on iptables and eBPF-IoT-MUD. For iptables, we use an additional parameter, `-limit-burst`, that allows small traffic bursts (in our case bursts of additional 5 packets). For eBPF-IoT-MUD, when the rate-limit is reached, the firewall starts dropping the incoming packets. This implementation difference explains why the traffic line in the Figure 7 and Figure 8 is straight when using eBPF-IoT-MUD, while when using iptables it is less smooth, but still meets the rate-limit with bursting allowed.

5.4 Rate limiting discussion

Although the results demonstrate the effectiveness of enforcing peak-based rules selected in our analysis, the scenario we presented assumes that the administrator - i.e., the user setting the MUD rules - may not have a comprehensive knowledge of the traffic flow generated in each communication that a device may engage in. In a more general case, where manufacturers define the MUD files, it becomes possible to enable a more precise approach to set rate limiting fields. For instance, a manufacturer can specify peak-based limits for every device they produce and for each outgoing communication (destination) from the device. This would help decrease the impact on device normal traffic and prevent DoS attacks from devices whose category aggregate rate limit is higher than their peak rate. We note that some DoS attacks might use traffic shaping, making them undetectable through rate limits, in which case other botnet detection approaches are needed.

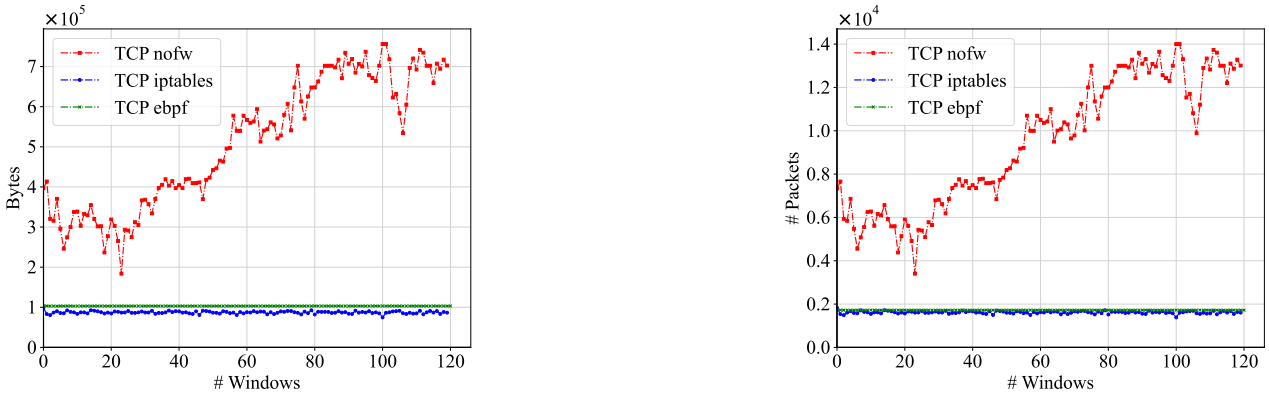
One of our key contributions in this research is the inclusion of a rate limit field for a MUD rule. The value of these rate limits can be determined either by the users or the device manufacturers, although this is a separate research problem that falls outside the scope of our paper. Our focus, instead, is on demonstrating the effectiveness of enforcing these rate limits at the device level as a means of managing network interactions and bandwidth usage for IoT devices. By doing so, our proposed solution allows both knowledgeable users and manufacturers to mitigate DoS attacks that may target the services required by the device to operate normally, while minimizing the impact on the device's regular traffic. It is important to note that while our approach can be integrated with other solutions and techniques, it is not intended to be a comprehensive solution on its own, but rather a vital piece in the broader fight against DDoS attacks.



(a) Appliances rate-limit bytes per minute

(b) Appliances rate-limit packets per minute

Figure 7: Appliances



(a) Smart hubs rate-limit bytes per minute

(b) Smart hubs rate-limit packets per minute

Figure 8: Smart hubs

6 COMPARATIVE STUDY OF EXISTING SOLUTIONS

To provide a better understanding of the contributions made in our work, we summarized in Table 4 the key characteristics of existing solutions that use MUD rules or whitelists to enhance security in IoT networks. Specifically, it emphasizes the following aspects: the network setting, the inclusion of rate-limiting support, adherence to the MUD standard, provision of an anomaly detector, and underlying technology used for rule enforcement. [16] presents a design that leverages programmable switches to enforce MUD rules in enterprise industrial networks, focusing on scalability. The implementation uses the P4 programming language for scaling the number of MUD rules that can be enforced in a network, but it does include a MUD manager in the current design. Furthermore, the architecture does not enforce rate limits to protect against DDoS Botnet attacks. Our inplementation does not require programmable switches, and it can run on legacy routers or Raspberry Pi, being ready to deploy in any network. [14] uses an ML-based anomaly detection that leverages OpenFlow to enforce MUD rules. An SDN controller collects traffic statistics for each rule, and the statistics sever as input to an online anomaly detector. The tool can detect several volumetric attacks from malicious IoT devices. [34] also enforces MUD rules in OpenFlow switches, however the focus of their work is determining how many rules can be installed at a switch, and how a proactive or reactive approach reacts to IoT network traffic. [13] leverages an online information aggregator (VirusTotal) to determine whether a destination (IP address) should be added to a whitelist or the communication should not be allowed. Their solution does not use the MUD standard and runs on OpenWRT router, and does not enforce rate limiting. IoTrim [28] finds the set of destinations contacted by an IoT device, and determines which of these destinations are needed to maintain device functionality. These destinations are usually related to the manufacturer, support or third-party analytics. If a destination is not needed for the device to operate normally, this destination will be blocked. The purpose of the work is to limit the information exposure of the IoT user because of privacy concerns. In our work, we present an integrated system that enforces MUD rules in order to avoid IoT devices contacting destinations that are not in the MUD file.

Table 4: Existing solutions

Work	Network setting	Rate-limiting Support	Standard Compliant	Anomaly Detector	Rule Enforcement Tech
Our	Smart homes	Yes (MUD standard ext.)	Yes	No	eBPF/iptables
[16]	Enterprise industrial	No	Yes (no MUD manager)	No	P4
[14]	Smart homes	No	Yes (no MUD manager)	Yes (ML-based)	OpenFlow
[34]	Enterprise industrial	No	Yes (no MUD manager)	No	OpenFlow
[13]	Smart homes	No	No	Yes (Virus Total)	OpenWRT
[28]	Smart homes	No	No	No (whitelist)	iptables

Furthermore, based on the extended MUD file with rate limits, we thwart volumetric attacks such as DDoS attacks towards legitimate cloud destinations. Conversely, other works [7, 13, 14, 34] focus on traffic filtering based on source/destination address of the packets, and do not consider bandwidth or data rate.

The idea of including bandwidth (packet rate and byte rate as a part of the MUD) has been discussed in a IETF RFC draft [25]. Andalibi et al. [2] discussed the advantage of peak request rate as a part of MUD when deploying MUD in fog computing environments, but did not provide an implementation. Similarly [8, 17] have presented the challenges associated with the implementation of traffic rate control at the router and firewall levels in MUD-compliant networks. Currently, to the best of our knowledge, there is no MUD implementation that evaluates rate-limitation enforced by MUD in MUD-compliant networks while leveraging eBPF and XDP for traffic filtering. Our work shows and evaluates the advantages of including rate-limiting policies in MUD files, proving their feasibility with quantitative performance results on Linux-based routers using osMUD.

7 RELATED WORK

Different organisations have developed prototypes of MUD manager (controller) implementations (middleware) in the last few years to allow seamless enforcement of the filtering rules specified in MUD files [12, 33]. Recent research surveys [3, 9, 17] discuss their limitations, challenges, and directions for future research [25]. One of the recognized challenges is the manufacturer's resistance to support MUD files and to make devices MUD-compliant in real deployment environments, in particular in absence of recognized IoT device usage patterns. One way to solve this problem is to create a behavioral fingerprint of the devices in the real environments and then use those fingerprints to design MUD files [29]. There have been significant efforts for creating behaviour fingerprints of IoT devices using their network traffic [21, 40]: the fingerprints are used to detect abnormal behaviours and take proper actions, either by limiting the incoming/outgoing traffic or by switching off the suspected anomalous devices. Our system can work in conjunction with IoT device identification systems that use machine learning models for device identification [21]. As a first step, a manufacturer can provide a MUD file with rate limits already defined for an IoT device, or that has been built using a tool such as [6, 15]. These rate limits can be further customized based on observed usage patterns using ML models for device identification. Furthermore, since smart homes can have different network connections and bandwidths, the network infrastructure conditions can further help to customize the MUD file. In this direction, our system exploits the eBPF technology, which has applications in networking, tracing and profiling, security, and monitoring. XDP ensures efficient packet processing on the RX datapath, facilitating the construction of DDoS defences. Studies such as [36] discuss the performance of packet filtering with eBPF. In [30] the authors present a hybrid system which uses XDP for traffic sampling and aggregation, and offloads DDoS mitigation rules to smartNICs. [1] uses eBPF and XDP for implementing traffic monitoring applications. [5] presents how XDP is integrated in the DDoS mitigation pipelines at Cloudflare to perform traffic analysis, aggregation, reaction and implement mitigation rules. Our work, eBPF-IoT-MUD, is the first to use eBPF in a smart home context, and implements a firewall using eBPF and XDP. Our custom system is integrated with the osMUD manager that provides it with the MUD rules to be enforced in order to prevent IoT devices from performing DDoS attacks on Internet destinations.

8 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we present the design of an extension to the MUD standard that rate-limits the outgoing traffic of IoT devices. This approach helps to ensure that IoT devices do not exceed their allocated bandwidth limits and do not negatively impact the performance of other devices on the network. We demonstrate a procedure to identify these rate-limits for consumer IoT devices. Next, we present the implementation and evaluation of an end-to-end system to enforce MUD rules on the home router through two new firewall backends, based on Linux standard iptables, and a custom system that uses eBPF (i.e., eBPF-IoT-MUD). Finally, we present novel experimental results evaluating the performance of our MUD extension and system prototype. We achieved this by using the osMUD manager and implementing custom MUD rules in iptables and eBPF-IoT-MUD, enforced at routers. In the future, we aim to recreate the emulated scenario on actual Linux-based systems like Raspberry Pi. This will serve as a versatile test-bed for executing various types of IoT-based attacks and evaluating our solution's performance against existing ones. Additionally, we intend to investigate methods that accurately identify rate limit values, by leveraging the eBPF technology.

ACKNOWLEDGMENTS

This work was supported in part by RCUK grants EP/T022493/1, EP/R03351X/1, EP/X040518/1 and EP/Y019229/1.

REFERENCES

[1] Marcelo Abranches, Oliver Michel, Eric Keller, and Stefan Schmid. 2021. Efficient Network Monitoring Applications in the Kernel with eBPF and XDP. In *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 28–34. https://doi.org/10.1109/NFV-SDN53031.2021.9665095

[2] Vafa Andalibi, DongInn Kim, and L. Jean Camp. 2019. Throwing MUD into the FOG: Defending IoT and Fog by expanding MUD to Fog network. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotedge19/presentation/andalibi

[3] Vafa Andalibi, Eliot Lear, DongInn Kim, and L. Jean Camp. 2021. On the Analysis of MUD-Files' Interactions, Conflicts, and Configuration Requirements Before Deployment. arXiv:2107.06372 [cs.CR]

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[5] Gilberto Bertin. 2017. XDP in practice:integrating XDP into our DDoS mitigation pipeline. https://legacy.netdevconf.info/2.1/papers/Gilberto_Bertin_XDP_in_practice.pdf.

[6] Anat Bremler-Barr, Bar Meyuhas, and Ran Shister. 2022. One MUD to Rule Them All: IoT Location Impact. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 1–5. https://doi.org/10.1109/NOMS54207.2022.9789828

[7] Gabriel Brown. 2019. Regulation of IoT Device Communications Using MUD Files and IPTables. https://gitlab.com/columbia.irt/riot/tree/master/Fall2019/CombineRouter

[8] Angelo Feraudo, Poonam Yadav, Richard Mortier, Paolo Bellavista, and Jon Crowcroft. 2020. SoK: Beyond IoT MUD Deployments - Challenges and Future Directions. *CoRR* abs/2004.08003 (2020). arXiv:2004.08003 https://arxiv.org/abs/2004.08003

[9] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. 2020. CoLearn: Enabling Federated Learning in MUD-Compliant IoT Edge Networks. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking* (Heraklion, Greece) *(EdgeSys '20)*. Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/3378679.3394528

[10] Matt Fleming. 2020. A thorough introduction to eBPF. https://lwn.net/Articles/740157/.

[11] Matt Fleming. 2021. A thorough introduction to eBPF. https://lwn.net/Articles/740157/.

[12] OSMUD Group. 2018. Open Source Manufacturer Usage Specification. https://osmud.org.

[13] Javid Habibi, Daniele Midi, Anand Mudgerikar, and Elisa Bertino. 2017. Heimdall: Mitigating the Internet of Insecure Things. *IEEE Internet of Things Journal* 4, 4 (2017), 968–978. https://doi.org/10.1109/JIOT.2017.2704093

[14] Ayyoob Hamza, Hassan Habibi Gharakheili, Theophilus A. Benson, and Vijay Sivaraman. 2019. Detecting Volumetric Attacks on LoT Devices via SDN-Based Monitoring of MUD Activity. In *Proceedings of the 2019 ACM Symposium on SDN Research* (San Jose, CA, USA) *(SOSR '19)*. Association for Computing Machinery, New York, NY, USA, 36–48. https://doi.org/10.1145/3314148.3314352

[15] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. 2018. Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (Budapest, Hungary) *(IoT S&P '18)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3229565.3229566

[16] S A Harish, Suvrima Datta, Hemanth Kothapalli, Praveen Tammana, Achmad Basuki, Kotaro Kataoka, Selvakumar Manickam, U. Venkanna, and Yung-Wey Chong. 2023. Scaling IoT MUD Enforcement using Programmable Data Planes. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 1–9. https://doi.org/10.1109/NOMS56928.2023.10154376

[17] José L. Hernández-Ramos, Sara N. Matheu, Angelo Feraudo, Gianmarco Baldini, Jorge Bernal Bernabe, Poonam Yadav, Antonio Skarmeta, and Paolo Bellavista. 2021. Defining the Behavior of IoT Devices Through the MUD Standard: Review, Challenges, and Research Directions. *IEEE Access* 9 (2021), 126265–126285. https://doi.org/10.1109/ACCESS.2021.3111477

[18] Hewlett-Packard. 2022. Netperf. https://hewlettpackard.github.io/netperf/.

[19] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The EXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (Heraklion, Greece) *(CoNEXT '18)*. Association for Computing Machinery, New York, NY, USA, 54–66. https://doi.org/10.1145/3281411.3281443

[20] Mahesh Jethanandani, Sonal Agarwal, Lisa Huang, and Dana Blair. 2019. YANG Data Model for Network Access Control Lists (ACLs). RFC 8519. https://doi.org/10.17487/RFC8519

[21] Roman Kolcun, Diana Andreea Popescu, Vadim Safronov, Poonam Yadav, Anna Maria Mandalari, Richard Mortier, and Hamed Haddadi. 2021. Revisiting IoT Device Identification. *TMA'21* abs/2107.07818 (2021). arXiv:2107.07818 https://arxiv.org/abs/2107.07818

[22] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and other botnets. *Computer* 50, 7 (2017), 80–84.

[23] Prabhakar Krishnan, Kurunandan Jain, Rajkumar Buyya, Pandi Vijayakumar, Anand Nayyar, Muhammad Bilal, and Houbing Song. 2022. MUD-Based Behavioral Profiling Security Framework for Software-Defined IoT Networks. *IEEE Internet of Things Journal* 9, 9 (2022), 6611–6622. https://doi.org/10.1109/JIOT.2021.3113577

[24] Eliot Lear, Ralph Droms, and Dan Romascanu. 2019. Manufacturer Usage Description Specification. RFC 8520. https://doi.org/10.17487/RFC8520

[25] Lear, E and Henry, J. 2020. Bandwidth Profiling Extensions for MUD. https://tools.ietf.org/html/draft-lear-opsawg-mud-bw-profile-01

[26] Ladislav Lhotka. 2016. JSON Encoding of Data Modeled with YANG. RFC 7951. https://doi.org/10.17487/RFC7951

[27] Franco Loi, Arunan Sivanathan, Hassan Habibi Gharakheili, Adam Radford, and Vijay Sivaraman. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (Dallas, Texas, USA) *(IoTS&P '17)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3139937.3139938

[28] Anna Maria Mandalari, Daniel J Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David Choffnes. 2021. Blocking Without Breaking: Identification and Mitigation of Non-Essential IoT Traffic. *Proceedings on Privacy Enhancing Technologies* 4 (2021), 369–388.

[29] Guðni Matthíasson, Alberto Giaretta, and Nicola Dragoni. 2020. IoT Device Profiling: From MUD Files to S×C Contracts. In *Open Identity Summit 2020*, Heiko Roßnagel, Christian H. Schunck, Sebastian Mödersheim, and Detlef Hühnlein (Eds.). Gesellschaft für Informatik e.V., Bonn, 143–154. https://doi.org/10.18420/ois2020_12

[30] Sebastiano Miano, Roberto Doriguzzi-Corin, Fulvio Risso, Domenico Siracusa, and Raffaele Sommese. 2019. Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case. *IEEE Access* 7 (2019), 107161–107170. https://doi.org/10.1109/ACCESS.2019.2933491

[31] Nour Moustafa. 2021. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society* 72 (2021), 102994. https://doi.org/10.1016/j.scs.2021.102994

[32] Mukrimah Nawir, Amiza Amir, Naimah Yaakob, and Ong Bi Lynn. 2016. Internet of Things (IoT): Taxonomy of security attacks. In *2016 3rd International Conference on Electronic Design (ICED)*. 321–326. https://doi.org/10.1109/ICED.2016.7804660

[33] NIST. 2019. Securing Small-Business and Home Internet of Things Devices: NIST SP 1800-15.

[34] Mudumbai Ranganathan, Douglas Montgomery, and Omar El Mimouni. 2019. Soft MUD: Implementing Manufacturer Usage Descriptions on OpenFlow SDN Switches. ThinkMind, Valencia, ES. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927289

[35] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 267–279. https://doi.org/10.1145/3355369.3355577

[36] Dominik Scholz, Daniel Raumer, Paul Emmerich, Alexander Kurtz, Krzysztof Lesiak, and Georg Carle. 2018. Performance Implications of Packet Filtering with Linux eBPF. In *2018 30th International Teletraffic Congress (ITC 30)*, Vol. 01. 209–217. https://doi.org/10.1109/ITC30.2018.00039

[37] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2019. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2019), 1745–1759. https://doi.org/10.1109/TMC.2018.2866249

[38] Turner, Aaron and Klassen, Fred. 2021. Tcpreplay. https://tcpreplay.appneta.com/wiki/tcpreplay.html

[39] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, and Luiz F. M. Vieira. 2020. Fast Packet Processing with EBPF and XDP: Concepts, Code, Challenges, and Applications. *ACM Comput. Surv.* 53, 1, Article 16 (Feb. 2020), 36 pages. https://doi.org/10.1145/3371038

[40] Poonam Yadav, Angelo Feraudo, Budi Arief, Siamak F. Shahandashti, and Vassilios G. Vassilakis. 2020. Position Paper: A Systematic Framework for Categorising IoT Device Fingerprinting Mechanisms. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (Virtual Event, Japan) *(AIChallengeIoT '20)*. Association for Computing Machinery, New York, NY, USA, 62–68. https://doi.org/10.1145/3417313.3429384