Using eBPF to inject IPv6 Extension Headers

Justin Iurman*, **Eric Vyncke**‡, **Benoit Donnet***,
∗ Université de Liège, Montefiore Institute – Belgium
‡ Cisco

Abstract

While IPv6 was already standardized in the 90's, only the last decade has seen a growth in its global adoption. In addition to dealing with IPv4 addresses exhaustion, IPv6 comes with a mechanism, called IPv6 Extension Header (IPv6 EH), allowing the protocol to be more flexible and extensible. However, according to recent studies trying to evaluate the survivability of IPv6 EHs, most of these IPv6 EHs do not easily survive over the global Internet (i.e., outside limited/controlled domains), which might be problematic if a specific service requires such an IPv6 EH. This paper discusses a use case in which an operator needs to test the survivability of specific IPv6 EHs, as a proof-of-concept, prior to service deployment. This paper explains how an eBPF program can find a suitable usage in building the service quickly as a proof-of-concept, by easily injecting IPv6 EHs in network traffic and without the need to modify existing tools or the kernel. We also evaluate our program for IPv6 EHs injection in terms of throughput.

Introduction

During the last decade, IPv6 has been more and more adopted [13]. If IPv6 allows for dealing with IPv4 address exhaustion [19], it also comes with a mechanism, called IPv6 *Extension Header* (IPv6 EH) [6, 4], that leads to more flexibility and innovation. Examples of such innovations based on IPv6 EHs are Segment Routing with IPv6 as forwarding plane [10, 11] and In-Situ Operations, Administration, and Maintenance (IOAM) [3]. The purpose of IPv6 EHs is to extend IPv6 without any modification to the core protocol. IPv6 EHs form a chain, using the IPv6 *Next Header* field, and are placed between the IPv6 header and the upper-layer protocol header. While new IPv6 EHs might be defined in the future, the current list mainly includes the `Hop-by-Hop Options` Header, the `Destination Options` Header, the `Routing` Header, the `Fragment` Header, the `Encapsulating Security Payload`, and the `Authentication` Header [6, 4].

Up to now, a few efforts have been made in assessing how operators process IPv6 EHs. RFC7045 [4] provides guidelines on how IPv6 EHs should be transmitted, also with a focus on middleboxes influence on the traffic. Gont et al. [12] analyze the security implication of IPv6 EHs and the implications of discarding or filtering packets. Hendrikx et al. [14] state that dropping all traffic containing any IPv6 EH is the de facto rule applied by any operator, for security reasons. To support their claim, they perform a limited measurement campaign on a national research network (CSNET) and a campus network (UTNET). In the same spirit, Padurean et al. [36] run large-scale `traceroute` measurements to find the presence of Segment Routing [10, 11] with IPv6 as forwarding plane deployment. They reported no presence of such a deployment, probably due to IPv6 EH filtering. Elkins et al. [7] focus on the Performance and Diagnostic (PDM) [8] `Destination Options` Header which provides sequence numbers and timing information as a basis for measurements. While they do not report any drop when measurements are performed between hosting services, they observe some drops when measurements are sent towards Alexa top 1M domains. Huston and Damas [18] report an improvement, over the years, in processing the IPv6 `Fragment` Header. They also notice that `Destination Options` Header and `Hop-by-Hop Options` Header are generally not supported on public Internet infrastructures. Finally, Léas et al. [37, 28] test IPv6 EHs in full mesh through multiple vantage points at the edge. Among others, they report that path traversal diminishes as the size of IPv6 EH increases. Generally speaking, most of the IPv6 EHs do not easily survive[1] over the global Internet, which goes against the flexibility purpose of IPv6 EHs.

In parallel to this, recent years have seen a growing interest in using eBPF to deploy network services. For instance, Tran and Bonaventure [34] leveraged eBPF to support use cases through injection of options in Multipath TCP. Xhonneux et al. [39] developed a framework allowing network operators to encode their own network functions as eBPF code that is automatically executed while processing specific packets. More recently, eBPF has also found an usage in telemetry [16], load balancing in clouds [27], or in security [38].

This paper discusses a service requiring to fragment large `DNS` packets and relies on the IPv6 `Fragment` Header, as an example. It has been shown that such an IPv6 EH does not easily survive over the Internet (see Léas et al. [28], for instance). We therefore believe that operators must first test the survivability of an IPv6 EH in the context of their service prior to any large-scale deployment.

[1]We define the IPv6 EHs survivability as the capacity of IPv6 EHs to traverse the Internet and arrive unmodified at the destination.

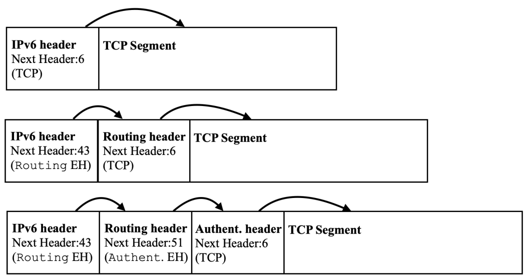

Figure 1: Example of a chain of pointers formed by the `Next Header` field in IPv6.

To ease the deployment of such a proof-of-concept, this paper introduces an eBPF [33] program (available online [22]) which allows to inject on the fly any IPv6 EH in existing traffic. With such a program, it is easy to inject dedicated IPv6 EHs on existing traffic, without modifying end-hosts. Doing so, an operator can easily test the survivability of its service. This paper also evaluates the performance of the eBPF program.

The remainder of this paper is organized as follows: next section provides some background with respect to IPv6 EHs; then, we describe how IPv6 EHs are injected with eBPF; after that, we discuss a use case in which IPv6 EHs can find a suitable usage and we evaluate our implementation; finally, we conclude this paper by summarizing its main achievements.

Background

The purpose of IPv6 EHs is to extend IPv6 without any modification to the core protocol. The IPv6 `Next Header` field specifies which upper-layer protocol comes after the IPv6 header. All IPv6 EHs share a common field in their respective headers, namely a `Next Header` field, whose name and purpose are identical to the one in the IPv6 header. This design allows for a chaining mechanism. Fig. 1 illustrates how it works with three examples: the first one represents a `TCP` segment, the second one represents a `Routing` Header followed by a `TCP` segment, and the third one represents a `Routing` Header followed by an `Authentication` Header followed by a `TCP` segment.

The Internet Assigned Numbers Authority (IANA) currently defines the following IPv6 EHs [2]: the `Hop-by-Hop Options` Header, the `Destination Options` Header, the `Routing` Header, the `Fragment` Header, the `Encapsulating Security Payload`, the `Authentication` Header, the `Mobility` Header, the `Host Identity Protocol` Header, and the `Shim6 Protocol` Header. The `Hop-by-Hop Options` Header is used to carry optional information, also called *Options*, that may be examined and processed by every node along a packet's delivery path, while the `Destination Options` Header is used to carry optional information to be examined only by the packet's destination. An example of `Hop-by-Hop Options` Header or `Destination Options` Header usage is In-Situ Operations, Administration, and Maintenance (IOAM) [3]. With IOAM, telemetry data is carried within packets rather than being sent through

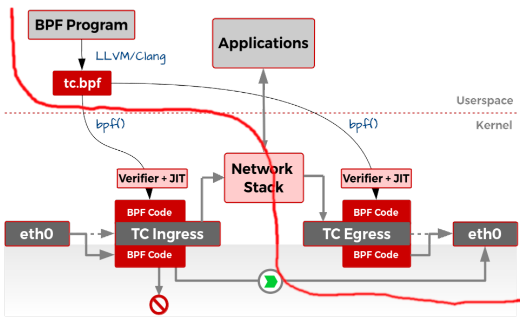

Figure 2: Overview of how tc works with eBPF [35].

```
# tc qdisc add dev eth0 clsact
# tc filter add dev eth0 egress bpf da obj ebpf_program.o sec
     section
```

Listing 1: Commands to attach an eBPF program to an interface.

packets specifically dedicated to that. The IOAM traffic is thus embedded in data traffic, but not part of the packet payload. The `Routing` Header is used by an IPv6 source to list one or more intermediate nodes to go through on the way to a packet's destination (i.e., to steer a packet), and has several types defined: Source route (type 0) and Nimrod (type 1) [5] which are both deprecated, Mobility support (type 2) [24], RPL (type 3) [17], and Segment Routing (type 4) [11]. The `Fragment` Header is used by an IPv6 source to send a packet larger than would fit in the path MTU to its destination. It works like IPv4 fragmentation except that only the packet source can fragment the packet. The `Authentication` Header (sender authentication, data integrity) [26] and `Encapsulating Security Payload` (sender authentication, data integrity, confidentiality) [25] are both part of the IPsec protocol suite. The `Mobility` Header is used to allow devices to move from one network to another while maintaining a permanent IPv6 address. The `Host Identity Protocol` Header is used to separate the end-point identifier and locator roles of IPv6 addresses [30]. The `Shim6 Protocol` Header is used to determine valid locator pairs that could be used when an outage is detected [31].

IPv6 EHs Injection with eBPF

This section describes how the eBPF program (both kernel and user) is implemented [22]. In order to inject one or multiple IPv6 EHs in outgoing traffic, the eBPF (kernel) program must be attached to an interface. More specifically, one needs to add a `clsact qdisc` [29] to an interface (Listing 1, line 1), which is like a scheduler holding only classifiers and that works both on ingress and egress. Then, the eBPF (kernel) program must be attached to an egress filter on that interface, with a specific section to be run (Listing 1, line 2). Both commands use `tc` [32], a traffic control tool as part of the `iproute2` [21] solution. Finally, the user program is used to configure the IPv6 EHs injection. Fig. 2 provides a high-level picture of how it works. What was previously described is inside the drawn red zone, on the right side of the figure.

```
1  struct exthdr_t {
2    struct bpf_spin_lock lock;
3    __u8 ip6nexthdr;
4    __u32 off_last_nexthdr;
5    __u32 bytes_len;
6  #define MAX_BYTES 2048 /* Feel free to increase if needed */
7    __u8 bytes[MAX_BYTES];
8  };
9
10 struct {
11   __uint(type, BPF_MAP_TYPE_ARRAY);
12   __uint(max_entries, 1);
13   __type(key, __u32);
14   __type(value, struct exthdr_t);
15   __uint(pinning, LIBBPF_PIN_BY_NAME);
16 } eh6_map SEC(".maps");
17
18 SEC("egress")
19 int egress_eh6(struct __sk_buff *skb) {
20   __u32 off, bytes_len, off_last_nexthdr, idx = 0;
21   struct exthdr_t *exthdr;
22   struct ipv6hdr *ip6;
23   __u8 ip6nexthdr, x;
24
25   ip6 = ipv6_header(skb, &off);
26   if (!ip6)
27     return TC_ACT_OK;
28
29   if (!pass_custom_filter(skb, ip6->nexthdr, off))
30     return TC_ACT_OK;
31
32   exthdr = bpf_map_lookup_elem(&eh6_map, &idx);
33   if (!exthdr)
34     return TC_ACT_OK;
35
36   bpf_spin_lock(&exthdr->lock);
37   bytes_len = exthdr->bytes_len;
38   ip6nexthdr = exthdr->ip6nexthdr;
39   off_last_nexthdr = exthdr->off_last_nexthdr;
40   bpf_spin_unlock(&exthdr->lock);
41
42   if (bytes_len < 8  bytes_len > MAX_BYTES)
43     return TC_ACT_OK;
44
45   x = ip6->nexthdr;
46   if (bpf_skb_adjust_room(skb, bytes_len, BPF_ADJ_ROOM_NET, 0))
47     return TC_ACT_OK;
48
49   if (bpf_skb_store_bytes(skb, off, exthdr->bytes, bytes_len,
          BPF_F_RECOMPUTE_CSUM))
50     return TC_ACT_SHOT;
51
52   if (off_last_nexthdr < MAX_BYTES && bpf_skb_store_bytes(skb, off
          + off_last_nexthdr, &x, sizeof(x), 0))
53     return TC_ACT_SHOT;
54
55   ip6 = ipv6_header(skb, &off);
56   if (!ip6)
57     return TC_ACT_SHOT;
58
59   ip6->nexthdr = ip6nexthdr;
60   ip6->payload_len = bpf_htons(skb->len - off);
61
62   return TC_ACT_OK;
63 }
```

Listing 2: Simplified example of the eBPF program. [22]

```
Usage: ./tc_ipv6_eh_user.o { --disable | --enable [ --force ]
      EXTHDR [ EXTHDR ... EXTHDR ] }

EXTHDR := { --hbh 8..2048 | --dest 8..2048 | --rh0 24..2040 |
      --rh2 | --rh3 24..2040 | --rh4 24..2040 | --fragA | --fragNA
      | --ah 16..1024 | --esp 16..2048 }

If a size is required, it MUST be an 8-octet multiple.
Routing Header sizes minus 8 MUST be 16-octet multiples.

Accepted chaining order, as per RFC8200 sec4.1:
- Hop-by-Hop Options header
- Destination Options header
- Routing header
- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header
```

Listing 3: User program. [22]

Listing 2 shows a simplified example of the eBPF program. Lines 1 to 8 represent the data structure used to inject IPv6 EHs, which is stored in a map (see lines 10 to 16). The map is pinned so that the user program is able to interact with it. This is also why the map is not per-cpu defined, since the user program needs to update the map (which is not possible for each cpu from user space). The function in line 19 is the handler for the *egress* section, and performs the following steps: (i) checks for a pointer overflow, which is required by the verifier (line 25); (ii) checks if the injection should happen for the current packet, based on editable rules (line 29); (iii) holds a lock to read the fields stored in the map (lines 36 to 40). Unfortunately, the lock can't be held when calling `bpf_skb_store_bytes`, which means the only safe solution would be to copy the buffer locally. However, this is impossible due to the verifier and the stack limit. Therefore, there is no safe solution and the worst case would be when the user program updates the buffer between the lock being released and the call to `bpf_skb_store_bytes`. As a consequence, the packet involved would probably be corrupted; (iv) applies some boundary checks required by the verifier (line 42); (v) makes room for the new bytes and inject them in the packet (lines 46 and 49); (vi) updates the last IPv6 EH's next header field to the upper-layer protocol used in the current packet (line 52), so that the IPv6 EH chain is complete; (vii) restores and rechecks pointers (line 55); $(viii)$ updates the IPv6 next header field and payload length according to which IPv6 EHs are inserted (lines 59 and 60).

Overall, the eBPF (kernel) program described in Listing 2 is completely independent of whether one or more IPv6 EHs are injected, or their order. The only thing that it knows is that it has to inject a buffer of bytes. Therefore, the overhead only depends on the number of bytes to inject (see next section for performance evaluation). Indeed, the buffer construction is delegated to the user program which is responsible for configuring what will be injected (one or more IPv6 EHs, their order, etc). Listing 3 shows the "`-help`" output of the user program to illustrate how it works and its possibilities. Very briefly, IPv6 EHs can be injected with constraints on respective sizes, and in any order. If the chosen order does not respect `RFC8200`, an error is returned as a security. Should the user really want to do it no matter what, the "`-force`" flag can be used to override the restriction.

IPv6 EHs Service Description

Let us consider an operator wanting to implement and deploy a new service that would require the use of large `DNS` requests and replies (i.e., sometimes above 1,280 bytes, sometimes up to 1,500 bytes and more). In IPv6, the minimum link `MTU` is 1,280 bytes, which means that bigger packet sizes are exposed to fragmentation, depending on the path `MTU`. The IPv6 fragmentation is always performed by the source and uses the `Fragment` Header. Based on some recent studies [20, 23, 12, 37, 18, 1, 28], the operator figures out that the `Fragment` Header does not always survive over the global Internet, which is problematic considering the service. However, those studies do not necessarily focus on the `Fragment` Header for the `DNS` service. Therefore, before implementing its entire new service

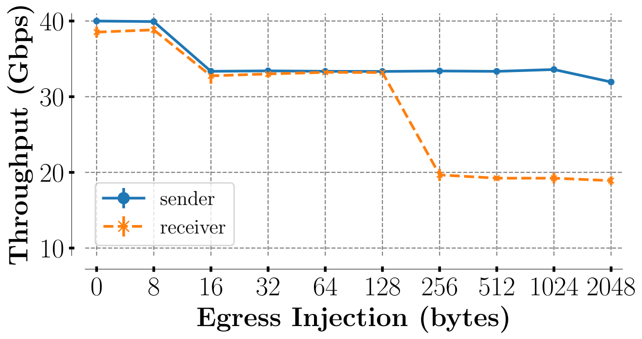

Figure 3: eBPF injection and its impact on throughput.

and in order to not waste time foolishly, the operator wants to make sure that all fragmented `DNS` requests and replies would go through and reach the target. On top of that, the operator would also like to use the `Minimum Path MTU Hop-by-Hop Option` [15] in order to detect the minimum path `MTU`. The aforementioned studies show that even a small `Hop-by-Hop Options` Header almost never survives over the global Internet. The eBPF program proposed in this paper could be a fast way of testing the service, i.e., an easy way to develop a proof-of-concept, by injecting either a `Fragment` Header or a `Hop-by-Hop Options` Header to some specific packets. There are two possible outcomes: either (i) the test is successful, and so the operator can confidently implement and deploy its service; or (ii) the test is not successful, and so the operator needs to think about an alternative for its new service.

Evaluation Methodology

To evaluate the eBPF program, we run measurements between two physical machines that are directly connected, each equipped with an `Intel XL710 2x40Gb QSFP+ NIC` (with Receive-Side Scaling enabled, and with an up-to-date firmware and up-to-date i40e driver). Each machine runs a Linux kernel v6.1 and has an Intel Xeon CPU E5-2630 v3 at 2.40GHz, 8 Cores, 16 Threads, and 32GB of RAM. `UDP` traffic (i.e., to mimic the `DNS` use case) is sent at line rate (40 Gbps) using iperf3 [9] with four clients (each sending at a constant rate of 10 Gbps) and four servers in parallel. The `MTU` is set to 8192 on both interfaces and the `UDP` payload is set to 6082 bytes, so that the maximum size inserted (i.e., 2048 bytes) would not make packet sizes to exceed.

As explained in previous section, only the size of the buffer to be injected has an impact on performance, not its content (i.e., whatever the combination of IPv6 EHs). Since most of these IPv6 EHs are limited to a maximum of 2048 bytes, we evaluate the size impact from 0 to 2048 bytes, although a combination of IPv6 EHs could result in a much bigger buffer (which is possible by simply increasing `MAX_BYTES` in the eBPF program). Therefore, based on the presented use case, we measure the injection of a `Hop-by-Hop Options` Header (because the `Fragment` Header has a fixed size of 8 bytes) and we vary the size: 0 (none), 8, 16, 32, 64, 128, 256, 512, 1024, and 2048 bytes. As a results, packet sizes are respectively 6144, 6152, 6160, 6176, 6208, 6272, 6400, 6656, 7168, and 8192 bytes.

Results

Fig. 3 shows a constant line rate throughput on both the sender and the receiver for an 8-byte injection. For a 16-byte injection and above, the throughput on the sender drops by approximately 17% and stays the same up to 2048 bytes. This sudden drop between 8 and 16 bytes and the fact that it is constant looks a bit suspicious. We still need to investigate to find an explanation. One of the reasons could be that the free room in socket buffers is exhausted, and therefore an implicit reallocation is needed. However, 16 bytes seems a bit low for that, and it would be unlucky to cross such a boundary so soon. Another more likely reason is the overuse of CPUs due to iperf3 clients generating traffic. Note that the receiver can keep up until 128 bytes, where a huge drop to approximately 20Gbps (half) is observed with 256 bytes and above. This observation on the receiver is provided for information only, since it is not directly what was measured.

This is important to remind readers that this section evaluates the worst case, i.e., line rate traffic. Overall, it is highly unlikely that an operator would need the eBPF program to inject IPv6 EHs at line rate in order to simulate and test a service.

Conclusion

In this paper, we leveraged eBPF for injecting IPv6 EHs into packets. We explained how we implemented our tool and demonstrate its potential usage through a use case deployed by a network service provider. As a consequence, we do believe that the proposed eBPF program could definitely be useful for operators to quickly develop proof-of-concept services based on IPv6 EHs.

Acknowledgments

This work has been supported by a Cisco grant CG#75919971 and by the CyberExcellence project funded by the Walloon Region, under number 2110186.

References

[1] APNIC Labs. Ipv6 fragmentation drop rate world map. Last Access: April, 17th 2023.

[2] Authority, I. I. A. N. Internet protocol version 6 (IPv6) parameters - IPv6 extension header types. Technical report, Internet Assigned Numbers Authority.

[3] Brockners, F.; Bhandari, S.; and Mizrahi, T. 2022. Data fields for in-situ operations, administration, and maintenance(IOAM). RFC 9197, Internet Engineering Task Force.

[4] Carpenter, B., and Jiang, S. 2013. Transmission and processing of IPv6 extension headers. RFC 7045, Internet Engineering Task Force.

[5] Castineyra, I.; Chiappa, N.; and Steenstrup, M. 1996. The Nimrod routing architecture. RFC 1992, Internet Engineering Task Force.

[6] Deering, S., and Hinden, R. 2017. Internet protocol, version 6 (ipv6) specification. RFC 8200, Internet Engineering Task Force.

[7] Elkins, N.; Ackermann, M.; and Deshpande, A. 2022. IPv6 extension headers (performance and diagnostic metics (PDM) destination option) testing across the Internet. Last Access: April, 19th 2023.

[8] Elkins, N.; Hamilton, R.; and Ackermann, M. 2017. IPv6 performance and diagnostic metrics (PDM) destination option. RFC 8250, Internet Engineering Task Force.

[9] ESnet. iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool.

[10] Filsfils, C.; Previdi, S.; Grinsberg, L.; Decraene, B.; Likowski, S.; and Shakir, R. 2018. Segment routing architecture. RFC 8402, Internet Engineering Task Force.

[11] Filsfils, C.; Dukes, D.; Previdi, S.; Leddy, J.; Matsushima, S.; and Voyer, D. 2020. Ipv6 segment routing header (srh). RFC 8754, Internet Engineering Task Force.

[12] Gont, F.; Linkova, J.; Chown, T.; and Liu, W. 2016. Observations on the dropping of packets with ipv6 extension headers in the real world. RFC 7872, Internet Engineering Task Force.

[13] Google. 2008–2023. IPv6 statistics. Last Access: April, 17th 2023.

[14] Hendrikx, L.; Velan, P.; Schmidts, R.; De Boer, P. T.; and Pras, A. 2017. Threats and surprises behind IPv6 extension headers. In *Proc. IFIP Network Traffic Measurement and Analysis (TMA)*.

[15] Hinden, R., and Fairhurst, G. 2022. IPv6 Minimum Path MTU Hop-by-Hop Option. RFC 9268, Internet Engineering Task Force.

[16] Hinz, J.-T.; Addanki, V.; Gyorgyi, C.; Jespen, T.; and Schmid, S. 2023. TCP's third eye: Leveraging eBPF for telemetry-powered congestion control. In *Proc. Workshop on eBPF and Kernel Extensions (eBPF)*.

[17] Hui, J.; Vasseur, J.-P.; Culler, D.; and Manral, V. 2012. An IPv6 routing header for source routes with the routing protocol for low-power and lossy networks (RPL). RFC 6554, Internet Engineering Task Force.

[18] Huston, G., and Damas, J. 2022. Ipv6 fragmentation and eh behaviours. Last Access: April, 17th 2023.

[19] Huston, G. 2013–2023. IPv4 address report. Last Acces: April, 17th 2023.

[20] Huston, G. 2017. Dealing with IPv6 fragmentation in the DNS. `https://blog.apnic.net/2017/08/22/dealing-ipv6-fragmentation-dns/`. Accessed: 2023-05-25.

[21] iproute2. Introduction to iproute2. See `https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.iproute2.html`.

[22] Iurman, J. 2023. IPv6 Extension Headers injection with eBPF. See `https://github.com/iurmanj/ebpf-ipv6-exthdr-injection`.

[23] Jaeggli, J.; Colitti, L.; Kumari, W.; Vyncke, E.; Kaeo, M.; and Taylor, T. 2013. Why operators filter framgents and what it implies. Internet Draft (Work in Progress) draft-taylor-v6ops-fragdrop-02, Internet Engineering Task Force.

[24] Johnson, D.; Perkins, C.; and Arkko, J. 2004. Mobility support in IPv6. RFC 3775, Internet Engineering Task Force.

[25] Kent, S., and Atkinson, R. 1998. IP encapsulating security payload (ESP). RFC 2406, Internet Engineering Task Force.

[26] Kent, S. 2005. IP authentication header. RFC 4302, Internet Engineering Task Force.

[27] Kogias, M., and Yang, R. 2023. HEELS: A host-enabled eBPF-based load balancing scheme. In *Proc. Workshop on eBPF and Kernel Extensions (eBPF)*.

[28] Léas, R.; Iurman, J.; Vyncke, E.; and Donnet, B. 2022. Measuring IPv6 extension headers survivability with james. In *Proc. ACM Internet Measurement Conference (IMC), Poster Session*.

[29] LWN.net. 2016. net, sched: add clsact qdisc. Accesses: 2023-06-01.

[30] Moskowitz, R.; Nikander, P.; Jokela, P.; and Henderson, T. 2008. Host identity protocol. RFC 5201, Internet Engineering Task Force.

[31] Nordmark, E., and Bagnulo, M. 2009. Shim6: Level 3 multihoming shim protocol for IPv6. RFC 55533201, Internet Engineering Task Force.

[32] tc. tc(8) – linux manual page. See `https://man7.org/linux/man-pages/man8/tc.8.html`.

[33] The Linux Foundation. 2021. eBPF. `https://ebpf.io`.

[34] Tran, V.-H., and Bonaventure, O. 2020. Beyond socket options: Towards fully extensible linux transport stacks. *Computer Communications* 1:118–138.

[35] Tuxology. 2017. An entertaining eBPF XDP adventure. Accesses: 2023-06-01.

[36] V.-A Padurean, Gasser, O.; Bush, R.; and Feldmann, A. 2022. SRv6: Is there anybordy out there? In *Proc. International Workshop on Traffic Measurements for Cybersecurity (WTMC)*.

[37] Vyncke, E.; Léas, R.; and Iurman, J. 2022. Just another measurement of extension header survivability (JAMES). Internet Draft (Work in Progress) draft-vyncke-v6ops-james-02, Internet Engineering Task Force.

[38] Wustrich, L.; Schacherbauer, M.; Budeus, M.; Freiherr von Kunbberg, D.; GAllenmuller, S.; Pahl, M.-O.; and Carle, G. 2023. Network profiles for detecting application-characteristic behavior using linux eBPF. In *Proc. Workshop on eBPF and Kernel Extensions (eBPF)*.

[39] Xhonneux, M.; Duchene, F.; and Bonaventure, O. 2018. Leveraging eBPF for programmable network functions with IPv6 segment routing. In *Proc. ACM CoNEXT*.