IPv6 In-Situ Operations, Administration, and Maintenance

J. Iurman, B. Donnet
Université de Liège, Montefiore Institute – Belgium

Abstract

In-situ Operations, Administration, and Maintenance (IOAM) is currently under standardization at the IETF. It allows for collecting telemetry and operational information along a path, within the data packet, as part of an existing (possibly additional) header. This paper introduces the very first implementation of IOAM for the Linux kernel with IPv6 as encapsulation protocol and discusses several use cases in which IOAM can find a suitable usage.

Keywords— IOAM, IPv6, Linux, Telemetry

1 Introduction

Operations, Administration, and Maintenance (OAM) refers to a set of techniques and mechanisms for performing fault detection and isolation, and for performance measurements. Throughout the years, multiple OAM tools have been developed for various layers in the protocol stack [1], going from basic `traceroute` [2] to *Bidirectional Forwarding Detection* (BFD [3]). Recently, OAM has been pushed further through *In-Situ* OAM (IOAM) [4, 5]. The term "In-Situ" directly refers to the fact that the OAM and telemetry data are carried within packets rather than being sent through packets specifically dedicated to OAM. The IOAM traffic is embedded in data traffic, but not part of the packet payload.

In a nutshell, IOAM gathers telemetry and operational information along a path, within the data packet (see Fig. 1), as part of an existing (possibly additional) header. It is included in IPv6 packets as an IPv6 *HopByHop extension header* [6, 7]. Typically, IOAM is deployed in a given domain, between the INGRESS and the EGRESS or between selected devices within the domain. Each node involved in IOAM may insert, remove, or update the extension header. IOAM data is added to a packet upon entering the domain and is removed from the packet when exiting the domain. There exist four IOAM types for which different IOAM data fields are defined. (*i*) the *Pre-allocated Trace Option*, where space for IOAM data is pre-allocated; (*ii*) the *Incremental Trace Option*, where nothing is pre-allocated and each node adds IOAM data while expanding the packet as well; (*iii*) the *Proof of Transit* (POT) and, (*iv*) the *Edge-to-Edge* (E2E) Option. Trace and POT options are both embedded in a *HopByHop extension header*, i.e., they are processed by every node on the path. On the contrary, E2E option is embedded in a *Destination extension header*, i.e., it is only processed by the destination node.

IOAM data fields are defined within IOAM *namespaces*, that are identified by a 16-bit identifier. They allow devices that are IOAM capable to determine, for example, whether an IOAM option header needs to be processed or ignored. IOAM namespaces can be used by an operator to distinguish different operational domains. They also

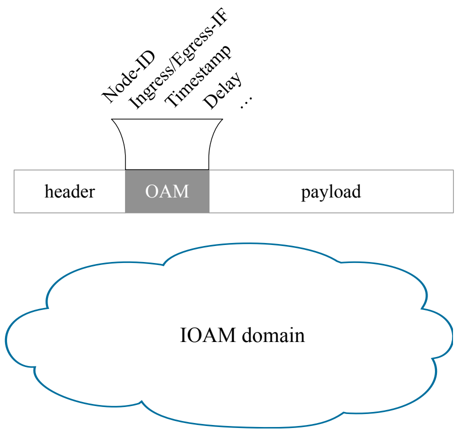

Figure 1: OAM data within data packet.

provide additional context for IOAM data fields, ensuring IOAM data is unique in case of several identical IOAM options, as well as allowing to identify different sets of devices.

We provide the very first Linux kernel implementation of IOAM, available online [8] and mainly based on the following two drafts: (*i*) the IOAM IPv6 draft [7] that defines how IOAM is carried by IPv6 and (*ii*) the IOAM data draft [5] that defines available data fields for IOAM. The implementation is compliant with RFC8200 [6] as it handles both the encapsulation and the inline insertion of IOAM. It is also per Linux network namespace oriented, i.e., it works with containers (e.g., Docker, Kubernetes) and virtual machines (VMs) too. A first official and more recent version [9] has been posted upstream. This is an improved version of the implementation proposed here, as an attempt to make IOAM part of the kernel. IOAM key impacts on the industry have the potential to be huge. Indeed, with IOAM, operators have now the opportunity to benefit from an efficient and lightweight tool for helping them to quickly detect and react to a network issue [10]. In this paper, through multiple use case scenarios (see Sec. 2), we describe those benefits and impacts for the industry

2 Interesting IOAM Use Cases

IOAM can find a suitable usage in multiple scenarios, increasing so its potential impact [10] on the industry. Among others, we can cite SLA verification, proof-of-transit, and geolocation. In this section, we discuss in particular three specific use cases: failure detection (Sec. 2.1), service selection (Sec. 2.2), and Cross-Layer Telemetry (Sec. 2.3).

2.1 Smart Traceroute: Failure Detection

Traditionally to detect and isolate network faults, `ping` and `traceroute`, or even BFD, are used. But in a complex network with micro-services or a large number of Unequal/Equal-Cost Multipath (U/E-CMP) being available, it would be difficult to detect and isolate such faults. Currently, failure detection takes tens of seconds in large networks, while failure isolation (using `ping` and `traceroute`) takes several minutes [11, 12]. Indeed, probing based on `traceroute` is slow as it typically requires $2 \times (n-1)$ packets for n nodes (i.e., two packets are generated for each TTL value, one from the sender, one from the intermediate node along the path). Therefore, one

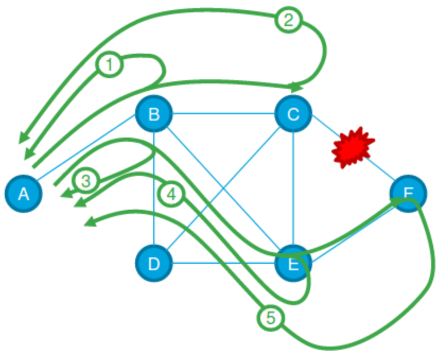

Figure 2: Failure Detection with IoAM.

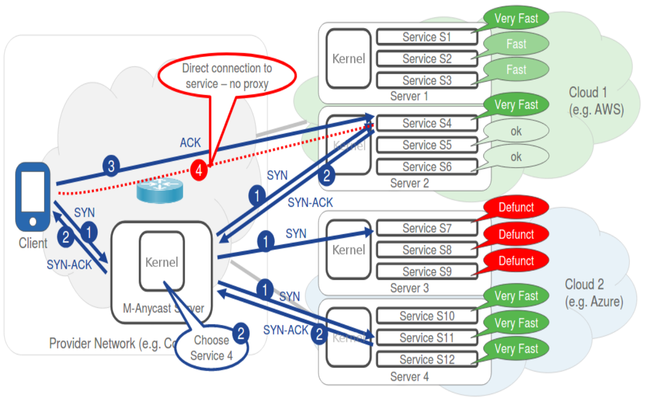

Figure 3: Service Selection and load balancing with IoAM

key impact is to significantly improve failure detection and isolation through efficient network probing, specially for hyper giant distribution networks (HGDNs, such as Facebook, Google, or Netflix) and large data center networks (DCNs).

This is exactly a context in which an active network probing relying upon UDP probes with IoAM can find a suitable usage. If one encounters connectivity issues between individual nodes, then IoAM tracing could be enabled between those nodes to understand where things are going wrong.

With IoAM, one can identify the exact path at a low probing cost [10] as it only requires n packets for n nodes. Indeed, IoAM injects a single packet in the network towards the destination, while each intermediate node along the path sends a copy back on the return path. Fig. 2 illustrates this. Node A sends a packet with IoAM to node F, through the path $A - B - C - F$. Each in turn, B (see step 1) and C (see step 2) send a copy back to A. Since there is a link failure between C and F, A do not receive anything from F. At this point, a failure is detected. To isolate it, repeat the same process for other paths (see steps 3-4-5). Step 5 allows A to detect that F is healthy and so exclude a node failure, leading to the conclusion that the failure is a link failure and can be isolated between C and F.

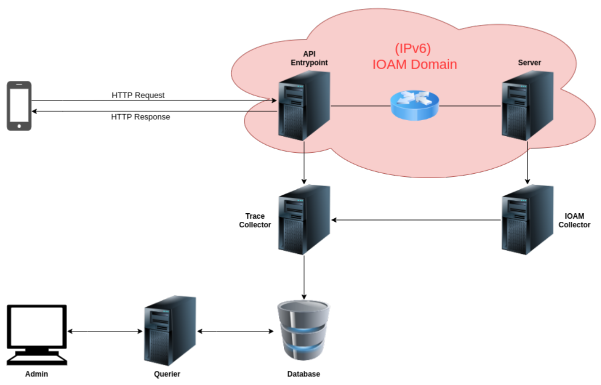

Figure 4: Cross-Layer Telemetry with IOAM.

2.2 M-Anycast: Service Selection

Another IOAM key impact is the possibility to provide an intelligent service selection. Let us take the example of highly redundant micro-services (e.g., video-chunk servers) hosted as containers in multiple public clouds, each service having an IPv6 address and sharing the same secondary anycast address. Access latencies, server load, service load, service liveliness, etc. differ from one to another. Therefore, the client needs to choose an appropriate service to connect to. Thanks to *M-Anycast* [13], as illustrated in Fig. 3, the client does not have to choose anymore. Indeed, the best service at that precise moment is provided by the server, acting as a proxy, to the client by leveraging Segment Routing [14] to steer traffic and IOAM for optimized service selection. This is something `ping`, `traceroute`, or even BFD cannot do, as IOAM also carries useful data and is designed for that purpose [10].

As a first step, the client initiates a TCP session with the service represented by an anycast address by starting the TCP three-way handshake (TCP `SYN` packet). Once received by the M-Anycast server, the packet is augmented with IOAM meta-data and replicated to a subset of available service instances. Doing so, telemetry information can be measured and propagated back to the M-Anycast server. In a second time, each service responds back in the TCP three-way handshake (TCP `SYN+ACK`). The optimal service is then selected by the M-Anycast server upon the IOAM meta-data received. The M-Anycast server is then responsible for forwarding the packet from the selected service back to the client. All others half-open connections are dropped and the connection states are cleaned up . Thirdly, the client receives a packet that contains Segment Routing policy that reveals its source (in this case, Service 4), avoiding so subsequent regular traffic to go through the M-Anycast server. The client then completes the three-way handshake process directly with the selected service (TCP `ACK`). Finally, once the TCP three-way handshake is completed, a TCP session is directly established between Service 4 and the client.

2.3 Cross-Layer Telemetry

Cross-Layer Telemetry (CLT) aims at making the entire network stack (L2 $\rightarrow$ L7) visible for distributed tracing tools (e.g., Jaeger), instead of the usual L5 $\rightarrow$ L7 visibility, leading so to a big impact on the industry as is allows for more efficient application debugging. Fig. 4 illustrates how it works. The IPv6 IOAM domain is where the magic happens. When a client request arrives, IOAM headers are inserted in the IOAM domain traffic by the "API entry point" (see Fig. 4) server and processed by each IOAM node on the path. Both the trace and span IDs from the tracing tool are injected in IOAM headers to allow for a future correlation. Then, an IOAM agent running on the end point server is responsible for gathering IOAM data and for sending everything to the IOAM collector, which one will correlate high level traces and network packets thanks to IOAM and send the result to the trace collector. Traces are stored on a database from where an operator can monitor them through a graphical user interface.

With CLT, an operator is now able to detect lower level issues. Take the example of an `SQL` request that takes ages and the service operator wants to debug. Thanks to CLT, the operator is now able to detect for instance a link failure, an interface misconfiguration, a heavy load on a queue, etc. associated to that `SQL` request issue. For that, IOAM is used to carry the trace and span IDs of the tracing tool in the dataplane, so that a correlation between network packets and high-level OpenTelemetry [15] traces can happen. The CLT code is available online [16] as well as a video to demonstrate what it looks like in action.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Mr. Iurman's work has been funded by a Cisco grant CG# 1343155.

References

[1] T. Mizrahi, N. Sprecher, E. Bellagamba, and Y. Weingarten, "An overview of operations, administration, and maintenance (OAM) tools," Internet Engineering Task Force, RFC 7276, June 2014.

[2] V. Jacobson et al., "traceroute," UNIX," man page, 1989, see source code: ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[3] D. Katz and D. Ward, "Bidirectional forwarding detection (BFD)," Internet Engineering Task Force, RFC 5880, June 2010.

[4] F. Brockners, S. Bhandari, and D. Bernier, "In-situ OAM deployment," Internet Engineering Task Force, Internet Draft (Work in Progress) draft-brockners-opsawg-ioam-deployment-01, March 2020.

[5] F. Brockners, S. Bhandari, and T. Mizrahi, "Data fields for in-situ OAM," Internet Engineering Task Force, Internet Draft (Work in Progress) draft-ietf-ippm-ioam-data-10, July 2020.

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

[7] S. Bhandari, F. Brockners, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, K. A., G. B., P. Lapukhov, S. M., K. S., and A. R., "In-situ OAM ipv6 options," Internet Draft (Work in Progress) draft-ietf-ippm-ioam-ipv6-options-02, July 2020.

[8] J. Iurman, "IPv6 IOAM patch for the linux kernel v4.12," 2019, see https://github.com/iurmanj/kernel_ipv6_ioam.

[9] ——, “Data plane support for ioam pre-allocated trace with ipv6,” June 2020, see https://lore.kernel.org/netdev/20200624192310.16923-1-justin.iurman@uliege.be/.

[10] J. Iurman, B. Donnet, and F. Brockners, “Implementation of ipv6 ioam in linux kernel,” in *Proc. Technical Conference on Linux Networking (Netdev 0x14)*, August 2020. [Online]. Available: https://netdevconf.info/0x14/pub/papers/32/0x14-paper32-talk-paper.pdf

[11] Facebook, “Udppinger,” see https://github.com/facebook/UdpPinger.

[12] ——, “fbtracert,” see https://github.com/facebook/fbtracert.

[13] Cisco, “M-anycast,” August 2017, see https://github.com/CiscoDevNet/iOAM/tree/master/M-Anycast.

[14] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, “Segment routing architecture,” Internet Engineering Task Force, RFC 8402, July 2018.

[15] OpenTelemetry, “Vendor-neutral APISs and instrumentation for distributed tracing,” 2020, https://opentelemetry.io.

[16] J. Iurman, “Cross layer telemetry,” 2020, see https://github.com/iurmanj/cross-layer-telemetry.

Required Metadata

Current Code Version

Nr.	Code Metadata Description
C1	v1.1
C2	`https://github.com/iurmanj/kernel_ipv6_ioam`
C3	
C4	GPL v2.0
C5	git
C6	C
C7	Linux kernel v4.12
C8	`https://github.com/iurmanj/kernel_ipv6_ioam/blob/master/README.md`
C9	`justin.iurman@uliege.be`

Current Executable Software Version

Nr.	**Executable Software Metadata Description**
S1	1.1
S2	`https://github.com/iurmanj/kernel_ipv6_ioam/blob/master/kernel_4_12.patch`
S3	
S4	GPL v2.0
S5	Linux kernel v4.12
S6	Linux kernel v4.12
S7	`https://github.com/iurmanj/kernel_ipv6_ioam/blob/master/README.md`
S8	`justin.iurman@uliege.be`