Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2021

H. Asai
Preferred Networks

2 November 2020

Separation of Data Path and Data Flow Sublayers in the Transport Layer
draft-asai-tsvwg-transport-review-00

Abstract

This document reviews the architectural design of the transport layer. In particular, this document separates the transport layer into two sublayers; the data path and the data flow layers. The data path layer provides functionality on the data path, such as connection handling, path quality and trajectory monitoring, waypoint management, and congestion control. The data flow layer provides additional functionality upon the data path layer, such as flow control for the receive buffer management, retransmission for reliable data delivery, and transport layer security. The data path layer multiplexes multiple data flow layer protocols and provides data path information to the data flow layer to control data transmissions, such as prioritization and inverse multiplexing for multipath protocols.

Table of Contents

1. Introduction

This document specifies two sublayers of the transport layer; the data path and the data flow layers. In this document, the transport layer's data path functionality, such as bidirectional connection handling, waypoint management, and congestion control, is separated from the data flow functionality, such as flow control for the receive buffer management, retransmission for reliable data delivery, and transport layer security. This document reviews the transport layer's functionality from the viewpoint of data paths and data flows in Section 2. It then specifies the data path layer and the data flow layer in Section 3. The data path and data flow layers provide the data path and the data flow functionality, respectively.

This document reviews the transport layer to clarify the transport layer's functionality and to invent data flow layer protocols for advanced Internet technologies, such as middleboxes and in-network computing. Hence, this document does not intend to obsolete the transport layer or violate the current Internet architecture.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Transport Layer Functionality Review

This section reviews the conventional layering and transport layer functionality. It then summarizes the transport layer functionality by distinguishing data paths and data flows.

2.1. Conventional Layering

RFC 1122 [RFC1122] defines three communication layers, link layer, Internet layer, transport layer, and the interfaces between these layers. Link-layer protocols provide hop-by-hop data communications. Internet layer protocols such as the Internet Protocol (IP), the Internet Control Message Protocol (ICMP), and the Internet Group Management Protocol (IGMP) provide fragmentation, hop-by-hop datagram forwarding, and end-to-end datagram delivery. RFC 1123 [RFC1123] defines the interface between the application layer and the transport layer.

The Internet design follows the end-to-end principle to keep the simplicity of the Internet layer. Thus, the main functionality of the Internet layer is to provide end-to-end reachability. It does not guarantee datagram integrity, which means that packet loss,

duplication, corruption, or reordering may occur. Over the Internet layer, the transport layer implements such functionality to provide datagram integrity on the end-hosts to achieve end-to-end communications.

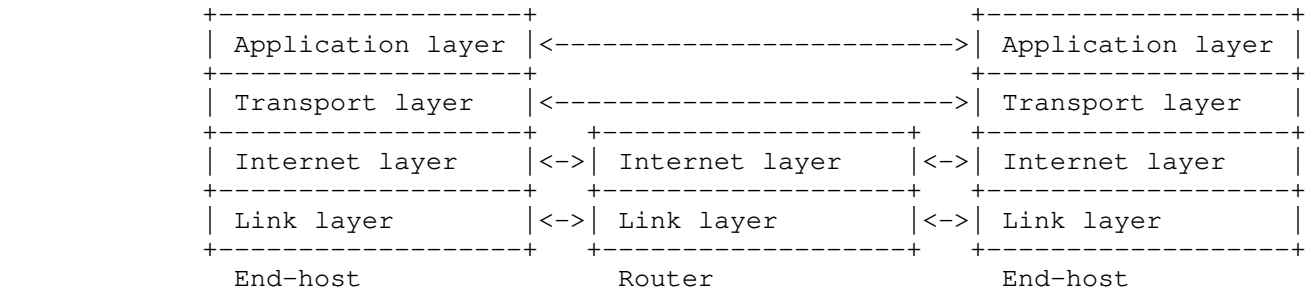


Figure 1: Conventional layering of Internet architecture

The transport layer provides various functions over the IP. User Datagram Protocol (UDP) [RFC0768] and Transmission Control Protocol (TCP) [RFC0793] are the most commonly used transport layer protocols. UDP is a connectionless transport protocol with a minimum header. TCP implements flow control according to the receiver's buffer capacity, retransmission for reliable communication, congestion control by a packet delivery status such as packet loss and delay. The transport layer may implement an end-to-end security function, Transport Layer Security (TLS) [RFC8446].

Figure 1 illustrates the conventional layering of Internet architecture. A router forwards an IP datagram to a next-hop router corresponding to the destination address. In this way, the Internet layer protocol, IP, provides end-to-end reachability through hop-by-hop routing. The transport layer provides additional functions for end-to-end communications.

2.2. Data Path-aware Networking

As described above, the Internet layer's main functionality has been end-to-end reachability with hop-by-hop packet forwarding based on destination IP address. Therefore, it is unaware of data paths, i.e., trajectories of packets. Best-effort communications over ''dumb'' networks without the quality of service (QoS) have been the Internet principle [RFC2768]. This end-to-end principle of the Internet has achieved a scalable architecture. However, computer networks' advancement introduced QoS and middleboxes to achieve high-quality data communication and optimize communication over distributed and heterogeneous computer networks. These technologies require to be aware of data paths associated with data flows.

The Internet layer has been extended to support QoS. Differentiated Services, DiffServ [RFC2474], is one of the technologies to implement QoS. It enables autonomous and scalable service discrimination using an IP header field, differentiated services codepoint (DSCP), to implement QoS for a data path. Segment Routing [RFC8402] enables data path configuration for individual flows by leveraging the source routing paradigm. Thus, Segment Routing is another technology to treat QoS.

Middleboxes are more complex than QoS. They add various functions such as firewalls, TCP offloading, transcoding, and content caches to a data path. These functions require to be aware of waypoints to be activated. Routing technologies with packet classification using the IP and the transport protocol headers have collectively been leveraged to route traffic through the middleboxes as the IP layer is not aware of data paths. If a middlebox is transparent to end-hosts, it should be installed at the network gateway or redirected by policy-based routing or segment routing to activate the function. Otherwise, an end-host should specify the middlebox as a target end-host. In addition to these middleboxes, distributed computing technologies, such as in-network computing and multi-access edge computing (MEC), also require to be aware of data paths.

In summary, advanced computer networks require to treat data paths for QoS, middleboxes, and distributed computing. Packet classification for data path treatment may use the header information of transport layer protocols such as port numbers as well as IP header fields.

2.3. Resource Management: Flow Control and Congestion Control

As the Internet layer does not provide resource management functionality, transport layer protocols may implement it, such as flow control and congestion control. Both flow control and congestion control mechanisms control packet transmission for resource management. However, the target resource is different. They control packet transmission according to the receiver’s buffer capacity and the network bandwidth capacity, respectively.

For example, in TCP's flow control, a receiver announces the remaining receive buffer size as the window size. Hence, flow control is not aware of network bandwidth capacity. On the other hand, congestion control is performed based on data communication quality information, such as packet loss and delay. The underlying Internet layer may provide congestion information by the Explicit Congestion Notification (ECN) [RFC3168]. In this manner, transport layer protocols control congestion depending on the data path's network resources. Therefore, congestion control is associated with a data path, while flow control is associated with the end-hosts.

As discussed above, congestion control should be performed on the associated data path. However, in current transport layer protocols such as TCP, Datagram Congestion Control Protocol (DCCP) [RFC4340], and Stream Control Transmission Protocol (SCTP) [RFC4960], an individual flow independently performs congestion control even if the same data path multiplexes multiple flows. Therefore, multiple flows cannot collectively perform congestion control for the data path. For example, when a data path multiplexes a TCP and a UDP flows, the TCP flow's congestion control may affect the other UDP flow's quality. Moreover, transport layer protocols utilize network resources on a fair basis. Thus, flow prioritization cannot be implemented at the transport layer, although a QoS technology such as DiffServ may be leveraged.

2.4. Multipath Protocols

Multipath TCP [RFC8684] and SCTP utilize multiple data paths over multiple endpoint addresses. As these multipath protocols are unaware of data paths, they distinguish the data paths by endpoint IP addresses. Accordingly, multiple flows of these protocols may use an identical data path without recognizing it.

Multipath protocols are also responsible for inverse multiplexing to split a data stream into multiple data paths. This inverse multiplexing is independent of data paths except for congestion control.

2.5. Reliable Data Communication

Transport layer protocols may implement retransmission and reordering functions to recover lost or reordered datagrams for reliable data communication. This functionality is independent of data paths, and consequently, should be implemented over data paths. Instead, ''smart'' end-hosts or middleboxes may implement it.

An end-host may transmit a duplicate packet for improving reliability [RFC8655]. This functionality is also independent of data paths. However, a router in a data path may be capable of duplicating a packet because the duplication process does not require significant computing resources, unlike retransmission.

2.6. Security

The transport layer may implement an end-to-end security function, such as Transport Layer Security (TLS) [RFC8446] Datagram Transport Layer Security (DTLS) [RFC6347]. As TLS does not encrypt the underlying transport protocol header, middleboxes such as TCP offloading can still work. However, some protocols implementing a transport layer protocol over DTLS, such as SCTP over DTLS used in WebRTC Data Channel [I-D.ietf-rtcweb-data-channel], encrypt the transport layer header, and consequently, have difficulty cooperating with these middleboxes.

2.7. Summary

As described above, the transport layer functionality is summarized by distinguishing data paths and data flows as follows:

The following functionality is categorized as data path functions.

* In-band trajectory monitoring
* Waypoint management
* Bidirectional connection establishment
* Data path quality (congestion) monitoring and congestion control
* Data flow multiplexing
* Packet duplication

The following functionality is categorized as data flow functions.

* Retransmission for reliable data communication
* Flow control for receive-buffer management
* Flow prioritization
* End-to-end security
* Inverse multiplexing for multipath protocols

3. Specifications of Data Path Layer and Data Flow Layer

As summarized in Section 2.7, this document separates data paths and data flows from the transport layer. Hence, this document divides the transport layer into two sublayers; the data path and the data flow layers. This section specifies the functionality of these two sublayers.

3.1. Data Path Layer

The data path layer provides the functionality to make the transport layer aware of data paths upon the Internet layer. The data path layer forwards packets without manipulating the payload. It means that the data path layer does not provide fragmentation, segmentation, or reassembly. Instead, the Internet layer or the data flow layer provide these functions.

The data path layer may implement in-band trajectory monitoring, bidirectional connection establishment, data path quality monitoring and congestion control, data flow multiplexing, and packet duplication. A data path layer protocol may implement other functions related to data paths.

3.1.1. In-band trajectory monitoring

The data path layer may provide an in-band trajectory monitoring function. The in-band trajectory monitoring function enables an upper-layer protocol to detect path change events and a single point of failure for multipath protocols.

DPH: Data path layer protocol header

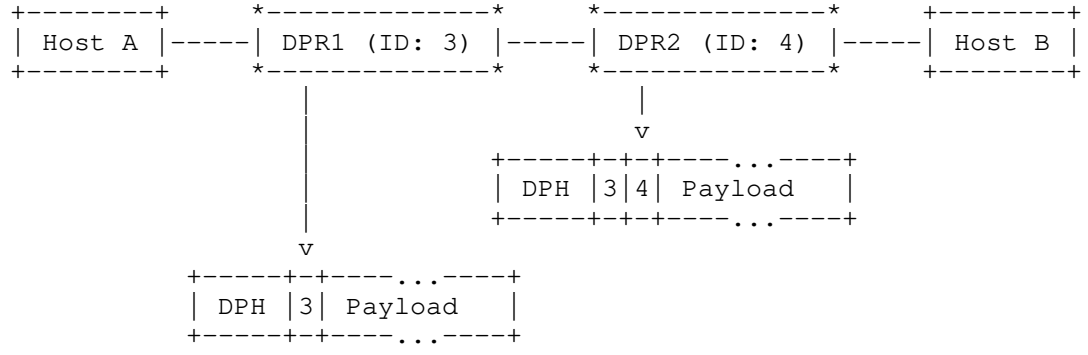


Figure 2: Appending data path router identifiers to distinguish the data path

One approach to implement this functionality is prepending, appending, or XOR-ing device identifiers like in-band network telemetry or in-situ OAM [I-D.ietf-ippm-ioam-data] by data path layer protocol's routers. Figure 2 shows an example of in-band trajectory monitoring. A router that handles a data path layer protocol appends the router identifier to the header. In case the upper-layer protocol does not require the path information but path change events, the router can apply an XOR operation with a hashed identifier to a fixed-length field. Other alternative approaches may be used.

The latter approach separates the control plane and the data plane. The control plane manages the data path, and the data plane monitors if packets pass through the dedicated data path. The data path control may leverage underlay protocols such as Segment Routing.

3.1.2. Waypoint management

The data path layer may support waypoint management functionality. The in-band trajectory monitoring functionality described above does not provide the functionality to designate the data path's waypoints. However, some middleboxes such as firewalls and in-network computing require to designate waypoints to activate the in-network functions.

A data path layer protocol may define a specification to control the waypoints of a data path. There are two approaches to designate waypoints over the Internet layer; 1) setting a waypoint as the destination IP address and 2) using routing technologies such as source routing and policy-based routing. The former approach includes Network Address Translation (NAT) [RFC3022] and proxy services. The latter approach may leverage underlay protocols to designate waypoints, such as Segment Routing and IPv6 Type 2 Routing Header [RFC6275]. Other alternative approaches may be used in a data path layer protocol.

3.1.3. Bidirectional connection establishment

The data path layer should support both unidirectional and bidirectional paths. A bidirectional path may be symmetric or asymmetric. As the characteristics of unidirectional and bidirectional paths are different, some data path layer protocols may only support either unidirectional or bidirectional paths.

A data path protocol supporting bidirectional data paths should implement a handshake mechanism to establish a bidirectional connection, such as a 3-way handshake of TCP and a 4-way handshake of SCTP. It may provide a 0-RTT connection establishment feature such as TLS 1.3 [RFC8446] and TCP Fast Open [RFC7413].

3.1.4. Data path quality (congestion) monitoring and congestion control

The data path layer should implement congestion control. However, Data path layer protocols supporting only unidirectional paths may not implement congestion control because it cannot implement congestion or data path quality reporting.

Congestion control is performed based on data path quality or congestion reporting from the receiver end-host or intermediate nodes that process a data path layer protocol. Data path quality or congestion signals may be packet losses, delay, or ECN, as used in many transport layer protocols. Other signals or metrics, such as in-band network telemetry information, may be used.

3.1.5. Data flow multiplexing

The data path layer enables us to collectively handle different characteristics (e.g., service level requirements) of transport layer protocols such as stream and datagram protocols.

3.1.6. Packet Duplication

On a lossy data path, a data path layer protocol may implement packet duplication for improving reliability. However, the data path layer should not provide retransmission because it requires significant resources.

3.2. Data Flow Layer

The data flow layer may implement various functions for end-to-end data communication over the data path layer. Data flow layer protocols may be stream, datagram, or message-oriented protocols. Middleboxes, MEC nodes, or in-network computing nodes may process data flow layer protocols. Therefore, a node processing data flow layer protocols should be as ``smart'' as end-hosts.

A data flow layer protocol may implement a retransmission function for reliable data communication, a flow control mechanism for receive-buffer management, flow prioritization for effective network resource utilization, a security extension such as TLS and DTLS, and a multipath transport protocol using multiple bidirectional paths over the data path layer.

3.2.1. Retransmission for reliable data communication

A data flow layer protocol may provide retransmission for reliable data communication. To this end, a node implementing the data flow layer protocol must equip a transmit buffer for retransmission to retransmit lost corrupted packets. It must also equip a receive buffer to reassemble a stream and a message from a sequence of packets in a stream and message-oriented data flow layer protocols. The receive buffer also handles packet reordering and duplication.

3.2.2. Flow control for receive-buffer management

Flow control is necessary for receive-buffer management. Therefore, a data flow layer protocol may implement the flow control mechanism. A receiver node advertises the available receive-buffer size in the data flow layer, like TCP's window size, when implementing the flow control mechanism. A sender node controls the transmission so that transmitted data do not exceed the receive-buffer size.

3.2.3. Flow prioritization

The prioritization of data flows multiplexed in a data path layer protocol may be implemented on the data flow layer using the data path layer information, such as monitored data path quality or congestion. A data flow layer protocol may provide a service code point or priority so that nodes processing the data flow protocol discriminate a flow.

3.2.4. End-to-end security

A data flow layer protocol may implement the end-to-end security function. TLS and DTLS can be used for stream and datagram protocols, respectively.

3.2.5. Inverse multiplexing for multipath protocols

A multipath data flow protocol may leverage multiple bidirectional data paths established by data path layer protocols. The data flow layer is responsible for the inverse multiplexing functionality. Therefore, the multipath data flow protocol divides a stream, a message, or a datagram into these multiple data paths. Note that the data path layer performs congestion control for each data path, while the data flow layer performs flow control.

4. Use Cases

This document does not specify any data path protocols or data flow protocols, but the data path and data flow layers' architectural design. For better understanding, this section describes the use cases of the data path and the data flow layers. In the use cases, a data path router (DPR) and a data flow layer node (DFN) denote a router that processes a data path layer protocol and a node that processes a data flow layer protocol, respectively.

4.1. Multipath Transport Protocols

Hosts A and B communicate with each other over multiple data paths; Path R1-DPR3-DPR2 and Path DPR4-DPR2. R1 is an IP router. DPR2, DPR3, and DPR4 are data path routers that treat data path layer protocols.

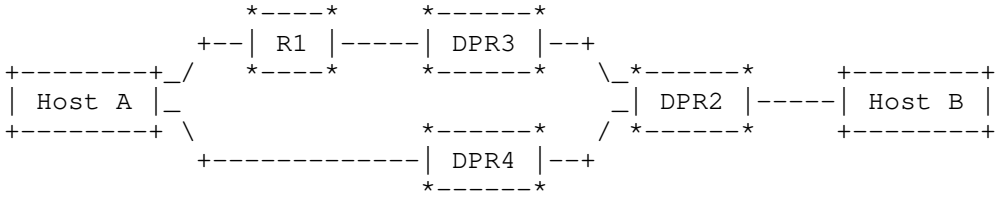


Figure 3: An example of multipath data communication

Figure 3 shows an example of multipath data communication over two data paths. The end-hosts A and B establish two bidirectional data paths; A-R1-DPR3-R2-B and A-DPR4-DPR2-B, and they use a data flow layer protocol over these data paths for end-to-end communication. The data path layer protocols monitor the data path quality and perform congestion control for each data path. The data flow layer protocol performs flow control and inverse multiplexing into these data paths.

Multipath transport protocols may leverage in-band trajectory monitoring to detect a shared waypoint. We assume that the data path routers manipulate the header of a data path layer protocol. They add the router identifier to the data path layer protocol's header to report the trajectory to the end-hosts. When sending packets from end-host A to end-host B over these paths, each packet reports trajectory A-DPR3-DPR2-B or A-DPR4-DPR2-B. In this way, end-host B recognizes two different data paths and detects a shared data path router, DPR2, on the multiple paths, potentially a single point of failure for end-to-end communication.

4.2. Congestion Control Acceleration

Some TCP congestion control acceleration functions proxy TCP sessions to shorten the perspective latency. The acceleration functions acknowledge receipt of packets. Using a loss-based congestion control algorithm, both congestion control and flow control use the acknowledgments (ACKs). For congestion control, missing ACKs report packet losses, and consequently, they present the data path quality to control the transmission rate to avoid congestion. For flow control, ACKs report successful data delivery. Then, the sender can release part of the transmit buffer. Otherwise, the sender retransmits the lost data.

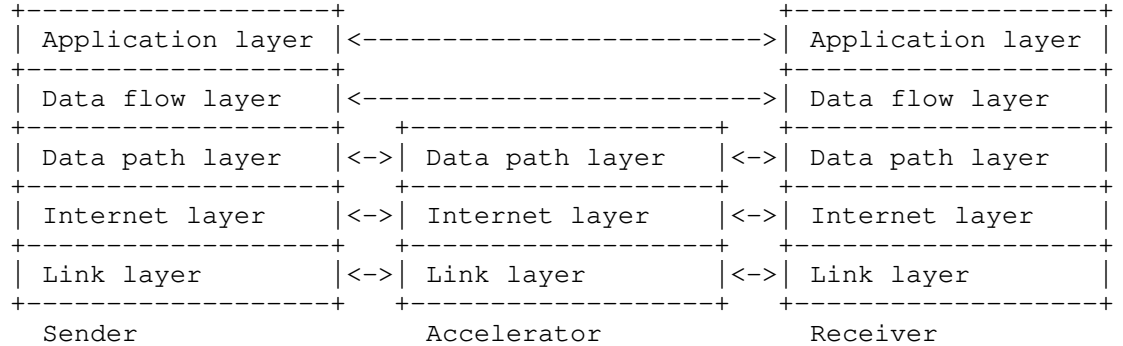


Figure 4: An example of a communication model for congestion control acceleration

Figure 4 illustrates an example of a communication model for congestion control acceleration using the data path layer. The accelerator or the receiver reports the data path quality to the sender on the data path layer to perform congestion control. There are various ways to measure data path quality. For example, data path routers may use the buffer capacity in the same way as ECN. If a data path router can detect packet losses for a connection of data path layer protocols, the packet losses or statistics may be used to report the data path quality. The data path quality report by the accelerator enables the fast response of congestion control algorithms. The data flow layer is responsible for retransmission for reliable communication and flow control for receive-buffer management.

4.3. In-Network Computing

In-network computing [I-D.kunze-coin-industrial-use-cases] is a novel distributed computing paradigm. It requires to be aware of the data path's waypoints because computing components are placed in between end-hosts. The message-oriented protocol may adopt the pub/sub communication model, widely used in machine-to-machine communication.

The data path layer establishes a data path while designating the waypoints to perform in-network computing. The data flow layer over the data path implements a message-oriented protocol or a stream protocol to feed data into in-network computing nodes. The message-oriented protocol may adopt the pub/sub communication model, widely used in machine-to-machine communication.

App., DF, DP, IP, and Link denote the application layer, the data flow layer, the data path layer, the IP, and the link layer, respectively. H1 and H2 are end-hosts, C1 and C2 are in-network computing nodes, P1 is a data path router, and R1 is an IP router.

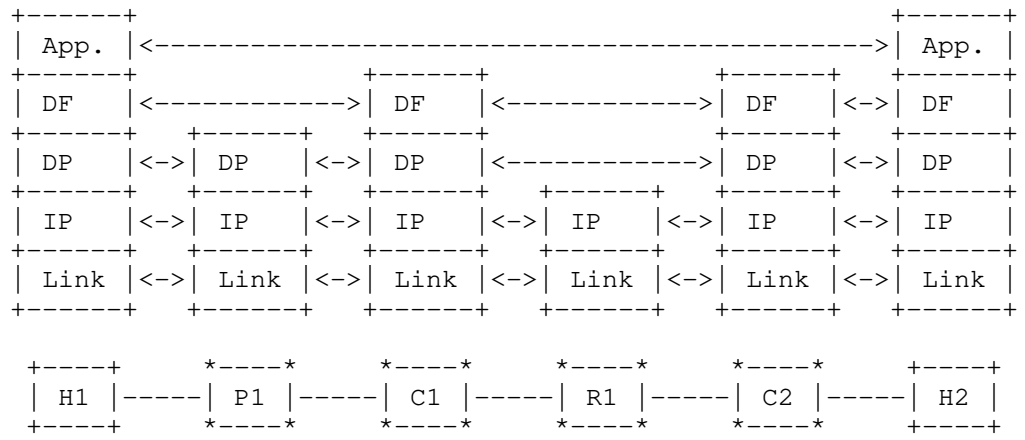


Figure 5: An example of communication model of in-network computing

Figure 5 illustrates an example of the communication model of in-network computing. A data path is established between H1 and H2. A data path layer protocol designates C1 and C2 as the waypoints. Over the established data path, H1 transmits messages to H2 using a data flow layer protocol. C1 and C2 process the messages according to the programs running on C1 and C2.

4.4. Flow Arbitration

The separation of the data path and the data flow layer enables flow-level prioritization. As the data path layer is responsible for congestion control, the multiplexed data flows over a data path can collectively perform resource arbitration for the data path bandwidth capacity.

For example, a data path multiplexes two data flows; one of them requires a constant data rate, and the other is tolerant of delayed data delivery. Flow control can prioritize the former data flow and decrease the transmission rate when the data path layer detects congestion. This flow arbitration is crucial in coexisting deadline-based and best-effort flows.

H1, H2, H3, and H4 are end-hosts. R1, R2, and R3 are IP routers. P1 is a data path router that support the data path quality monitoring function.

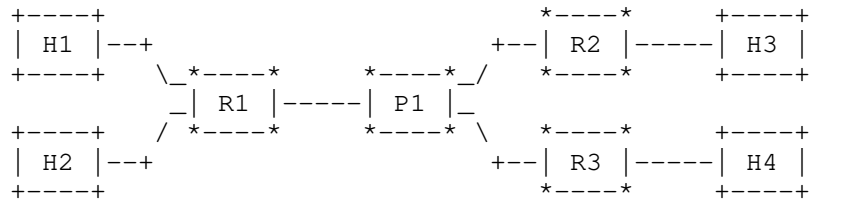


Figure 6: An example of flow arbitration over multiple data paths

Moreover, the data path layer may support data path quality monitoring for multiple data paths. If data path routers monitor multiple data paths' bandwidth capacity, they can report the estimated capacity to arbitrate multiple flows over the data paths. Figure 6 shows an example flow arbitration over multiple data paths. Suppose data flows X and Y are over the data paths H1-R1-P1-R2-H3 and H2-R1-P1-R3-H4, respectively; the data path router P1 can monitor the data path quality such as packet losses for these data paths. If the bandwidth utilization reaches the capacity, the data path router requests resource arbitration. A flow may reduce the data rate using the flow control mechanism in the data flow layer if its priority is not high. Thus, these two flows can autonomously perform the flow arbitration. This flow arbitration may fail and perform best-effort communication, such in case both flows are the high priority and do not reduce the data rate.

5. Implementation Considerations

Although this document does not specify the data path layer and the data flow layer protocols, it discusses the implementation of these protocols' specifications. The data path layer may be implemented over UDP for interoperability with the current Internet operations because other protocols than ICMP, TCP, and UDP may be filtered on the Internet. Data path layer protocols must implement the maximum transmission unit (MTU) discovery [RFC8899].

LH: Link-layer protocol header, IPH: IP header, UDPH: UDP header, DPH: Data path layer protocol header, DFH: Data flow layer protocol header

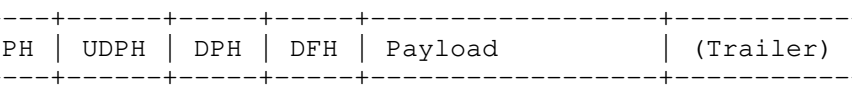


Figure 7: An implementation of data path layer and data flow layer protocols over UDP

Figure 7 illustrates an example packet format of data path layer and data flow layer protocols over UDP. In this figure, the UDP header is not intended to provide the transport layer's functionality but is introduced for interoperability with existing middleboxes on the Internet.

A data path layer protocol may use hop-by-hop UDP connections to designate waypoints like an overlay network. Otherwise, data path routers require to leverage routing technologies such as Segment Routing and policy-based routing to support waypoint management.

6. IANA Considerations

This document does not have IANA Considerations.

7. Security Considerations

End-to-end encryption becomes critical to Internet protocols for privacy and security. The data path layer must be visible to intermediate nodes, such as routers and middleboxes, by design to process and manipulate a data path layer protocol. Therefore, this document proposes to implement transport layer security at the data flow layer. It exposes data path information. Therefore, data path layer protocols must not include privacy-sensitive information in the protocol header.

Data path layer protocols may not be visible to intermediate nodes if an underlay protocol encrypts the payload. For example, IPsec [RFC6071] encrypts the payload of IP datagrams. In such cases, intermediate nodes cannot process or manipulate data path layer protocols.

8. Acknowledgements

We thank Kenichi Murata, Ryokichi Onishi, Yusuke Doi, and Masahiro Ishiyama for their comments and review of this document.

9. References

9.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC2768] Aiken, B., Strassner, J., Carpenter, B., Foster, I., Lynch, C., Mambretti, J., Moore, R., and B. Teitelbaum, "Network Policy and Services: A Report of a Workshop on Middleware", RFC 2768, DOI 10.17487/RFC2768, February 2000, <https://www.rfc-editor.org/info/rfc2768>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC6071] Frankel, S. and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap", RFC 6071, DOI 10.17487/RFC6071, February 2011, <https://www.rfc-editor.org/info/rfc6071>.

[RFC6275] Perkins, C., Ed., Johnson, D., and J. Arkko, "Mobility Support in IPv6", RFC 6275, DOI 10.17487/RFC6275, July 2011, <https://www.rfc-editor.org/info/rfc6275>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/info/rfc8402>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8655] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <https://www.rfc-editor.org/info/rfc8655>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

[RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

9.2. Informative References

[I-D.ietf-ippm-ioam-data]
Brockners, F., Bhandari, S., and T. Mizrahi, "Data Fields for In-situ OAM", Work in Progress, Internet-Draft, draft-ietf-ippm-ioam-data-10, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ippm-ioam-data-10.txt>.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", Work in Progress, Internet-Draft, draft-ietf-rtcweb-data-channel-13, 4 January 2015, <http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-data-channel-13.txt>.

[I-D.kunze-coin-industrial-use-cases]
Kunze, I. and K. Wehrle, "Industrial Use Cases for In-Network Computing", Work in Progress, Internet-Draft, draft-kunze-coin-industrial-use-cases-03, 8 September 2020, <http://www.ietf.org/internet-drafts/draft-kunze-coin-industrial-use-cases-03.txt>.

Author's Address

Hirochika Asai
Preferred Networks, Inc.
1-6-1 Otemachi, Chiyoda, Tokyo
100-0004
Japan

Email: panda@wide.ad.jp

LISP Working Group
Internet-Draft
Intended status: Experimental
Expires: March 10, 2020

S. Barkai
B. Fernandez-Ruiz
S. ZionB
R. Tamir
Nexar Inc.
A. Rodriguez-Natal
F. Maino
Cisco Systems
A. Cabellos-Aparicio
J. Paillissé Vilanova
Technical University of Catalonia
D. Farinacci
lispers.net
October 10, 2019

Network-Hexagons: H3-LISP Based Mobility Network
draft-barkai-lisp-nexagon-11

Abstract

This document specifies combined use of H3 and LISP for mobility-networks:
- Enabling real-time tile by tile indexed annotation of public roads
- For sharing: hazards, blockages, conditions, maintenance, furniture..
- Between MobilityClients producing-consuming road geo-state information
- Using addressable grid of channels of physical world state representation

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 4, 2019.

Table of Contents

1. Introduction

(1) The Locator/ID Separation Protocol (LISP) [RFC6830] splits current IP addresses in two different namespaces, Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). LISP uses a map-and-encap approach that relies on (1) a Mapping System (distributed database) that stores and disseminates EID-RLOC mappings and on (2) LISP tunnel routers (xTRs) that encapsulate and decapsulate data packets based on the content of those mappings.

(2) H3 is a geospatial indexing system using a hexagonal grid that can be (approximately) subdivided into finer and finer hexagonal grids, combining the benefits of a hexagonal grid with hierarchical subdivisions. H3 supports sixteen resolutions. Each finer resolution has cells with one seventh the area of the coarser resolution. Hexagons cannot be perfectly subdivided into seven hexagons, so the finer cells are only approximately contained within a parent cell. Each cell is identified by a 64bit HID.

(3) The Berkeley Deep Drive (BDD) Industry Consortium investigates state-of-the-art technologies in computer vision and machine learning for automotive applications, and, for taxonomy of published automotive scene classification.

These standards are combined to create in-network-state which reflects the condition of each hexagon tile (~1sqm) in every road. The lisp network maps & encapsulates traffic between MobilityClients endpoint-identifiers (EID), and, addressable (HID=>EID) tile-states. States are aggregated byH3Service EIDs.

The H3-LISP mobility network bridges timing-location gaps between the production and consumption of information by MobilityClients:
- o vision, sensory, LIADR, AI applications - information producers
- o driving-apps, smart-infrastructure, command & control - who consume it

This is achieved by putting the physical world on a shared addressable geo-state grid at the edge, a low-latency production-consumption indirection. Tile by tile based geo-state mobility-network solves key issues in todays' vehicle to vehicle networking, where observed hazards are expected to be relayed or "hot-potato-tossed" (v2v without clear-reliable convergence i.e. given a situation observable by some of traffic, it is unclear if the rest of the relevant traffic will receive consistent, conflicting, multiple, or no indication what so ever - using peer-to-peer propagation.

For example, when a vehicle experiences a sudden highway slow-down,"sees" many brake-lights or "feels" accelerometer, there is no clear way for it to share this annotation with vehicles 20-30sec away for preventing potential pile-up. Or, when a vehicle crosses an intersection, observing opposite-lane obstruction - construction, double-park, commercial-loading / un-loading, garbage truck, or stopped school-bus - there is no clear way for it to alert vehicles turning in to that situation as it drives away.

Geo-state indirection also helps solve communicating advanced machine-vision and radar annotations. These are constantly evolving technologies, however, communicating the road enumerations they produce using peer-to-peer protocols poses a significant interoperability challenge - testing each new annotation by any sensor / OEM vendor and any other OEM and driving application vendor.

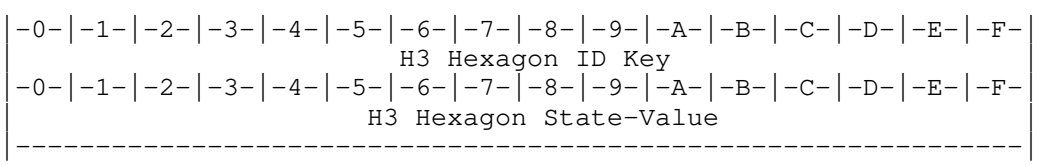
These peer-to-peer limitations are inherit yet unnecessary, as in most road situations vehicles are not really proper peers. They just happen to be in the same place at the same time. The H3-LISP mobility network solves limitations of direct vehicle to vehicle communication because it anchors per each geo-location: timing, security, privacy, interoperability. Anchoring is by

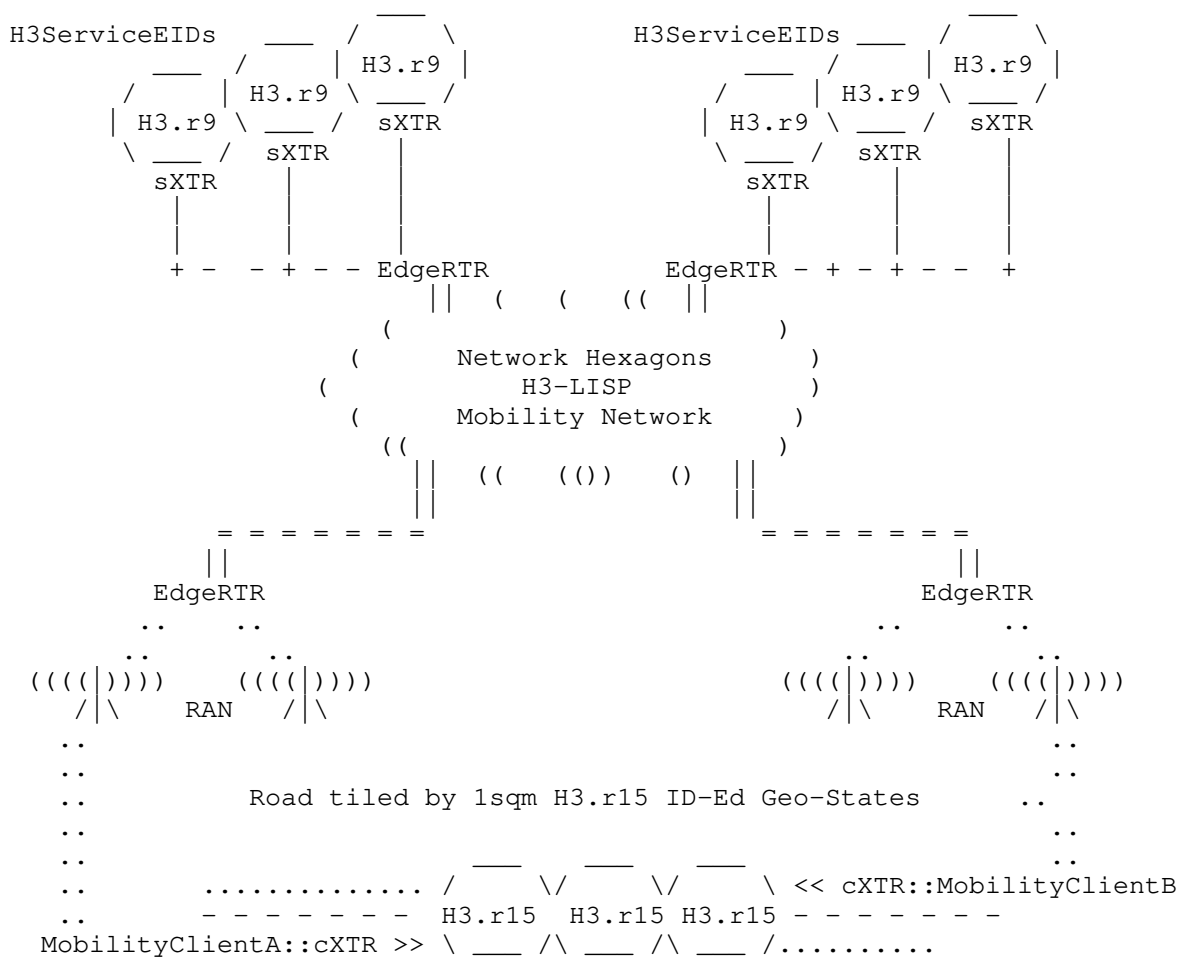
MobilityClients communicating through in-network geo-states. Addressable tiles are aggregated and maintained by LISP H3ServiceEIDs.

An important set of use-cases for state propagation of information to MobilityClients is to provide drivers heads-up alerts on hazards and obstacles beyond line of sight of both the drivers and in-car sensors: over traffic, around blocks, far-side-junction, beyond turns, and surface-curvatures. This highlights the importance of networks in providing road-safety.

To summarize the H3-LISP solution outline:

(1) MicroPartition: 64bit indexed geo-spatial H3.r15 road-tiles
(2) EnumState: 64bit state values compile tile condition representation
(3) Aggregation: H3.r9 H3ServiceEID group individual H3.r15 road-tiles
(4) Channels: H3ServiceEIDs function as multicast state update channels
(5) Scale: H3ServiceEIDs distributed for in-network for latency-throughput
(6) Mapped Overlay: tunneled-network routes the mobility-network traffic
(7) Signal-free: tunneled overlay is used to map-register for mcast channels
(8) Aggregation: tunnels used between MobilityClients/H3ServiceEIDs <> edge
(9) Access: ClientXTRs/ServerXTRs tunnel traffic to-from the LISP EdgeRTRs
(10) Control: EdgeRTRs register-resolve H3ServiceEIDs and mcast subscription





- MobilityClientA has seen MobilityClientB (20-30 sec) future, and, vice versa

- Clients share information using addressable shared-state routed by LISP Edge
- ClientXTR (cXTR): tunnel encapsulation through access network to LISP Edge
- ServerXTR (sXTR): tunnel encapsulation through cloud network to LISP Edge
- The H3-LISP Mobility overlay starts in the cXTR and terminates in the sXTR
- The updates are routed to the appropriate tile geo-state by the LISP network
- EdgeRTRs perform multicast replication to edges and then native or to cXTRs
- Clients receive tile-by-tile geo-state updates via the multicast channels

Each H3.r9 hexagon is an EID Service with corresponding H3 hexagon ID. Bound to that service is a LISP xTR, called a ServerXTR, resident to deliver encapsulated packets to and from the H3ServiceEID and LISP Edge. EdgeRTRs are used to re-tunnel packets from MobilityClients to H3ServiceEIDs. Each H3ServiceEID is also a source multicast address for updating MobilityClients on the state of the H3.r15 tiles aggregated-represented by the H3ServiceEID.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definition of Terms

H3ServiceEID: Is an addressable aggregation of H3.r15 state-tiles. It is a designated source for physical world reported annotations, and an (s,g) source of multicast public-safety update channels. H3ServiceEID is itself an H3 hexagon, large enough to provide geo-spatial conditions context, but not too large as to over-burden (battery powered, cellular connected) subscribers with too much information. For Mobility Network it is H3.r9. It has a light-weight LISP protocol stack to tunnel packets aka ServerXTR. The EID is an IPv6 EID that contains the H3 64-bit address numbering scheme. See IANA consideration for details.

ServerXTR: Is a light-weight LISP protocol stack implementation that co-exists with H3ServiceEID process. When the server roams, the xTR roams with it. The ServerXTR encapsulates and decapsulates packets to/from EdgeRTRs.

MobilityClient: Is a roaming application that may be resident as part of an automobile, as part of a navigation application, part of municipal, state, of federal government command and control application, or part of live street view consumer type of application. It has a light-weight LISP protocol stack to tunnel packets aka ClientXTR.

MobilityClient EID: Is the IPv6 EID used by the Mobility Client applications to source packets. The destination of such packets are only H3ServiceEIDs. The EID format is opaque and is assigned as part of the MobilityClient network-as-a-service (NaaS) authorization.

ClientXTR: Is the light-weight LISP protocol stack implementation that is co-located with the Mobility Client application. It encapsulates packets sourced by applications to EdgeRTRs and decapsulates packets from EdgeRTRs.

EdgeRTR: Is the core scale and structure of the LISP mobility network. EdgeRTRs proxy H3ServiceEIDs and MobilityClient H3ServiceEID channel registration. EdgeRTRs aggregate MobilityClients and H3Services using tunnels to facilitate hosting-providers and mobile-hosting flexibility - for accessing the nexagon mobility network.
EdgeRTRs decapsulate packets from ClientXTRs and ServerXTRs and re-encapsulates
packets to the clients and servers tunnels. EdgeRTRs glean H3ServiceEIDs and glean MobilityClient EIDs when it decapsulates packets. EdgeRTRs store H3ServiceEIDs and their own RLOC of where the H3ServiceEID is currently reachable from in the map-cache. These mappings are registered to the LISP mapping system so other EdgeRTRs know where to encapsulate for such EIDs. EdgeRTRs do not register MobilityClients' EIDs at the mapping service as

these are temporary-renewed while using the mobility network. Enterprises may provide their own client facing EdgeRTRs to mask their clients geo-whereabouts while using the mobility network.

4. Deployment Assumptions

The specification described in this document makes the following deployment assumptions:

(1) Unique 64-bit HID is associated with each H3 geo-spatial tile
(2) MobilityClients and H3ServiceEIDs share this well known index
(3) 64-bit BDD state value is associated with each H3-indexed tile
(4) Tile state is compiled 16 fields of 4-bits, or max 16 enums

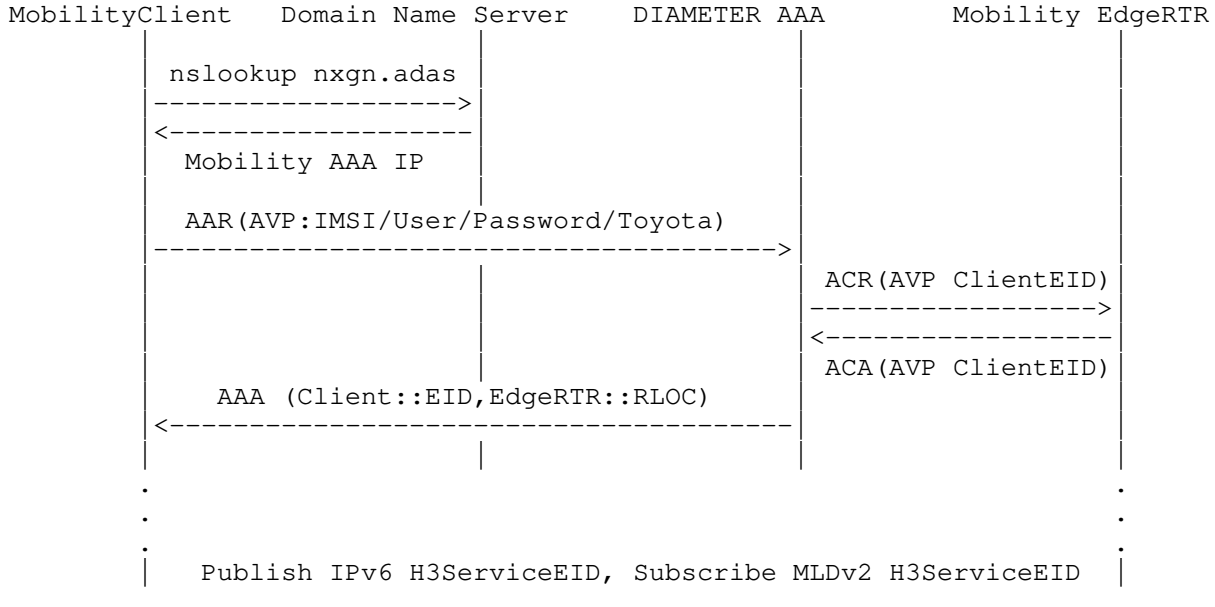
```
|-0-|-1-|-2-|-3-|-4-|-5-|-6-|-7-|-8-|-9-|-A-|-B-|-C-|-D-|-E-|-F-|
0123012301230123012301230123012301230123012301230123012301230123
```

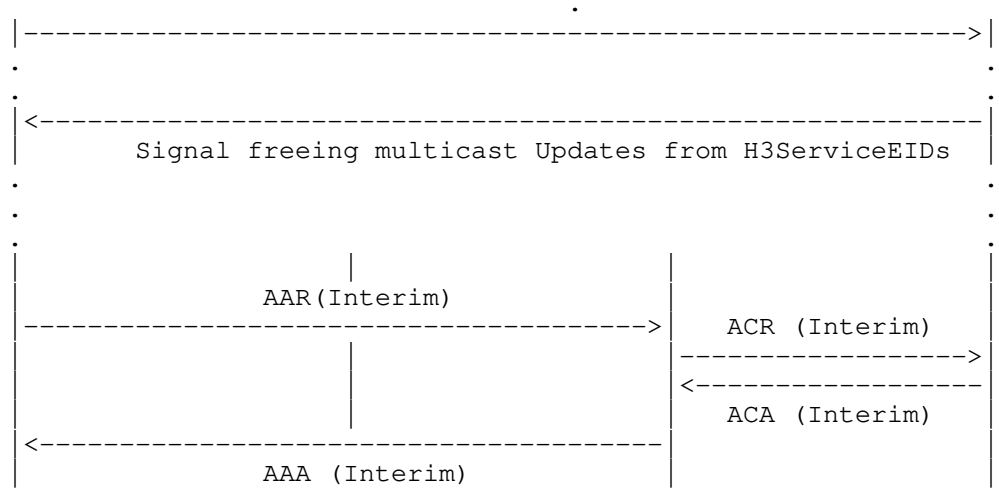
Subscription of MobilityClients to the mobility network is temporary-renewed while on the move and is not intended as means of basic connectivity. This is why MobilityClients use DNS/AAA to obtain temporary EIDs and EdgeRTRs and why they use (LISP) data-plane tunnels to communicate using their temporary EIDs with the dynamically assigned EdgeRTRs.

MobilityClient are otherwise unaware of the LISP network mechanism or mapping system and simply regard the data-plane tunnels application specific virtual private network (VPN) that supports IPv6 EID addressable geo-state for publish (Ucast), Subscribe (Mcast) H3Services.

In order to get access to the MobilityVPN MobilityClients first authenticate with the MobilityVPN AAA Server. DIAMETER based AAA is typically done at the provider-edge PE by edge gateways. However the typical case involves handful of customer-premise equipment(CPE/UE) types physically connected by wireline, or, by wireless spectrum to a specific service-provider. The Mobility VPN overlays potentially a number of wireless network providers and cloud-edge providers, and it involves dozens of Car-OEM, Driving-Applications, Smart-infrastructure vendors. It is therefore required to first go through AAA in-order to get both a MobilityClientEID and EdgeRTR gateway RLOC opened.

ClientXTR performs the following steps in-order to use the mobility network:
1) obtain the address of the mobility network AAA server using DNS
2) obtain MobilityClientEID and EdgeRTR(s) from AAA server using DIAMETER
3) renew authorization from AAA while using the mobility network T1 minutes





Using this network-login / re-login method we ensure that:
- the MobilityClientEIDs serve as credentials with the specific EdgeRTRs
- EdgeRTRs are not tightly coupled to H3.r9 areas for privacy/load-balance
- Mobility Clients do not need to update EdgeRTRs while roaming in a metro

The same EdgeRTR may serve several H3.r9 areas for smooth ride continuity, and, several EdgeRTRs may load balance a H3.r9 area with high density of originating MobilityClient rides. When a MobilityClient ClientXTR is homed to EdgeRTR it is able to communicate with H3ServiceEIDs.

5. Mobility Clients-Network-Services

The mobility network functions as a standard LISP VPN overlay.
The overlay delivers unicast and multicast packets across:
- multiple access-network-providers / radio-access-technologies.
- multiple cloud-edge hosting providers, public, private, hybrid.

We use data-plane XTRs in the stack of each mobility client and server. ClientXTRs and ServerXTRs are homed to one or more EdgeRTRs at the LISP edge. This structure allows for MobilityClients to "show-up" at any time, behind any network-provider in a given mobility network administrative domain (metro), and for any H3ServiceEID to be instantiated, moved, or failed-over to - any rack in any cloud-provider. The LISP overlay enables these roaming mobility network elements to communicate un-interrupted. This quality is insured by the LISP RFCs. The determinism of identities for MobilityClients to always refer to the correct H3ServiceEID is insured by H3 geospatial HIDs.

There are two options for how we associate ClientXTRs with LISP EdgeRTRs:

I. Semi-random load-balancing by DNS/AAA

In this option we assume that in a given metro edge a pool of EdgeRTRs can distribute the Mobility Clients load randomly between them and that EdgeRTRs are topologically more or less equivalent. Each RTR uses LISP to tunnel traffic to and from other EdgeRTRs for MobilityClient with H3Service exchanges
.
MobilityClients can (multi) home to EdgeRTRsRTRs throughout while moving.

II. Topological by any-cast

In this option we align an EdgeRTR with topological aggregation like in the Evolved Packet Core (EPC) solution. Mobility Clients currently roaming in an area home to that RTR and so is the H3 Server. There is only one hop across the edge overlay between clients and servers and mcast replication is more focused, but clients need to keep re-homing as they move.

To summarize the H3LISP mobility network layout:

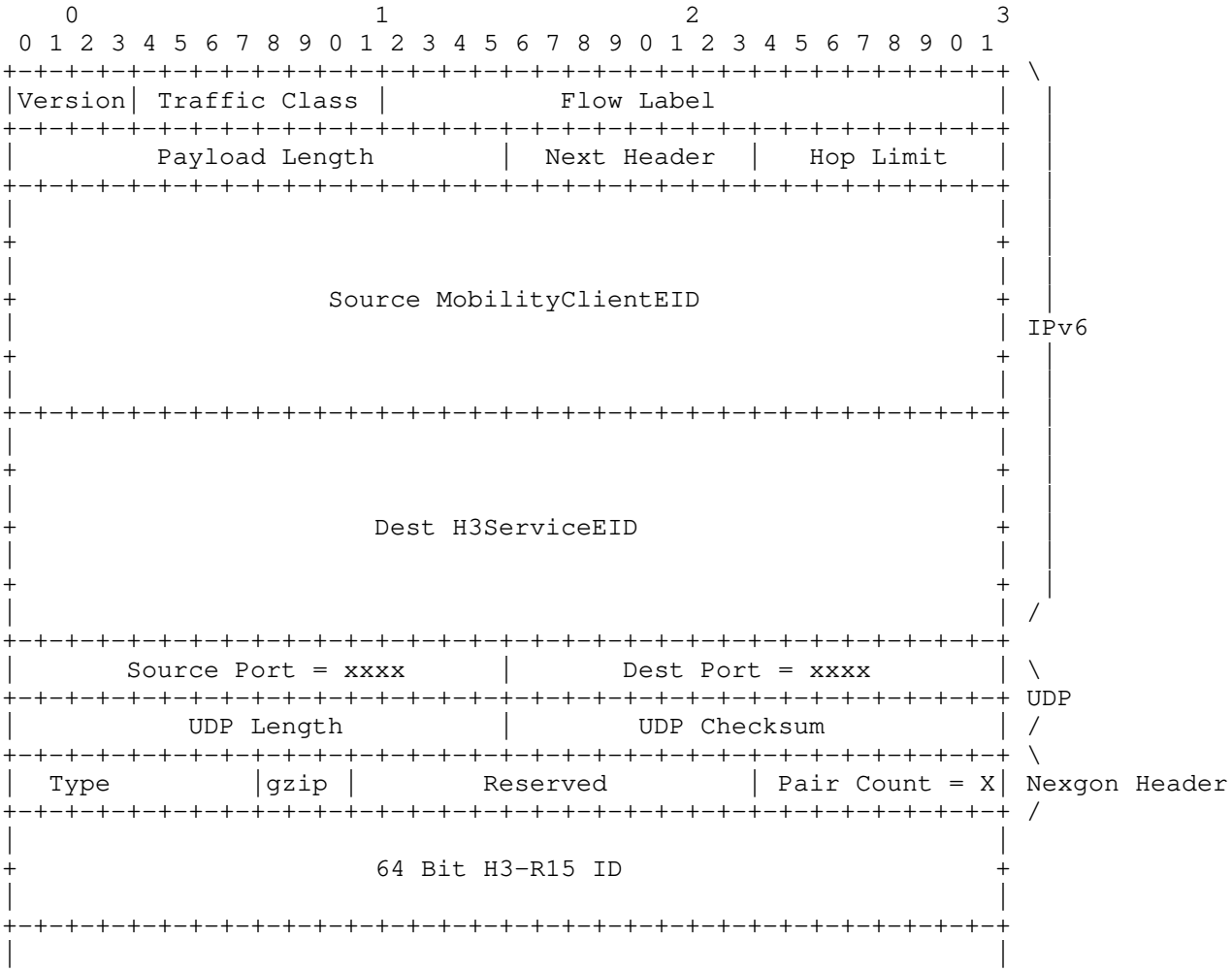
(1) Mobility-Clients traffic is tunneled via data-plane ClientXTRs
ClientXTRs are (multi) homed to EdgeRTR(s)
(2) H3ServiceEID traffic is tunneled via data-plane ServerXTR
ServerXTRs are (multi) homed to EdgeRTR(s)
(3) EdgeRTRs use mapping service to resolve Ucast HIDs to RTR RLOCs
EdgeRTRs also register to (Source, Group) H3ServiceEID multicasts

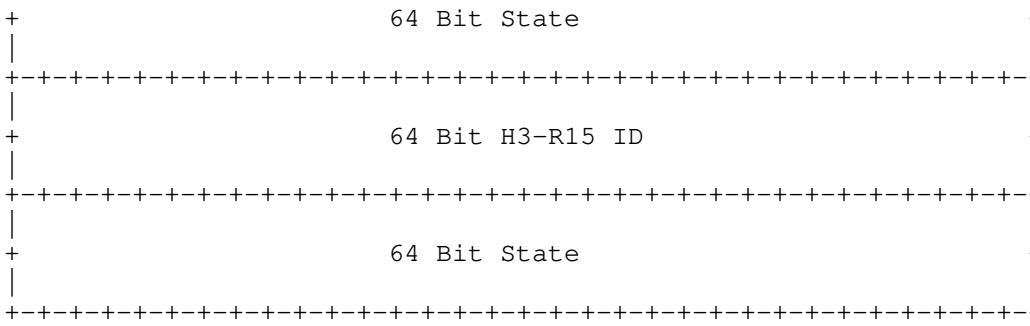
```
MobilityClients <> ClientXTR <Access Provider > EdgeRTR  v
                                                         v
v      << Map-Assisted Mobility-Network Overlay <<       v
v
>> EdgeRTR <Cloud Provider> ServerXTR <> H3ServiceEID
```

6. Mobility Unicast and Multicast

Which ever way a ClientXTR is homed to an Edge RTR an authenticated MobilityClient EID can send: [64bitH3.15ID :: 64bitState] annotation to the H3.r9 H3ServiceEID. The H3.r9 IP HID can be calculated by clients algorithmically form the H3.15 localized snapped-to-tile annotation.

The ClientXTR encapsulates MobilityClient EID and H3ServiceEID in a packet sourced from the ClientXTR, destined to the EdgeRTR RLOC IP, Lisp port. EdgeRTRs then re-encapsulate annotation packets either to remote EdgeRTR (optionI) or to homed H3ServiceEID ServerXTR (option2).
The remote EdgeRTR aggregating H3ServiceEIDs re-encapsulates MobilityClient EID to ServerXTR and from there to the H3ServiceEID.





To Summarize Unicast:

(1) MobilityClients can send annotation state localized an H3.r15 tile
These annotations are sent to an H3.r9 mobility H3ServiceEIDs
(2) MobilityClient EID and H3ServiceEID HID are encapsulated:
XTR <> RTR <> RTR <> XTR
* RTRs can map-resolve re-tunnel HIDs
(3) RTRs re-encapsulate original source-dest to ServerXTRs
ServerXTRs decapsulate packets to H3ServiceEID

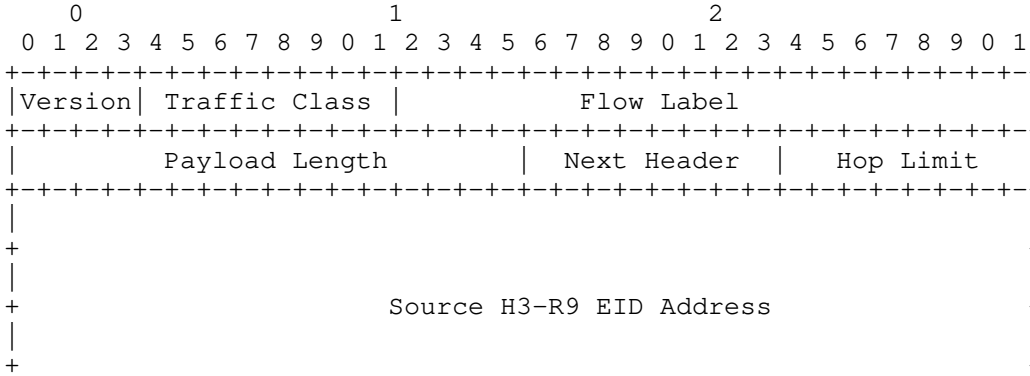
Each H3.r9 Server is used by clients to update H3.r15 tile state is also an IP Multicast channel Source used to update subscribers on the aggregate state of the H3.r15 tiles in the H3.r9 Server.

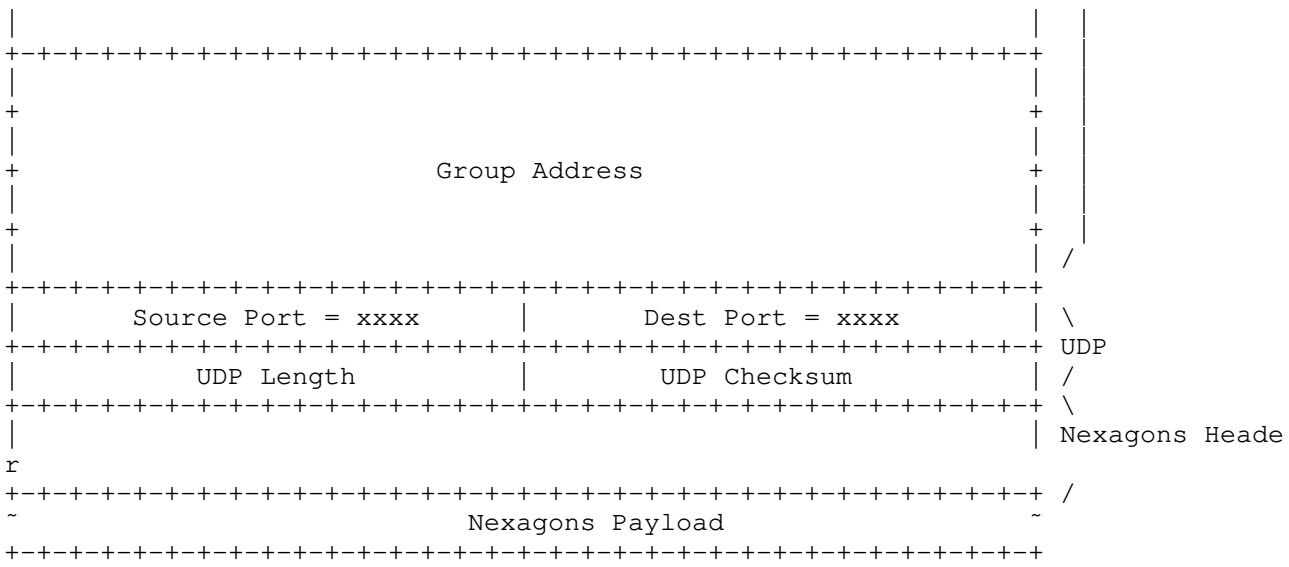
We use rfc8378 signal free multicast to implement mcast channels in the overlay. The mobility network has many channels and relatively few subscribers per each. MobilityClients driving through or subscribing to a a H3.r9 area can explicitly issue an rfc4604 MLDv2 in-order to subscribe, or, may be subscribed implicitly by the EdgeRTR gleaning to ucast HID dest.

The advantage of explicit client MLDv2 registration trigger to rfc8378 is that the clients manage their own mobility mcast hand-over according to their location-direction moment vectors, and that it allows for otherwise silent, or
,
non annotating clients. The advantage of EdgeRTR implicit registration is less signaling required.

MLDv2 signaling messages are encapsulated between the ClientXTR and the LISP EdgeRTR, therefore there is no requirement for the underlying network to support native multicast. If native access multicast is supported (for example native 5G multicast), then MobilityClient registration to H3ServiceEID safety channels may be integrated to it, in which case the evolved-packet-core (EPC) element supporting it (eNB) will use this standard to register with the appropriate H3.r9 channels in its area.

Multicast update packets are of the following structure:

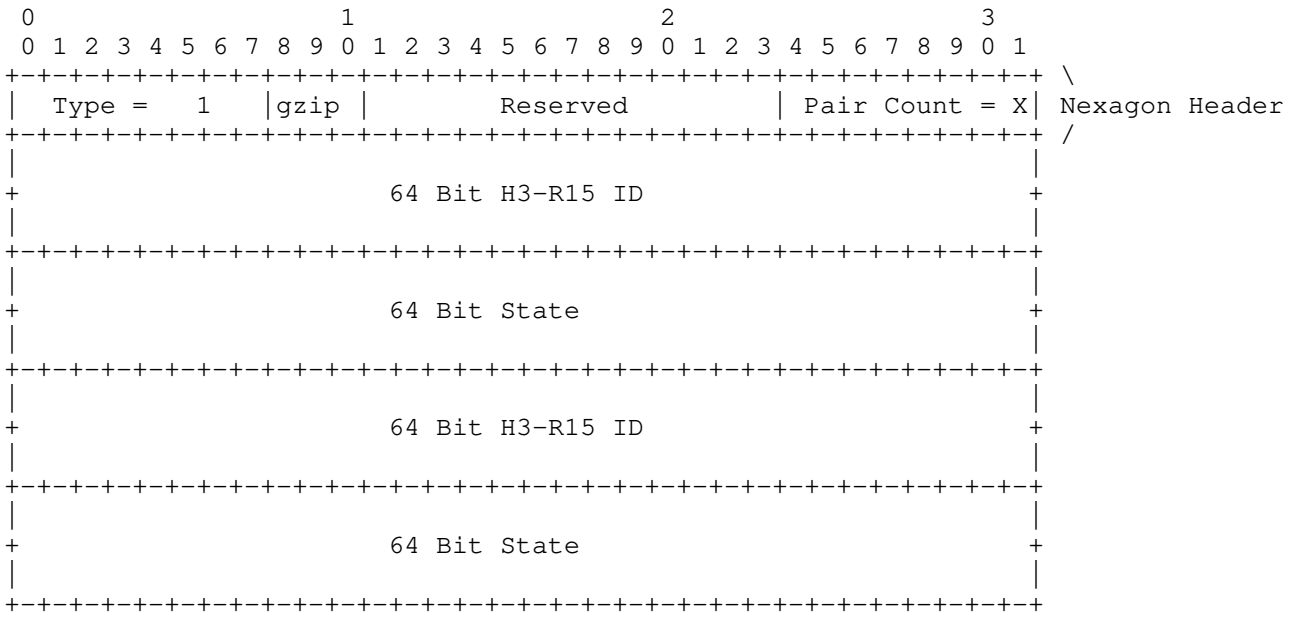


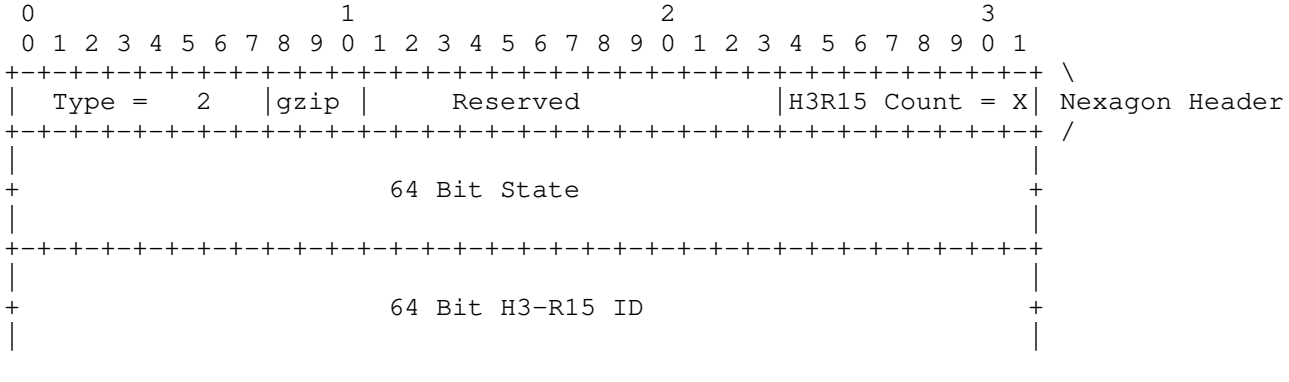


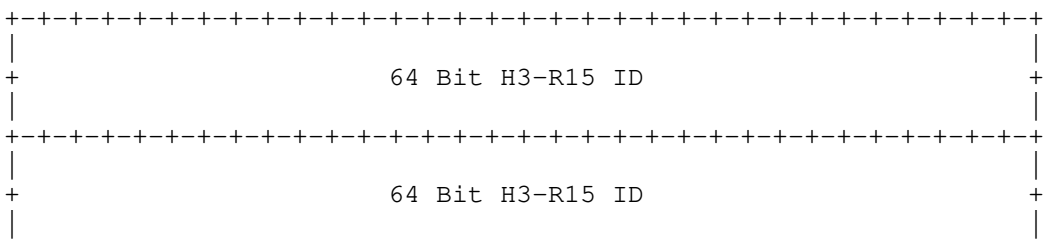
Outer headers = 40 (IPv6) + 8 (UDP) + 8 (LISP) = 56
Inner headers = 40 (IPv6) + 8 (UDP) + 4 (Nexagon Header) = 52

1500 (MTU) - 56 - 52 = 1392 bytes of effective payload

Type 1:key-value, key-value.. 1392 / (8 + 8) = 87 pairs
Type 2:value, key,key,key.. (1392 - 8) / 8 = 173 H3-R15 IDs







` The remote EdgeRTRs homing MobilityClients in-turn replicate the packet to the MobilityClients registered with them.

We expect an average of 600 H3.r15 tiles of the full 7^6 (~100K) possible in H3.r9 to be part of any road. The H3.r9 server can transmit the status of all 600 or just those with meaningful state based on update SLA and policy.

To Summarize:

(1) H3LISP Clients tune to H3.r9 mobility updates using rfc8378
H3LISP Client issue MLDv2 registration to H3.r9 HIDs
ClientXTRs encapsulate MLDv2 to EdgeRTRs who register (s,g)

(2) ServerXTRs encapsulate updates to EdgeRTRs who map-resolve (s,g) RLOCs
EdgeRTRs replicate mobility update and tunnel to registered EdgeRTRs
Remote EdgeRTRs replicate updates to registered ClientXTRs

7. Security Considerations

The nexagon layer3 v2v/v2i/c&c network is inherently more secure and private then alternatives because of the indirection. No car or infrastructure element ever communicates directly with MobilityClients. All information is conveyed using shared / addressable geo-state. MobilityClients are supposed to receive information only from the network as a trusted broker without indication as to the origin of the information. This is an important step towards better privacy, security, extendability, and interoperability.

In order to be able to use the nexagon mobility network for a given period, the mobility clients go through a DNS/AAA stage by which they obtain their clientEID identifiers-credentials and the RLOCs of EdgeRTRs they may use as gateways to the network. This MobilityClient <> EdgeRTR is the most sensitive interface in the network as far as privacy-security.

The traffic on the MobilityClient<>EdgeRTR interface is tunneled and its UDP content may be encrypted, still, the EdgeRTR will know based on the LISP headers alone the MobilityClient RLOC and H3-R9 (~0.1sqkm) geo-spatial area a given client publishes in or subscribes to.

For this reason we envision the ability of enterprise or groups of users to "bring their own" EdgeRTRs. BYO-RTR masks individual clients' IP-RLOC to H3-R9 association and is pre-provisioned to be able to use the mapping system and be on a white-list of EdgeRTRs aggregating H3ServiceEIDs.

Beyond this sensitive hop, the mapping system does not hold MobilityClientEIDs and remote EdgeRTRs are only aware of MobilityClient ephemeral EIDs not their actual IP RLOC or any other mobile-device identifiers. EdgeRTRs register in th
e
mapping (s,g) H3-R9 multicast groups, but which clients reside beyond which EdgeRTR is not in the mapping system. The H3ServiceEIDs them selves of-course decrypt and parse actual H3-R15 annotations, they also consider during this th

e
MobilityClientEID credentials to avoid "fake-news", but again these are only temporary EIDs allocated to clients in-order to be able to use the mobility network and not for their basic communications.

8. Acknowledgments

This work is partly funded by the ANR LISP-Lab project #ANR-13-INFR-009 (https://lisplab.lip6.fr).

9. IANA Considerations

I. Formal H3 to IPv6 EID mapping

II. State enum fields of H3 tiles:

Field 0x: Traffic Direction {
0x - null
1x - Lane North
2x - Lane North + 30
3x - Lane North + 60
4x - Lane North + 90
5x - Lane North + 120
6x - Lane North + 150
7x - Lane North + 180
8x - Lane North + 210
9x - Lane North + 240
Ax - Lane North + 270
Bx - Lane North + 300
Cx - Lane North + 330
Dx - junction
Ex - shoulder
Fx - sidewalk
}

field 1x: Persistent or Structural {
0x - null
1x - pothole light
2x - pothole severe
3x - speed-bump low
4x - speed-bump high
5x - icy
6x - flooded
7x - snow-cover
8x - snow-deep
9x - construction cone
Ax - gravel
Bx - choppy
Cx - blind-curve
Dx - steep-slope
Ex - low-bridge
}

field 2x: Transient Condition {
0x - null
1x - pedestrian
2x - bike scooter
3x - stopped car / truck
4x - moving car / truck
5x - first responder vehicle
6x - sudden slowdown
7x - oversized over-height vehicle
8x - red-light-breach
9x - light collision (fender bender)
Ax - hard collision / casualty
Bx - collision course car/structure

```
 Cx - recent collision residues
 Dx - hard brake
 Ex - sharp cornering
 Fx - freeing-parking
}

field 3x: Traffic-light Cycle {
 0x - null
 1x - 1 seconds to green
 2x - 2 seconds to green
 3x - 3 seconds to green
 4x - 4 seconds to green
 5x - 5 seconds to green
 6x - 6 seconds to green
 7x - 7 seconds to green
 8x - 8 seconds to green
 9x - 9 seconds to green
 Ax - 10 seconds or less
 Bx - 20 seconds or less
 Cx - 30 seconds or less
 Dx - 60 seconds or less
 Ex - green now
 Fx - red now
}

field 4x: Impacted tile from neighboring {
 0x - null
 1x - epicenter
 2x - light yellow
 3x - yellow
 4x - light orange
 5x - orange
 6x - light red
 7x - red
 8x - light blue
 9x - blue
 Ax - green
 Bx - light green
}

field 5x: Transient, Cycle, Impacted, Valid for Next{
 0x - null
 1x - 1sec
 2x - 5sec
 3x - 10sec
 4x - 20sec
 5x - 40sec
 6x - 60sec
 7x - 2min
 8x - 3min
 9x - 4min
 Ax - 5min
 Bx - 10min
 Cx - 15min
 Dx - 30min
 Ex - 60min
 Fx - 24hours
}

field 6x: LaneRightsSigns {
 0x - null
 1x - yield
 2x - speedLimit
 3x - straightOnly
 4x - noStraight
 5x - rightOnly
 6x - noRight
```

```
 7x - rightStraight
 8x - leftOnly
 9x - leftStraight
 Ax - noLeft
 Bx - noUTurn
 Cx - noLeftU
 Dx - bikeLane
 Ex - HOVLane
 Fx - Stop
}

field 7x: MovementSigns {
0x - null
1x - keepRight
2x - keepLeft
3x - stayInLane
4x - doNotEnter
5x - noTrucks
6x - noBikes
7x - noPeds
8x - oneWay
9x - parking
Ax - noParking
Bx - noStandaing
Cx - noPassing
Dx - loadingZone
Ex - railCross
Fx - schoolZone
}

field 8x: CurvesIntersectSigns {
0x - null
1x - turnsLeft
2x - turnsRight
3x - curvesLeft
4x - curvesRight
5x - reversesLeft
6x - reversesRight
7x - windingRoad
8x - hairPin
9x - pretzelTurn
Ax - crossRoads
Bx - crossT
Cx - crossY
Dx - circle
Ex - laneEnds
Fx - roadNarrows
}

field 9x: Current Tile Speed {
0x - null
1x - < 5kmh
2x - < 10kmh
3x - < 15kmh
4x - < 20kmh
5x - < 30kmh
6x - < 40kmh
7x - < 50kmh
8x - < 60kmh
9x - < 80kmh
Ax - < 100kmh
Bx - < 120kmh
Cx - < 140kmh
Dx - < 160kmh
Ex - > 160kmh
Fx - queuedTraffic
}
```

field Ax: Vehicle / Pedestrian Traffic {
0x - null
1x - probability of ped/vehicle on tile close to 100%, packed
2x - 95%
3x - 90%
4x - 85%
5x - 80%
6x - 70%
7x - 60%
8x - 50%
9x - 40%
Ax - 30%
Bx - 20%
Cx - 15%
Dx - 10%
Ex - 5%
Fx - probability of ped/vehicle on tile close to 0%, empty
}

filed Bx - reserved platooning lineup
field Cx - reserved objects of interest
field Dx - reserved
field Ex - reserved
field Fx - reserved

10. Normative References

[I-D.ietf-lisp-rfc6833bis]
Fuller, V., Farinacci, D., and A. Cabellos-Aparicio, "Locator/ID Separation Protocol (LISP) Control-Plane", draft-ietf-lisp-rfc6833bis-07 (work in progress), December 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <https://www.rfc-editor.org/info/rfc6830>.

[RFC8378] Farinacci, D., Moreno, V., "Signal-Free Locator/ID Separation Protocol (LISP) Multicast", RFC8378, DOI 10.17487/RFC8378, May 2018, <https://www.rfc-editor.org/info/rfc8378>.

Authors' Addresses

Sharon Barkai
Nexar
CA
USA

Email: sbarkai@gmail.com

Bruno Fernandez-Ruiz
Nexar
London

UK

Email: b@getnexar.com

S ZionB
Nexar
Israel

Email: sharon@fermicloud.io

Rotem Tamir
Nexar
Israel

rotem.tamir@getnexar.com

Alberto Rodriguez-Natal
Cisco Systems
170 Tasman Drive
San Jose, CA
USA

Email: natal@cisco.com

Fabio Maino
Cisco Systems
170 Tasman Drive
San Jose, CA
USA

Email: fmaino@cisco.com

Albert Cabellos-Aparicio
Technical University of Catalonia
Barcelona
Spain

Email: acabello@ac.upc.edu

Jordi Paillissé-Vilanova
Technical University of Catalonia
Barcelona
Spain

Email: jordip@ac.upc.edu

Dino Farinacci
lispers.net
San Jose, CA
USA

Email: farinacci@gmail.com

COINRG
Internet-Draft
Intended status: Informational
Expires: March 12, 2021

I. Fink
K. Wehrle
RWTH Aachen University

September 8, 2020

Enhancing Security and Privacy with In-Network Computing
draft-fink-coin-sec-priv-01

Abstract

With the growing interconnection of devices, cyber-security and data protection are of increasing importance. This is especially the case regarding cyber-physical systems due to their close entanglement with the physical world. Misbehavior and information leakage can lead to financial and physical damage and endanger human lives and well-being. Thus, hard security and privacy requirements are necessary to be met. Furthermore, a thorough investigation of incidents is essential for ultimate protection. In-network computing allows the processing of traffic and data directly in the network and at line-rate. Thus, the in-network computing paradigm presents a promising solution for efficiently providing security and privacy mechanisms as well as event analysis. This document discusses select mechanisms to demonstrate how in-network computing concepts can be applied to counter existing shortcomings of cyber-security and data privacy.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 12, 2021.

Copyright Notice

Table of Contents

1. Introduction

Several deficiencies emerge from cyber-physical systems (CPS) such as the (Industrial) Internet of Things (IoT). Everyday things are equipped with sensors and CPUs to allow for automatization and make life more comfortable. The deployment of additional sensors supports the processing efficiency in Industrial Control Systems (ICS). The entanglement of the sensors with the physical world leads to high sensitivity of the transmitted and collected data. At the same time, devices are increasingly connected to the Internet to enable, e.g., processing of data on cloud servers or exchange with other systems.

Devices in CPS are often resource-constrained and do not offer the possibility to implement elaborate security mechanisms. Furthermore, legacy devices and communication protocols are often still used in industrial networks but were not designed to face the security and privacy challenges the new interconnection brings. Thus, communication and access are often unprotected, providing new attack surfaces with severe consequences: leakage of private data endangers

the users' privacy. The leakage of business secrets bears the risk of severe financial damage. Manipulation of ICSs can lead to downtimes in the best case or, financially worse, faulty production results. Last, the failure of CPS can lead to personal injury or even death. As a consequence of the described risks, we need security and privacy measures tailored to the new situation. Upgrading legacy devices with protection mechanisms is an effortful and expensive procedure. A promising approach for retrofitting security nevertheless is the deployment of suitable mechanisms within the network. To date, this is mainly realized using middle-boxes, leading to overhead and need for additional hardware.

While proper prevention and detection of attacks in the (Industrial) IoT is an unresolved issue, the after-treatment of incidents in networks offers room for general improvement. We can use network forensics to retrace and comprehend the origin and course of malicious events. However, the underlying monitoring of network traffic requires special hardware leading to high costs in traditional networks.

The common problem of all shortcomings is that traditional networking devices only allow for fixed-function deployment. Software-defined networking (SDN) enables more flexible traffic handling in the network by separating control and data plane. However, the use of fixed-function switches still restricts primary approaches like OpenFlow. Those switches match traffic against a fixed set of protocol headers to decide if and where it should be forwarded. Furthermore, consultation of the remote control plane leads to communication overhead and delays, which is especially unfavorable in the context of time-sensitive applications, e.g., in industry.

INC, in contrast, covers the shortfalls of traditional networks and SDN by allowing actual programming of the switches. This programmability leads to dynamic and custom processing of network packets at line-rate. Thus, security-related functions and packet inspection can be implemented and applied right at the switch.

This draft explores the opportunities of INC for improving security and privacy as follows: we first describe feasible mechanisms for preventing attacks and intrusion in the first place. Then, we present which mechanisms we can implement with INC for detecting intrusion and undesired behavior when it has already taken place. Last, we explore how INC can improve network forensics for analyzing and following up incidents, preventing future attacks.

2. Protection Mechanisms

The common ground for providing security and data privacy is to protect against unauthorized access. That protection is primarily provided by deploying the basic security mechanisms encryption, integrity checking, authentication, and authorization. Those are especially often missing in resource-constrained environments. [RFC7744] thoroughly discusses the need for authentication and authorization in resource-restrained environments. [RFC8576] presents security and privacy risks and challenges specific to the IoT. In the following, we describe how INC can help to retrofit suitable mechanisms.

2.1. Encryption and Integrity Checks

Encryption is critical to preserve confidentiality when transmitting data. Integrity checks prevent undetected manipulation, which can remain unnoticed even despite encryption, e.g., in case of flipped bits. Due to resource-constraints, many devices in CPS do not provide encryption or calculation of check-sums.

Complex cryptography is not supported by current programmable switches either. However, this might change in the future, which would allow retrofitting encryption and integrity checks at networking devices. Concretely, using INC with suitable hardware, data could be encrypted and supplemented with a check-sum directly at the first networking device passed by the respective data packet. The packet is then forwarded through the network or Internet to its designated destination. Decryption and integrity checks can be executed at the last networking device before the destination. Alternatively, this can be implemented at the destination if supported by the respective device. This approach does not require deployment or forwarding to additional middle-boxes. Thus, no additional attack surface or processing overhead is introduced, which is essential for time-sensitive processes as often at hand in the industry.

Overall, INC has the potential to help maintain confidentiality and integrity efficiently, and thus the availability of resource-constrained or legacy devices. Questions to clarify are if and at which costs hardware for enabling cryptographic calculations could and should be embedded in future generations of programmable networking devices.

2.2. Authorization and Authentication

Authorization and authentication mechanisms are needed to avoid unauthorized access to devices and their manipulation in the first place. With INC, networking devices can flexibly decide whether to forward packets, thus enforce authorization and authentication checks.

One possibility for authorization is to conduct a handshake between the sender and networking device before starting the communication with the industrial device. If not feasible in switch hardware, the respective calculations can be conducted in the control plane. In the case of success, the sender is added to a list of authorized communication partners. The decision is then enforced by the switch. Since authorization is only needed when starting or refreshing a connection, the necessity and overhead for consulting the control plane are limited.

The sender can append a secret token for authentication to packets directed to an industrial device. Then, the last networking device can authenticate the sender and forward the actual data only in case of success and drop it otherwise. One possibility to avoid eavesdropping of the token is the use of hash chains. Secure reinitialization can again be done using the control plane, which usually has the resources for conducting encrypted communication.

In the case of unsuccessful authorization or authentication, networking devices can inform the network administrator about possible intrusion of the system.

Undesired traffic can emerge even from authorized and authenticated devices. A solution is to add policy-based access control, on which we elaborate in the next subsection.

2.3. Behavioral and Enterprise Policies

Control processes can include communication between various parties. Even despite authorization and authentication mechanisms, undesired behavior can occur. For instance, malicious third-party software might be installed at the approved device. Regarding communication between two legacy devices, authentication might not be possible at all. An effective way to exclude malicious behavior nevertheless is policy-based access control.

[RFC8520] proposes the Manufacturer Usage Description (MUD), a standard for defining the communication behavior of IoT devices, which use specific communication patterns. The definition is primarily based on domain names, ports, and protocols (e.g., TCP and

UDP). Further characteristics as the TLS usage [I-D.draft-reddy-opsawg-mud-tls-03] or the required bandwidth of a device [I-D.draft-lear-opsawg-mud-bw-profile-01] can help to define connections more narrowly.

By defining the typical behavior, we can exclude deviating communication, including undesired behavior. Likewise to IoT devices, industrial devices usually serve a specific purpose. Thus, the application of MUD or similar policies is possible in industrial scenarios as well.

The problem that remains to date is the efficient enforcement of such policies through fine-granular and flexible traffic filtering. While middle-boxes increase costs and processing overhead, primary SDN approaches as OpenFlow allow only filtering based on match-action rules regarding fixed protocol header fields. Evaluation of traffic statistics for, e.g., limiting the bandwidth, requires consultation of the remote controller. This leads to latency overheads, which are not acceptable in time-sensitive scenarios.

In contrast, the INC paradigm allows flexible filtering even concerning the content of packets and connection metadata. Furthermore, traffic filtering can be executed at line-rate in the switch.

Going one step further, not only network communication behavior of devices can be defined in policies. As [KANG] shows, INC can be used to consider additional (contextual) parameters, e.g., the time of day or activity of other devices in the network. Furthermore, companies can define advanced policies to, e.g., authorize specific users or subnets.

While the presented policies aim to restrict communication to its designated purpose, we can use access control to explicitly address individual devices' security vulnerabilities as described next.

2.4. In-Network Vulnerability Patches

Resource-constrained devices are typically hard to update. Thus, device vulnerabilities often cannot be fixed after deployment. As a remedy and special case of policies, rules can be defined to describe known attacks' signatures. By enforcing these rules at programmable networking devices, e.g., by dropping matching traffic, INC offers an efficient way to avoid exploitation of device vulnerabilities. Further advantages are the potentially easy and extensive roll-out of such "in-network patches" in the form of (automatic) software updates of the network device.

Future research is needed to evaluate the potential and benefits of in-network patches compared to traditional security measures, e.g., firewalls, and provide proof of concepts using existing devices and vulnerabilities.

Besides presented security mechanisms, data protection mechanisms are required to preserve business secrets and the privacy of individuals. We show in the following subsection how INC can contribute to data anonymization.

2.5. Anonymization

Due to its interconnection with the physical world, the generation of sensitive data is inherent to CPS. Smart infrastructure leads to the collection of sensitive user data. In industrial networks, information about confidential processes is gathered. Such data is increasingly shared with other entities to increase production efficiency or enable automatic processing.

Despite the benefits of data exchange, manufacturers and individuals, might not want to share sensitive information. Again, deployment of privacy mechanisms is usually not possible at resource-constrained or legacy devices. INC has the potential to flexibly apply privacy mechanisms at line-rate.

Data can be pseudonymized at networking devices by, e.g., extracting and replacing specific values. Furthermore, elaborate anonymization techniques can be implemented in the network by sensibly decreasing the data accuracy. For example, concepts like k-Anonymity can be applied by aggregating the values of multiple packets before forwarding the result. Noise addition can be implemented by adding a random number to values. Similarly, the state-of-the-art technique differential privacy can be implemented by adding noise to responses to statistical requests.

Even though the INC paradigm shows the potential to deploy described privacy mechanisms within the network, research is needed to clarify the proposed concepts' feasibility.

3. Intrusion and Anomaly Detection

Ideally, attacks are prevented from the outset. However, in the case of incidents, fast detection is critical for limiting damage. Deployment of sensors, e.g., in industrial control systems, can help to monitor the system state and detect anomalies. This can be used in combination with INC to detect intrusion and to provide advanced safety measures, as described in the following.

3.1. Intrusion Detection

Data of sensors or communication behavior can be compared against expected patterns to detect intrusion. Even if intrusion prevention is deployed and connections are allowed when taken individually, subtle attacks might still be possible. For example, a series of values might be out of line if put into context even though the individual values are unobtrusive. Anomaly detection can be used to detect such abnormalities and notify the network administrator for further assessment.

While anomaly detection is usually outsourced to middle-boxes or external servers, INC provides the possibility to detect anomalies at-line rate, e.g., by maintaining statistics about traffic flows. This decreases costs and latency, which is valuable for a prompt reaction.

Besides intrusion, anomalies can also imply safety risks. In the following, we pick up the potential of INC to support safety.

3.2. Dead Man's Switch

[I-D.draft-kunze-coin-industrial-use-cases-03] addresses the potential of INC for improving industrial safety. Detection of an anomaly in the sensor data or operational flow can be used to automatically trigger an emergency shutdown of a system or single system components if the data indicates an actual hazard. Apart from that, other safety measures like warning systems or isolation of areas can be implemented. While we do not aim at replacing traditional dead man's switches, we see the potential of INC to accelerate the detection of failures. Thus, INC can valuably complement existing safety measures.

4. Incident Investigation

After detecting an incident, it is essential to conduct Network Forensics to investigate the origin and spreading of the related activity. The results of this analysis can be used to allow for consistent recovery, to adapt protection mechanisms, and prevent similar events in the future. For enabling potential investigation, traffic records are constantly collected for each flow in a network. This requires additional hardware in large networks. Furthermore, it might be preferable to exclude, e.g., specific subnets from the analysis. This is not easily possible with traditional networking devices, leading to storage and processing overhead.

With INC, flow records can be created directly at the switch when forwarding a packet. Furthermore, record generation can be done more

flexibly, e.g., by applying fine-granular traffic filtering. Also, header fields of particular interest can be efficiently extracted. Therefore, INC can considerably decrease the load and increase the efficiency of network forensics. This leads, in turn, to a better understanding of attacks and security.

5. Security Considerations

When implementing security and privacy measures in networking devices, the networking devices' security and failure resistance is critical. Related research questions to clarify in the future are stated in [I-D.draft-kutscher-coinrg-dir-01].

6. IANA Considerations

N/A

7. Conclusion

INC has the potential to improve and retrofit security and privacy, especially in concern of resource-restrained and legacy devices.

First, INC can provide intrusion prevention mechanisms like authentication and efficient enforcement of (context-based) policies. Easily deployable in-network patches of device vulnerabilities could further improve security. Encryption and integrity checks are limited by the current hardware but might be realizable in the future.

Second, INC allows examining packet contents at networking devices, which can be used to implement fast anomaly and intrusion detection in the network.

Last, INC can contribute to an efficient and targeted incident analysis.

Investigation of the feasibility of the presented mechanisms is subject to future research.

8. Informative References

[I-D.draft-kunze-coin-industrial-use-cases-03]
Kunze, I. and K. Wehrle, "Industrial Use Cases for In-Network Computing", draft-kunze-coin-industrial-use-cases-03 (work in progress), September 2020.

[I-D.draft-kutscher-coinrg-dir-01]
Kutscher, D., Karkkainen, T., and J. Ott, "Directions for Computing in the Network", draft-kutscher-coinrg-dir-01 (work in progress), November 2019.

[I-D.draft-lear-opsawg-mud-bw-profile-01]
Lear, E. and O. Friel, "Bandwidth Profiling Extensions for MUD", draft-lear-opsawg-mud-bw-profile-01 (work in progress), July 2019.

[I-D.draft-reddy-opsawg-mud-tls-03]
Reddy, T., Wing, D., and B. Anderson, "MUD (D)TLS profiles for IoT devices", draft-reddy-opsawg-mud-tls-03 (work in progress), January 2019.

[KANG] Kang, Q., Morrison, A., Tang, Y., Chen, A., and X. Luo, "Programmable In-Network Security for Context-aware BYOD Policies", In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), August 2020, <https://www.usenix.org/conference/usenixsecurity20/presentation/kang>.

[RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <https://www.rfc-editor.org/info/rfc7744>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

[RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <https://www.rfc-editor.org/info/rfc8576>.

Authors' Addresses

Ina Berenice Fink
RWTH Aachen University
Ahornstr. 55
Aachen D-52062
Germany

Phone: +49-241-80-21419
Email: fink@comsys.rwth-aachen.de

Klaus Wehrle
RWTH Aachen University
Ahornstr. 55
Aachen D-52062
Germany

Phone: +49-241-80-21401
Email: wehrle@comsys.rwth-aachen.de

COINRG
Internet-Draft
Intended status: Informational
Expires: September 10, 2020

I. Kunze
K. Wehrle
RWTH Aachen University

March 09, 2020

Industrial Use Cases for In-Network Computing
draft-kunze-coin-industrial-use-cases-02

Abstract

Cyber-physical systems and the Industrial Internet of Things are characterized by diverse sets of requirements which can hardly be satisfied using standard networking technology. One example are latency-critical computations which become increasingly complex and are consequently outsourced to more powerful cloud platforms for feasibility reasons. The intrinsic physical propagation delay to these remote sites can already be too high for given requirements. The challenge is to develop techniques that bring together these requirements. Utilizing available computational capabilities within the network for in-network computing concepts can be a solution to this challenge. This document discusses selected industrial use cases to demonstrate how in-network computing concepts can be applied to the industrial domain and to point out essential requirements of industrial applications.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2020.

Copyright Notice

Table of Contents

1. Introduction

The Internet is based on a best-effort network that provides limited guarantees regarding the timely and successful transmission of packets. This design-choice is suitable for general Internet-based applications, but specialized industrial applications demand a number of strict performance guarantees, e.g., regarding real-time capabilities, which cannot be provided over regular best-effort networks.

Enhancements to the standard Ethernet such as Time-Sensitive-Networking [TSN] try to achieve the requirements on the link layer by statically reserving shares of the bandwidth. These concepts are well-suited for industrial settings with well understood communication patterns where the communication paths are encapsulated at the factory sites. In the Industrial Internet of Things (IIoT), however, more and more parts of the industrial production domain are interconnected. This increases the complexity of the industrial

networks, makes them more dynamic, and creates more diverse sets of requirements. Furthermore, process control is imagined to be exercised from remote clouds for feasibility reasons which is why solutions on the link layer alone are not sufficient in these scenarios.

Common components of the IIoT can be divided into three categories as illustrated in Figure 1. Following [I-D.mcbride-edge-data-discovery-overview], EDGE DEVICES, such as sensors and actuators, constitute the boundary between the physical and digital world. They communicate the current state of the physical world to the digital world by transmitting sensor data or let the digital world interact with the physical world by executing actions after receiving (simple) control information. The processing of the sensor data and the creation of the control information is done on COMPUTING DEVICES. They range from small-powered controllers close to the EDGE DEVICES, to more powerful edge or remote clouds in larger distances. The connection between the EDGE and COMPUTING DEVICES is established by NETWORKING DEVICES. In the industrial domain, they range from standard devices, e.g., typical Ethernet switches, which can interconnect all Ethernet-capable hosts, to proprietary equipment with proprietary protocols only supporting hosts of specific vendors.

The challenge is to develop concepts that can include off-premise entities (such as distant cloud platforms) as well as proprietary

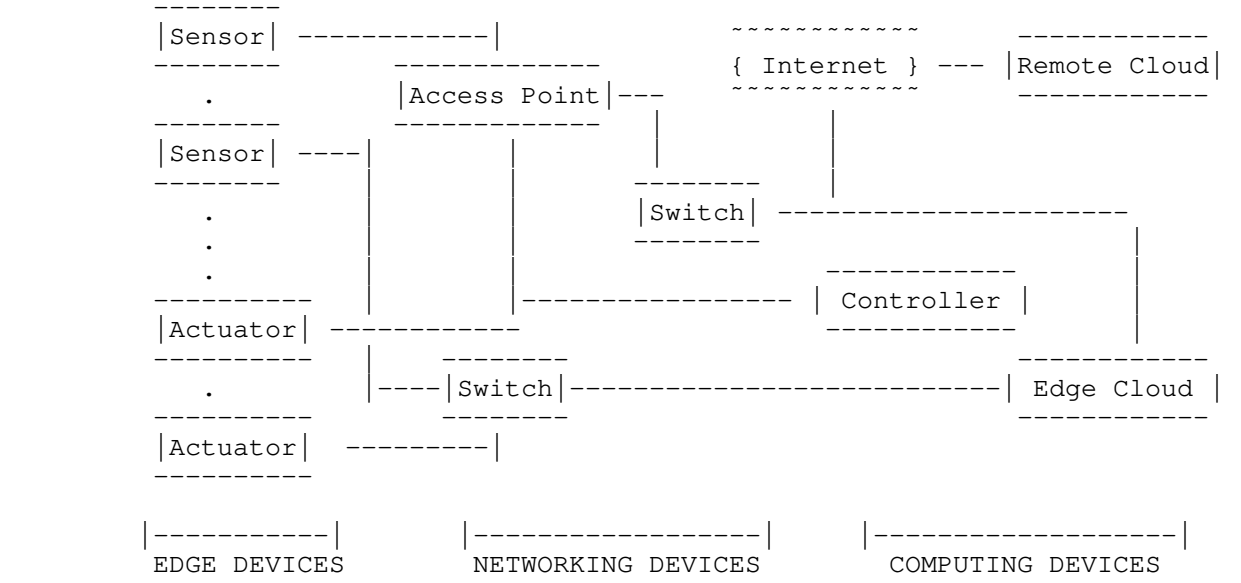


Figure 1: Industrial networks show a high level of heterogeneity.

hosts into the communication and still satisfy the performance requirements of modern industrial networks. The in-network computing paradigm presents a promising starting point because (pre-)processing data within the network can speed up the communication, e.g., by reducing the amount of transmitted data and thus congestion. Flexibly distributing the computation tasks across the network helps to manage dynamic changes. Specifying general requirements for the different application scenarios is difficult due to the mentioned diversity. This draft characterizes and analyzes three distinct scenarios to showcase potential requirements for the industrial production domain and to illustrate how in-network computations can be helpful.

2. In-Network Control / Time-sensitive applications

The control of physical processes and components of a production line is essential for the growing automation of production and ideally allows for a consistent quality level. Traditionally, the control has been exercised by control software running on programmable logic controllers (PLCs) located directly next to the controlled process or component. This approach is best-suited for settings with a simple model that is focussed on a single or few controlled components.

Modern production lines and shop floors are characterized by an increasing amount of involved devices and sensors, a growing level of dependency between the different components, and more complex control models. A centralized control is desirable to manage the large amount of available information which often has to be pre-processed or aggregated with other information before it can be used. PLCs are not designed for this array of tasks and computations could theoretically be moved to more powerful devices. These devices are no longer close to the controlled objects and induce additional latency.

It is worthwhile to investigate whether the outsourcing of control functionality to distant computation platforms is viable because these platforms have a high level of flexibility and scalability. In the following, we describe the requirements and characteristics of the control setting in more detail.

2.1. Characterization and Requirements

A control process consists of two main components as illustrated in Figure 2: a system under control and a controller. In feedback control, the current state of the system is monitored, e.g., using sensors and the controller influences the system based on the difference between the current and the reference state to keep it close to this reference state.

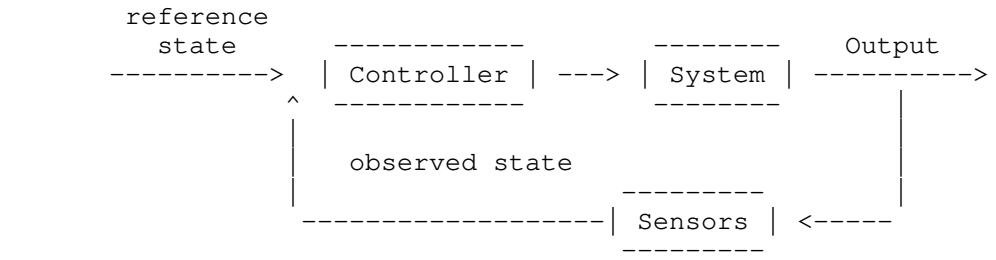


Figure 2: Simple feedback control model.

Apart from the control model, the quality of the control primarily depends on the timely reception of the sensor feedback, because the controller can only react if it is notified of changes in the system state. Depending on the dynamics of the controlled system, the control can be subject to tight latency constraints, often in the single-digit millisecond range. While low latencies are essential, there is an even greater need for stable and deterministic levels of latency, because controllers can generally cope with different levels of latency, if they are designed for them, but they are significantly challenged by dynamically changing or unstable latencies. The unpredictable latency of the Internet exemplifies this problem if off-premise cloud platforms are included.

The main requirements for the industrial control scenario are low and stable latencies to ensure that processes can work continuously and that no machines are damaged.

2.1.1. Approaches

Control models, in general, can become involved but there is a variety of control algorithms that are composed of simple computations such as matrix multiplication. As these are supported by programmable network devices, it is possibile to compose simplified approximations of the more complex algorithms and deploy them in the network. While the simplified versions induce a more inaccurate control, they allow for a quicker response and might be sufficient to operate a basic tight control loop while the overall control can still be exercised from the cloud. The problem, however, is that networking devices typically only allow for integer precision computation while floating-point precision is needed by most control algorithms. Early approaches like [RUETH] have already shown the general applicability of such ideas, but there are still a lot of open research questions not limited to the following:

- How can one derive the simplified versions of the overall controller?

    * How complex can they become?

    * How can one take the limited computational precision of networking devices into account when making them?

  o How does one distribute the simplified versions in the network?

  o How does the overall controller interact with the simplified versions?

3. Large Volume Applications/ Traffic Filtering

In the IIoT, processes and machines can be monitored more effectively resulting in more available information. This data can be used to deploy machine learning (ML) techniques and consequently help to find previously unknown correlations between different components of the production which in turn helps to improve the overall production system. Newly gained knowledge can be shared between different sites of the same company or even between different companies.

Traditional company infrastructure is neither equipped for the management and storage of such large amounts of data nor for the computationally expensive training of ML approaches. Similar to the considerations in Section 2, off-premise cloud platforms offer cost-effective solutions with a high degree of flexibility and scalability. While the unpredictable latency of the Internet is only a subordinate problem for this use case, moving all data to off-premise locations primarily poses infrastructural challenges which are presented in more detail in the following.

3.1. Characterization and Requirements

Processes in the industrial domain are monitored by distributed sensors which range from simple binary (e.g., light barriers) to sophisticated sensors measuring the system with varying degrees of resolution. Sensors can further serve different purposes, as some might be used for time-critical process control while others are only used as redundant fallback platforms. Overall, there is a high level of heterogeneity which makes managing the sensor output a challenging task.

Depending on the deployed sensors and the complexity of the observed system, the resulting overall data volume can easily be in the range of several Gbit/s [GLEBKE]. Using off-premise clouds for managing the data requires uploading or streaming the growing volume of sensor data using the companies' Internet access which is typically limited to a few hundred of Mbit/s. While large networking companies can simply upgrade their infrastructure, most industrial companies rely

on traditional ISPs for their Internet access. Higher access speeds are hence tied to higher costs and, above all, subject to the supply of the ISPs and consequently not always available. A major challenge is thus to devise a methodology that is able to handle such amounts of data over limited access links.

Another aspect is that business data leaving the premise and control of the company further comes with security concerns, as sensitive information or valuable business secrets might be contained in it. Typical security measures such as encrypting the data make in-network computing techniques hardly applicable as they typically work on unencrypted data. Adding security to in-network computing approaches, either by adding functionality for handling encrypted data or devising general security measures, is thus an auspicious field for research which we describe in more detail in Section 5.

3.2. Approaches

There are at least two concepts which might be suitable for reducing the amount of transmitted data in a meaningful way:

1. filtering out redundant or unnecessary data

2. aggregating data by applying pre-processing steps within the network

Both concepts require detailed knowledge about the monitoring infrastructure at the factories and the purpose of the transmitted data.

3.2.1. Traffic Filters

Sensors are often set up redundantly, i.e., part of the collected data might also be redundant. Moreover, they are often hard to configure or not configurable at all which is why their resolution or sampling frequency is often larger than required. Consequently, it is likely that more data is transmitted than is needed or desired. A trivial idea for reducing the amount of data is thus to filter out redundant or undesired data before it leaves the premise using simple traffic filters that are deployed in the on-premise network. There are different approaches to how this topic can be tackled. A first step would be to scale down the available sensor data to the data rate that is needed. For example, if a sensor transmits with a frequency of 5 kHz, but the control entity only needs 1 kHz, only every fifth packet containing sensor data is let through. Alternatively, sensor data could be filtered down to a lower frequency while the sensor value is in an uninteresting range, but let through with higher resolution once the sensor value range

becomes interesting. It is important that end-hosts are informed about the filtering so that they can distinguish between data loss and data filtered out on purpose.

In this context, the following research questions can be of interest:

- How can traffic filters be designed?
- How can traffic filters be coordinated and deployed?
- How can traffic filters be changed dynamically?
- How can traffic filtering be signaled to the end-hosts?

3.2.2. In-Network (Pre-)Processing

There are manifold computations that can be performed on the sensor data in the cloud. Some of them are very complex or need the complete sensor data during the computation, but there are also simpler operations which can be done on subsets of the overall dataset or earlier on the communication path as soon as all data is available. One example is finding the maximum of all sensor values which can either be done iteratively on each intermediate hop or at the first hop, where all data is available.

Using expert knowledge about the exact computation steps and the concrete transmission path of the sensor data, simple computation steps can be deployed in the on-premise network to reduce the overall data volume and potentially speed up the processing time in the cloud.

Related work has already shown that in-network aggregation can help to improve the performance of distributed ML applications [SAPIO]. Investigating the applicability of stream data processing techniques to programmable networking devices is also interesting, because sensor data is usually streamed. In this context, the following research questions can be of interest:

- Which (pre-)processing steps can be deployed in the network?
  - How complex can they become?
- How can applications incorporate the (pre-)processing steps?
- How can the programming of the techniques be streamlined?

4. Industrial Safety (Dead Man's Switch)

Despite increasing automation in production processes, human workers are still often necessary. Consequently, safety measures have a high priority to ensure that no human life is endangered. In traditional factories, the regions of contact between humans and machines are well-defined and interactions are simple. Simple safety measures like emergency switches at the working positions are enough to provide a decent level of safety.

Modern factories are characterized by increasingly dynamic and complex environments with new interaction scenarios between humans and robots. Robots can either directly assist humans or perform tasks autonomously. The intersect between the human working area and the robots grows and it is harder for human workers to fully observe the complete environment.

Additional safety measures are essential to prevent accidents and support humans in observing the environment. The increased availability of sensor data and the detailed monitoring of the factories can help to build additional safety measures if the corresponding data is collected early at the correct position.

4.1. Characterization and Requirements

Industrial safety measures are typically hardware solutions because they have to pass rigorous testing before they are certified and deployment-ready. Standard measures include safety switches, which need to be triggered manually, and light barriers. Additionally, the working area can be explicitly divided into 'contact' and 'safe' areas, indicating when workers have to watch out for interactions with machinery.

These measures are static solutions, potentially relying on specialized hardware, and are challenged by the increased dynamics of modern factories where the factory configuration can be changed on demand. Software solutions offer higher flexibility as they can dynamically respect new information gathered by the sensor systems. Depending on the corresponding occupational safety laws, the software has to satisfy stringent requirements which cannot be satisfied by regular best-effort networks.

4.1.1. Approaches

Software-based solutions can take advantage of the large amount of available sensor data. Different safety indicators within the production hall can be combined within the network so that programmable networking devices can give early responses if a

potential safety breach is detected. A rather simple possibility could be to track the positions of human workers and robots. Whenever a robot gets too close to a human in a non-working area or if a human enters a defined safety zone, robots are stopped to prevent injuries. More advanced concepts could also include image data or combine arbitrary sensor data.

In this context, the following research questions can be of interest:

- How can the software give guaranteed safety over best-effort networks?
- Which sensor information can be combined and how?

5. Security Considerations

Current in-network computing approaches typically work on unencrypted plain text data because today's networking devices usually do not have crypto capabilities. As is already mentioned in Section 3.1, this above all poses problems when business data, potentially containing business secrets, is streamed into remote computing facilities and consequently leaves the control of the company. Insecure on-premise communication within the company and on the shop-floor is also a problem as machines could be intruded from the outside. It is thus crucial to deploy security and authentication functionality on on-premise and outgoing communication although this might interfere with in-network computing approaches. Ways to implement and combine security measures with in-network computing are described in more detail in [I-D.fink-coin-sec-priv].

6. IANA Considerations

N/A

7. Conclusion

In-network computing concepts have the potential to improve industrial applications. There are at least three scenarios for which in-network processing can be beneficial, each having a unique set of requirements.

In the control scenario, tight latency constraints in the single digit millisecond range have to be satisfied despite the use of cloud platforms and the corresponding unstable latency of the Internet.

In a second scenario, large amounts of data have to be transmitted to cloud platforms for further evaluation. One important task here is to reduce the amount of data that needs to be transmitted as the

available Internet access speed is most likely non-sufficient. Apart from that, security measures have to be implemented as business data is transmitted to the Internet.

Regarding safety, software-based measures often lack the required guarantees and do not withstand the testing for certification. In-network processing with its potential for early responses can be a solution by combining different sensor outputs early and acting quickly.

8. Informative References

[GLEBKE] Glebke, R., "A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems", DOI: 10125/60162, in HICSS, January 2019.

[I-D.fink-coin-sec-priv] Fink, I. and K. Wehrle, "Enhancing Security and Privacy with In-Network Computing", draft-fink-coin-sec-priv-00 (work in progress), March 2020.

[I-D.mcbride-edge-data-discovery-overview] McBride, M., Kutscher, D., Schooler, E., and C. Bernardos, "Edge Data Discovery for COIN", draft-mcbride-edge-data-discovery-overview-03 (work in progress), January 2020.

[RUETH] Rueth, J., "Towards In-Network Industrial Feedback Control", DOI: 10.1145/3229591.3229592, in ACM SIGCOMM NetCompute, August 2018.

[SAPIO] Sapio, A., "Scaling Distributed Machine Learning with In-Network Aggregation", 2019, <https://arxiv.org/abs/1903.06701>.

[TSN] "Time-Sensitive Networking (TSN) Task Group", 2019, <https://1.ieee802.org/tsn/>.

Authors' Addresses

Ike Kunze
RWTH Aachen University
Ahornstr. 55
Aachen D-50274
Germany

Phone: +49-241-80-21422
Email: kunze@comsys.rwth-aachen.de

Klaus Wehrle
RWTH Aachen University
Ahornstr. 55
Aachen D-50274
Germany

Phone: +49-241-80-21401
Email: wehrle@comsys.rwth-aachen.de

COINRG
Internet-Draft
Intended status: Informational
Expires: 6 May 2021

I. Kunze
K. Wehrle
RWTH Aachen University
D. Trossen
Huawei Technologies Duesseldorf GmbH

2 November 2020

Transport Protocol Issues of In-Network Computing Systems
draft-kunze-coinrg-transport-issues-03

Abstract

Today's transport protocols offer a variety of functionality based on the notion that the network is to be treated as an unreliable communication medium. Some, like TCP, establish a reliable connection on top of the unreliable network while others, like UDP, simply transmit datagrams without a connection and without guarantees into the network. These fundamental differences in functionality have a significant impact on how COIN approaches can be designed and implemented. Furthermore, traditional transport protocols are not designed for the multi-party communication principles that underlie many COIN approaches. This document raises several questions regarding the use of transport protocols in connection with COIN.

Table of Contents

1. Introduction

A fundamental design choice of the Internet is that the network should be kept as simple as possible while complexity in the form of processing should be located on end-hosts at the edges of the network. This choice is reflected in the end-to-end principle which states that end-hosts directly address each other and perform all relevant computations while the network only delivers the packets without modifying them. Transport protocols are consequently designed to facilitate the direct communication between end-hosts.

In practice, the end-to-end principle is often violated by intransparent middleboxes which alter transmitted packets, e.g., by dropping or changing header fields. Contrary to that, computing in the network (COIN) encourages explicit computations in the network which introduces an intertwined complexity as the computations on the end-hosts depend on the functionality deployed in the network. It further challenges traditional end-to-end transport protocols as they are not designed to address in-network computation entities or to include more than two devices into a communication. Some of these problems are already presented in [I-D.draft-kutscher-coinrg-dir-02].

This draft discusses potential problems for traditional transport protocols in more detail to raise questions that the COIN community needs to solve before a widespread application of COIN functionality is sensible. Collaboration with other IRTF and IETF groups, such as PANRG, the IETF transport area in general, or the LOOPS BOF, can help in finding suitable solutions.

2. Terminology

COIN element: Device on which COIN functionality can be deployed

3. Addressing

The traditional addressing concept of the Internet is that end-hosts directly address each other with all computational intelligence residing at the network edges. With COIN, computations move into the network and need to be integrated into the established infrastructure. In systems where the whole network is under the control of the network operator this integration can be implemented by explicitly adjusting the communication schemes based on the COIN functionality. Considering larger scales, this approach of manually adjusting traffic patterns and applications to correctly incorporate changes made by the network is not feasible.

What is needed are ways to specify which kind of functionality should be applied to the transmitted data on the path inside the network and maybe even where or by whom the execution should take place. Such functionality could for example be implemented using an indirection mechanism which routes a packet along a pre-defined or dynamically chosen path which then realizes the desired functionality.

One possibility is to directly route on service or functionality identifiers instead of sending individual packets between locator-addressed network elements [I-D.draft-sarathchandra-coin-appcentres-03]. While this aligns the routing more clearly with the communication between computational elements, selecting the 'right' computational endpoint (out of possibly several ones) becomes critical to the proper functioning of the overall service.

Further related concepts are Segment and Source Routing as well as (Service/Network) Function Chaining/Composition.

Main challenges/questions are:

1. How should end-hosts address the COIN functionality?

2. How can the treatment of the transmitted data, i.e., which COIN functionality to execute, be represented in the addressing of the request?

3. How can end-hosts direct computational requests to different computational endpoints of the same service in different network locations, i.e., decide where the COIN functionality is executed?

4. How to decide which computational endpoint to choose (from possibly several ones existing in the network)?

5. How can devices which do not implement COIN functionality be integrated into the systems without breaking the COIN or legacy functionality?

4. Flow granularity

Core networking hardware pipelines such as backbone switches are built to process incoming packets on a per-packet basis, keeping little to no state between them. This is appropriate for the general task of forwarding packets, but might not be sufficient for COIN as information that is needed for the computations can be spread across several packets. In a TCP stream, for example, data is dynamically distributed across different segments which means that the data needed for application-level computations might also be split up. In contrast to that the content of UDP datagrams is defined by the application itself which is why the datagrams could either be self-contained or information can be cleverly distributed onto different datagrams. Summarizing, different transport protocols induce different meanings to the packets that they send out which needs to be accounted for in COIN elements as they have to know how the received data is to be interpreted. There are at least three options for this.

1. Every packet is treated individually. This respects the possibilities that are already offered by all networking equipment.

2. Every packet is treated as part of a message. In this setting, the packet alone does not have enough information for the computations. Instead, it is important to know the content of the surrounding packets which together form the overall message.

3. Every packet is treated as part of a byte stream. Here, all previous packets and potentially even all subsequent packets need to be taken into consideration for the computations as the current packet could, e.g., be the first of a group of packets, a packet in the middle, or the final packet.

Along those options above, the question arises how shorter-term 'messages' (or transactions) of the computation should be handled compared to the often longer-term management of the network resources needed to transmit the packets across one or more such messages. For instance, error control may be best applied to the individual messages between computational endpoints, while congestion control may be applied across several messages at the level of the relation between the network elements hosting the computational endpoints. In this view, the notion of a 'flow' may separate message or transaction handling from the resource management aspect, where a flow may be divided into sub-flows (said messages or transactions) with error control being applied to those sub-flows but resource management being applied to the overall flow. Such choice of flow granularity would consequently have a significant impact on how and where computations can be performed as well as ensuring that end-hosts know who has altered the data and how.

5. Collective Communication

COIN scenarios may exhibit a collective communication semantic, i.e., a communication between one and more computational endpoints, as is for example illustrated by use cases in [I-D.draft-sarathchandra-coin-appcentres-03]. With this, unicast and multicast transmissions become almost equal forms of communication, as also observed in work on information-centric networking (ICN).

Yet, these many-point relations may be ephemeral down to the granularity of individual service requests between computational endpoints which questions the viability of stateful routing and transport approaches used for long-lived multicast scenarios such as liveTV transmissions.

This is particularly pertinent for the transport layer where reliability and flow control among a quickly changing set of receivers is a challenging problem. The ability to divide receiver groups with the support of in-network COIN elements may provide solutions that will cater to the possible dynamics of collective communication among computational endpoints.

6. Authentication

The realisation of COIN legitimizes and actively promotes that data transmitted from one host to another can be altered on the way inside the network. This opens the door for foul play as all intermediate network elements - no matter if they are malicious or misbehaving by accident, COIN elements, or 'traditional' middleboxes - could simply start altering parts of the original data and potentially cause harm to the end-hosts. What is needed are mechanisms with which the

receiving host can verify (a) how and (b) by whom the data has been altered on the way. In fact, these might very well be two distinct mechanisms as one (a) only focusses on the changes that are made to the data while (b) requires a scheme with which COIN elements can be uniquely identified (could very well relate to Section 3) and subsequently authenticated. The Proof of Transit [I-D.draft-ietf-sfc-proof-of-transit-08] concept of the SFC WG could be applicable for proving that a packet has indeed passed the desired COIN elements, although it does not provide means to validate which changes were made by the known nodes.

Main challenges/questions are:

1. How are changes to the data within the network communicated to the end-hosts?

2. How are the COIN elements that are responsible for the changes communicated to the end-hosts?

3. How are changes made by the COIN elements authenticated?

7. Security

Many early COIN concepts require an unencrypted transmission of data. At the same time, there is a general tendency towards more and more security features in communication protocols. QUIC, e.g., encrypts all payload data and almost all header content already inside the transport layer. This makes current COIN concepts infeasible in settings where QUIC connections are used as the COIN elements do not have access to any packet content. Using COIN thus also depends on how well security mechanisms like encryption can be integrated into COIN frameworks.

8. Transport Features

Depending on application needs, different transport protocols provide different features. These features shape the behavior of the protocol and have to be taken into account when developing COIN functionality. In this section, we focus on the impact of reliability as well as flow and congestion control to create awareness for the multifaceted interaction between the transport protocols and COIN elements.

8.1. Reliability

Applications require a reliable transport whenever it is important that all data is transmitted successfully. TCP[RFC0793] provides such a reliable communication as it first sets up a dedicated connection and then ensures the successful reception of all data. In contrast, UDP[RFC0768] is a connectionless protocol without guarantees and COIN elements working on UDP transmissions must be robust to lost information. This is not the case for applications on top of TCP, but the retransmissions and the TCP state, which TCP uses to achieve the reliability, make packet processing for COIN more complex due to at least three reasons.

The concept of retransmissions bases on the end-to-end principle as retransmissions are performed by the sender if it has determined that the receiver did not receive the corresponding original message. Both participants can then act knowing that parts of the overall data are still missing. For simple COIN elements, which are not aware of the involved TCP states and which do not track sequence numbers, it is difficult to identify (a) that a packet in the sequence is missing and (b) that a packet is a retransmission. One question is whether COIN elements should incorporate an understanding for retransmissions on the basis of existing transport mechanisms or if a COIN-capable transport should include dedicated signals for the COIN elements.

Apart from challenges in identifying retransmissions, there is also the fact that they are sent out of order with the original packet sequence. Depending on the chosen flow granularity (see Section 4), COIN elements might have to hold contextual information for a prolonged time once they identify an impeding retransmission. Moreover, they might have to postpone or cancel computations if data is missing and instead schedule later computations. The main question arising from this is: to what extent should COIN elements be capable of incorporating retransmissions into their computation schemes and how much additional storage capabilities are required for this?

When incorporating COIN elements into the retransmission mechanisms, it is also an interesting question whether it should be possible to request or perform retransmissions from COIN elements. Considering a setting with COIN elements that are capable of detecting missing packets and retransmission requests, it might improve the overall performance if the COIN element directly requests or performs the retransmission instead of forwarding the packet/request through the complete sequence of elements. This is especially interesting in the context of collective communication where reliability mechanisms could make use of the multi-source nature of the communication and leverage the presence of many COIN elements in the network, for

instance by using network coding techniques, which in turn may benefit from COIN elements participating in the reliability mechanism. In all cases, the aforementioned storage capabilities are relevant so that the COIN elements can store enough information. The general question, i.e., which nodes in the sequence should do the retransmission, has already been worked on in the context of multicast transport protocols.

Depending on the extent of realization of the presented retransmission features, COIN elements might almost have to implement some of TCP's state to fulfil their tasks. Considering that different COIN elements have different computational and storage capacities, it is very likely that not every form of transport integration into COIN can be supported by every available COIN platform. The choice of devices included into the communication will hence certainly affect the types of transport protocols that can be operated on the COIN networks.

Another aspect to consider is the 'unit' that needs to be reliably transferred. In stream-based transport protocols, such as TCP, packets represent the smallest unit of transfer. However, different choices in the flow granularity and a possible move to larger-than-a-packet messages or transactions, as suggested in Section 4, might make other approaches to reliability viable that operate on the basis of such messages.

Challenges/questions regarding reliability are:

1. What is the unit of reliable transfer?

2. How to utilize more than one computational endpoint in the reliability mechanism?

3. Should COIN elements be aware of retransmissions?

4. How can COIN elements identify missing packets or retransmissions?

5. Should COIN elements be explicitly notified about retransmissions?

6. To what extent should COIN elements be capable of incorporating retransmissions into their computation schemes?

7. How much storage capabilities are required for incorporating retransmissions?

8. How can COIN elements incorporate missing packets into their computations?

9. How to deal with state changes in COIN elements caused by data lost later in the communication chain and then retransmitted?

10. Should COIN elements be capable of requesting retransmissions/ answering retransmission requests?

11. Which devices should perform retransmissions?

12. Do COIN elements have to keep transport state?

13. How much transport state do COIN elements have to keep?

8.2. Flow/Congestion Control

TCP incorporates mechanisms to avoid overloading the receiving host (flow control) and the network (congestion control) and determines its sending rate as the minimum value of what both mechanisms determine as feasible for the system. This approach is based on the notion that computing and forwarding hosts are separated and is challenged by the inclusion of COIN elements, i.e., computing nodes in the network.

Flow control bases on explicit end-host information as the participating end-hosts notify each other about how much data they are capable of processing and consequently do not transmit more data as the other host can handle. This only changes if one of the end-hosts updates its flow control information.

Congestion control, on the other hand, interprets volatile feedback from the network to guess a sending rate that is possible given the current network conditions. Most congestion control algorithms hereby follow a cyclical procedure where the sending end-hosts constantly increase their sending rate until they detect network congestion. They then decrease their sending rate once and start to increase it again.

In this traditional two-fold approach, loss, delay, or any other congestion signal (depending on the congestion control algorithm) induced by COIN elements (only in case that they are the bottleneck) is interpreted as network congestion and thus accounted for in the congestion control mechanism. This means that the sending end-host may repeatedly overload the computational capabilities of the COIN elements when probing for the current network conditions instead of respecting general device capabilities as is done by flow control.

In the context of COIN, the granularity of flows may see a division into sub-flows or messages to better represent the used computational semantic as discussed in Section 4. This raises the question whether flow and congestion control should be applied to longer term flows (of many sub-flows or messages) or directly to sub-flows. Eventually, this could possibly lead to a separation of error control (for sub-flows) and flow control (for longer-term flows). A subsequent challenge is then how to reconcile the possible volatile nature of sub-flow relations (between computational endpoints) with the longer-term relationship between network endpoints that will see a flow of messages between them. This is particularly pertinent in collective communication scenarios, where many forward unicast sub-flows may lead to a single multicast sub-flow response albeit only for that one response message. Reconciling the various unicast resource regimes into a single (ephemeral) multicast one poses a significant challenge.

Consequently, the question arises whether COIN elements should be able to participate in end-to-end flow control.

Main challenges/questions are:

1. Should COIN elements be covered by congestion control?

2. Should COIN elements be able to participate in end-to-end flow control?

3. How could a resource constraint scheme similar to flow control be realized for COIN elements?

4. How to reconcile message-level flexibility in transport relations between computational endpoints with longer-term resource stability between network elements participating in the computational scenario?

9. Security Considerations

COIN changes the traditional paradigm of a simple network and the corresponding end-to-end principle as it encourages computations in/by the network. Approaches designed to protect transmitted data, such as Transport Layer Security (TLS) which is even embedded into newer transport protocols like QUIC, rely on the end-to-end principle and are thus conceptually not compatible with COIN. Additionally, COIN elements often do not support required cryptographic functionality. Thus, there exist no out-of-the-box security solutions for COIN which means new security concepts have to be developed to allow for a secure use of COIN.

10. IANA Considerations

N/A

11. Conclusion

The advent of COIN comes with many new use cases and promises improved solutions for various problems. It is, however, not directly compatible with the end-to-end nature of transport protocols. To enable a transport-based communication, it is thus important to answer key questions regarding COIN and transport protocols.

All in all, there is a wide range of transport features which offer improved performance in certain settings and for certain application combinations. However, as presented, it is unlikely that all of the features/types of transport protocols can be supported on every COIN element. It might make sense to define different classes of COIN-ready transport protocols which can then be deployed depending on the concretely available networking/hardware elements. Alternatively, each of the features could be treated separately and they could then be composed based on the demands of an application at-hand.

The general question summarizing this document is:

Which transport features should be supported by COIN, how can they be identified, and how can they be provided to application designers?

12. Informative References

[I-D.draft-ietf-sfc-proof-of-transit-08]
Brockners, F., Bhandari, S., Mizrahi, T., Dara, S., and S. Youell, "Proof of Transit", Work in Progress, Internet-Draft, draft-ietf-sfc-proof-of-transit-08, 1 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-sfc-proof-of-transit-08.txt>.

[I-D.draft-kutscher-coinrg-dir-02]
Kutscher, D., Karkkainen, T., and J. Ott, "Directions for Computing in the Network", Work in Progress, Internet-Draft, draft-kutscher-coinrg-dir-02, 31 July 2020, <http://www.ietf.org/internet-drafts/draft-kutscher-coinrg-dir-02.txt>.

[I-D.draft-sarathchandra-coin-appcentres-03]
Trossen, D., Sarathchandra, C., and M. Boniface, "In-Network Computing for App-Centric Micro-Services", Work in Progress, Internet-Draft, draft-sarathchandra-coin-

appcentres-03, 23 October 2020, <http://www.ietf.org/internet-drafts/draft-sarathchandra-coin-appcentres-03.txt>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

Authors' Addresses

Ike Kunze
RWTH Aachen University
Ahornstr. 55
D-52074 Aachen
Germany

Email: kunze@comsys.rwth-aachen.de

Klaus Wehrle
RWTH Aachen University
Ahornstr. 55
D-52074 Aachen
Germany

Email: wehrle@comsys.rwth-aachen.de

Dirk Trossen
Huawei Technologies Duesseldorf GmbH
Riesstr. 25C
D-80992 Munich
Germany

Email: Dirk.Trossen@Huawei.com

COINRG D. Kutscher
Internet-Draft University of Applied Sciences Emden/Leer
Intended status: Experimental T. Kaerkkaeinen
Expires: February 1, 2021 J. Ott
Technical University Muenchen
July 31, 2020

Directions for Computing in the Network
draft-kutscher-coinrg-dir-02

Abstract

In-network computing can be conceived in many different ways - from active networking, data plane programmability, running virtualized functions, service chaining, to distributed computing.

This memo proposes a particular direction for Computing in the Networking (COIN) research and lists suggested research challenges.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 1, 2021.

Copyright Notice

Table of Contents

1. Introduction

Recent advances in platform virtualization, link layer technologies and data plane programmability have led to a growing set of use cases where computation near users or data consuming applications is needed - for example, for addressing minimal latency requirements for compute-intensive interactive applications (networked Augmented Reality, AR), for addressing privacy sensitivity (avoiding raw data copies outside a perimeter by processing data locally), and for speeding up distributed computation by putting computation at convenient places in a network topology.

In-network computing has mainly been perceived in five variants so far: 1) Active Networking [ACTIVE], adapting the per-hop-behavior of network elements with respect to packets in flows, 2) Edge Computing as an extension of virtual-machine (VM) based platform-as-a-service, 3) programming the data plane of SDN switches (through powerful programmable CPUs and programming abstractions, such as P4 [SAPIO]), 4) application-layer data processing frameworks, and 5) Service Function Chaining (SFC).

Active Networking has not found much deployment in the past due to its problematic security properties and complexity.

Programmable data planes can be used in data centers with uniform infrastructure, good control over the infrastructure, and the feasibility of centralized control over function placement and scheduling. Due to the still limited, packet-based programmability model, most applications today are point solutions that can demonstrate benefits for particular optimizations, however, often without addressing transport protocol services or data security that would be required for most applications running in shared infrastructure today.

Edge Computing (in the ETSI Multi-access Edge Computing [MEC] variant, as traditional cloud computing) has a fairly coarse-grained (VM-based) computation-model and is hence typically deploying centralized positioning/scheduling though virtual infrastructure management (VIM) systems. Besides such industry-driven activities, manifold research approaches to edge computing with varying granularity and orchestration approaches, among other differiating elements, have been pursued [EDGESURVEY] [FOGEDGE].

Microservices can be seen as a (lightweight) extension of the cloud computing model (application logic in containers and orchestrators for resource allocation and other management functions), leveraging more lightweight platforms and fine-grained functions. Compared to traditional VM-based systems, microservice platforms typically employ a "stateless" approach, where the service/application state is not tied to the compute platform, thus achieving fault tolerance with respect to compute platform/process failures.

Application-layer data processing such as Apache Flink [FLINK] provide attractive dataflow programming models for event-based stream processing and light-weight fault-tolerance mechanisms - however systems such as Flink are not designed for dynamic scheduling of compute functions.

Modern distributed applications frameworks such as Ray [RAY], Sparrow [SPARROW] or Canary [CANARY] are more flexible in this regard - but

since they are conceived as application-layer frameworks, their scheduling logic can only operate with coarse-granular cost information. For example, application-layer frameworks in general, can only infer network performance, anomalies, optimization potential indirectly (through observed performance or failure), so most scheduling decisions are based on metrics such as platform load.

Service Function Chaining (SFC, [RFC7665]) is about establishing IP tunnels between processing functions that are expected to work on packets or flows - for applications such as inspection and classification, so that some of these functions could be seen as elements in a COIN context as well.

2. Terminology

We are using the following terms in this memo:

Program: a set of computations requested by a user

Program Instance: one currently executing instance of a program

Function: a specific computation that can be invoked as part of a program

Execution Platform: a specific host platform that can run function code

Execution Environment: a class of target environments (execution platforms) for function execution, for example, a JVM-based execution environment that can run functions represented in JVM byte code

3. Computing in the Network vs Networked Computing vs Packet Processing

Many applications that might intuitively be characterized as "computing in the network" are actually either about connecting compute nodes/processes or about IP packet processing in fairly traditional ways.

Here, we try to contrast these existing and widely successful systems (that probably do not require new research) with a more novel "computing in the network (COIN)" approach that revisits the function split between computing and networking.

3.1. Networked Computing

Networked Computing exists in various facets today (as described in the Introduction). Fundamentally, these systems make use of networking to connect compute instances - be it VMs, containers, processes or other forms of distributed computing instances.

There are established frameworks for connecting these instances, from general purpose Remote Method Invocation/Remote Procedure Calls to system-specific application-layer protocols. With that, these systems are not actually realizing "computing in the network" - they are just using the network (and taking connectivity as granted).

Most of the challenges here are related to compute resource allocation, i.e., orchestration methods for instantiating the right compute instance on a corresponding platform - for achieving fault tolerance, performance optimization and cost reduction.

Examples of successful applications of networked computing are typical overlay systems such as CDNs. As overlays they do not need to be "in the network" - they are effectively applications. (Note: we sometimes refer to CDN as an "in-network" service because of the mental model of HTTP requests that are being directed and potentially forwarded by CDN systems. However, none of this happens "in the network" - it is just a successful application of HTTP and underlying transport protocols.)

3.2. Packet Processing

Packet processing is a function "in the network" - in a sense that middleboxes reside in the network as transparent functions that apply processing functions (inspection, classification, filtering, load management etc.) - mostly _transparent_ to endpoints. Some middlebox functions (TCP split proxies, video optimizers) are more invasive in a sense that they do not only operate on IP flows but also try to impersonate transport endpoints (or interfere with their behavior).

Since these systems can have severe impacts on service availability, security/privacy, and performance they are typically not very _programmable_ - they just execute (usually) static code for predefined functions.

Active Networking can be characterized as an attempt to offer abstractions for programmable packet processing from an "endpoint perspective", i.e., by using data packets to specify intended behavior in the network with the aforementioned security problems.

Programmable Data Plane approaches such as P4 are providing abstractions of different types of network switch hardware (NPUs, CPUs, FPGA, PISA) from a switch/network programming perspective. The corresponding programs are constrained by the capabilities (instruction set, memory) of the target platform and typically operate on packets/flow abstractions (for example match-action-style processing).

Network Functions Virtualization (NFV) is essentially a "Networked Computing" approach (after all, Network Functions are just virtualized compute functions that get instantiated on compute platforms by an orchestrator). However, some Virtual Network Functions (VNFs) happen to process/forward packets (e.g., gateways in provider networks, NATs or firewalls). Still, that does not affect their fundamental properties as virtualized computing functions.

When connecting VNFs, there is the question of how to steer packet flows so that packets reach the right functions (and pass through them in the right order). One way is through configuration and network control/management (SDN), i.e., the VNFs are places in a virtual network, and there are configurations for meaningful next-hop IP addresses etc.

A more dynamic way is through Service Function Chaining (SFC, [RFC7665]), where a dynamic chain of IP-addressable packet processors can be specified (in an encapsulation packet header structure) and where forwarding nodes are equipped to interpret these headers and forward the packets to the appropriate next hops.

The SFC [RFC7665]) framework works with IP addresses for function (host) identifiers. Name-Based Service Function Forwarding [I-D.trossen-sfc-name-based-sff] takes this one step further by adding another layer of indirection and by identifying the Service Functions using a name rather than a routable IP endpoint (or Layer 2 address). In addition to the naming concept, [I-D.trossen-sfc-name-based-sff] also described the possibility of using different transport and application layer protocols for the communication between functions - which could in principle extend the applicability from mere packet processing to some form of distributed computing.

3.3. Computing in the Network

In some deployments, networked computing and packet processing go well together, for example, when network virtualization (multiplexing physical infrastructure for multiple isolated subnetworks) is achieved through data-plane programming (SDN-style) to provide connectivity for VMs of a tenant system.

While such deployments are including both computing and networking, they are not really doing computing _in the network_. VM/containers are virtualized hosts/processes using the existing network, and packet processing/programmable networks is about packet-level manipulation. While it is possible to implement certain optimizations (for example, processing logic for data aggregation) - the applicability is rather limited especially for applications where application-data units do not map to packets and where additional transport protocols and security requirements have to be considered.

Multi-access Edge Computing [MEC] is a particular architecture that leverages the virtual host platform concept, and that is focused on management and orchestration for such platforms. MEC can be combined with virtual networking concepts such as "Network Slicing" in 5G [MEC5G] to assure a certain QoS for connectivity to MEC platform instances. It should be noted that there may be other forms of edge computing that are not VM-based.

Distributed Computing (stream processing, edge computing) on the other side is an area where many application-layer frameworks exist that actually _could_ benefit from a better integration of computing and networking, i.e., from a new "computing in the network" approach.

For example, when running a distributed application that requires dynamic function/process instantiation, traditional frameworks typically deploy an orchestrator that keeps track of available host platforms and assigned functions/processes. The orchestrator typically has good visibility of the availability of and current load on host platforms, so it can pick suitable candidates for instantiating a new function.

However, it is typically agnostic of the network itself - as application layer overlays the function instances and orchestrators take the network as a given, assuming full connectivity between all hosts and functions. While some optimizations may still be feasible (for example co-locating interacting functions/processes on a single host platform), these systems cannot easily reason about

- shortest paths between function instances; function off-loading
- opportunities on topologically convenient next hops; and
- availability of new, not yet utilized resources in the network.

While it is possible to perform optimizations like these in application layers overlays, it involves significant monitoring effort and would often duplicate information (topology, latency) that is readily available inside the network. In addition to the

associated overhead, such systems also operate at different time scales so that direct reaction in fine-grained computing environments is difficult to achieve.

When asking the question of how the network can support distributed computing better, it may be helpful to characterize this problem as a resource allocation optimization problem: Can we integrate computing and networking in a way that enables a joint optimization of computing and networking resource usage? Can we apply this approach to achieve certain optimization goals such as:

- low latency for certain function calls or compute threads;
- high throughput for a pipeline of data processing functions;
- high availability for an overall application/service;
- load management (balancing, concentration) according to performance/cost constraints; and
- consideration of security/privacy constraints with respect to platform selection and function execution?
- Also: can we do this at the speed of network dynamics, which may be substantially higher than the rate at which distributed computing applications change?

Considering computing and networking resource holistically could be the key for achieving these optimization goals (without considerable overhead through telemetry, management and orchestration systems). If we are able to dissolve the layer boundaries between the networking domain (that is typically concerned with routing, forwarding, packet/flow-level load balancing) and the distributed computing domain (that is typically concerned with 'processor' allocation, scaling, reaction to failure for functions/processes), we might get a handle to achieve a joint resource optimization and enable the distributed computing layer to leverage network-provided mechanisms directly.

For example, if distributing information about available/suitable compute platform could be a routing function, we might be able to obtain and utilize this information in a distributed fashion. If instantiating a new function (or offloading some piece of computation) could consider live performance data obtained from a in-network forwarding/offloading service (similar to IP packet forwarding in traditional IP networks), the "next-hop" decision could be based both on network performance and node load/availability).

Integrating computing and networking in this manner would not rule out highly optimized systems leveraging sophisticated orchestrators. Instead, it would provide a (possibly somewhat uniform) framework that could allow several operating and optimization modes, including totally distributed modes, centralized orchestration, or hybrid forms, where policies or intents are injected into the distributed decision-making layer, i.e., as parameters for resource allocation and forwarding decisions.

3.4. Elements for Computing in the Network

In-network computing requires computing resources (CPU, possibly GPUs, memory, ...), physical or virtualized to some extent by a suitable platform. These computing resources may be available in a number of places, as partly already discussed above, including the following:

- They may be found on dedicated machines co-locating with the routing infrastructure, e.g., having a set of servers next to each router as one may find in access network concentrators. This would come closest to today's principles of edge computing.

- They may be integrated with routers or other network operations infrastructure and thus be tightly integrated within the same physical device.

- They may be integrated within switches, similar to the (limited) P4 compute capabilities offered today.

- They may be located on NICs (in hosts) or line cards (routers) and be able to proactively perform some application functions, in the sense of a generalized variant of "offloading" that protocol stacks perform to reduce main CPU load.

- They might add novel types of dedicated hardware to execute certain functions more efficiently, e.g., GPU nodes for (distributed) analytics.

- They might include low-end (embedded) devices such as microcontrollers that support decentralized computation at low cost and limited performance.

- They may also encompass additional resources at the edge of the network, such as sensor nodes. Associated sensors could be physical (as in IoT) or logical (as in MIB data about a network device).

- Even user devices along the lines of crowd computing [CROWD] or mist computing [MIST] may contribute compute resources and dynamically become part of the network.

Depending on the type of execution platform, as already alluded to above, a suitable execution framework must be put in place: from lambda functions to threads to processes or process VMs to unikernels to containers to full-blown VMs. This should support mutual isolation and, depending on the service in question, a set of security features (e.g., authentication, trustworthy execution, accountability). Further, it may be desirable to be able to compose the executable units, e.g., by chaining lambda functions or allowing unikernels to provide services to each other - both within a local execution platform and between remote platform instances across the network.

The code to be executed may be pre-installed (as firmware, as microcode, as operating system functions, as libraries, as *aaS offering, among others) or may be dynamically supplied. While the former is governed by the entity operating the execution device or supplying it (the vendor), the code to be executed may have different origins. Fundamentally, we can distinguish between two cases:

1. The code may be "centrally" provisioned, originating from an application or other service provider inside the network. This is analogous to CDNs, in which an application provider contracts a CDN provider to host content and service logic on its behalf. The deployment is usually long-term, even if instantiations of the code may vary. The code thus originates from rather few - known - sources. In this setting, applications only invoke this code and pass on their parameters, context, data, etc.

2. The code may be provided in a decentralized manner from a user device or other service that requires a certain function or service to be carried out. At the coarse granularity of entire application images, this has been explored as "code offloading"; recent approaches have moved towards finer granularities of offloading (sets of) functions, for which also some frameworks for smartphones were developed, leading to finer granularities down to individual functions. In this setting, application transfer mobile code - along with suitable parameters, etc. - into the network that is executed by suitable execution platforms. This code is naturally expected to be less trusted as it may come from an arbitrary source.

Obviously, 1. and 2. may be combined as mobile code may make use of other in-network functions and services, allowing for flexible application decomposition. Essentially, computing in the network may

support everything from full application offloading to decomposing an application into small snippets of code (e.g., at class, objects, or function granularity) that are fully distributed inside the network and executed in a distributed fashion according to the control flow of the application. This may lead to iterative or recursive calling from application code on the initiating host to mobile code to pre-provisioned code.

Another dimension beyond where the code comes from is how tightly the code and the data are coupled. At one extreme, approaches like Active Messages combine the data and the code that operates (only) on that data into transmission units, while at the other extreme approaches like Network Function Virtualization are only concerned with the instantiation of the code in the network. The underlying architectural question is whether the goal is to enable the network to perform computations on the data passing through it, or whether the goal is to enable distributed computational processes to be built in the network. And, of course, complete applications may leverage both approaches.

With these different existing and possibly emerging platforms and execution environments and different ways to provision functions in the network, it does not seem useful to assume any particular platform and any particular "mobile code" representation as _the_ "computing in the network" environment. Instead, it seems more promising to reason about properties that are relevant with respect to distributed program semantics and protocols/interfaces that would be used to integrate functions on heterogeneous platforms into one application context. We discuss these ideas and associated challenges in the following section.

4. Examples

4.1. Compute-First Networking with ICN

[CFN] is an example of a computing-in-the-network system that is based on computation graph representation for distributed programs. These programs are composed of stateful actors and stateful functions that are dynamically instantiated on available compute resources.

The first motivating use case was a real-time health monitoring system that analyzed audio samples from coughing noises which involves processing several audio feeds for noise addition and subtraction and for feature extraction.

The key concept of CFN is to provide a general-purpose distributed computing framework that can be programmed without knowledge about

the runtime environment but that can leverage the dynamic resource properties automatically, and with reasonable efficiency.

CFN can lay out compute graphs over the available computing platforms in a network to perform flexible load management and performance optimizations, taking into account function/actor location and data location, as well as platform load and network performance.

In CFN, compute nodes that can execute functions within a given program instance are called workers. The allocation of functions and actors to workers happens in a distributed fashion. A CFN system knows the current utilization of available resources and the least cost paths to copies of needed input data. It can dynamically decide which worker to use, performing optimizations such as instantiating functions close to big data inputs. The bindings that control which execution platforms host which program interfaces (or individual functions/actors) is maintained through a computation graph.

To realize this distributed scheduling, workers in each resource pool advertise their available resources. This information is shared among all workers in the pool. A worker execution environment can decide, without a centralized scheduler, which set of workers to prefer to invoke a function or to instantiate an actor. In order to direct function invocation requests to specific worker nodes, CFN utilizes the underlying ICN network's forwarding capabilities - the network performs late binding through name-based forwarding and workers can provide forwarding hints to steer the flow of work.

4.2. Akka Toolkit

The Akka toolkit [1] for building concurrent and distributed applications on the the JVM that is used by frameworks such as Apache Flink [2]. Akka is implements the Actor model, a way of realizing distributed computing as asynchronous message-based communication between concurrent processes that encapsulate application logic.

Communication between distributed actors is based on symmetric peer-to-peer model (actors can send each other messages) and is implemented by TCP-based protocols [3].

Akka actors are logically organized in a tree hierarchy [4], and there are two addressing concepts: 1) Actor References that unique identify an actor instance and 2) Actor Paths, hierarchically structured names that specify the logical position of an actor instance in system tree. Actor path can have an address component that specified location information (e.g., host and port number).

Akka has a routing concept [5] that can duplicate and distribute messages to a set of actors (for example for map-reduce like parallelism).

The Akka toolkit support cluster features [6], i.e., the management of a collection of JVMs that can be monitored for resource and failure management.

5. Research Challenges

Conceiving computing in the network as a joint resource optimization problem as described above incurs a set of interesting, novel research challenges that are particularly relevant from an Internet Research perspective.

5.1. Categorization of Different Use Cases for Computing in the Network

There are different applications but also different configuration classes of Computing in the Network systems. For example, a data processing pipeline might be different from a distributed application employing some stateful actor components. It is worthwhile analyzing different typical use cases and identify commonalities (for example, fundamental protocol elements etc.) and differences.

5.2. Networking and Remote-Method-Invocation Abstractions

In distributed systems, there are different classes of functions that can be distinguished, for example:

1. Strictly stateless functions that do not keep any context state beyond their activation time

2. Stateful functions/modules/programs that can be instantiated, invoked and eventually destroyed that do keep state over a series of function invocations

Modern frameworks such as Ray are offering a clear separation of stateless functions and stateful actors and offer corresponding abstractions in their programming environment. The aforementioned analysis of use cases should provide a diverse set of use cases for deriving a minimal yet sufficient set of function classes.

Beyond this fundamental categorization of functions/actors, there is the question of interfaces and protocols mechanisms - as building blocks to utilize functions in programs. For example, stateful functions are typically invoked through some Remote Method Invocation (RMI) protocol that identifies functions, allows for specifying/ transferring parameters and function results etc. Stateful actors

could provide class-like interfaces that offer a set of functions (some of which might manipulate actor state).

Another aspect is about identity (and naming) of functions and actors. For actors that are typically used to achieve real-world effects or to enable multiple invocations of functions manipulating actor state over time, it is obvious that there needs to be a concept of specific instances. Invoking an actor function would then require specifying some actor instance identifier.

Stateless functions may be different: an invoking instance may be oblivious with respect to the specific function instance and locus (on an execution platform) and might just want to leave it to the network to find the "best" instance or locus for a new instantiation. Some fine-granular functions might just be instantiated for one invocation. On the other hand, a function might be tied to a particular execution platform, for example an GPU-supported host system. The naming and identity framework must allow for specifying such a function (or at least equivalence classes) accordingly.

Stateful functions may share state within the same program context, i.e., across multiple invocations by the same application (as, e.g., holds for web services that preserve context - locally or on the client side). But stateful functions may also hold state across applications and possibly across different instantiations of a function on different compute nodes. Such will require data synchronization mechanisms and the implementation of suitable data structure to achieve a certain degree of consistency. The targeted degree of consistency may vary depending on the function and so may the mechanisms used to achieve the desired consistency.

Finally, execution platforms will require efficient resource management techniques to operate with different types of stateless and stateful functions and their associated resources, as well as for dynamically instantiated mobile code. Besides the aforementioned location of suitable compute platforms and scheduling (possibly queuing) functions and function invocations, this also includes resource recovery ("garbage collection").

5.3. Transport Abstractions

When implementing Computing in the Network and building blocks such as function invocation it seems that IP packet processing is not the right abstraction. First of all, carrying the context for some function invocation might require many IP packets - possibly something like Application Data Units (ADUs). But even if such ADUs could be fit into network layer packets, other problems still need to

be addressed, for example message formats, reliability mechanisms, flow and congestion control etc.

It could be argued that today's distributed computing overlays solve that by using TCP and corresponding application layer formats (such as HTTP) - however this begs the question whether a fine-granular distributed computing system, aiming to leverage the network for certain tasks, is best served by a TCP/IP-based approach that entails issues such as

- need for additional resolution/mapping system to find IP addresses for functions;
- possible overhead for establishing TCP connections for fine-granular function invocation;
- defining and managing security properties of such connections and coping with the associated setup/validation overhead; and
- mismatch between TCP end-to-end semantics and the intention to defer next-hop selection etc. to the network.

Moreover, some Computing in the Network applications such as Big Data processing (Hadoop-style etc.) can benefit significantly from data-oriented concepts such as

- in-network caching (of data objects that represent function parameters or results);
- reasoning about the tradeoffs between moving data to function vs. moving code to data assets; and
- sharing data (e.g., function results) between sets of consuming entities.

RMI systems such as RICE [RICE] enable Remote Method Invocation of ICN (data-oriented network/transport). Research questions include investigating how such approaches can be used to design general-purpose distributed computing systems. More specifically, this would involve questions such as:

- What is the role of network elements in forwarding RMI requests?
- What visibility into load, performance and other properties should endpoints and the network have to make forwarding/offloading decisions and how can such visibility be afforded?

- What is the notion of transport services in this concept and how intertwined is traditional transport with RMI invocation?

- What kind of feedback mechanisms would be desirable for supporting corresponding transport services?

5.4. Programming Abstractions

When creating SDKs and programming environments (as opposed to individual point solutions) questions arise such as:

- How to use concepts such as stateless functions, actor models and RMI in actual programs, i.e., what are minimal/ideal bindings or extensions to programming languages so that programmers can take advantage of Computing in the Network?

- Are there additional, potentially higher-layer, abstractions that are needed/useful, for example data set synchronization, data types for distributed computing such as CRDTs?

In addition to programming languages, bindings, and data types, there is the question of execution environments and mobile code representation. With the vast number of different platforms (CPUs, GPUs, FPGAs etc.) it does not seem useful to assume exactly one environment. Instead, interesting applications might actually benefit from running one particular function on a highly optimized platform but are agnostic with respect to platforms for other, less performance-critical functions. Being able to support a heterogenous, evolving set of execution environments brings about questions such as:

- How to discover available platforms (and understand their properties)?

- How to specify application needs and map them to available platforms?

- Can a certain function/application service be provided with different fidelity levels, e.g., can an application leverage a GPU platform if available and fall back to a reduced feature set in case such a platform is not available?

In this context, updates and versioning could entail another dimension of variability for Computing in the Network:

- How to manage coexistence of multiple versions of functions and services, also for service routing and request forwarding?

- Is there potential for fallback and version negotiation if needed (considering the risk of "bidding downs" attacks?)
- How to retire old versions?
- How to securely and reliably deal with function updates and corresponding maintenance tasks?

5.5. Security, Privacy, Trust Model

Computing in the Network has interesting security-related challenges, including:

- How can a caller trust that a remote function works as expected? This entails several questions such as
  - How to securely bind "function names" to actual function code?
  - How to trust the execution platform (in its entirety)?
  - How to trust the network that is forwards requests (and result messages) reliably and securely?
  - How to ascertain that a function does what it claims to do?
- What levels of authentication are needed for callers (assuming that not everybody can invoke any function)?
- How to authenticate and achieve confidentiality for requests, their parameters and result data (especially when considering sharing of results)?

Many of these questions are related to other design decisions such as

- What kind of session concept do we assume, i.e., is there a concept of distributed application session that represents a trust domain for its members?
- Where is trust anchored? Can the system enable decentralized operation?

All of these questions are not new, but conceiving networking and computing holistically seems to revisit distributed systems and network security - because some established concepts and technologies may not be directly applicable (such as transport layer security and corresponding web PKI).

5.6. Coordination

For distributed systems, coordination is a key function and involves several functions such as configuration management, service discovery, application state management, and consensus schemes.

How these functions are implemented depends a lot on the nature of specific systems. For example, Apache ZooKeeper [7] is a logically centralized coordination service that provides coordination primitives to client application modules. The ZooKeeper itself is implemented as a distributed system consisting of a set of tightly coupled server instances that replicate the ZooKeeper state.

Other systems, such as the ICN-based CFN (Section 4.1) implement these services in a distributed way, employing different mechanisms for synchronization and consensus building.

While the fundamental concepts and mechanisms for coordination services are well understood, applying these concepts and mechanisms to a specific system design requires careful consideration.

5.7. Fault Tolerance, Failure Handling, Debugging, Management

Distributed computing naturally provides different types of failures and exceptions. In fine-granular distributed computing, some failures may by more tolerable (think microservices), i.e., platform crash or function abort due to isolated problems could be handled by just re-starting/re-running a particular function. Similarly, "message loss" or incorrect routing information may be repairable by the system itself (after time).

When failure cannot be repaired (or just tolerated) by the distributed computing framework, this raises questions such as:

- What are strategies for retrying vs aborting function invocation?
- How to signal exceptions and enable robust response to failures?

Failure handling and debugging also has a management aspect that leads to questions such as:

- What monitoring and instrumentation interfaces are needed?
- How can we represent, visualize, and understand the (dynamically changing) properties of Computing in the Network infrastructure as well as of the currently running/instantiated entities?

6. Acknowledgements

The authors would like to thank Dave Oran, Michal Krol, Spyridon Mastorakis, Yiannis Psaras, Eve Schooler, Dirk Trossen, and Phil Eardley for previous fruitful discussions on Computing in the Network topics and for feedback on this draft.

7. ChangeLog

7.1. 02

- fixed errors and updates references
- new Section 5.6 on Coordination
- renamed Section 5.7 to Fault Tolerance, Failure Handling, Debugging, Management
- new Section 4.2 on Akka in Section 4

7.2. 01

- added explanation of MEC and network slicing in Section 3.
- added clarification that edge computing is not limited to MEC
- added description of named service function chaining
- new Section 4 with a description of CFN-ICN

8. References

8.1. Informative References

[ACTIVE] Tennenhouse, D. and D. Wetherall, "Towards an active network architecture", ACM SIGCOMM Computer Communication Review Vol. 26, pp. 5-17, DOI 10.1145/231699.231701, April 1996.

[CANARY] Qu et al, H., "Canary -- A scheduling architecture for high performance cloud computing", 2016, <https://arxiv.org/abs/1602.01412>.

[CFN] KrA^3l, M., Mastorakis, S., Oran, D., and D. Kutscher, "Compute First Networking", Proceedings of the 6th ACM Conference on Information-Centric Networking, DOI 10.1145/3357150.3357395, September 2019.

[CROWD] Murray, D., Yoneki, E., Crowcroft, J., and S. Hand, "The case for crowd computing", Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds - MobiHeld '10, DOI 10.1145/1851322.1851334, 2010.

[EDGESURVEY] Mach et al, P., "Mobile Edge Computing -- A Survey on Architecture and Computation Offloading", 2017, <https://ieeexplore.ieee.org/document/7879258>.

[FLINK] Katsifodimos, A. and S. Schelter, "Apache Flink: Stream Analytics at Scale", 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), DOI 10.1109/ic2ew.2016.56, April 2016.

[FOGEDGE] Salaht, F., Desprez, F., and A. Lebre, "An Overview of Service Placement Problem in Fog and Edge Computing", ACM Computing Surveys Vol. 53, pp. 1-35, DOI 10.1145/3391196, July 2020.

[I-D.trossen-sfc-name-based-sff] Trossen, D., Purkayastha, D., and A. Rahman, "Name-Based Service Function Forwarder (nSFF) component within SFC framework", draft-trossen-sfc-name-based-sff-07 (work in progress), May 2019.

[MEC] ETSI, ., "Multi-access Edge Computing (MEC)", 2020, <https://www.etsi.org/technologies/multi-access-edge-computing>.

[MEC5G] Sami Kekki et al, ., "MEC in 5G Networks", 2018, <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf>.

[MIST] Barik, R., Dubey, A., Tripathi, A., Pratik, T., Sasane, S., Lenka, R., Dubey, H., Mankodiya, K., and V. Kumar, "Mist Data: Leveraging Mist Computing for Secure and Scalable Architecture for Smart and Connected Health", Procedia Computer Science Vol. 125, pp. 647-653, DOI 10.1016/j.procs.2017.12.083, 2018.

[RAY] Moritz et al, P., "Ray -- A Distributed Framework for Emerging AI Applications", 2018, <http://dl.acm.org/citation.cfm?id=3291168.3291210>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[RICE] KrA^3l, M., Habak, K., Oran, D., Kutscher, D., and I. Psaras, "RICE", Proceedings of the 5th ACM Conference on Information-Centric Networking, DOI 10.1145/3267955.3267956, September 2018.

[SAPIO] Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come", Proceedings of the 16th ACM Workshop on Hot Topics in Networks, DOI 10.1145/3152434.3152461, November 2017.

[SPARROW] Ousterhout, K., Wendell, P., Zaharia, M., and I. Stoica, "Sparrow", Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13, DOI 10.1145/2517349.2522716, 2013.

8.2. URIs

[1] https://akka.io/

[2] https://flink.apache.org/

[3] https://doc.akka.io/docs/akka/2.3/scala/remoting.html

[4] https://doc.akka.io/docs/akka/current/general/addressing.html

[5] https://doc.akka.io/docs/akka/current/typed/routers.html

[6] https://doc.akka.io/docs/akka/current/typed/cluster.html

[7] https://zookeeper.apache.org/

Authors' Addresses

Dirk Kutscher
University of Applied Sciences Emden/Leer
Constantiaplatz 4
Emden D-26723
Germany

Email: ietf@dkutscher.net

Teemu Kaerkkaeinen
Technical University Muenchen
Boltzmannstrasse 3
Munich
Germany

Email: kaerkkae@in.tum.de

Joerg Ott
Technical University Muenchen
Boltzmannstrasse 3
Munich
Germany

Email: jo@in.tum.de

COINRG
Internet-Draft
Intended status: Standards Track
Expires: May 5, 2021

M. McBride
Futurewei
D. Kutscher
Emden University
E. Schooler
Intel
CJ. Bernardos
UC3M
D. Lopez
Telefonica
X. de Foy
InterDigital Communications
Nov 1, 2020

Edge Data Discovery for COIN
draft-mcbride-edge-data-discovery-overview-05

Abstract

This document describes the problem of distributed data discovery in edge computing, and in particular for computing-in-the-network (COIN), which may require both the marshalling of data at the outset of a computation and the persistence of the resultant data after the computation. Although the data might originate at the network edge, as more and more distributed data is created, processed, and stored, it becomes increasingly dispersed throughout the network. There needs to be a standard way to find it. New and existing protocols will need to be developed to support distributed data discovery at the network edge and beyond.

Table of Contents

1. Introduction

Edge computing is an architectural shift that migrates Cloud functionality (compute, storage, networking, control, data management, etc.) out of the back-end data center to be more proximate to the IoT data being generated and analyzed at the edges

of the network. Edge computing provides local compute, storage and connectivity services, often required for latency- and bandwidth-sensitive applications. Thus, Edge Computing plays a key role in verticals such as Energy, Manufacturing, Automotive, Video Surveillance, Retail, Gaming, Healthcare, Mining, Buildings and Smart Cities.

1.1. Edge Data

Edge computing is motivated at least in part by the sheer volume of data that is being created by endpoint devices (sensors, cameras, lights, vehicles, drones, wearables, etc.) at the very network edge and that flows upstream, in a direction for which the network was not originally designed. In fact, in dense IoT deployments (e.g., many video cameras are streaming high definition video), where multiple data flows collect or converge at edge nodes, data is likely to need transformation (to be transcoded, subsampled, compressed, analyzed, annotated, combined, aggregated, etc.) to fit over the next hop link, or even to fit in memory or storage. Note also that the act of performing computation on the data creates yet another new data stream! Preservation of the original data streams is needed sometimes but not always.

In addition, data may be cached, copied and/or stored at multiple locations in the network on route to its final destination. With an increasing percentage of devices connecting to the Internet being mobile, support for in-the-network caching and replication is critical for continuous data availability, not to mention efficient network and battery usage for endpoint devices.

Additionally, as mobile devices' memory/storage fill up, in an edge context they may have the ability to offload their data to other proximate devices or resources, leaving a bread crumb trail of data in their wakes. Therefore, although data might originate at edge devices, as more and more data is continuously created, processed and stored, it becomes increasingly dispersed throughout the physical world (outside of or scattered across managed local data centers), increasingly isolated in separate local edge clouds or data silos. Thus, there needs to be a standard way to find it. New and existing protocols will need to be identified/developed/enhanced for these purposes. Being able to discover distributed data at the edge or in the middle of the network will be an important component of Edge computing.

1.2. Background

Several IETF T2T RG Edge Computing discussions have been held over the last couple years. A comparative study on the definition of Edge computing was presented in multiple sessions in T2T RG in 2018 and an Edge Computing I-D was submitted early 2019. An IETF BEC (beyond edge computing) effort has been evaluating potential gaps in existing edge computing architectures. Edge Data Discovery is one potential gap that was identified and that needs evaluation and a solution. The newly proposed COIN RG highlights the need for computations in the network to be able to marshal potentially distributed input data and to handle resultant output data, i.e., its placement, storage and/or possible migration strategy.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.4. Terminology

- Edge: The edge encompasses all entities not in the back-end cloud. The device edge represents the very leaves of the network and encompasses the entities found in the last mile network. Sensors, gateways, compute nodes are included. Because the things that populate the IoT can be both physical and/or cyber, in some solutions, particularly in software-defined or digital-twin contexts, the device edge can include logical (vs physical) entities. The infrastructure edge includes equipment on the network operator side of the last mile network including cell towers, edge data centers, cable headends, POPs, etc. See Figure 1 for other possible tiers of edge clouds between the device edge and the back-end cloud data center.

- Edge Computing: Distributed computation that is performed near the network edge, where nearness is determined by the system requirements. This includes high performance compute, storage and network equipment on either the device or infrastructure edge.

- Edge Data Discovery: The process of finding required data from edge entities, i.e., from databases, file systems, and device memory that might be physically distributed in the network, and providing access to it logically as if it were a single unified source, perhaps through its namespace, that can be evaluated or searched.

- ICN: Information Centric Networking. An ICN-enabled network routes data by name (vs address), caches content natively in the network, and employs data-centric security. Data discovery may require that data be associated with a name or names, a series of descriptive attributes, and/or a unique identifier.

2. Edge Data Discovery Problem Scope

Our focus is on how to define and scope the edge data discovery problem. This requires some discussion of the evolving definition of the edge as part of a cloud-to-edge continuum and in turn what is meant by edge data, as well as the meta-data about the edge data.

2.1. A Cloud-Edge Continuum

Although Edge Computing data typically originates at edge devices, there is nothing that precludes edge data from being created anywhere in the cloud-to-edge computing continuum (Figure 1). New edge data may result as a byproduct of computation being performed on the data stream anywhere along its path in the network. For example, infrastructure edges may create new edge data when multiple data streams converge upon this aggregation point and require transformation (e.g., to fit within the available resources, to smooth raw measurements to eliminate high-frequency noise, or to obfuscate data for privacy).

Initially our focus is on discovery of edge data that resides at the Device Edge and the Infrastructure Edge.

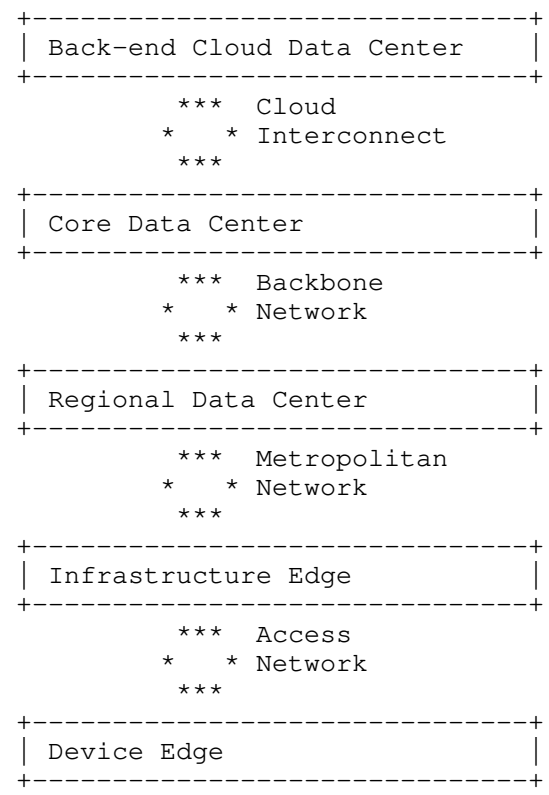


Figure 1: Cloud-to-edge computing continuum

2.2. Types of Edge Data

Besides classically constrained IoT device sensor and measurement data accumulating throughout the edge computing infrastructure, edge data may also take the form of higher frequency and higher volume streaming data (from a continuous sensor or from a camera), meta data (about the data), control data (regarding an event that was triggered), and/or an executable that embodies a function, service, or any other piece of code or algorithm. Edge data also could be created after multiple streams converge at an edge node and are processed, transformed, or aggregated together in some manner.

Regardless of edge data type, a key problem in the Cloud-Edge continuum is that data is often kept in silos. Meaning, data is often sequestered within the Edge where it was created. A goal of this discussion is to consider the prospect that different types of edge data will be made accessible across disparate edges, for example to enable richer multi-modal analytics. But this will happen only if

data can be described, searched and discovered across heterogeneous edges in a standard way. Having a mechanism to enable granular edge data discovery is the problem that needs solving either with existing or new protocols. The mechanisms shouldn't care to which flavor cloud or edge the request for data discovery is made.

3. Edge Scenarios Requiring Data Discovery

1. A set of data resources appears (e.g., a mobile node hosting data joins a network) and they want to be discoverable by an existing but possibly virtualized and/or ephemeral data directory infrastructure.

2. A device wants to discover data resources available at or near its current location. As some of these resources may be mobile, the available set of edge data may vary over time.

3. A device wants to discover to where best in the edge infrastructure to opportunistically upload its data, for example if a mobile device wants to offload its data to the infrastructure (for greater data availability, battery savings, etc.).

4. Edge Data Discovery

How can we discover data on the edge and make use of it? There are proprietary implementations that collect data from various databases and consolidate it for evaluation. We need a standard protocol set for doing this data discovery, on the device or infrastructure edge, in order to meet the requirements of many use cases. We will have terabytes of data on the edge and need a way to identify its existence and find the desired data. A user requires the need to search for specific data in a data set and evaluate it using their own tools. The tools are outside the scope of this document, but the discovery of that data is in scope.

4.1. Types of Discovery

There are many aspects of discovery and many different protocols that address each aspect.

Discovery of new devices added to an environment. Discovery of their capabilities/services in client/server environments. Discovery of these new devices automatically. Discovering a device and then synchronizing the device inventory and configuration for edge services. There are many existing protocols to help in this discovery: UPnP, mDNS, DNS-SD, SSDP, NFC, XMPP, W3C network service discovery, etc.

Edge devices discover each other in a standard way. We can use DHCP, SNMP, SMS, COAP, LLDP, and routing protocols such as OSPF for devices to discover one another.

Discovery of link state and traffic engineering data/services by external devices. BGP-LS is one such solution.

The question is if one or more of these protocols might be a suitable contender to extend to support edge data discovery?

4.2. Early Stage of Discovery

The different types of discovery may involve mobile devices, which can be the source, or target, of discovery operations. Mobile devices may have an influence on discovery in COIN, and early stage discovery may be necessary in some scenarios.

In many cases (e.g. crowds, drones or vehicular scenarios), multiple networks, or attachment points, are available to a mobile device. This type of device needs to efficiently select among multiple interfaces, or multiple attachment points, which one(s) to use for discovery. An early discovery stage should provide enough information to perform such a selection and therefore reduce power consumption, service latency, and impact on network usage.

To select among (already attached) multiple interfaces, we can leverage provisioning domains, router advertisements, DHCP, etc. to convey information about service or data. To select among multiple attachment points, pre-attachment discovery (e.g. 802.11aq, or obtaining provisioning domains through a control plane) or a discovery protocol over a control plane (e.g. as described in 3GPP edge computing) can be used.

What are suitable protocols to extend to support this early stage of discovery? There is also a tradeoff between the amount of exposed information and the limited resources available at this early stage. Trust and privacy are also important early stage discovery factors.

4.3. Naming the Data

Information-Centric Networking (ICN) RFC 7927 [RFC7927] is a class of architectures and protocols that provide "access to named data" as a first-order network service. Instead of host-to-host communication as in IP networks, ICNs often use location-independent names to identify data objects, and the network provides the services of processing (answering) requests for named data with the objective to finally deliver the requested data objects to a requesting consumer.

Such an approach has profound effects on various aspects of a networking system, including security (by enabling object-based security on a message/packet level), forwarding behavior (name-based forwarding, caching), but also on more operational aspects such as bootstrapping, discovery etc.

The CCNx and NDN (https://named-data.net) variants of ICN are based on a request/response abstraction where consumers (hosts, application requesting named data) send INTEREST messages into the network that are forwarded by network elements to a destination that can provide the requested named data object. Corresponding responses are sent as so-called DATA messages that follow the reverse INTEREST path.

Each unique data object is named unambiguously in a hierarchical naming scheme and is authenticated through Public-Key cryptography (data objects can also optionally be encrypted in different ways). The naming concept and the object-based security approach lay the foundation for location-independent operation. The network can generally operate without any notion of location, and nodes (consumers, forwarders) can forward requests for named data objects directly, i.e., without any additional address resolution. Location independence also enables additional features, for example the possibility to replicate and cache named data objects. On-path caching is a standard feature in many ICN systems -- typically for enhancing reliability and performance.

In CCNx and NDN, forwarders are stateful, i.e., they keep track of forwarded INTEREST to later match the received DATA messages. Stateful forwarding (in conjunction with the general named-based and location-independent operation) also empowers forwarders to execute individual forwarding strategies and perform optimizations such as in-network retransmissions, multicasting requests (in cases there are several opportunities for accessing a particular named data object) etc.

Naming data and application-specific naming conventions are naturally important aspects in ICN. It is common that applications define their own naming convention (i.e., semantics of elements in the name hierarchy). Such names can often be directly derived from application requirements, for example a name like /my-home/living-room/light/switch/main could be relevant in a smart home setting, and corresponding devices and applications could use a corresponding convention to facilitate controllers finding sensors and actors in such a system with minimal user configuration.

The aforementioned features make ICN amenable to data discovery. Because there is no name/address chasm as in IP-based systems, data can be discovered by sending an INTEREST to named data objects

directly (assuming a naming convention as described above). Moreover, ICN can authenticate received data objects directly, for example using local trust anchors in the network (for example in a home network).

Advanced ICN features for data discovery include the concept of manifests in CCNx, i.e., ICN objects that describe data collections, and data set synchronization protocols in NDN (https://named-data.net/publications/li2018sync-intro/) that can inform consumers about the availability of new data in a tree-based data structure (with automatic retrieval and authentication). Also, ICN is not limited to accessing static data. Frameworks such as Named Function Networking (http://www.named-function.net) and RICE can provide the general ICN feature for discovery not only for data but also for name functions (for in-network computing) and for their results.

5. Use Cases of Edge Data Discovery

5.1. Autonomous Vehicles

Autonomous vehicles rely on the processing of huge amounts of complex data in real-time for fast and accurate decisions. These vehicles will rely on high performance compute, storage and network resources to process the volumes of data they produce in a low latency way. Various systems will need a standard way to discover the pertinent data for decision making.

5.2. Video Surveillance

The majority of the video surveillance footage will remain at the edge infrastructure (not sent to the cloud data center). This footage is coming from vehicles, factories, hotels, universities, farms, etc. Much of the video footage will not be interesting to those evaluating the data. A mechanism, perhaps a set of protocols, is needed to identify the interesting data at the edge. What constitutes interesting will be context specific, e.g., a video frame might be considered interesting if and only if it includes a car, or person, or bicyclist, or a backyard nocturnal creature, or etc. Interesting video data may be stored longer in storage systems at the very edge of the network and/or in networking equipment further away from the device edge that has access to data in flight as it transits the network.

5.3. Elevator Networks

Elevators are one of many industrial applications of edge computing. Edge equipment receives data from hundreds of elevator sensors. The data coming into the edge equipment is vibration, temperature, speed,

level, video, etc. We need the ability to identify where the data we need to evalute is located.

5.4. Service Function Chaining

Service function chaining (SFC) allows the instantiation of an ordered set of service functions (SFs) and the subsequent "steering" of traffic through them. Service functions are expected to be deployed at the edge of the network, as a feasible deployment of "Compute In the Network", with multiple types of potential use cases (e.g., fog robotics, Industry 4.0 automation, etc). Service functions provide a specific treatment of received packets, therefore they need to be discoverable so they can be used in a given service composition via SFC. In addition, these functions can be producers and/or consumers of data. So far, how the functions are discovered and composed has been out of the scope of discussions in the IETF. While there are some mechanisms that can be used and/or extended to provide this functionality, more work needs to be done. An example of this can be found in [I-D.bernardos-sfc-discovery].

In an SFC environment deployed at the edge, the discovery protocol may also need the following kind of meta-data information per (service) function:

- Service Function Type: identifying the category of function provided.
- SFC-aware: Yes/No. Indicates if the function is SFC-aware.
- Route Distinguisher (RD): IP address indicating the location of the function.
- Pricing/costs details.
- Migration capabilities of the function: whether a given function can be moved to another provider (potentially including information about compatible providers topologically close).
- Mobility of the device hosting the function, with e.g. the following sub-options:

  Level: no, low, high; or a corresponding scale (e.g., 1 to 10).

  Current geographical area (e.g., GPS coordinates, post code).

  Target moving area (e.g., GPS coordinates, post code).

- Power source of the device hosting the function, with e.g. the following sub-options:

  Battery: Yes/No. If Yes, the following sub-options could be defined:

  Capacity of the battery (e.g., mmWh).

  Charge status (e.g., %).

  Lifetime (e.g., minutes).

Discovery of resources in an NFV environment: virtualized resources do not need to be limited to those available in traditional data centers, where the infrastructure is stable, static, typically homogeneous and managed by a single admin entity. Computational capabilities are becoming more and more ubiquitous, with terminal devices getting extremely powerful, as well as other types of devices that are close to the end users at the edge (e.g., vehicular onboard devices for infotainment, micro data centers deployed at the edge, etc.). It is envisioned that these devices would be able to offer storage, computing and networking resources to nearby network infrastructure, devices and things (the fog paradigm). These resources can be used to host functions, for example to offload/ complement other resources available at traditional data centers, but also to reduce the end-to-end latency or to provide access to specialized information (e.g., context available at the edge) or hardware. Similar to the discovery of functions, while there are mechanisms that can be reused/extended, there is no complete solution yet defined. An example of work in this area is [I-D.bernardos-intarea-vim-discovery]. The availability of this meta-data about the capabilities of nearby physical as well as virtualized resources can be made discoverable through edge data discovery mechanisms.

5.5. Ubiquitous Witness

Ubiquitous Witness (UW) is the name of a use case that has been presented in past COINRG and ICNRG meetings at the IETF. It describes what might occur in dense IoT deployments when an anomaly occurs. There are many "witnesses" to report on what happened within a limited region of interest and around an approximate point in time. The use case highlights the need for upstream data discovery and management. It is agnostic to where the dense IoT deployment resides, whether in a factory, home, commercial building, city, entertainment venue, et cetera. For example, as cameras and other sensors have become ubiquitous in Smart Cities, it would be helpful to be able to discover and examine data from all devices and sensors

that witnessed an accident in a city intersection; this could be data from cameras mounted at the intersection itself, on nearby buildings, in cars, and cell phones of individuals on location.

If an anomaly were to automatically trigger independent upstream flows of video data from all of the witnesses (within a proximal vicinity and time window), the data flows would naturally converge at shared collection or aggregation points in the network. These edge nodes might opt to vault any data deemed part of a safety-related anomaly, which would enable interested parties (the car owner, the car manufacturer, an insurance company, a city traffic planner) to investigate the root cause of the anomaly after the fact. The implication however is that enough meta data has been generated alongside the data itself (e.g., a data name, an identifier, or a geo location and timestamp), to allow the retrieval of this distributed data, provided those asking have proper authorization to access it.

The UW streams are contextually-related and as such it can be advantageous also to be able to process them simultaneously, at the time they are first generated. For example if collection nodes could derive that groups of data streams are contextually-related, they could stitch streams together to create a 360-degree view of the anomalous event (e.g., to walk around in the data), or to winnow the set of vaulted data to only the "best" video (e.g., highest resolution, unoccluded views) or to perform compute-in-the-network to enable them to fit within the available resources (e.g., at the receiving node due to the convergence or implosion of upstream data, or over the next hoplink). Ubiquitous Witness data doesn't have to be video data, but video illustrates why one might want to jointly process upstream flows in real-time.

6. IANA Considerations

N/A

7. Security Considerations

Security considerations will be a critical component of edge data discovery particularly as intelligence is moved to the extreme edge where data is to be extracted.

An assumption is that all data will have associated policies (default, inherited or configured) that describe access control permissions. Consequently, the discoverability of data will be a function of who or what has requested access. In other words, the discoverable view into the available data will be limited to those who are authorized. Discovering edge data that is exclusively private is out of scope of this document, the assumption being that

there will be some edge clouds that do not expose or publish the availability of their data. Although edge data may be sent to the back-end cloud as needed, there is nothing that precludes it from being discoverable if the cloud offers it as public.

A trust relationship may be needed between the source and target of a discovery operation to avoid denial of service attacks from a malicious source or target of the operation. And discovery information, which is exposed by a node or network, may need to be protected for privacy purposes, e.g. not leak information in the presence of a certain type of data in a network.

8. Acknowledgement

The authors thank Dave Oran, Greg Skinner and Lixia Zhang for contributing to this document.

9. Normative References

[I-D.bernardos-intarea-vim-discovery]
Bernardos, C. and A. Mourad, "IPv6-based discovery and association of Virtualization Infrastructure Manager (VIM) and Network Function Virtualization Orchestrator (NFVO)", draft-bernardos-intarea-vim-discovery-04 (work in progress), September 2020.

[I-D.bernardos-sfc-discovery]
Bernardos, C. and A. Mourad, "Service Function discovery in fog environments", draft-bernardos-sfc-discovery-05 (work in progress), September 2020.

[I-D.irtf-icnrg-ccnxmessages]
Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format", draft-irtf-icnrg-ccnxmessages-09 (work in progress), January 2019.

[I-D.irtf-icnrg-ccnxsemantics]
Mosko, M., Solis, I., and C. Wood, "CCNx Semantics", draft-irtf-icnrg-ccnxsemantics-10 (work in progress), January 2019.

[I-D.kutscher-icnrg-rice]
Krol, M., Habak, K., Oran, D., Kutscher, D., and I. Psaras, "Remote Method Invocation in ICN", draft-kutscher-icnrg-rice-00 (work in progress), October 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7927] Kutscher, D., Ed., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", RFC 7927, DOI 10.17487/RFC7927, July 2016, <https://www.rfc-editor.org/info/rfc7927>.

Authors' Addresses

Mike McBride
Futurewei

Email: michael.mcbride@futurewei.com

Dirk Kutscher
Emden University

Email: ietf@dkutscher.net

Eve Schooler
Intel

Email: eve.m.schooler@intel.com
URI: http://www.eveschooler.com

Carlos J. Bernardos
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: http://www.it.uc3m.es/cjbc/

Diego R. Lopez
Telefonica

Email: diego.r.lopez@telefonica.com
URI: https://www.linkedin.com/in/dr2lopez/

Xavier de Foy
InterDigital Communications, LLC
1000 Sherbrooke West
Montreal
Canada

Email: Xavier.Defoy@InterDigital.com

COIN
INTERNET-DRAFT
Intended Status: Informational
Expires: April 25, 2021

D. Trossen
Huawei
C. Sarathchandra
InterDigital Inc.
M. Boniface
University of Southampton

October 23, 2020

In-Network Computing for App-Centric Micro-Services
draft-sarathchandra-coin-appcentres-03

Abstract

The application-centric deployment of 'Internet' services has increased over the past ten years with many millions of applications providing user-centric services, executed on increasingly more powerful smartphones that are supported by Internet-based cloud services in distributed data centres, the latter mainly provided by large scale players such as Google, Amazon and alike. This draft outlines a vision for evolving those data centres towards executing app-centric micro-services; we dub this evolved data centre as an AppCentre. Complemented with the proliferation of such AppCentres at the edge of the network, they will allow for such micro-services to be distributed across many places of execution, including mobile terminals themselves, while specific micro-service chains equal today's applications in existing smartphones. We outline the key enabling technologies that needs to be provided for such evolution to be realized, including references to ongoing IETF work in some areas.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Table of Contents

1 Introduction

With the increasing dominance of smartphones and application markets, the end-user experiences today have been increasingly centered around the applications and the ecosystems that smartphone platforms create. The experience of the 'Internet' has changed from 'accessing a web site through a web browser' to 'installing and running an application on a smartphone'. This app-centric model has changed the way services are being delivered not only for end-users, but also for business-to-consumer (B2C) and business-to-business (B2B) relationships.

Designing and engineering applications is largely done statically at design time, such that achieving significant performance improvements thereafter has become a challenge (especially, at runtime in response to changing demands and resources). Applications today come prepackaged putting them at disadvantage for improving efficiency due to the monolithic nature of the application packaging. Decomposing application functions into micro-services [MSERVICE1] [MSERVICE2] allows applications to be packaged dynamically at run-time taking varying application requirements and constraints into consideration. Interpreting an application as a chain of micro-services, allows the application structure, functionality, and performance to be adapted dynamically at runtime in consideration of tradeoffs between quality of experience, quality of service and cost.

Interpreting any resource rich networked computing (and storage) capability not just as a pico or micro-data centre, but as an application-centric execution data centre (AppCentre), allows distributed execution of micro-services. Here, the notion of an 'application' constitutes a set of objectives being realized in a combined packaging of micro-services under the governance of the 'application provider'. These micro-services may then be deployed on the most appropriate AppCentre (edge/fog/cloud resources) to satisfy requirements under varying constraints. In addition, the high degree of distribution of application and data partitions, and compute resources offered by the execution environment decentralizes control between multiple cooperating parties (multi-technology, multi-domain, multi-ownership environments). Furthermore, compute resource availability may be volatile, particularly when moving along the spectrum from well-connected cloud resources over edge data centres to user-provided compute resources, such as (mobile) terminals or home-based resources such as NAS and IoT devices.

We believe that the emergence of AppCentreS will democratize infrastructure and service provision to anyone with compute resources with the notion of applications providing an element of governing the execution of micro-services. This increased distribution will lead to new forms of application interactions and user experiences based on

cooperative AppCentreS (pico-micro and large cloud data centres), in which applications are being designed, dynamically composed and executed.

2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3 Use Cases

Although our motivation for the 'AppCentre' term stems from the (mobile) application ecosystem, the use cases in this section are not limited to mobile applications only. Instead, we interpret 'applications' as a governing concept of executing a set of micro-services where the 'application provider' can reach from those realizing mobile applications over novel network applications to emerging infrastructure offerings serving a wide range of applications in a purpose- (and therefore application-)agnostic manner. The following use cases provide examples for said spectrum of applications.

3.1 Mobile Application Function Offloading

Partitioning an application into micro-services allows for denoting the application as a collection of functions for a flexible composition and a distributed execution, e.g., most functions of a mobile application can be categorized into any of three, "receiving", "processing" and "displaying" function groups.

Any device may realize one or more of the micro-services of an application and expose them to the execution environment. When the micro-service sequence is executed on a single device, the outcome is what you see today as applications running on mobile devices. However, the execution of functions may be moved to other (e.g., more suitable) devices which have exposed the corresponding micro-services to the environment. The result of the latter is flexible mobile function offloading, for possible reduction of power consumption (e.g., offloading CPU intensive process functions to a remote server) or for improved end user experience (e.g., moving display functions to a nearby smart TV).

The above scenario can be exemplified in an immersive gaming application, where a single user plays a game using a VR headset. The headset hosts functions that "display" frames to the user, as well as the functions for VR content processing and frame rendering combining with input data received from sensors in the VR headset. Once this

application is partitioned into micro-services and deployed in an app-centric execution environment, only the "display" micro-service is left in the headset, while the compute intensive real-time VR content processing micro-services can be offloaded to a nearby resource rich home PC, for a better execution (faster and possibly higher resolution generation).

Figure 1 shows one realization of the above scenario, where a 'DPR app' is running on a mobile device (containing the partitioned Display(D), Process(P) and Receive(R) micro services) over an SDN network. The packaged applications are made available through a localized 'playstore server'. The application installation is realized as a 'service deployment' process (Section 5.2.), combining the local app installation with a distributed micro-service deployment (and orchestration) on most suitable AppCentreS ('processing server').

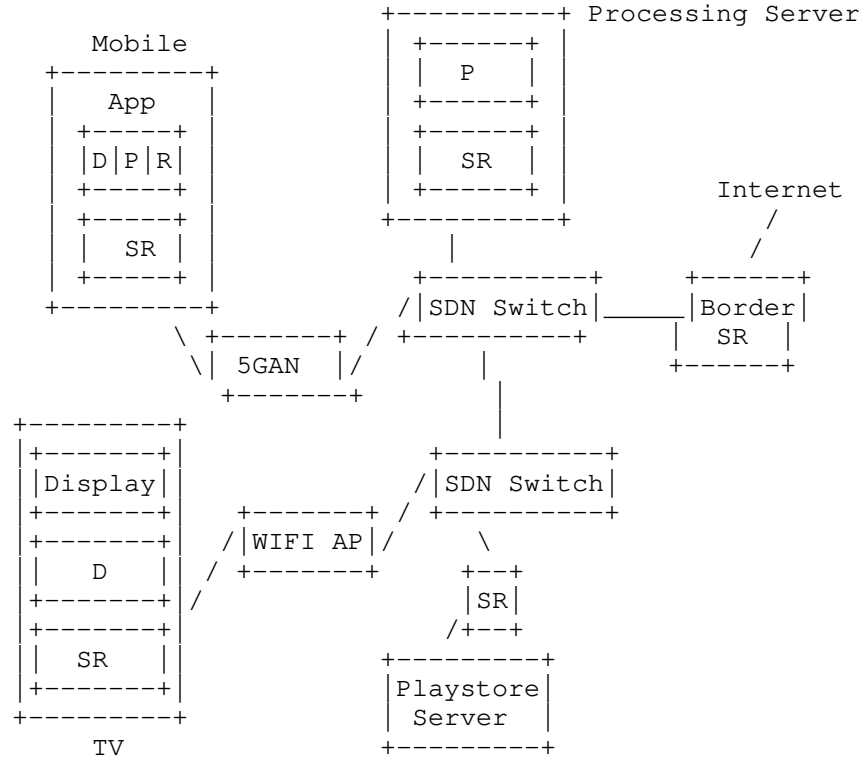


Figure 1: Application Function Offloading Example

Such localized deployment could, for instance, be provided by a visiting site, such as a hotel or a theme park. Once the 'processing'

micro-service is terminated on the mobile device, the 'service routing' (SR) elements in the network (Section 5.4.) routes requests to the previously deployed 'processing' micro-service running on the 'processing server' AppCentre over an existing SDN network. As an extension to the above scenarios, we can also envision that content from one processing micro-service may be distributed to more than one display micro-service, e.g., for multi/many-viewing scenarios.

3.2 Interactive Real-time Applications

There has been a recent shift from applications that provide single-user experiences, such as the ones described in the previous section to collaborative/cooperative experiences that are highly interactive with strict real-time requirements, such as collaborative working, education, multi-user gaming, and mixed/virtual reality. This leads to increasing interaction where input (e.g., gesture, gaze, touch, movement) and output (e.g., visual display, sound, and actuation) needs to be processed within strict timing constraints and synchronized to ensure temporal and spatial consistency with local and distant users. App-centric design allows functions with high data and process coupling to be modularized, deployed and executed, such that the subset of micro-services is cooperatively executed towards optimizing the interactive experiences.

The same example of the previous section can be envisaged in a multi-player gaming scenario. Here the micro-services that need to be executed cooperatively are executed in a localized and synchronized manner to ensure player coordination and synchronized state between collaborating players.

3.3 Distributed AI

There is a growing range of use cases demanding for the realization of AI capabilities among distributed endpoints. Such demand may be driven by the need to increase overall computational power for large-scale problems. Other solutions may desire the localization of reasoning logic, e.g., for deriving attributes that better preserve privacy of the utilized raw input data. Examples for large-scale AI problems include biotechnology and astronomy related reasoning over massive amounts of observational input data. Examples for localizing input data for privacy reasons include radar-like application for the development of topological mapping data based on (distributed) radio measurements at base stations (and possibly end devices), while the processing within radio access networks (RAN) already constitute a distributed AI problem to a certain extent albeit with little flexibility in distributing the execution of the AI logic.

Reasoning frameworks, such as TensorFlow, may be utilized for the

realization of the (distributed) AI logic, building on remote service invocation through protocols such as gRPC [GRPC] or MPI [MPI] with the intention of providing an on-chip NPU (neural processor unit) like abstraction to the AI framework.

3.4 Content Delivery Networks

Delivery of content to end users often relies on Content Delivery Networks (CDNs) storing said content closer to end users for latency reduced delivery with DNS-based indirection being utilized to serve the request on behalf of the origin server. From the perspective of this draft, a CDN can be interpreted as a (network service level) application with distributed logic for distributing content from the origin server to the CDN ingress and further to the CDN replication points which ultimately serve the user-facing content requests.

Studies such as those in [FCDN] have shown that content distribution at the level of named content, utilizing efficient (e.g., Layer 2) multicast for replication towards edge CDN nodes, can significantly increase the overall network and server efficiency. It also reduces indirection latency for content retrieval as well as reduces required edge storage capacity by benefiting from the increased network efficiency to renew edge content more quickly against changing demand.

3.5 Compute-Fabric-as-a-Service (CFaaS)

App-centric execution environments, consisting of Layer 2 connected AppcentreS, provide the opportunity for infrastructure providers to offer CFaaS type of offerings to application providers. Those app providers utilize the compute fabric exposed by this CFaaS offering for the purposes defined through their applications. In other words, the compute resources can be utilized to execute the desired micro-services of which the application is composed, while utilizing the inter-connection between those compute resources to do so in a distributed manner.

We foresee those CFaaS offerings to be tenant-specific, a tenant here defined as the provider of at least one application. For this, we foresee an interaction between CFaaS provider and tenant to dynamically select the appropriate resources to define the demand side of the fabric. Conversely, we also foresee the supply side of the fabric to be highly dynamic with resources being offered to the fabric through, e.g., user-provided resources (whose supply might depend on highly context-specific supply policies) or infrastructure resources of intermittent availability such as those provided through road-side infrastructure in vehicular scenarios. The resulting dynamic demand-supply matching establishes a dynamic nature of the

compute fabric that in turn requires trust relationships to be built dynamically between the resource provider(s) and the CFaaS provider. This also requires the communication resources to be dynamically adjusted to interconnect all resources suitably into the (tenant-specific) fabric exposed as CFaaS.

4 Requirements Derived from Use Cases

The following requirements are derived from the presented use cases in Section 3.1. to 3.5., numbered according to the use case numbers (as main item numbers) although those requirements apply in some cases across more than one of the presented use cases.

Req 1.1: Any app-centric execution environment MUST provide means for routing of service requests between resources in the distributed environment.

Req 1.2: Any app-centric execution environment MUST provide means for dynamically choosing the best possible micro-service sequence (i.e., chaining of micro-services) for a given application experience. Means for discovering suitable micro-service SHOULD be provided.

Req 1.3: Any app-centric execution environment MUST provide means for pinning the excution of a specific micro-service to a specific resource instance in the distributed environment.

Req 1.4: Any app-centric execution environment SHOULD provide means for packaging micro-services for deployments in distributed networked computing environments. The packaging MAY include any constraints regarding the deployment of service instances in specific network locations or compute resources. Such packaging SHOULD conform to existing application deployment models, such as mobile application packaging, TOSCA orchestration templates or tar balls or combinations thereof.

Req 2.1: Any app-centric execution environment MUST provide means for real-time synchronization and consistency of distributed application states.

Req 3.1: Any app-centric execution environment MUST provide means to specify the constraints for placing (AI) execution logic in certain logical execution points (and their associated physical locations).

Req 3.2: Any app-centric execution environment MUST provide support for app/micro-service specific invocation protocols.

Req 4.1: Any app-centric execution environment SHOULD utilize Layer 2 multicast transmission capabilities for responses to concurrent service requests.

Req 5.1: Any app-specific execution environment SHOULD expose means to specify the requirements for the tenant-specific compute fabric being utilized for the app execution.

Req 5.2: Any app-specific execution environment SHOULD allow for dynamic integration of compute resources into the compute fabric being utilized for the app execution; those resources include, but are not limited to, end user provided resources.

Req 5.3: Any app-specific execution environment MUST provide means to optimize the inter-connection of compute resources, including those dynamically added and removed during the provisioning of the tenant-specific compute fabric.

Req 5.4: Any app-specific execution environment MUST provide means for ensuring availability and usage of resources is accounted for.

5 Enabling Technologies

This section discusses a number of enabling technologies relevant for the realization of the app-centric micro-service vision laid out in this draft.

EDITOR NOTE: Section 5 will be updated to include the addressing of specific requirements listed in Section 4.

5.1 Application Packaging

Applications often consist of one or more sub-elements (e.g., audio, visual, hepatic elements) which are 'packaged' together, resulting in the final installable software artifact. Conventionally, application developers perform the packaging process at design time, by packaging a set of software components as a (often single) monolithic software package, for satisfying a set of predefined application requirements.

Decomposing micro-services of an application, and then executing them on peer execution points in AppCentreS (e.g., on an app-centric serverless runtime [SRVLESS]) can be done with design-time planning. Micro-service decomposition process involves, defining clear boundaries of the micro-service (e.g., using wrapper classes for

handling input/output requests), which could be done by the application developer at design-time (e.g., through Android app packaging by including, as part of the asset directory, a service orchestration template [TOSCA] that describes the decomposed micro-services). Likewise, the peer execution points could be 'known' to the application (e.g., using well-known and fixed peer execution points on AppCentreS) and incorporated with the micro-services by the developer at design-time.

Existing programming frameworks address decomposition and execution of applications centering around other aspects such as concurrency [ERLANG]. For decomposing at runtime, application elements can be profiled using various techniques such as dynamic program analysis or dwarf application benchmarks. The local profiler information can be combined with the profiler information of other devices in the network for improved accuracy. The output of such a profiler process can then be used to identify smaller constituting sub-components of the application in forms of pico-services, their interdependencies and data flow (e.g., using caller/callee information, instruction usage). Due to the complex nature of resulting application structure and therefore its increased overhead, in most cases, it may not be optimal to decompose applications at the pico level. Therefore, one may cluster pico-services into micro-services with common characteristics, enabling a meaningful (e.g., clustering pico-services with same resource dependency) and a performant decomposition of applications. Characteristics of micro-services can be defined as a set of concepts using an ontology language, which can then be used for clustering similar pico-services into micro-services. Micro-services may then be partitioned along their identified borders. Moreover, mechanisms for governance, discovery and offloading can be employed for 'unknown' peer execution points on AppCentreS with distributed loci of control.

Therefore, with this app-centric model, application packaging can be done at runtime by constructing micro-service chains for satisfying requirements of experiences (e.g., interaction requirements), under varying constraints (e.g., temporal consistency between multiple players within a shared AR/VR world)[SCOMPOSE]. Such packaging includes mechanisms for selecting the best possible micro-services for a given experience at runtime in the multi-technology environment. These run-time packaging operations may continuously discover the 'unknown' and adapt towards an optimal experience. Such decision mechanisms handle the variability, volatility and scarcity within this multi-X framework.

5.2 Service Deployment

The service function chains, constituting each individual

application, will need deployment mechanisms in a true multi-X (multi-user, multi-infrastructure, multi-domain) environment [SDEPLOY1][SDEPLOY2]. Most importantly, application installation and orchestration processes are married into one, as a set of procedures governed by device owners directly or with delegated authority. However, apart from extending towards multi-X environments, the process also needs to cater for changes in the environment, caused, e.g., by movement of users, new pervasive sensors/actuators, and changes to available infrastructure resources. Methods are needed to deploy service functions as executable code into chosen service execution points. Those methods need to support the various endpoint (e.g., device stacks, COTS stacks, etc.) and service function realizations, e.g., through utilizing existing and emerging virtualization techniques.

A combination of application installation procedure and orchestrated service deployment can be achieved by utilizing the application packaging with integrated service deployment templates described in Section 5.1 such that the application installation procedure on the installing device is being extended to not only install the local application package but also extract the service deployment template for orchestrating with the localized infrastructure, using, for instance, REST APIs for submitting the template to the orchestrator.

5.3 Compute Inter-Connection at Layer 2

While Layer2 switching technologies have long proliferated in data centre deployments, recent developments have advanced the capabilities for interconnecting distributed computing resources over Layer2 technologies. For instance, the efforts in 3GPP on so-called '5G LAN' (or Vertical LAN) [SA2-5GLAN] allow for establishing a Layer2 bearer between participating compute entities, using a vertical-specific LAN identifier for packet forwarding between the distributed Layer2 entities. Combined with Layer2 technology in data centres as well as office and home networks alike, this enables the deployment of services in vertical (Layer2) networks, interconnecting with other Internet-based services through dedicated service gateways.

Real deployments and realizations will have to show the scalability of this approach but it points into a direction where application or service-specific deployments could potentially 'live' entirely in their own vertical network, interconnecting only based on need (which for many services may not exist). From the application's or service's perspective, the available compute resource pool will look no different from that being realized in a single data centre albeit with the possibility to being highly distributed now among many (e.g., edge) data centres as well as mobile devices.

In such a deployment, it is interesting to study the realization of suitable service routing capabilities, leading us to the next technology area of interest.

5.4 Service Routing

Routing service requests is a key aspect within a combined compute and network infrastructure in order to enable true end-to-end experiences across distributed application execution points provisioned on distant cloud, edge and device-centric resources. Once the micro-services are packaged and deployed in such highly distributed micro-data centres, the routing mechanisms must ensure efficient information exchange between corresponding micro-services, e.g., at the level of service requests, within the multi-technology execution environment.

Routing here becomes a problem of routing micro-service requests, not just packets, as done through IP. This calls for some form of 'flow affinity' that allows for treating several packets as part of a request semantic. This is important, e.g., for mobility (avoiding to send some packets of a larger request to one entity, while other packets are sent to another one, therefore creating incomplete information at both entities as a result). Also, when applying constraints to the forwarding of packets (discussed in more detail in Section 5.6), it is important to apply the actions across the packets of the request rather than individually.

Another key aspect is that of addressing services. Traditionally, the combination of the Domain Naming Service (DNS) and IP routing has been used for this purpose. However, the advent of virtualization with use cases such as those outlined in Section 3 (such as those on app-specific micro-services on mobile devices) have made it challenging to further rely on the DNS. Apart from the initial delay observed when resolving a service name into a locator for the first time, the long delay in updating DNS entries to 'point' to the right micro-service instances prohibits offloading to dynamically created service instances. If one was to use the DNS, one would be updating the DNS entries at a high rate, caused by the diversity of trigger, e.g., through movement. DNS has not been designed for such frequent update, rendering it useless for such highly dynamic applications. With many edge scenarios in the VR/AR space demanding interactivity and being latency-sensitive, efficient routing will be key to any solution.

Various ongoing work on service request forwarding [RFC8677] with the service function chaining [RFC7665] framework as well as name-based routing [ICN5G][ICN4G] addresses some aspects described above albeit with a focus on HTTP as the main invocation protocol. Extensions will

be required to support other invocation protocols, such as GRPC or MPI (for distributed AI use cases, as outlined in Section 3.3.). Proposals such as those in [DYN-CAST] suggest extensions to the IP anycast scheme to enable the flexible routing of service requests to one or more service instances. Common to those proposals is the use of a semantic identifier, often a service identifier akin to a URL, in the routing decision within the network.

5.5 Constraint-based Forwarding Decisions

Allocating the right resources to the right micro-services is a fundamental task when executing micro-services across highly distributed micro-data centres (e.g., resource management in cloud [CLOUDFED]). This is particularly important in the light of volatile resource availability as well as concurrent and highly dynamic resource access. Once the specific set of micro-services for an application has been identified, requirements (e.g., QoS) must be ensured by the execution environment. Therefore, all micro-data centres and the execution environment will need to realize mechanisms for ensuring the utilization of specific resources within a pool of resources for a specific set of micro-services belonging to one application, while also ensuring integrity of the wider system.

In relation to the service routing capability discussed in the previous sub-section, constraints may need to be introduced into the forwarding decisions for service requests. Such constraints will likely go beyond network load and latency, as often applied in scenarios such as load balancing in CDNs. Instead, those constraints are generally app/service-specific and will need a suitable representation for the use within network nodes, i.e., the routers that are forwarding service requests. Moreover, individual router decisions (e.g., realized through matching operations such as min/max/equal over a constraint representation) may be coordinated to achieve a distribution of service requests among many service instances, effectively realizing a service scheduling capability in the network, optimized around service-specific constraints, not unlike many existing data centre service switching schemes.

5.6 Collective Communication

Many micro-service scenarios may exhibit some form of collective communication beyond 'just' unicast communication, therefore requiring support for 1:M, M:1, and M:N communication. It is important to consider here that such collective communication is often extremely short-lived and can even take place at the level of a single request, i.e., a following request may exhibit a different communication pattern, even at least a different receiver group for the same pattern, such as in the case of an interactive game. It is

therefore required that solutions for supporting such collective communication must support the spontaneous formation of multicast relations, as observed in those scenarios.

5.7 State Synchronization

Given the highly distributed nature of app-centric micro-services, their state exchange and synchronization is a very crucial aspect for ensuring in-application and system wide consistency. Mechanisms that ensure consistency will ensure that data is synchronized with different spatial, temporal and relational data within a given time period. From the perspective of support through in-network compute capabilities, such as provided through technologies like P4, it is important to consider what system and protocol support is required to utilize such in-network capabilities.

5.8 Dynamic Contracts

NOTE: left for future revision

6 Security Considerations

The use of semantic (or service) identifiers for routing decisions, as mentioned in Section 5.4October 1, 2018April 4, 2019, requires methods to ensure the privacy and security of the communication through avoiding the exposure of service semantic (which is realized at the application layer) to the network layer, therefore opening up the opportunity for traffic inspection, among other things. The use of cryptographic information, e.g., through self-certifying identifiers, should be investigated to mitigate potential security and privacy risks.

7 IANA Considerations

N/A

8 Conclusion

This draft positions the evolution of data centres as one of becoming execution centres for the app-centric experiences provided today mainly by smart phones directly. With the proliferation of data centres closer to the end user in the form of edge-based micro data centres, we believe that app-centric experiences will ultimately be executed across those many, highly distributed execution points that this increasingly rich edge environment will provide, such as smart glasses and IoT devices. Although a number of activities are currently underway to address some of the challenges for realizing such AppCentre evolution, we believe that the proposed COIN research

group will provide a suitable forum to drive forward the remaining research and its dissemination into working systems and the necessary standardization of key aspects and protocols.

9 References

9.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7665] Halpern, J., Ed., and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

9.2 Informative References

[MSERVICE1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday,today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195-216). Springer, Cham.

[MSERVICE2] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software, 33(3), 42-52.

[SRVLESS] C. Cicconetti, M. Conti and A. Passarella, "An Architectural Framework for Serverless Edge Computing: Design and Emulation Tools," 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, 2018, pp. 48-55. doi: 10.1109/CloudCom2018.2018.00024

[TOSCA] Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. OASIS Standard. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>.

[ERLANG] Armstrong, Joe, et al. "Concurrent programming in ERLANG." (1993).

[SCOMPOSE] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations", ACM Computing Surveys,46(4):1-34, Mar. 2014

[SDEPLOY1] Lu, H., Shtern, M., Simmons, B., Smit, M., & Litoiu, M. (2013, June). Pattern-based deployment service for next generation clouds. In 2013 IEEE Ninth World Congress on Services (pp. 464-471). IEEE.

[SDEPLOY2] Eilam, T., Elder, M., Konstantinou, A. V., & Snible, E. (2011, May). Pattern-based composite application deployment. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops (pp. 217-224). IEEE.

[RFC8677] Trossen, D., Purkayastha, D., Rahman, A., "Name-Based Service Function Forwarder (nSFF) Component within a Service Function Chaining (SFC) Framework", RFC 8677, November 2019.

[ICN5G] Ravindran, R., Suthar, P., Trossen, D., Wang, C., White, G., "Enabling ICN in 3GPP's 5G NextGen Core Architecture", <https://tools.ietf.org/html/draft-ravi-icnrg-5gc-icn-03> (work in progress), March 2019.

[ICN4G] Suthar, P., Jangam, Ed., Trossen, D., Ravindran, R., "Native Deployment of ICN in LTE, 4G Mobile Networks", <https://tools.ietf.org/html/draft-irtf-icnrg-icn-lte-4g-03> (work in progress), March 2019.

[CLOUDFED] M. Liaqat, V. Chang, A. Gani, S. Hafizah Ab Hamid, M. Toseef, U. Shoaib, R. Liaqat Ali, "Federated cloud resource management: Review and discussion", Elsevier Journal of Network and Computer Applications, 2017.

[GRPC] High performance open source universal RPC framework, https://grpc.io/

[MPI] A. Vishnu, C. Siegel, J. Daily, "Distributed TensorFlow with MPI", https://arxiv.org/pdf/1603.02339.pdf

[FCDN] M. Al-Naday, M. J. Reed, J. Riihijarvi, D. Trossen, N. Thomos, M. Al-Khalidi, "fCDN: A Flexible and Efficient CDN Infrastructure without DNS Redirection of Content Reflection", https://arxiv.org/pdf/1803.00876.pdf

[DYN-CAST] Y. Li, J. He, L. Geng, P. Liu, Y. Cui, "Framework of Compute First Networking (CFN)", <https://tools.ietf.org/html/draft-li-rtgwg-cfn-framework-00> (work in progress), November 2019

[SA2-5GLAN] 3gpp-5glan, "SP-181129, Work Item Description,

Vertical_LAN(SA2), 5GS Enhanced Support of Vertical and LAN Services", 3GPP, <http://www.3gpp.org/ftp/tsg_sa/TSG_SA/Docs/SP-181120.zip>

Authors' Addresses

Dirk Trossen
Huawei Technologies Duesseldorf GmbH
Riesstr. 25C
80992 Munich
Germany

Email: Dirk.Trossen@Huawei.com

Chathura Sarathchandra
InterDigital Europe, Ltd.
64 Great Eastern Street, 1st Floor
London EC2A 3QR
United Kingdom

Email: Chathura.Sarathchandra@InterDigital.com

Michael Boniface
University of Southampton
University Road
Southampton SO17 1BJ
United Kingdom

Email: mjb@it-innovation.soton.ac.uk