Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 20 November 2025

D. Lou
L. Iannone
Y. Li
C. Zhang
Huawei
K. Yao
China Mobile

19 May 2025

Signaling In-Network Computing operations (SINC)
draft-lou-rtgwg-sinc-04

Abstract

This memo introduces "Signaling In-Network Computing operations" (SINC), a mechanism to enable signaling in-network computing operations on data packets in specific scenarios like NetReduce, NetDistributedLock, NetSequencer, etc. In particular, this solution allows to flexibly communicate computational parameters, to be used in conjunction with the payload, to in-network SINC-enabled devices in order to perform computing operations.

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 20 November 2025.

Copyright Notice

Table of Contents

1. Introduction

According to the original design, the Internet performs just "store and forward" of packets, and leaves more complex operations at the end-points. However, new emerging applications could benefit from in-network computing to improve the overall system efficiency ([GOBATTO], [ZENG]). It is different from what the IETF Computing-Aware Traffic Steering (CATS) working group is chartered for service instance selection based on network and compute metrics between clients of a service and sites offering the service. The in-network computing is more about "light" data calculation/operation performed in the network to reduce the computation work load for the end hosts.

The formation of the COmputing In-Network (COIN) Research Group [COIN], in the IRTF, encourages people to explore this emerging technology and its impact on the Internet architecture. The "Use Cases for In-Network Computing" document [I-D.irtf-coinrg-use-cases] introduces some use cases to demonstrate how real applications can benefit from COIN and show essential requirements demanded by COIN applications.

Recent research has shown that network devices undertaking some computing tasks can greatly improve the network and application performance in some scenarios, like for instance aggregating path-computing [NetReduce], key-value(K-V) cache [NetLock], and strong consistency [GTM]. Their implementations mainly rely on programmable network devices, by using P4 [P4] or other languages. In the context of such heterogeneity of scenarios, it is desirable to have a generic and flexible framework, able to explicitly signaling the computing operation to be performed by network devices, which should be applicable to many use cases, enabling easier deployment.

This document specifies such a Signaling In-Network Computing (SINC) framework for, as the name states, in-network computing operation. The computing functions are hosted on network devices, which, in this memo, are generally named as SINC switches/routers.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. SINC Relevant Use Cases

Hereafter a few relevant use cases are described, namely NetReduce, NetDistributedLock, and NetSequencer, in order to help understanding the requirements for a framework. Such a framework, should be generic enough to accommodate a large variety of use cases, besides the ones described in this document.

3.1. NetReduce

Over the last decade, the rapid development of Deep Neural Networks (DNN) has greatly improved the performance of many applications like computer vision and natural language processing. However, DNN training is a computation intensive and time consuming task, which has been increasing exponentially in the past 10 years. Scale-up techniques concentrating on the computing capability of a single

device cannot meet the expectation. Distributed DNN training approaches with synchronous data parallelism like Parameter Server [PARAHUB] and All-Reduce [MGWFBP] are commonly employed in practice, which on the other hand, become increasingly a network-bound workload since communication becomes a bottleneck at scale.

Comparing to host-oriented solutions, in-network aggregation approaches like SwitchML [SwitchML], SHArP [SHArP], and NetReduce [NetReduce] can potentially reduce to nearly half the bandwidth needed for data aggregation, by offloading gradients aggregation from the host to network switches. However, they are limited to one single specific operation, namely aggregation.

SwitchML is designed to implement in-network workers performing data aggregation relying on Remote Direct Memory Access (RDMA) [ROCEv2] and the application layer logic. In principle this allows to repurpose relatively easily the system at the cost of deploying new workers since there is no in-network operation signaling.

NetReduce [NetReduce] does tackle the same problem like SwitchML, including the use of RDMA, but introduces an In-Network Aggregation (INA) header, allowing easy identification of data fragments. Yet, the only possible operation remains the aggregation, there is no mechanism to signal a different operation.

SHArP [SHArP], uses as well RDMA and introduces as well a custom header to simplify in-network handling of the packets. Similarly to NetReduce, SHArP remains a solution targeting only the aggregation function, relying on a rigid tree topology and proposing a header that allows only aggregation function and no other operation, hence, like NetReduce, hard to be converted for other purposes.

3.2. NetDistributedLock

In the majority of distributed system, the lock primitive is a widely used concurrency control mechanism. For large distributed systems, there is usually a dedicated lock manager that nodes contact to gain read and/or write permissions of a resource. The lock manager is often abstracted as Compare And Swap (CAS) or Fetch Add (FA) operations.

The lock manager is typically running on a server, causing a limitation on the performance by the speed of disk I/O transaction. When the load increases, for instance in the case of database transactions processed on a single node, the lock manager becomes a major performance bottleneck, consuming nearly 75% of transaction time [OLTP]. The multi-node distributed lock processing superimposes the communication latency between nodes, which makes the performance

even worse. Therefore offloading the lock manager function from the server to the network switch might be a better choice, since the switch is capable of managing lock function efficiently. Meanwhile it liberate the server for other computation tasks.

The test results in NetLock [NetLock] show that the lock manager running on a switch is able to answer 100 million requests per second, nearly 10 times more than what a lock server can do.

3.3. NetSequencer

Transaction managers are centralized solutions to guarantee consistency for distributed transactions, such as GTM in Postgre-XL ([GTM], [CALVIN]). However, as a centralized module, transaction managers have become a bottleneck in large scale high-performance distributed systems. The work by Kalia et al. [HPRDMA] introduces a server based networked sequencer, which is a kind of task manager assigning monotonically increasing sequence number for transactions. In [HPRDMA], the authors shows that the maximum throughput is 122 Million requests per second (Mrps), at the cost of an increased average latency. This bounded throughput will impact the scalability of distributed systems. The authors also test the bottleneck for varies optimization methods, including CPU, DMA bandwidth and PCIe RTT, which is introduced by the CPU centric architecture.

For a programmable switch, a sequencer is a rather simple operation, while the pipeline architecture can avoid bottlenecks. It is worth implementing a switch-based sequencer, which sets the performance goal as hundreds of Mrps and latency in the order of microseconds.

4. In-Network Generic Operations

The COIN use case draft [I-D.irtf-coinrg-use-cases] illustrates some general requirements for scenarios where the aforementioned use cases belong to. One of the requirements is that any in-network computing system must provide means to specify the constraints for placing execution logic in certain logical execution points (and their associated physical locations). In case of NetReduce, NetDistributedLock, and NetSequencer, data aggregation, lock management and sequence number generation functions can be offloaded onto the in-network device. It can be observed that those functions are based on "simple" and "generic" operators, as shown in Table 1. Programmable switches are capable of performing basic operations by executing one or more operators, without impacting the forwarding performance ([NetChain], [ERIS]).

Use Case	Operation	Description
NetReduce	Sum value (SUM)	The in-network device sums the data together and outputs the resulting value.
NetLock	Compare And Swap or Fetch-and-Add (CAS or FA)	By comparing the request with the status of its own lock, the in-network device sends out whether the host has the acquired the lock. Through the CAS and FA, host can implement shared and exclusive locks.
NetSequencer	Fetch-and-Add (FA)	The in-network device provides a monotonically increasing counter number for the host.

Table 1: Example of in-network operations.

5. SINC Framework Overview

This section describes the various elements of the SINC framework and explains how they work together.

The SINC protocol and extensions are designed for deployment in limited domains, such as a data center network and/or a storage network, rather than deployment across the open Internet. The requirements and semantics are specifically limited, as defined in the previous sections.

Figure 1 shows the overall SINC framework, consisting of Hosts, the SINC Ingress Proxy, SINC switch/router (SW/R), the SINC Egress Proxy and normal switches/routers(if any).

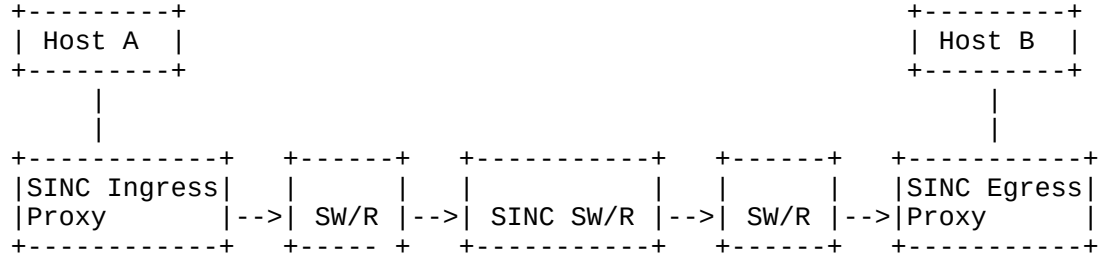

Figure 1: General SINC deployment.

In the SINC domain, a host MUST be SINC-aware. It defines the data operation to be executed. However, it does not need to be aware of where the operation will be executed and how the traffic will be steered in the network. The host sends out packets with a SINC header containing the definition and parameters of data operations. The SINC header could be placed directly after the transport layer, before the computing data as part of the payload. However, the SINC header can also potentially be positioned at layer 4, layer 3, or even layer 2, depending on the network context of the applications and the deployment consideration. This will be discussed in further details in [I-D.zhou-rtgwg-sinc-deployment-considerations].

The SINC proxies are responsible for encapsulating/decapsulating packets in order to steer them through the right network path and nodes. The SINC proxies may or may not be collocated with hosts.

The SINC Ingress Proxy encapsulates and forwards packets containing a SINC header, to the right node(s) with SINC operation capabilities. Such an operation may involve the use of protocols like Service Function Chaining (SFC [RFC7665]), LISP [RFC9300], Geneve [RFC8926], or even MPLS [RFC3031]. Based on the definition of the required data processing and the network capabilities, the SINC ingress proxy can determine whether the data processing defined in the SINC header could be executed in a single node or in multiple nodes. The SINC Egress Proxy is responsible for decapsulating packets before forwarding them to the destination host.

The SINC switch/router is the node equipped with in-network computing capabilities. It MUST look for the SINC header and perform the required operations if any. It could be done from the encapsulation protocols that contain a field of "next protocols". Otherwise, the SINC switch/router should be able to perform a deep packet inspection to identify the location of the SINC header. The detection of the location of the SINC header will be further depicted in [I-D.zhou-rtgwg-sinc-deployment-considerations]. Upon receiving a SINC packet, the SINC switch/router data-plane processes the SINC header, executes required operations, updates the payload with results (if necessary) and forwards the packet to the destination.

The SINC workflow is as follows:

1. Host A transmits a packet with the SINC header and data to the SINC Ingress proxy.

2. The SINC Ingress proxy encapsulates and forwards the original packet to a SINC switch/router(s).

3. The SINC switches/routers verifies the source, checks the integrity of the data and performs the required data processing defined in the SINC header. When the computing is done, if necessary, the payload is updated with the result and then forwarded to the SINC Egress proxy.

4. When the packet reaches the SINC Egress Proxy, the encapsulation will be removed and the inner packet will be forwarded to the final destination (Host B).

6. Data Operation Mode

According to the SINC scenarios, the SINC processing can be divided into two modes: individual computing mode and batch computing mode.

Individual operations include all operations that can be performed on data coming from a single packet (e.g., Netlock). Conversely, batch operations include all operations that require to collect data from multiple packets (e.g., NetReduce data aggregation).

6.1. Individual Computing Mode

The NetLock is a typical scenario involving individual operations, where the SINC switch/router acts as a lock server, generating a lock for a packet coming from one host.

This kind of operation has some general aspects to be considered:

* Initialization of the context on the computing device. The context is the information necessary to perform operations on the packets. For instance, the context for a lock operation includes selected keys, lock states (values) for granting locks.

* Error conditions. Operations may fail and, as a consequence, sometimes actions need to be taken, e.g. sending a message to the source host. Another option is to forward the packet unchanged to the destination host. The destination host will in this case perform the operation. If the operation fails again, the destination host will handle the error condition and may send a message back to the source host. In this way SINC switches/router operation remains relatively simple.

6.2. Batch Computing Mode

The batch operations require to collect data from multiple sources before actually being able to perform the required operations. For instance, in the NetReduce scenario, the gradient aggregation requires packets carrying gradient arrays from each host to generate the desired result array.

In this mode, the data operation is collective. The data coming from multiple sources may be aggregated in multiple aggregation nodes in a hierarchy. Hence a tree topology should be created from the control plane for each batch computing request, which will be dismissed once the batch computing is done. A message is required to signal the start and the end of the operation.

Each aggregation node executes the calculation when data is arrived. If some of packets do not arrive or arrive too late, the batch computing may fail. The time the packets are temporarily cached on the SINC switch/router should be carefully configured. On the one hand, it has to be sufficiently long to receive all required packets. On the other hand, it has to be sufficiently short so that no retransmissions are triggered at the transport or application layers on the end hosts. Similarly to the error condition for the individual operations, if the SINC switch/router does not receive all required packets in the configured time interval, it can either signal an error message back to the source host, or simply forward the packets to the end host that handles the packet losses.

7. SINC Header

The SINC header carries the data operation information and it has a fixed length of 16 octets, as shown in Figure 2.

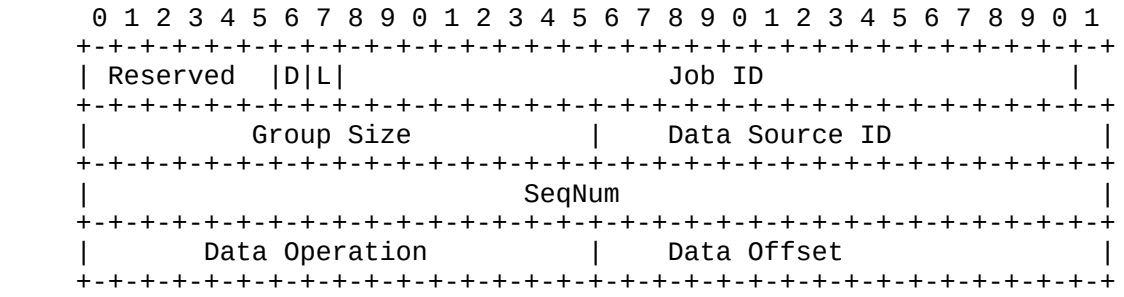

Figure 2: SINC Header.

* Reserved: Flags field reserved for future use. MUST be set to zero on transmission and MUST be ignored on reception.

* Done flag (D): Zero (0) indicates that the request operation is not yet performed. One (1) indicates the operation has been done. The source host MUST set this bit to 0. The in-network switch/ router performing the operation MUST set this flag to 1 after the operation is executed.

* Loopback flag (L): Zero (0) indicates that the packet SHOULD be sent to the destination after the data operation. One (1) indicates that the packet SHOULD be sent back to the source node after the data operation.

* Job ID: The Job ID identifies different groups. Each group is associated with one task.

* Group Size: Total number of data source nodes that are part of the group.

* Data Source ID: Unique identifier of the data source node of the packet.

* Sequence Number (SeqNum): The SeqNum is used to identify different requests within one group.

* Data Operation: The operation to be executed by the SINC switch/ router.

* Data Offset: The in-packet offset from the SINC header to the data required by the operation. This field is useful in cases where the data is not right after the SINC header, the offset indicates directly where, in the packet, the data is located.

8. Control Plane Considerations

The SINC control plane is responsible for the creation and configuration of the computing network topology with SINC capable network elements, as well as the monitoring and management of the system, to ensure the proper execution of the computing task. The SINC framework can work with either centralized (e.g. SDN like), distributed (by utilizing dynamic signaling protocols), or hybrid control planes. However, this document does not assume any specific control plane design.

A computing network topology needs to be created in advance to support the required in network computing tasks. The topology could be as simple as a explicit path with SINC capable nodes for individual computing mode (e.g. NetLock and NetSequencer), as well as a logical tree topology supporting more complex batch computing mode (e.g. NetReduce). After the completion of the computing task, the control plane needs to delete the topology and release relevant resources accordingly for that task or job.

The following features are necessary in control plane for the topology creation/deletion in the SINC network:

* Topology computation: When receiving the computing request from the Host, the SINC control plane needs to compute a set of feasible paths with SINC capable nodes to support individual or batch computations.

* Topology establishment: The topology has to be sent/configured/ signaled to the network device, so SINC packets could pass through the right SINC capable nodes to perform the required data computing in the network. Once done, the control plane will signal the application to kick off the packet transmission.

* Topology deletion: Once the application finishes the action, it will inform the control plane to delete the topology and release the reserved resources for other applications and purposes.

Distributed control plane signaling can be useful in creating the logical in-network computing topology with some "in-band" style. Consider a relatively complex case that the logical computing topology is a multilevel tree. If the centralized controlling manager is used, the tree needs to be established before the end points send the data to be aggregated. Distributed signaling can make the in-band tree creation coupled with the pull-based data aggregation. Each participating end point sends the in-band signal to create the tree. The SINC capable switch receiving them reserves the resource and aggregates such signals. Information from the end points carried in the signaling packet helps the switch determine whether it is the root. Such information can be the total group member counter and the sibling member counter. When the accumulated sibling member counter has the same value as the total gorup member counter, the switch considers it the root. When the root receives the signals from all the participating members, it then starts to pull the data from the end points. Distributed signaling directly initiates the data aggregation procedures without waiting for the control plane setting every ready in advance so that it improves the reusability with the more dynamic way in topology creation and deletion.

SINC packets are supposed to pass SINC capable nodes without traffic and computing congestion, which demands sufficient resource reservation. There are multiple types of resources (e.g., computing resource, buffer resource, and bandwidth resource) in the network that should be reserved to ensure the smooth execution of the computing tasks.

The performance monitoring (PM) is required to detect any potential issues during the data operation. It could be done actively or passively. By injecting OAM packets into the network to estimate the performance of the network, the active PM might affect the overall performance of the network. SINC does not introduces any constrains and pre-existing monitoring infrastructures can continue to be used.

The service protection contains two parts: the computing service protection and network service protection.

* The in-network computing service must be protected. If a SINC node of an in-network operation fails, the impact should be minimized by guaranteeing as much as possible that the packets are at least delivered to the end node, which will perform the requested operation (cf. Section 6). The control plane will take care to recover the failure, possibly using a different SINC node and re-routing the traffic.

* The network service must be able to deliver packet to the designated SINC nodes even in case of partial network failures (e.g. link failures). To this end existing protection and re-route solution may be applied.

From the above discussion about the control plane, the following basic requirements can be identified:

* The ability to exchange computing requirements (e.g. computing tasks, performance, bandwidth, etc.) and execution status with the application (e.g., via a User Network Interface). SINC tasks should be carefully coordinated with (other) host tasks.

* The ability to gather the resources available on SINC-capable devices, which demands regular advertisement of node capabilities and link resources to other network nodes or to network controller(s).

* The ability to dynamically create, modify and delete computing network topologies based on application requests and according to defined constraints. It includes, but it is not limited to, topology creation/update, explicit path determination, link bandwidth reservation and node computing resource reservation.

The created topology should be able to execute computing task requested by the application with no (ot low) impact on the packet transmission.

* The ability to monitor the performance of SINC nodes and link status to ensure that they meet the requirements.

* The ability to provide failover mechanism in order to handle errors and failures, and improves the resilience of the system. A fallback mechanism is required in case that in-network resources are not sufficient for processing SINC tasks, in which case, end host might provide some complementary computing capabilities.

9. Security Considerations

In-network computing exposes computing data to network devices, which inevitably raises security and privacy considerations. The security problems faced by in-network computing include, but are not limited to:

* Trustworthiness of participating devices

* Data hijacking and tampering

* Private data exposure

This document assumes that the deployment is done in a trusted environment. For example, in a data center network or a private network.

A detailed security analysis will be provided in future revisions of this memo.

10. IANA Considerations

This document makes no requests to IANA.

Acknowledgements

This document received contribution from Yujing Zhou as well as valuable feedback from Dirk Trossen, which was of great help in improving the content. The authors would like to thank all of them for their contributions.

References

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/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Informative References

[CALVIN] Thomson, A., Diamond, T., Weng, S., Ren, K., Shao, P., and D. Abadi, "Calvin: fast distributed transactions for partitioned database systems", ACM, Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, DOI 10.1145/2213836.2213838, May 2012, <https://doi.org/10.1145/2213836.2213838>.

[COIN] "Computing in the Network, COIN, proposed IRTF group", n.d., <https://datatracker.ietf.org/rg/coinrg/about/>.

[ERIS] Li, J., Michael, E., and D. R. K. Ports, "Eris:/ Coordination-Free Consistent Transactions Using In-Network Concurrency Control", SOSP ’17:/ Proceedings of the 26th Symposium on Operating Systems Principles , 2017.

[GOBATTO] Reinehr Gobatto, L., Rodrigues, P., Tirone, M., Cordeiro, W., and J. Azambuja, "Programmable Data Planes meets In-Network Computing: A Review of the State of the Art and Prospective Directions", Journal of Integrated Circuits and Systems, Journal of Integrated Circuits and Systems vol. 16, no. 2, pp. 1-8, DOI 10.29292/jics.v16i2.497, August 2021, <https://doi.org/10.29292/jics.v16i2.497>.

[GTM] "GTM and Global Transaction Management", n.d., <https://www.postgres-xl.org/documentation/xc-overview-gtm.html>.

[HPRDMA] Kalia, A., Kaminsky, M., and D. G. Andersen, "Design Guidelines for High Performance RDMA Systems", 2016 USENIX Annual Technical Conference (USENIX ATC 16) , 2016, <https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia>.

[I-D.irtf-coinrg-use-cases] Kunze, I., Wehrle, K., Trossen, D., Montpetit, M., de Foy, X., Griffin, D., and M. Rio, "Use Cases for In-Network

Computing", Work in Progress, Internet-Draft, draft-irtf-coinrg-use-cases-07, 4 December 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-coinrg-use-cases-07>.

[I-D.zhou-rtgwg-sinc-deployment-considerations]
Lou, D., Iannone, L., Zhou, Y., Yang, J., and Zhangcuimin, "Signaling In-Network Computing operations (SINC) deployment considerations", Work in Progress, Internet-Draft, draft-zhou-rtgwg-sinc-deployment-considerations-00, 23 February 2023, <https://datatracker.ietf.org/doc/html/draft-zhou-rtgwg-sinc-deployment-considerations-00>.

[MGWFBP] Shi, S., Chu, X., and B. Li, "MG-WFBP:/ Efficient data communication for distributed synchronous SGD algorithms", IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE , 2019.

[NetChain] Jin, X., Li, X., and H. Zhang, "NetChain:/ Scale-free sub-RTT coordination", 2018.

[NetLock] Z, Y., Y, Z., and B. V, "Netlock:/ Fast, centralized lock management using programmable switches", Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. , 2020.

[NetReduce]
Liu, S., Wang, Q., Zhang, J., Wu, W., Lin, Q., Liu, Y., Xu, M., Canini, M., Cheung, R., and J. He, "In-Network Aggregation with Transport Transparency for Distributed Training", ACM, Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 pp. 376-391, DOI 10.1145/3582016.3582037, March 2023, <https://doi.org/10.1145/3582016.3582037>.

[OLTP] R, J., I, P., and A. A, "Improving OLTP scalability using speculative lock inheritance", Proceedings of the VLDB Endowment , 2009.

[OPENAI] "OpenAI. AI and compute", 2018, <https://openai.com/blog/ai-and-compute/>.

[P4] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and D. Walker, "P4: programming protocol-

independent packet processors", Association for Computing Machinery (ACM), ACM SIGCOMM Computer Communication Review vol. 44, no. 3, pp. 87-95, DOI 10.1145/2656877.2656890, July 2014, <https://doi.org/10.1145/2656877.2656890>.

[PARAHUB] L, L., J, N., and C. L, "Parameter hub:/ a rack-scale parameter server for distributed deep neural network training", Proceedings of the ACM Symposium on Cloud Computing. , 2018.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/rfc/rfc3031>.

[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/rfc/rfc7665>.

[RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed., "Geneve: Generic Network Virtualization Encapsulation", RFC 8926, DOI 10.17487/RFC8926, November 2020, <https://www.rfc-editor.org/rfc/rfc8926>.

[RFC9300] Farinacci, D., Fuller, V., Meyer, D., Lewis, D., and A. Cabellos, Ed., "The Locator/ID Separation Protocol (LISP)", RFC 9300, DOI 10.17487/RFC9300, October 2022, <https://www.rfc-editor.org/rfc/rfc9300>.

[ROCEv2] "InfiniBand Architecture Specification Release 1.2.1 Annex A17 RoCEv2", InfiniBand Trade Association , September 2014, <https://cw.infinibandta.org/document/dl/7781>.

[SHArP] Graham, R., Koushnir, V., Levi, L., Margolin, A., Ronen, T., Shpiner, A., Wertheim, O., Zahavi, E., Bureddy, D., Lui, P., Rosenstock, H., Shainer, G., Bloch, G., Goldenerg, D., Dubman, M., and S. Kotchubievsky, "Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction", IEEE, 2016 First International Workshop on Communication Optimizations in HPC (COMHPC) pp. 1-10, DOI 10.1109/comhpc.2016.006, November 2016, <https://doi.org/10.1109/comhpc.2016.006>.

[SwitchML] Sapio, A., Canini, M., Ho, C., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D., and P. Richt叩rik, "Scaling Distributed Machine Learning with In-Network Aggregation", arXiv, DOI 10.48550/ARXIV.1903.06701, 2019, <https://doi.org/10.48550/ARXIV.1903.06701>.

[ZENG] Zeng, D., Ansari, N., Montpetit, M., Schooler, E., and D. Tarchi, "Guest Editorial: In-Network Computing: Emerging Trends for the Edge-Cloud Continuum", Institute of Electrical and Electronics Engineers (IEEE), IEEE Network vol. 35, no. 5, pp. 12-13, DOI 10.1109/mnet.2021.9606835, September 2021, <https://doi.org/10.1109/mnet.2021.9606835>.

Contributors

Jinze Yang
China
Email: jz.yang@live.com

Authors' Addresses

Zhe Lou
Huawei Technologies
Riesstrasse 25
80992 Munich
Germany
Email: zhe.lou@huawei.com

Luigi Iannone
Huawei Technologies France S.A.S.U.
18, Quai du Point du Jour
92100 Boulogne-Billancourt
France
Email: luigi.iannone@huawei.com

Yizhou Li
Huawei Technologies
Nanjing
China
Email: liyizhou@huawei.com

Cuimin Zhang
Huawei Technologies
Huawei base in Bantian, Longgang District
Shenzhen
China
Email: zhangcuimin@huawei.com

Kehan Yao
China Mobile
China
Email: yaokehan@chinamobile.com