

COINRG
Internet-Draft
Intended status: Informational
Expires: 28 April 2022

I. Kunze
K. Wehrle
RWTH Aachen University
D. Trossen
Huawei
25 October 2021

Transport Protocol Issues of In-Network Computing Systems
draft-kunze-coinrg-transport-issues-05

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.

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 28 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Technology Areas	3
3.1. Addressing	4
3.1.1. Research questions and challenges	4
3.1.2. Related concepts and efforts	5
3.2. Flow granularity	6
3.2.1. Research questions and challenges	7
3.2.2. Related concepts and efforts	8
3.3. Collective Communication	8
3.3.1. Research questions and challenges	9
3.3.2. Related concepts and efforts	9
3.4. Authentication	9
3.4.1. Research questions and challenges	10
3.4.2. Related concepts and efforts	10
3.5. Security	10
3.5.1. Research questions and challenges	10
3.5.2. Related concepts and efforts	10
3.6. Transport Features	11
3.6.1. Reliability	11
3.6.2. Flow/Congestion Control	14
4. Summary of related research and standardization efforts . . .	15
5. Gap Analysis	16
5.1. Addressing	17
5.2. Flow granularity	17
5.3. Collective Communication	17
5.4. Authentication	17
5.5. Security	17
5.6. Transport Features	17
5.6.1. Reliability	17
5.6.2. Flow/Congestion Control	17
6. Summary of Issues Identified	17
7. Security Considerations	17
8. IANA Considerations	18
9. Conclusion	18
10. Informative References	18
Authors' Addresses	22

1. Introduction

A fundamental consideration for the Internet's design is that functions can be implemented correctly and completely only with the knowledge of the applications, as formulated in [E2E]. This choice is reflected in the end-to-end (E2E) principle [RFC1958],[RFC2775] in that end-hosts perform most, if not all, relevant computations. The network only performs transparent, reasonable operations such as delivering the packets without modifying them with transport protocols designed to facilitate the direct communication between those end-hosts.

[E2E], however, does consider that "sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement". We link this consideration to the field of computing in the network (COIN), which encourages explicit computations in the network, introducing an intertwined complexity as the computations on the end-hosts depend on the functionality deployed in the network.

Such thinking, to some extent, 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, even for inherent functionalities provided by the transport protocol. Some of the resulting problems when considering in-network computation in the context of an overall E2E problem are already presented in [I-D.draft-kutscher-coinrg-dir-02].

This draft focusses on the potential opportunities and research questions for the design of transport protocols that may assume the availability of in-network computing capabilities, including the possible collaboration with other IRTF and IETF groups, such as PAN RG, the IETF transport area in general, or the LOOPS BOF, for finding suitable solutions. In particular, the draft first describes how different aspects of transport protocols might be affected by in-network computing functionality before it is analyzed how existing transport protocols map to the identified questions and challenges.

2. Terminology

COIN element: Device on which COIN functionality can be deployed

3. Technology Areas

3.1. 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. From an identification perspective, addressing may not only need to specify the set of functionality that is being desired but also enable to provide affinity to a member of the set of computational nodes that provide said functionality.

For instance, orchestration functionality may 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-irtf-coinrg-use-cases]. 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.

3.1.1. Research questions and challenges

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 (and encode the decision) 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?

3.1.2. Related concepts and efforts

- * Segment and Source Routing (see [SPRING-WG])

Source Routing allows a sender to (partially) define the route of a packet through the network. This mechanism can be leveraged to steer the traffic along COIN nodes and thus trigger desired COIN functionality. The SPRING WG is scoped to define procedures around Segment Routing [SR], a modern variant of Source Routing for IPv6 and MPLS.

- * (Service/Network) Function Chaining/Composition (see [SFC-WG])

Service Function Chaining (SFC) describes a process to first define an ordered list of service functions (e.g., firewalls) and then steer traffic through these functions [SFC-PS]. The SFC WG is tasked with defining suitable orchestration techniques for SFC. The existing SFC architecture [SFC-Arch] and the Network Service Header [SFC-NSH] already provide fundamental mechanisms. Interpreting COIN functionality as service functions could make SFC applicable to COIN at Layer 2 and Layer 3, but also at name-based, e.g., HTTP level [RFC8677]

- * Internet services over ICN (see [ICNIP])

Work in the ICN RG [ICNRG] has generally studied the addressing of information rather than endpoints, opening up the possibility for providing information from different sources, including in-network elements, such as for caching purposes. The work in [ICNIP] utilizes the ICN capabilities to address services directly as a named entity, including IP endpoints, in order to support concepts like virtualization of service endpoints and provisioning within edge and in-network locations. The solution in [ICNIP] proposes the use of a Layer 2 path-based forwarding with service identifiers used to address the specific service endpoint.

- * Flexible Addressing (see [I-D.draft-jia-intarea-scenarios-problems-addressing])

Although the work in the INT Area of the IETF does not postulate a specific solution, it outlines a number of communication scenarios and challenges, some of which aligns with those outlined above. The companion draft [I-D.draft-jia-intarea-internet-addressing-gap-analysis] provides a

deeper gap analysis of existing solutions (the above mentioned solutions are presented here, too), identifying a number of issues that arise from the specific point solutions realized by those solutions. The authors argue for both flexibility and extensibility of addressing; key aspects that any solution addressing the research questions outlined above would benefit from.

* Semantic Routing (see
[I-D.draft-king-irtf-semantic-routing-survey])

The survey at [I-D.draft-king-irtf-semantic-routing-survey] provides an overview of efforts on addressing and routing that incorporate a semantic beyond the one defined by IPv6, covering both existing IETF solutions as well as ongoing research, defined as 'semantic routing' in the draft. The companion draft at [I-D.draft-king-irtf-challenges-in-routing] outlines a number of challenges that exist for such extension of the addressing semantic, some of which align with the issues identified in this document. More importantly, the draft discusses the possible deployment of semantic routing solutions, e.g., as an overlay or limited to a single domain (following the Limited Domain concept of [RFC8799]). Some of the challenges identified in [I-D.draft-king-irtf-challenges-in-routing] apply to a COIN environment, while not being limited to it. For instance, the intended scope of any enhanced addressing (e.g., identifying COIN elements on-path in a scenario) or the description of path characteristics that COIN traffic would need to adhere to.

3.2. 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 maps to the capabilities of existing 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.

3.2.1. Research questions and challenges

1. Which flow granularities are sensible for which scenarios and upper layer protocols?
2. How do the different flow granularities map to error and congestion control?
3. How is flow granularity used for creating affinity in, e.g., routing choices?
4. How may flow granularity information be used in COIN elements, e.g., to support routing and transport protocol realizations?

3.2.2. Related concepts and efforts

As mentioned above, flow granularities are defined in transport protocols through their semantic for the unit of transfer, which can be a 'datagram' or a 'flow'. Upper layer protocols, such as HTTP, map their application data into this semantic, resulting, for instance, in a flow of HTTP requests. Note that the flow identified by the 5-tuple for the transport connection usually also carries the reverse direction of communication, e.g., in the form of HTTP responses. The introduction of 'TCP re-use' in HTTP/1.1 introduced the capability of sending many HTTP request/response interactions in a single TCP flow. The notion of flow granularity is being used in [DYNCAST] to link the relation of one or more application level interactions to a specific service instance in deployment scenarios where more than one service instance may serve requests for a given service; [DYNCAST] refers to the problem of 'instance affinity', i.e., the need to send one or more such interactions to the same instance before being able to choose another instance (e.g., based on computing or network metrics, as suggested in [DYNCAST]). At this point of the work, the potential realization of such 'instance affinity' and the relation to transport (as well as application) protocols has not been discussed yet.

Within the concept of Service Function Chaining (SFC) [SFC-Arch], a chain of services is formed and expressed through the next service header (NSH) [SFC-NSH], which provides entries into a next hop table maintained at each Service Function Forwarder (SFF) [SFC-Arch]. Packet classification takes place at the entry point of the chain, therefore providing a notion of flow granularity where the chain is treated as the 'unit of transfer'. Chaining can take place at Layer 2 or Layer 3, but also at a name-based layer (such as HTTP), as proposed in [RFC8677].

3.3. 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-04]. With this, unicast and multicast transmissions become almost equal forms of communication, as also observed in work on information-centric networking (ICN) [ICNRG].

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.

3.3.1. Research questions and challenges

1. How to handle ephemeral transport relations at the request level across more than one endpoint?
2. How to separate longer-term resource management from shorter-term transaction handling for, e.g., error and flow control?
3. What role could COIN elements play in improving on solutions for questions 1 and 2?

3.3.2. Related concepts and efforts

As stated above, work in the [ICNRG] has long considered multicast and unicast delivery as two communication models, realized by the same communication method that utilizes the interest-data model of ICN. The work in [ICNIP] utilizes a different approach by relying on path-based forwarding of packets identified through service-level identifiers (such as URLs but also IP addresses), where return path multicast is achieved through binary operations over the path information of incoming service requests. The utilized transport network technology is that of 5GLAN or SDN, where the latter uses an OpenFlow-compatible approach to path-based forwarding with constant state requirements for the in-network forwarders. A similar approach is used in [BIER-MC] albeit at the level of a BIER overlay network. [ICNIP] also discusses, albeit briefly only, the separation of longer-lived resource management from shorter-lived transaction handling to increase efficiency of the ephemeral return path communication at the transport level.

3.4. 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 3.1) and subsequently authenticated.

3.4.1. Research questions and challenges

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?

3.4.2. Related concepts and efforts

* Proof of Transit [SFC-PoT]

The Proof of Transit concept of the SFC WG allows for proving that packets have passed a defined path. Using this concept, it could at least be possible to make sure that a packet has indeed passed the desired COIN elements. However, it does not provide means to validate which changes were made by the known nodes.

3.5. 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.

3.5.1. Research questions and challenges

To be added.

3.5.2. Related concepts and efforts

To be added.

3.6. 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.

3.6.1. Reliability

Applications require a reliable transport whenever it is important that all data is transmitted successfully. TCP[TCP] 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[UDP] 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 3.2), COIN elements might have to hold contextual information for a prolonged time once they identify an impending 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 3.2, might make other approaches to reliability viable that operate on the basis of such messages.

3.6.1.1. Research questions and challenges

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?

3.6.1.2. Related concepts and efforts

* Transmission Control Protocol [TCP]

TCP provides reliable, ordered, and error-checked delivery of a byte stream. As such, TCP does not allow for payload changes. This means that COIN elements could only make changes to lower header information.

* Stream Control Transmission Protocol [SCTP]

SCTP provides ensures a reliable exchange of messages. In contrast to TCP, it decouples reliability from in-order delivery and thus allows for sending messages without ordering. Additionally, it has also been extended to provide partial reliability, i.e., controlling the desired reliability on a per-message basis [SCTP-PR].

* Constrained Application Protocol [CoAP]

CoAP is a specialized protocol targeting nodes that are constrained, e.g., in terms of compute power or available bandwidth. It is message-based and distinguishes between confirmable and non-confirmable messages, i.e., similar to SCTP, allows for controlling the reliability on a per-message basis.

* User Datagram Protocol [UDP]

UDP is a message-based protocol that does not provide any guarantees regarding reliability to the application layer.

3.6.2. 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 3.2. 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.

3.6.2.1. Research questions and challenges

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?

3.6.2.2. Related concepts and efforts

* Transmission Control Protocol [TCP]

TCP implements flow and congestion control. The traffic is controlled using TCP's receiver and congestion windows.

* Separation of Data Path and Data Flow

[I-D.draft-asai-tsvwg-transport-review-02] proposes to explicitly divide transport protocols into two parts: a data path and a data flow layer. Essentially, the data path layer is responsible for handling path-related tasks, such as congestion control, and as such spans multiple flows. The data flow layer on top then handles flow-related tasks such as retransmissions and flow control. As indicated by the early stage of the document, the concrete structure is still up for debate. Yet, explicitly dividing congestion and flow control could give the opportunity to devise more sophisticated approaches to incorporate COIN elements.

4. Summary of related research and standardization efforts

Issue	Efforts
Addressing	Segment and Source Routing: <ul style="list-style-type: none"> - [SPRING-WG] - Segment Routing [SR] (Service/Network) Function Chaining/Composition: <ul style="list-style-type: none"> - [SFC-WG] - SFC Problem Statement [SFC-PS] - SFC Architecture [SFC-Arch] - SFC Network Service Header [SFC-NSH] - Internet Services over IP [ICNIP]
Flow Granularity	Service Function Chaining [SFC-Arch], [SFC-NSH] Use cases and problem statement for dynamic anycast [DYNCAST]
Collective Communication	Information-centric networking [ICNRG] Internet Services over IP [ICNIP] HTTP multicast over BIER [BIER-MC]
Authentication	SFC Proof of Transit [SFC-PoT]
Reliability	Transmission Control Protocol [TCP] Stream Control Transmission Protocol [SCTP] Constrained Application Protocol [CoAP] User Datagram Protocol [UDP]
Flow/Congestion Control	[I-D.draft-asai-tsvwg-transport-review-02]

Figure 1: Related research and standardization efforts.

5. Gap Analysis

This section provides a gap analysis within the identified technology areas with respect to existing IETF solutions and ongoing efforts in transport protocols that were summarized in the previous section.

The goal of such analysis is to identify issues with those existing solutions through and within COIN environments. From the viewpoint of structuring the gap analysis, approaches such as those taken in [I-D.draft-jia-intarea-internet-addressing-gap-analysis] may be used as examples as well as direct (if suitable) input into the gap analysis performed here.

5.1. Addressing

TBD

5.2. Flow granularity

TBD

5.3. Collective Communication

TBD

5.4. Authentication

TBD

5.5. Security

TBD

5.6. Transport Features

5.6.1. Reliability

TBD

5.6.2. Flow/Congestion Control

TBD

6. Summary of Issues Identified

This section will summarize the main issues across the investigated transport technology areas of the previous section.

7. 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 without a consistent view on how in-network compute elements would fit into the traditional end-to-end model.

Additionally, COIN elements often do not support required cryptographic functionality.

Thus, there may be a need for new security concepts specific to COIN environment that may have to be developed to allow for a secure use within COIN environments.

8. IANA Considerations

N/A

9. Conclusion

The advent of COIN may bring many new use cases, as documented in [I-D.draft-irtf-coinrg-use-cases], with promises of improved solutions for various problems. The concept of in-network computing capabilities, however, is not directly compatible with the end-to-end nature of transport protocols, thereby posing a number of key questions regarding COIN and transport protocols.

Those key questions, positioned across key technology areas for transport protocols, lead us to look at possible gaps that may be found in existing solutions when it comes to suitably supporting COIN environments and scenarios. The gap analysis performed in this document and the issues identified as a result of this analysis are positioned as possible input into shaping a research agenda for new transport protocols that have in-network computing capabilities in mind and support them securely as well as the best performance possible.

10. Informative References

- [BIER-MC] Trossen, D., Rahman, A., Wang, C., and T. T. Eckert, "Applicability of BIER Multicast Overlay for Adaptive Streaming Services", Work in Progress, Internet-Draft, draft-ietf-bier-multicast-http-response-06, 10 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-bier-multicast-http-response-06.txt>>.
- [CoAP] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [DYNCAST] Liu, P., Willis, P., and D. Trossen, "Dynamic-Anycast (Dyncast) Use Cases and Problem Statement", Work in Progress, Internet-Draft, February 2021, <<https://datatracker.ietf.org/doc/draft-liu-dyncast-ps-usecases/>>.

- [E2E] Saltzer, J., Reed, D., and D. Clark, "End-to-end arguments in system design", ACM Transactions on Computer Systems Vol. 2, pp. 277-288, DOI 10.1145/357401.357402, November 1984, <<https://doi.org/10.1145/357401.357402>>.
- [I-D.draft-asai-tsvwg-transport-review-02]
Asai, H., "Separation of Data Path and Data Flow Sublayers in the Transport Layer", Work in Progress, Internet-Draft, draft-asai-tsvwg-transport-review-02, 15 September 2021, <<https://www.ietf.org/archive/id/draft-asai-tsvwg-transport-review-02.txt>>.
- [I-D.draft-irtf-coinrg-use-cases]
Kunze, I., Wehrle, K., Trossen, D., and M. Montpetit, "Use Cases for In-Network Computing", Work in Progress, Internet-Draft, draft-irtf-coinrg-use-cases-00, 17 February 2021, <<https://www.ietf.org/archive/id/draft-irtf-coinrg-use-cases-00.txt>>.
- [I-D.draft-jia-intarea-internet-addressing-gap-analysis]
Jia, Y., Trossen, D., Iannone, L., Shenoy, N., and P. Mendes, "Gap Analysis in Internet Addressing", Work in Progress, Internet-Draft, draft-jia-intarea-internet-addressing-gap-analysis-00, 12 July 2021, <<https://www.ietf.org/archive/id/draft-jia-intarea-internet-addressing-gap-analysis-00.txt>>.
- [I-D.draft-jia-intarea-scenarios-problems-addressing]
Jia, Y., Trossen, D., Iannone, L., Shenoy, N., Mendes, P., 3rd, D. E. E., and P. Liu, "Challenging Scenarios and Problems in Internet Addressing", Work in Progress, Internet-Draft, draft-jia-intarea-scenarios-problems-addressing-01, 12 July 2021, <<https://www.ietf.org/archive/id/draft-jia-intarea-scenarios-problems-addressing-01.txt>>.
- [I-D.draft-king-irtf-challenges-in-routing]
King, D. and A. Farrel, "Challenges for the Internet Routing Infrastructure Introduced by Changes in Address Semantics", Work in Progress, Internet-Draft, draft-king-irtf-challenges-in-routing-03, 14 June 2021, <<https://www.ietf.org/archive/id/draft-king-irtf-challenges-in-routing-03.txt>>.

- [I-D.draft-king-irtf-semantic-routing-survey]
King, D. and A. Farrel, "A Survey of Semantic Internet Routing Techniques", Work in Progress, Internet-Draft, draft-king-irtf-semantic-routing-survey-02, 28 June 2021, <<https://www.ietf.org/archive/id/draft-king-irtf-semantic-routing-survey-02.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-04]
Trossen, D., Sarathchandra, C., and M. Boniface, "In-Network Computing for App-Centric Micro-Services", Work in Progress, Internet-Draft, draft-sarathchandra-coin-appcentres-04, 26 January 2021, <<https://www.ietf.org/internet-drafts/draft-sarathchandra-coin-appcentres-04.txt>>.
- [ICNIP] Trossen, D., Robitzsch, S., Essex, U., AL-Naday, M., and J. Riihijarvi, "Internet Services over ICN in 5G LAN Environments", Work in Progress, Internet-Draft, draft-trossen-icnrg-internet-icn-5glan-04, 1 October 2020, <<http://www.ietf.org/internet-drafts/draft-trossen-icnrg-internet-icn-5glan-04.txt>>.
- [ICNRG] "Information-Centric Networking, IRTF Research Group", <<https://datatracker.ietf.org/rg/icnrg/about/>>.
- [RFC1958] Carpenter, B., Ed., "Architectural Principles of the Internet", RFC 1958, DOI 10.17487/RFC1958, June 1996, <<https://www.rfc-editor.org/info/rfc1958>>.
- [RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, DOI 10.17487/RFC2775, February 2000, <<https://www.rfc-editor.org/info/rfc2775>>.
- [RFC8677] Trossen, D., Purkayastha, D., and A. Rahman, "Name-Based Service Function Forwarder (nSFF) Component within a Service Function Chaining (SFC) Framework", RFC 8677, DOI 10.17487/RFC8677, November 2019, <<https://www.rfc-editor.org/info/rfc8677>>.

- [RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <<https://www.rfc-editor.org/info/rfc8799>>.
- [SCTP] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [SCTP-PR] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <<https://www.rfc-editor.org/info/rfc3758>>.
- [SFC-Arch] 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>>.
- [SFC-NSH] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <<https://www.rfc-editor.org/info/rfc8300>>.
- [SFC-PoT] 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, <<https://www.ietf.org/internet-drafts/draft-ietf-sfc-proof-of-transit-08.txt>>.
- [SFC-PS] Quinn, P., Ed. and T. Nadeau, Ed., "Problem Statement for Service Function Chaining", RFC 7498, DOI 10.17487/RFC7498, April 2015, <<https://www.rfc-editor.org/info/rfc7498>>.
- [SFC-WG] "Service Function Chaining, IETF Working Group", <<https://datatracker.ietf.org/wg/sfc/about/>>.
- [SPRING-WG] "Source Packet Routing in Networking, IETF Working Group", <<https://datatracker.ietf.org/wg/spring/about/>>.
- [SR] 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>>.

- [TCP] Postel, J., "Transmission Control Protocol", STD 7,
RFC 793, DOI 10.17487/RFC0793, September 1981,
<<https://www.rfc-editor.org/info/rfc793>>.
- [UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768,
DOI 10.17487/RFC0768, August 1980,
<<https://www.rfc-editor.org/info/rfc768>>.

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
Internet-Draft
Intended status: Experimental
Expires: February 1, 2021

D. Kutscher
University of Applied Sciences Emden/Leer
T. Kaerkkainen
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

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	4
3. Computing in the Network vs Networked Computing vs Packet Processing	4
3.1. Networked Computing	5
3.2. Packet Processing	5
3.3. Computing in the Network	6
3.4. Elements for Computing in the Network	9
4. Examples	11
4.1. Compute-First Networking with ICN	11
4.2. Akka Toolkit	12
5. Research Challenges	13
5.1. Categorization of Different Use Cases for Computing in the Network	13
5.2. Networking and Remote-Method-Invocation Abstractions	13
5.3. Transport Abstractions	14
5.4. Programming Abstractions	16
5.5. Security, Privacy, Trust Model	17
5.6. Coordination	18
5.7. Fault Tolerance, Failure Handling, Debugging, Management	18
6. Acknowledgements	19
7. ChangeLog	19
7.1. 02	19
7.2. 01	19
8. References	19
8.1. Informative References	19
8.2. URIs	21
Authors' Addresses	21

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 through virtual infrastructure management (VIM) systems. Besides such industry-driven activities, manifold research approaches to edge computing with varying granularity and orchestration approaches, among other differentiating 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

- o shortest paths between function instances; function off-loading
- o opportunities on topologically convenient next hops; and
- o 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:

- o low latency for certain function calls or compute threads;
- o high throughput for a pipeline of data processing functions;
- o high availability for an overall application/service;
- o load management (balancing, concentration) according to performance/cost constraints; and
- o consideration of security/privacy constraints with respect to platform selection and function execution?
- o 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:

- o 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.
- o They may be integrated with routers or other network operations infrastructure and thus be tightly integrated within the same physical device.
- o They may be integrated within switches, similar to the (limited) P4 compute capabilities offered today.
- o 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.
- o They might add novel types of dedicated hardware to execute certain functions more efficiently, e.g., GPU nodes for (distributed) analytics.
- o They might include low-end (embedded) devices such as microcontrollers that support decentralized computation at low cost and limited performance.
- o 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).

- o 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 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

- o need for additional resolution/mapping system to find IP addresses for functions;
- o possible overhead for establishing TCP connections for fine-granular function invocation;
- o defining and managing security properties of such connections and coping with the associated setup/validation overhead; and
- o 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

- o in-network caching (of data objects that represent function parameters or results);
- o reasoning about the tradeoffs between moving data to function vs. moving code to data assets; and
- o 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:

- o What is the role of network elements in forwarding RMI requests?
- o 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?

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

- o 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?
- o 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 heterogeneous, evolving set of execution environments brings about questions such as:

- o How to discover available platforms (and understand their properties)?
- o How to specify application needs and map them to available platforms?
- o 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:

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

- o Is there potential for fallback and version negotiation if needed (considering the risk of "bidding downs" attacks?)
- o How to retire old versions?
- o 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:

- o 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 forwards requests (and result messages) reliably and securely?
 - * How to ascertain that a function does what it claims to do?
- o What levels of authentication are needed for callers (assuming that not everybody can invoke any function)?
- o 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

- o 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?
- o 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 be 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:

- o What are strategies for retrying vs aborting function invocation?
- o 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:

- o What monitoring and instrumentation interfaces are needed?
- o 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

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

7.2. 01

- o added explanation of MEC and network slicing in Section 3.
- o added clarification that edge computing is not limited to MEC
- o added description of named service function chaining
- o 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] KrA³¹, 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 Kaerkkäeinen
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

RTGWG
INTERNET-DRAFT
Intended Status: Informational
Expires: May 7, 2020

Y. Li
J. He
Huawei Technologies
L. Geng
P. Liu
China Mobile
Y. Cui
Tsinghua University
November 4, 2019

Framework of Compute First Networking (CFN)
draft-li-rtgwg-cfn-framework-00

Abstract

Compute First Networking (CFN) leverages both computing and networking status to help determine the optimal edge among multiple edge sites with different geographic locations to serve a specific edge computing request. Requests for the same service can be determined and dispatched to different edges based on service requirements, network and computing resource conditions and other factors to achieve better load balancing and system efficiency. The request needs to be dispatched to the selected edge in real time and the subsequent packets from the same flow should be served by the same edge for flow affinity. This document describes a framework of CFN to achieve the desired features.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

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

The list of Internet-Draft Shadow Directories can be accessed at

<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. CFN Framework	4
2.1 CFN Service Overview	5
2.2 Generic Workflow	7
3. Control Plane and Data Plane	7
3.1 Control plane	7
3.2 Data plane	9
4. Summary	12
5. Security Considerations	12
6. IANA Considerations	12
7. Acknowledgements	12
8. References	13
8.1 Normative References	13
8.2 Informative References	13
Authors' Addresses	13

1. Introduction

Compute First Networking (CFN) scenarios and requirements document [CFN-req] shows the usage scenarios that require an edge to be dynamically selected from multiple edge sites with different geographic locations to serve an edge computing request based on computing resource consumption and network status in real time. For instance, edge site in residential area receives low request volume during working hours and high request volume during non-working hours. And the request volume received by the edge site in industrial park is the opposite. Such a pattern causes a big difference of computing load on different edge sites. Traditional static or hashing based service dispatch can not adapt to the unbalanced nature of computing load or rapid change of it on different edge sites. One edge such as the closest one to the client may have been overloaded and at the same time the other edges may still have plenty of computing resources to serve the requests. To efficiently leveraging the computing resources hosted on all edges, service requests should be dispatched and handled dynamically to make the computing and network resources consumed in a balanced way.

CFN assumes there are multiple service equivalent edges to serve a single service. A single edge has limited computing resources and different edges may have different resources available for serving a specific service at a specific time. In concept, multiple edges are interconnected and collaborated with each other to balance the service load in CFN. Computing resource available to serve a request is the top metric to be considered when dispatching a request. At the same time, the quality of the network path to an edge varies over time. CFN is a network based approach so that the request is dispatched to the optimal edge in terms of both computing resources available and network status on the fly.

This document presents a CFN framework which can support service equivalency and dynamics in edge computing to achieve better load balancing with no application dependency.

1.1 Terminology

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] [RFC8174] when, and only when, they appear in all capitals, as shown here.

CFN: Computing First Networking

2. CFN Framework

Edge computing is expanding from a single edge site to networked and collaborated multiple edge sites to solve the issues of low efficiency and low resource reuse. CFN enables large scale edge interconnection and collaboration, providing optimal service access and load balancing to adapt to service dynamics. Based on the real-time computing capacity available and the network conditions, CFN dynamically schedules computing request to appropriate service node, thus the resource utilization and user experience is improved.

Figure 1 shows the network topology of CFN. CFN node is the basic function entity in CFN network to provide the capability to exchange the information about the computing resource consumption information of service nodes attached to it and/or provide the CFN service access to the clients. Edge site (edge for short) is normally the site where the edge computing is hosted. CFN node can be a network virtual function (NFV) co-located with service node in a server. CFN node's function can also be provided by physical equipment like access router in access ring or metro network.

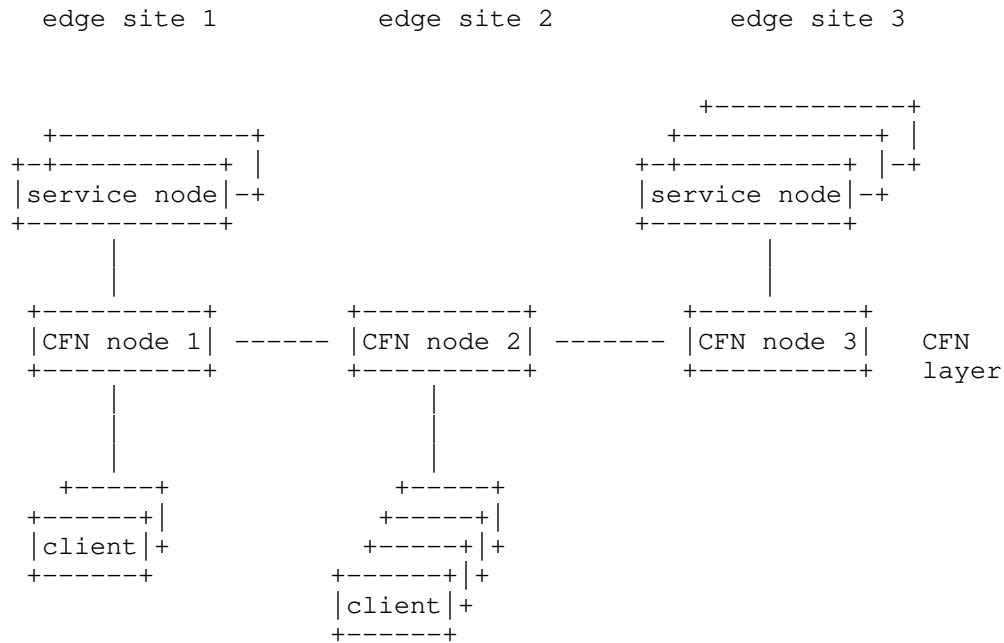


Figure 1. CFN Network Topology

2.1 CFN Service Overview

CFN uses Service ID (SID) to identify a particular service provided by service nodes on multiple edges. The end devices always use SID to initiate an access to a service. SID in current system is an anycast address. Request to a single SID can potentially be served by different edge sites. The end device does not know in advance which edge to serve the request. The procedures to make such determination is called the service dispatch. During service dispatch, the most appropriate edge site (i.e. CFN egress) is selected and it is the edge to which the service node that handles this specific request is attached to. A binding IP (BIP) address to the requested SID is known by CFN egress. BIP is a unicast IP address accessible to a particular service node providing service SID.

As shown in figure 2, service with SID 2 can be served by either CFN node 2 with binding IP BIP22 or CFN node 3 with BIP32. When the service request from the end device to SID2 reaches the ingress CFN (which is CFN node 1 in this case), the ingress CFN node should determine on the fly which egress CFN this request should be sent to. Then, the de facto service node is determined, and all the subsequent

data packets from the same flow to access this service should always be sent to the binding IP of the selected service node.

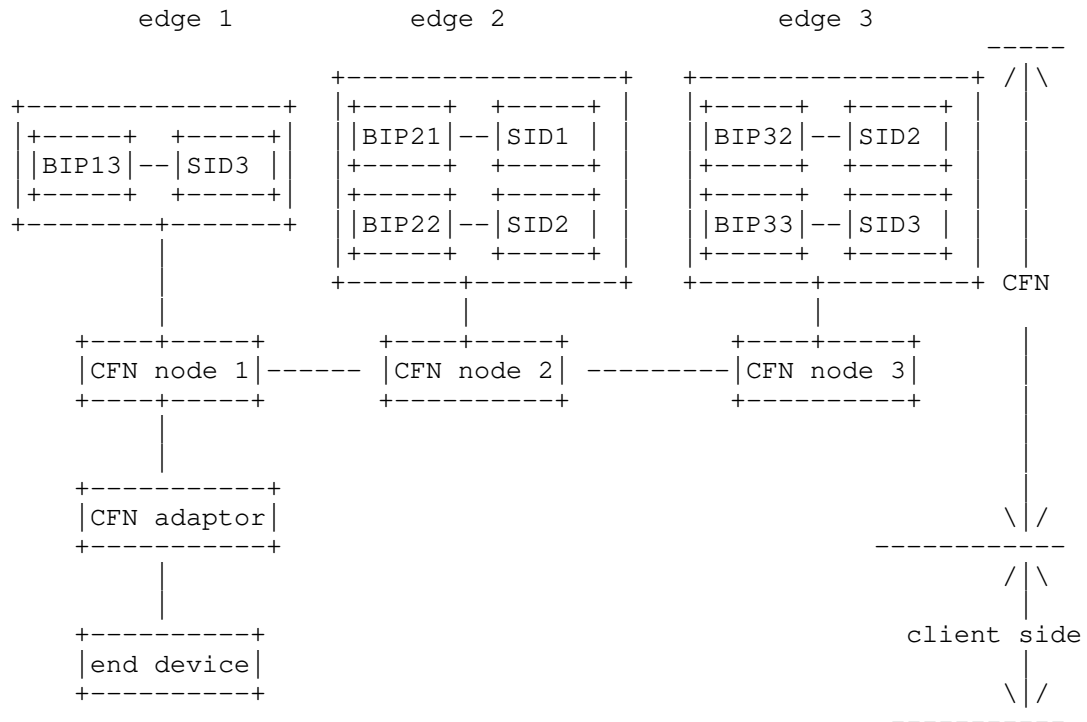


Figure 2. CFN System Overview

CFN adaptor shown in figure 2 is an entity to help the end device working with CFN in a way of keeping the binding information, identifying the initial request packet, and so on. It can be implemented as a part of CFN node (internal mode) or on a separate equipment (external mode). Figure 2 shows an external mode of CFN adaptor which can be deployed at the client side, on a virtual gateway connecting multi user equipments (UEs), as a user Plane Function (UPF) in the mobile network, or on Broadband Remote Access Server (BRAS) in the fixed network. The reason to have such an external mode is that CFN adaptor can be put closer to the clients, and then CFN node is put at some aggregated point with multiple CFN adaptors attached to it. Compared to the internal mode, external CFN adaptor keeps less binding information of the clients. It results in

less memory requirements on CFN node. CFN adaptor has no control plane.

2.2 Generic Workflow

The following procedures describe how CFN works in general.

- 1) CFN adaptor identifies a new service request from the end device, possibly by the special anycast address range for a SID.
- 2) CFN adaptor sends the request to its attaching CFN node which is CFN ingress.
- 3) CFN ingress determines the most appropriate CFN egress based on the computing resource consumption of the service nodes, the network status to the egress nodes and other information. CFN ingress forwards the request to the selected CFN egress. CFN ingress can select itself to serve the request. In this case, it is both ingress and egress in concept.
- 4) CFN egress receives the request from the CFN ingress and explicitly uses the binding IP BIP as destination address to access the required service.
- 5) CFN adaptor of ingress keeps the binding information on (SID, CFN egress) for the flow.
- 6) CFN ingress sends the subsequent packets for the same service from the same flow to the bound CFN egress to ensure the flow affinity.
- 7) CFN nodes distribute the service nodes status like available computing resources for specific services to each other on a regular base.

3. Control Plane and Data Plane

3.1 Control plane

CFN node needs to notify each other about service IDs (SIDs) attaching to it and the computing load information available corresponding to each service ID. This is used for service discovery and dispatch when a request to access a SID is received. Such information can be carried in current BGP [RFC4760] /IGP routing protocol extension. The network cost to a CFN node can be distributed in the same way. A sample service status information to be stored on

a CFN edge is shown in figure 2.

Destination	Computing Load	Network Cost	Next Hop
SID 1	3	5	CFN Egress node 1

Figure 2. Example of service status information in CFN

Computing load can be calculated from different weighted dimensions, e.g. CPU used, number of session being served, query per second, computation delay and so on. Such information needs to be refreshed regularly. In order to avoid fluctuation, it is distributed only when the metrics variation exceeds a threshold or the updating timer is expired. At the same time, the most appropriate egress node selected by the CFN ingress does not necessarily mean the one with the lowest load. Request can be sent to one selected from those egresses with relatively low computing load to avoid fluctuation.

Since SID is an anycast address, CFN ingress determines which CFN egress to forward the request to a specific SID to based on a combination of computing load and network cost.

Figure 3 shows how CFN control plane works in general. It depicts that CFN node 3 distributes computing information for service SID2. CFN node 2 should distribute service SID 2 information in the similar way as shown in figure 3. Definition and operations to extend control plane routing protocol to support CFN information distribution, and schemes/criteria to select CFN egress with anycast address from those information are to be added.

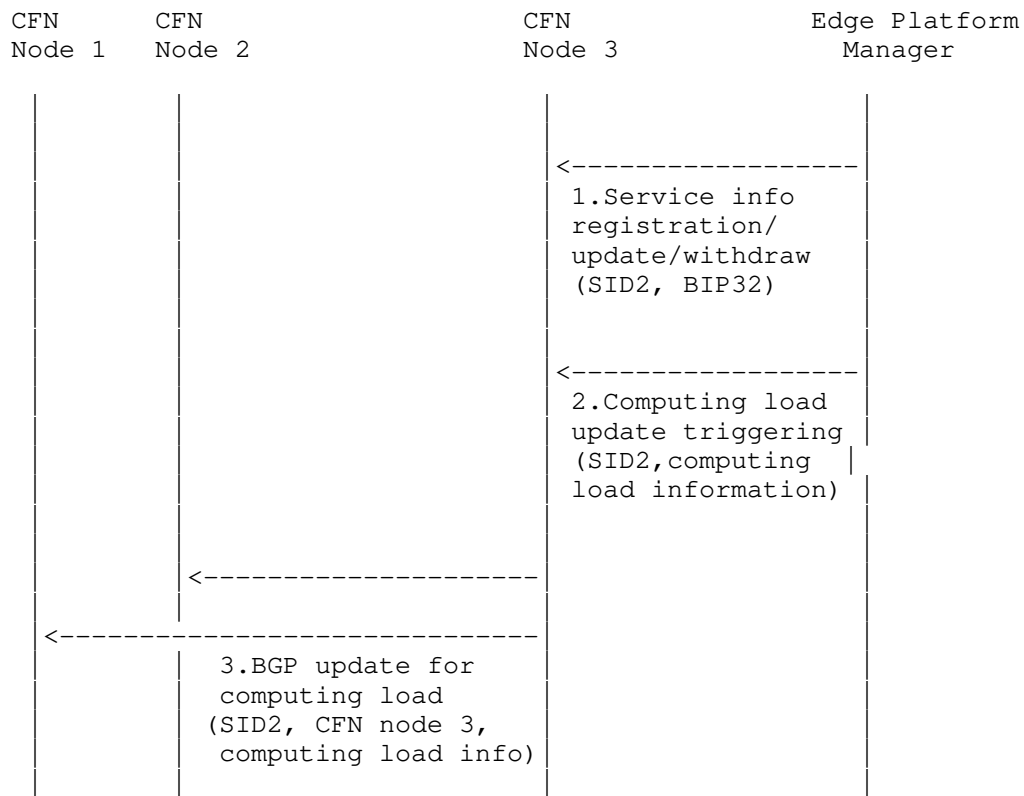


Figure 3. CFN control plane

3.2 Data plane

The traditional anycast is normally used for single request single response style communication as different requests may be sent to different places when the network status changes. CFN used in edge computing may require multiple request multiple response style communication between the end device and the service node. Therefore the data plane must maintain flow affinity to ensure that the requests from the same flow are always processed by the same edge and that edge is determined at the time when the first anycast request is received by CFN ingress. The service access to the same SID from different end hosts attaching to the same CFN ingress may be dispatched to different CFN egresses. We call such a feature dynamic anycast or Dynicast in this document.

Dynacast puts some requirements on the data plane. The flow affinity table needs to be maintained by CFN ingress. On the other hand, large number of end hosts may attach to a CFN node. Therefore CFN ingress may require large memory space, such as tens of thousands of entries, to maintain such a big table of (flow, service ID, egress CFN). It is preferable to place such a binding table on an external CFN adaptor as CFN adaptor only needs to maintain a much smaller table, usually less than a hundred.

Figure 4 shows how CFN data plane works in general.

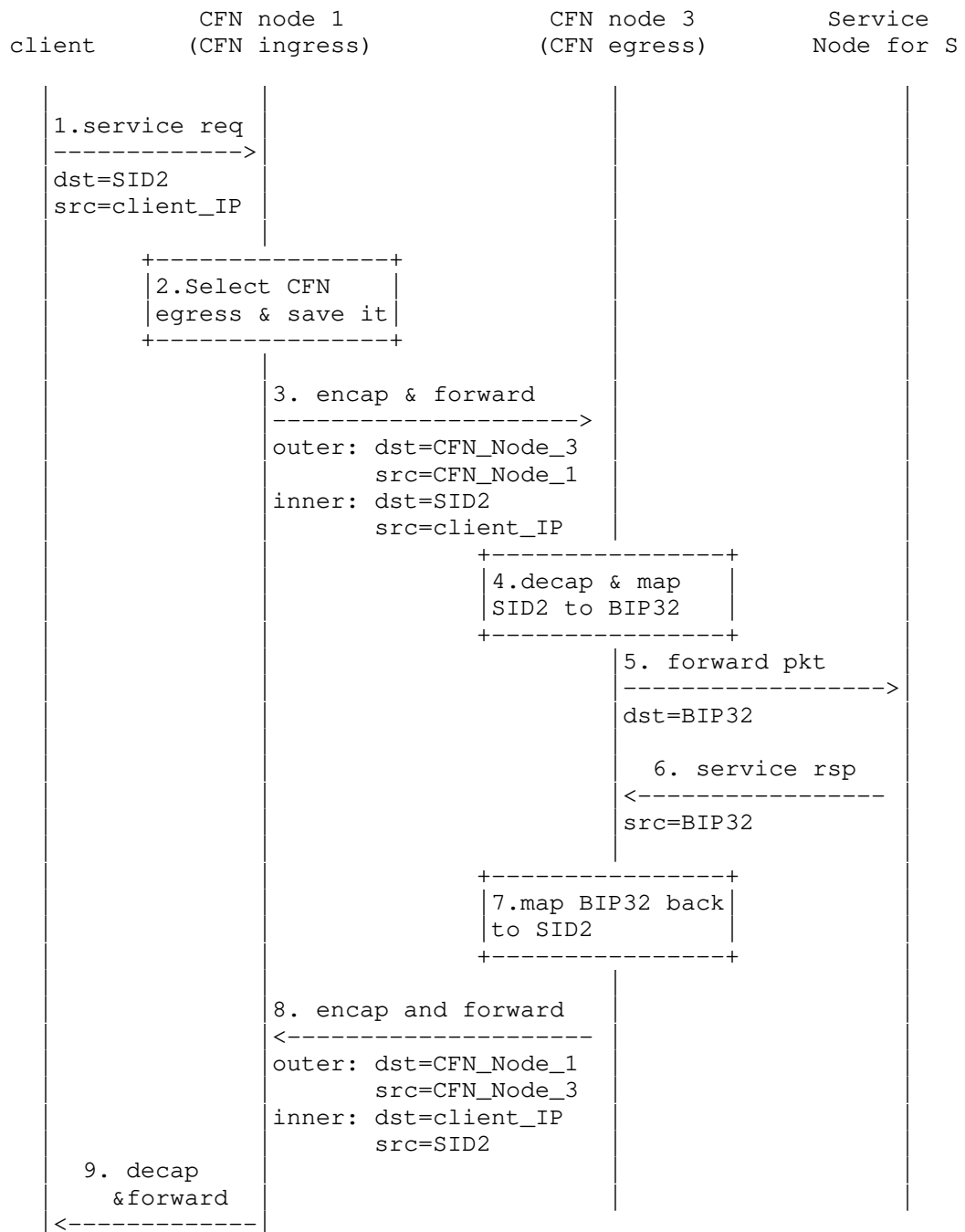


Figure 4. CFN data plane for the first request of a flow

The data plane supports the following functions.

- CFN ingress forwards the first service access request packet of a flow to the selected CFN egress by encapsulation, source routing or segment routing. Figure 4 shows the example of forwarding by encapsulation.
- CFN ingress can inform the external CFN adaptor (if there is) about the binding information on (flow, service ID, egress CFN).
- CFN adaptor (internal or external to CFN ingress) maintains the binding information table for all end hosts attaching to it and forwards the subsequent packets based on the binding information if any.

4. Summary

This draft introduces a CFN framework that enables the service request to be sent to an optimal edge to improve the overall system load balancing. It can dynamically adapt to the computing resources consumption and network status on edges and avoid overloading a single load. CFN is a network based solution that supports a large number of edges and is independent of the applications or services hosted on the edge.

This present document is a strawman for defining CFN framework. A routing protocol (BGP [RFC4760]/IGP based) extension to distribute computing resource information and a late binding based dynamic anycast are to be defined on control plane and data plane respectively.

5. Security Considerations

The computing resource information changes over time very fast with the creation and termination of service instance handlers. When such information is carried in routing protocol, too many updates can make the network fluctuate. Section 3.1 gives a brief idea on avoiding sending too much updates.

6. IANA Considerations

No IANA action is required so far.

7. Acknowledgements

8. References

8.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2 Informative References

- [RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, January 2007.
- [CFN-req] Geng, L., et al, "Compute First Networking (CFN) Scenarios and Requirements", draft-geng-rtgwg-CFN-req-00, November 2019.

Authors' Addresses

Yizhou Li
Huawei Technologies

Email: liyizhou@huawei.com

Jeffrey He
Huawei Technologies

Email: jeffrey.he@huawei.com

Liang Geng
China Mobile
Email: gengliang@chinamobile.com

Peng Liu
China Mobile
Email: liupengyjy@chinamobile.com

Yong Cui
Tsinghua University

Email: cuiyong@tsinghua.edu.cn

Computing in Network Research Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2021

P. Liu
L. Geng
China Mobile
July 12, 2020

Requirement of Computing in network
draft-liu-coinrg-requirement-03

Abstract

New technology such as IOT, edge computing, etc. propose the requirement of computing in network, so the convergence of network and computing has become a trend. It will bring some new directions and areas to be considered, such as the relationship between network and computing, the influence of integrating computing to the network, and so on.

This document points out the requirements of computing in network according to the development of new Industry, including the network and computing requirements.

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].

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 January 13, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Overview	2
2. Requirements of Network	3
2.1. Precision	3
2.2. Concurrent	4
2.3. Addressing	4
2.4. Information interaction	4
3. Requirements of computing	5
3.1. Computing resource deployment	5
3.2. Computing resource discovery	5
3.3. Computing resource reservation	5
3.4. Computing aware scheduling	6
3.5. Computing resource OAM	6
4. Requirements of management	6
4.1. Cross domain management	6
4.2. Joint optimisation	6
4.3. Multi user access	7
5. Conclusion	7
6. Security Considerations	7
7. IANA Considerations	7
8. Normative References	7
Authors' Addresses	8

1. Overview

The new services' provider expects a user experience with lower latency and high reliability, which put forwards immense challenges to cloud computing and traditional network. Centralized computing requires a long transmission distance of traffic flow, and the existing network technology is to the best of its ability. Network operators start to think about how to meet the higher needs of service provider and users. Computing in the network may solve the

problems because it can provide a flexible network and computing integration system.

To integrate the computing resource to the network, it need to find suitable computing nodes to handle service's request, as well as a forwarding path to them. How much computing resources will affect the delay of service processing, which could also affect the whole network latency. Just as the measurement of network performance has one more dimension, it will interact and cooperate with others. So there are some requirments for both network and computing.

2. Requirements of Network

The network requirements includes precision, concurrent, addressing and information interaction.

2.1. Precision

Precision of the network refers to the deterministic of latency, packet loss rate and perception of computing resources.

* Latency: The traditional network's best-effort forwarding mode can no longer meet the demand of such services for network latency. The deterministic latency brings forward a new measure latitude for network, which changes from in-time to on-time.

* Packet loss rate: It is another factor to evaluate the precision of the network. Utilizing the ubiquitous computing capability of the network, network prediction and segment-by-segment path retransmitting are realized based on AI, network transmission can be optimized and service QoS can be ensured.

* Perception of computing resources: how to precisely obtain the status of computing resource to meet the requirements of business requests is also a challenge to the network. It considers the network status and the performance status of computing resources can be matched dynamicly. So the user experience, utilization rate of computing resources and the network efficiency can be optimum.

For the latency and packet loss rate, some technologies such as time-sensitive network TSN, deterministic network DetNet, etc., have proposed corresponding technical means to provide network bearers with deterministic latency(IEEE802.1Qbv, IEEE802.1Qbu) and packet loss rate and guarantee the user's business experience. However, it also needs to consider how to guarantee the service's end-to-end latency, packet loss rate and resource utilization rate.

For the perception of computing resources, we may consider about the OAM and telemetry to achieve it, however, the performance and information collection strategies are issues that need attention.

2.2. Concurrent

There will be number of computing nodes deployed in the network, or computing functions integrated in the network device for network computing. A service's computing request may be distributed in several computing nodes in order to respond quickly to the client. So there may be a lot of parallel computing task, which causes too much connection among the nodes but consumes less bandwidth. It will bring great challenges to the concurrent network connection including how to build and deploy these distributed computing nodes to ensure the processing capability of the network, as well as the storage, call of the database are worth studying.

2.3. Addressing

Traditional application-based addressing can not accurately grasp the network performance in real time. The comprehensive performance of addressing results based on application layer may not be the best. It is always to find the consistent host's address and go through a long distance internet, which results in poor business experience.

It needs to find some new way to improve the addressing process. For example, in the function based addressing, the application deconstruction components on the server side are distributed on the cloud platform, and the business logic in the server is transferred to the client side. So the client only needs to care about the computing function itself, not about the computing resources such as server, virtual machine, container and so on.

2.4. Information interaction

The network needs to have the ability to sense application's requirements and expose network and computing status. For example, application can tell the network requirements including bandwidth, latency and jitter, as well as the computing requirements, such as CPU, storage and memory. The network also can have the capability to be aware of the application's requirements. Thus it can effectively support the network programming, which could meet the future business requirements.

3. Requirements of computing

The computing requirements includes computing resource deployment, discovery, reservation, scheduling and OAM.

3.1. Computing resource deployment

If some computing tasks in the network is planned to be implemented, it needs to consider about what kinds of chips and where should them be deployed. On the one hands, different kinds of computing require different kinds of chips, such as CPU, GPU and memory chip. On the other hands, those chips may be put into router, switch, server or some dedicated machines, which are connected by the network.

There is an example about AI algorithm which might be discussed before. The AI algorithm has several steps including training and matching, and they also have different requirements of chips. In network computing, those steps could be distributed in different computing nodes.

3.2. Computing resource discovery

The network needs to have the ability to discover computing resources. when the computing nodes are deployed in the network, it need to be registered to the network management system, and the information of computing resource or routing can update. In this way, when there are computing tasks to be executed, the network can reasonably allocate resources according to the needs of the application.

3.3. Computing resource reservation

There might be serial distributed computing model of computing in the network, and different resources need to be reserved for different nodes. For example, AI algorithm now has a model of step-by-step iteration at multiple nodes. The previous iteration will affect the next calculation results, and the computing resources required for each iteration are not the same. From the perspective of network standard, we hope to regard computing resources as the dimensions to measure network performance, such as the same bandwidth, path, etc., while the traditional technologies of resource reservation have not considered the reservation of computing resources, and have not considered the differentiated resource reservation model. Therefore, new protocol or extension of existing protocol is needed to meet the requirement.

3.4. Computing aware scheduling

Computing in network needs a reasonable scheduling strategy, which means computing aware scheduling. According to the business requests, dynamically computing power matching is carried out based on network status and performance of computing resources to achieve optimal user experience, optimal utilization of computing resources and optimal network efficiency. In computing aware scheduling, computing is seen as "link state" and the computing resource information should be exposed.

3.5. Computing resource OAM

The ability of OAM can be used to continuously update the current computing power resources, and perform some troubleshooting tasks. However, OAM of computing resource is more complex than network. Network monitoring is relatively simple, like bandwidth, latency, jitter, while computing can be divided into many categories, different application need different kinds of computing. So it need to implement fine-grained OAM of computing resource.

4. Requirements of management

The management requirements includes cross domain management, joint optimisation and multi user access.

4.1. Cross domain management

The computing in the network should ensure the end-to-end network management to meet the needs of different network topology, performance and function, which involves cross domain network arrangement. In the process of network data transmission, different services will forward in different ways or different network protocols, and computing resources may be distributed in different network domains. Effective cross domain management will enhance the performance of network and computing.

4.2. Joint optimisation

As computing resources are integrated in the network, and may be used as one of the measurement dimensions of network performance, joint optimization is also a very important part. Network and computing resources will affect each other, including performance, scheduling and so on. So It need a good joint optimization scheduling strategy.

4.3. Multi user access

Many existing applications, such as games, remote video conferencing, are usually multi--accessed and interacted by several users at the same time. This brings about the problem of service consistency, that is, users accessing to the same game or video need the consistency of SLA, otherwise it will seriously affect the experience of other users. Service consistency can be achieved through network management or application layer control.

5. Conclusion

Based on the requirements of new business, this document puts forward the requirements of computing in network, and gives some reference technologies and use cases. Computing in network is a new direction, some details need more in-depth discussion and research.

6. Security Considerations

Computing In network has brought the trend of network convergence in different regions. For example, 5g network of operators can go deep into the vertical industry user site to provide users with higher quality network services, which will bring the convergence of operator's network and user site network. Besides, industrial Internet brings the trend of integration of industrial OT network and IT network to further improve the production efficiency of the industry. It need to ensure the security of the network, including the mutual trust and non aggression of information among regions, which may require further protection and detection measures.

7. IANA Considerations

TBD.

8. Normative References

[I-D.kutscher-coinrg-dir]

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

[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>>.

Authors' Addresses

Peng Liu
China Mobile
Beijing 100053
China

Email: liupengyjy@chinamobile.com

Liang Geng
China Mobile
Beijing 100053
China

Email: gengliang@chinamobile.com