

COIN
Internet-Draft
Intended status: Informational
Expires: January 5, 2020

I. Kunze
J. Rueth
K. Wehrle
RWTH Aachen University
July 4, 2019

Industrial Use Cases for In-Network Computing
draft-kunze-coin-industrial-use-cases-00

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 5, 2020.

Copyright 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 (<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. In-Network Control / Time-sensitive applications	4
2.1. Characterization and Requirements	5
2.1.1. Approaches	5
3. Large Volume Applications/ Traffic Filtering	6
3.1. Characterization and Requirements	6
3.2. Approaches	7
3.2.1. Traffic Filters	7
3.2.2. In-Network (Pre-)Processing	8
4. Industrial Safety (Dead Man's Switch)	9
4.1. Characterization and Requirements	9
4.1.1. Approaches	9
5. Security Considerations	10
6. IANA Considerations	10
7. Conclusion	10
8. Informative References	11
Authors' Addresses	11

1. Introduction

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

Enhancements to the standard Ethernet such as Time-Sensitive-Networking [TSN] try to achieve the requirements on the link layer by statically reserving shares of the bandwidth. These concepts are

well-suited for traditional industrial settings where the communication paths are encapsulated at the respective factory sites and where the communication patterns are well understood. Following the vision of the Industrial Internet of Things (IIoT), more and more parts of the industrial production domain are interconnected. This increases the complexity of the industrial networks, making them more dynamic and creating more diverse sets of requirements. Furthermore, process control is imagined to be exercised from remote clouds for feasibility reasons which is why solutions on the link layer alone are not sufficient in these scenarios.

Common components of the IIoT can be divided into three categories as illustrated in Figure 1. Following

[I-D.draft-mcbride-edge-data-discovery-overview-01], EDGE DEVICES, such as sensors and actuators, constitute the boundary between physical and digital world. They communicate the current state of the physical world to the digital world by transmitting sensor data or let the digital world interact with or manipulate the physical world by executing actions after receiving (simple) control information. The processing of the sensor data as well as the creation of the control information is done on COMPUTING DEVICES. They range from small-powered controllers in close proximity to the EDGE DEVICES, to more powerful edge or remote clouds in larger distances. The connection between the EDGE and COMPUTING DEVICES is established by NETWORKING DEVICES. In the industrial domain, they range from standard devices, e.g. typical Ethernet switches, which can interconnect all Ethernet-capable hosts, to proprietary equipment with proprietary protocols which only supports hosts of specific vendors.

The challenge is to develop concepts which can include off-premise entities (such as distant cloud platforms) as well as proprietary hosts into the communication and still satisfy the performance requirements of modern industrial networks. The in-network computing paradigm presents a promising starting point because (pre-)processing data within the network can speed up the communication, e.g., by reducing the amount of transmitted data and thus congestion. Flexibly distributing the computation tasks across the network helps to manage dynamic changes. Specifying general requirements for the different application scenarios is difficult due to the mentioned diversity. In an effort to showcase potential requirements for the domain of industrial production, we characterize and analyze three distinct scenarios to illustrate how in-network computations can be helpful.

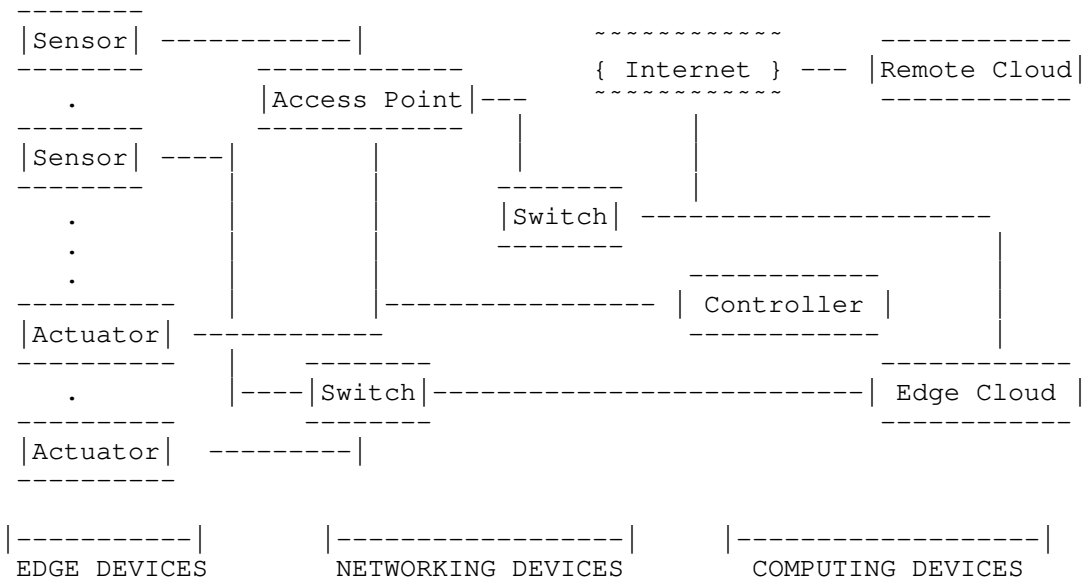


Figure 1: Industrial networks show a high level of heterogeneity.

2. In-Network Control / Time-sensitive applications

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

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

It is worthwhile to investigate whether the outsourcing of control functionality to distant computation platforms is viable, because these platforms have a high level of flexibility and scalability. In

the following, we describe the requirements and characteristics of the control setting in more detail.

2.1. Characterization and Requirements

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

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

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

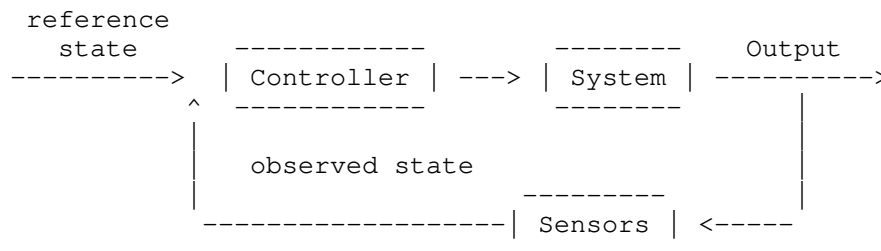


Figure 2: Simple feedback control model

2.1.1. Approaches

Control models in general can become complex but there is a variety of control algorithms that are composed of simple computations such as matrix multiplication. As these are supported by programmable network devices, it is a possibility to compose simplified approximations of the more complex algorithms and deploy them in the network. While the simplified versions induce a more inaccurate

control, they allow for a quicker response and might be sufficient to operate a basic tight control loop while the overall control can still be exercised from the cloud. The problem, however, is that networking devices typically only allow for integer precision computation while floating point precision is needed by most control algorithms. Early approaches like [RUETH] have already shown the general applicability of such ideas, but there are still a lot of open research questions not limited to the following:

- o How can one derive the simplified versions of the overall controller?
 - * How complex can they become?
 - * How can one take the limited computational precision of networking devices into account when making them?
- o How does one distribute the simplified versions in the network?
- o How does the overall controller interact with the simplified versions?

3. Large Volume Applications/ Traffic Filtering

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

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

3.1. Characterization and Requirements

Processes in the industrial domain are monitored by distributed sensors which range from simple binary (e.g., light barriers) to complex sensors measuring the system with varying degrees of resolution. Sensors can further serve different purposes, as some

might be used for the time-critical process control while others are only used as redundant fall back platforms. Overall, there is a high level of heterogeneity which makes managing the sensor output a challenging task.

Depending on the deployed sensors and the complexity of the observed system, the resulting overall data volume can easily be in the range of several Gbit/s [GLEBKE]. Using off-premise clouds for managing the data requires uploading or streaming the growing volume of sensor data using the companies' Internet access which is typically limited to a few hundred of Mbit/s. While large networking companies can simply upgrade their infrastructure, most industrial companies rely on traditional ISPs for their Internet access. Higher access speeds are hence tied to higher costs and, above all, subject to the supply of the ISPs and consequently not always available. A major challenge is thus to devise methodology which is able to handle such amounts of data over limited access links.

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

3.2. Approaches

While there is no work on the question of security yet, there are at least two concepts which might be suitable for reducing the amount of transmitted data in a meaningful way:

1. filtering out redundant or unnecessary data
2. aggregating data by applying preprocessing steps within the network

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

3.2.1. Traffic Filters

Sensors are often set up redundantly, i.e., part of the collected data might also be redundant. Moreover, they are often hard to configure or not configurable at all which is why their resolution or

sampling frequency is often larger than required. Consequently, it is likely that more data is transmitted than is actually needed or desired. A trivial idea for reducing the amount of data is thus to filter out redundant or undesired data before it leaves the premise using simple traffic filters that are deployed in the on-premise network. In this context, the following research questions can be of interest:

- o How can traffic filters be designed?
- o How can traffic filters be coordinated and deployed?
- o How can traffic filters be changed dynamically?

3.2.2. In-Network (Pre-)Processing

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

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

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

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

4. Industrial Safety (Dead Man's Switch)

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

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

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

4.1. Characterization and Requirements

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

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

4.1.1. Approaches

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

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

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

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

5. Security Considerations

N/A

6. IANA Considerations

N/A

7. Conclusion

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

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

In a second scenario, large amounts of data have to be transmitted to cloud platforms for further evaluation. One important task here is to reduce the amount of data that needs to be transmitted as the available Internet access speed is most likely non-sufficient. Apart from that, security measures have to be implemented as business data is transmitted to the Internet.

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

8. Informative References

- [GLEBKE] Glebke, R., "A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems", DOI: 10125/60162, in HICSS, January 2019.
- [I-D.draft-mcbride-edge-data-discovery-overview-01]
McBride, M., Kutscher, D., Schooler, E., and C. Bernardos, "Overview of Edge Data Discovery", draft-mcbride-edge-data-discovery-overview-01 (work in progress), March 2019.
- [RUETH] Rueth, J., "Towards In-Network Industrial Feedback Control", DOI: 10.1145/3229591.3229592, in ACM SIGCOMM NetCompute, August 2018.
- [SAPIO] Sapio, A., "Scaling Distributed Machine Learning with In-Network Aggregation", 2019, <<https://arxiv.org/abs/1903.06701>>.
- [TSN] "Time-Sensitive Networking (TSN) Task Group", 2019, <<https://1.ieee802.org/tsn/>>.

Authors' Addresses

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

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

Jan Rueth
RWTH Aachen University
Ahornstr. 55
Aachen D-50274
Germany

Phone: +49-241-80-21417
Email: rueth@comsys.rwth-aachen.de

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

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

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

I. Kunze
K. Wehrle
RWTH Aachen University
D. Trossen
Huawei Technologies Duesseldorf GmbH
2 November 2020

Use Cases for In-Network Computing
draft-kunze-coin-industrial-use-cases-04

Abstract

Computing in the Network (COIN) comes with the prospect of deploying functionality on networking devices, such as switches and network interface cards. While such functionality can be beneficial in several contexts, it has to be carefully placed into the context of the general Internet communication. This document discusses some use cases to demonstrate how real applications can benefit from COIN and to showcase essential requirements that have to be fulfilled by COIN applications.

Status of This Memo

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

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

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

This Internet-Draft will expire on 6 May 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	3
3. Industrial Use Cases	3
3.1. IIoT Network Scenario	4
3.2. In-Network Control / Time-sensitive applications	5
3.2.1. Characterization and Requirements	5
3.2.2. Approaches	6
3.3. Large Volume Applications/ Traffic Filtering	7
3.3.1. Characterization and Requirements	7
3.3.2. Approaches	8
3.4. Industrial Safety (Dead Man's Switch)	9
3.4.1. Characterization and Requirements	10
3.4.2. Approaches	10
4. Security Considerations	11
5. Immersive Devices	11
5.1. Mobile Application Offloading	11
5.2. Edge AR/VR	11
6. Infrastructure Services	11
6.1. Distributed AI	11
6.2. Content Delivery Networks	11
6.3. CFaaS	12
7. Taxonomy	12
8. IANA Considerations	12
9. Conclusion	12
10. Informative References	12
Authors' Addresses	14

1. Introduction

The Internet bases on a best-effort network with limited guarantees regarding the timely and successful transmission of packets. Functionality is generally provided by the end-hosts while the network is kept simple and only intended to forward the packets. This design-choice is suitable for general Internet-based applications and has helped in the rapid growth of the Internet. However, there are several domains which, e.g., demand a number of strict performance guarantees that cannot be provided over regular best-effort networks. In this context, flexibly distributing the computation tasks across the network can help to achieve the guarantees and increase the overall performance.

The different domains, however, have different requirements and it is unclear whether there can be a common solution to all COIN scenarios or if solutions have to be tailored to each scenario.

This document first presents applications and requirements of some domains to illustrate the importance of COIN for realizing advanced applications. Based on these discussion, the draft then creates a taxonomy of COIN scenarios with the goal of guiding future work.

2. Terminology

Programmable network devices (PNDs): Network devices, such as network interface cards and switches, which are programmable, e.g., using P4

3. Industrial Use Cases

The industrial domain is characterized by diverse sets of requirements which often cannot be provided over regular best-effort networks. Consequently, there is a large number of specialized applications and protocols designed to give the required strict performance guarantees, e.g., regarding real-time capabilities. Time-Sensitive-Networking [TSN] as an enhancement to the standard Ethernet, e.g., tries to achieve these requirements on the link layer by statically reserving shares of the bandwidth. In the Industrial Internet of Things (IIoT), however, more and more parts of the industrial production domain are interconnected. This increases the complexity of the industrial networks, makes them more dynamic, and creates more diverse sets of requirements. In these scenarios, solutions on the link layer alone are not sufficient.

The challenge is to develop concepts that can satisfy the dynamic performance requirements of modern industrial networks. COIN presents a promising starting point because it allows to flexibly distribute computation tasks across the network which can help to manage dynamic changes. As specifying general requirements for the industrial production domain is difficult due to the mentioned diversity, this document next characterizes and analyzes three distinct scenarios to showcase potential requirements for the industrial production domain, thereby illustrating how COIN can be helpful.

3.1. IIoT Network Scenario

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

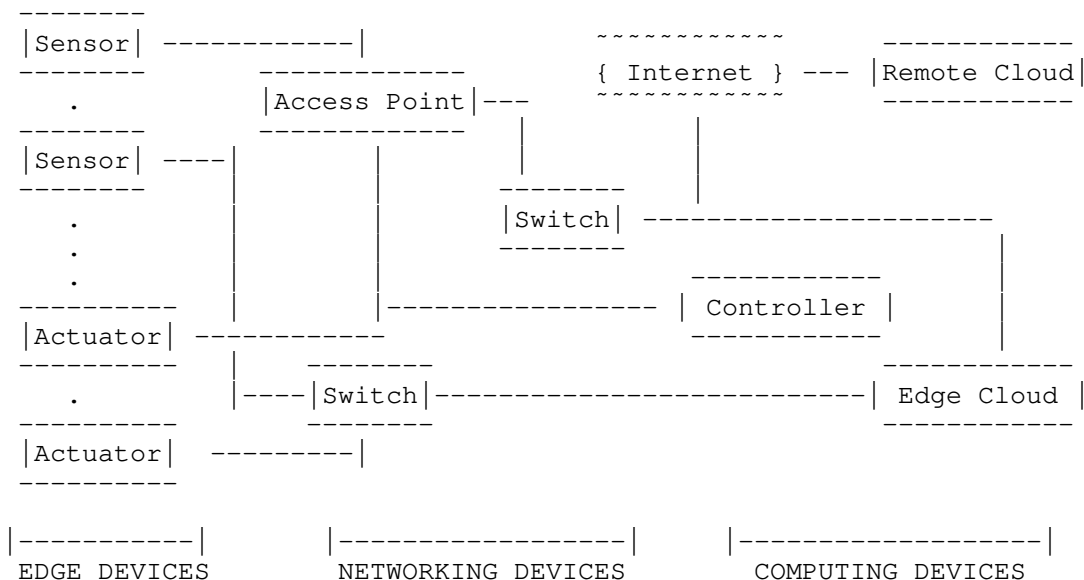


Figure 1: Industrial networks show a high level of heterogeneity.

3.2. In-Network Control / Time-sensitive applications

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

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

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

3.2.1. Characterization and Requirements

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

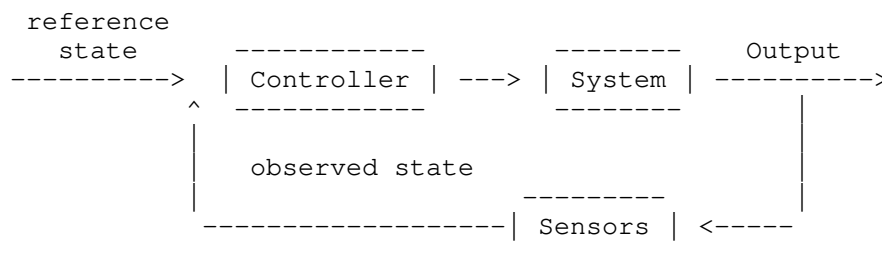


Figure 2: Simple feedback control model.

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

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

3.2.2. Approaches

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

- * How can one derive the simplified versions of the overall controller?
 - How complex can they become?
 - How can one take the limited computational precision of networking devices into account when making them?
- * How does one distribute the simplified versions in the network?
- * How does the overall controller interact with the simplified versions?

3.3. Large Volume Applications/ Traffic Filtering

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

Traditional company infrastructure is neither equipped for the management and storage of such large amounts of data nor for the computationally expensive training of ML approaches. Similar to the considerations in Section 3.2, off-premise cloud platforms offer cost-effective solutions with a high degree of flexibility and scalability. While the unpredictable latency of the Internet is only a subordinate problem for this use case, moving all data to off-premise locations primarily poses infrastructural challenges which are presented in more detail in the following.

3.3.1. Characterization and Requirements

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

Depending on the deployed sensors and the complexity of the observed system, the resulting overall data volume can easily be in the range of several Gbit/s [GLEBKE]. Using off-premise clouds for managing the data requires uploading or streaming the growing volume of sensor data using the companies' Internet access which is typically limited to a few hundred of Mbit/s. While large networking companies can simply upgrade their infrastructure, most industrial companies rely on traditional ISPs for their Internet access. Higher access speeds are hence tied to higher costs and, above all, subject to the supply of the ISPs and consequently not always available. A major challenge is thus to devise a methodology that is able to handle such amounts of data over limited access links.

Another aspect is that business data leaving the premise and control of the company further comes with security concerns, as sensitive information or valuable business secrets might be contained in it. Typical security measures such as encrypting the data make in-network

computing techniques hardly applicable as they typically work on unencrypted data. Adding security to in-network computing approaches, either by adding functionality for handling encrypted data or devising general security measures, is thus an auspicious field for research which we describe in more detail in Section 4.

3.3.2. Approaches

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

1. filtering out redundant or unnecessary data
2. aggregating data by applying pre-processing steps within the network

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

3.3.2.1. Traffic Filters

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

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

- * How can traffic filters be designed?
- * How can traffic filters be coordinated and deployed?

- * How can traffic filters be changed dynamically?
- * How can traffic filtering be signaled to the end-hosts?

3.3.2.2. In-Network (Pre-)Processing

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

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

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

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

3.4. Industrial Safety (Dead Man's Switch)

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

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

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

3.4.1. Characterization and Requirements

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

These measures are static solutions, potentially relying on specialized hardware, and are challenged by the increased dynamics of modern factories where the factory configuration can be changed on demand. Software solutions offer higher flexibility as they can dynamically respect new information gathered by the sensor systems, but in most cases they cannot give guaranteed safety. Yet, it is worthwhile to investigate whether such solutions can introduce additional safety measures.

3.4.2. Approaches

Software-based solutions can take advantage of the large amount of available sensor data. Different safety indicators within the production hall can be combined within the network so that programmable networking devices can give early responses if a potential safety breach is detected. A rather simple possibility could be to track the positions of human workers and robots. Whenever a robot gets too close to a human in a non-working area or if a human enters a defined safety zone, robots are stopped to prevent injuries. More advanced concepts could also include image data or combine arbitrary sensor data.

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

- * Which additional safety measures can be provided?

- Do these measures actually improve safety?
- * Which sensor information can be combined and how?

4. Security Considerations

Note: This section will need consolidation once new use cases are added to the draft. Current in-network computing approaches typically work on unencrypted plain text data because today's networking devices usually do not have crypto capabilities. As is already mentioned in Section 3.3.1, this above all poses problems when business data, potentially containing business secrets, is streamed into remote computing facilities and consequently leaves the control of the company. Insecure on-premise communication within the company and on the shop-floor is also a problem as machines could be intruded from the outside. It is thus crucial to deploy security and authentication functionality on on-premise and outgoing communication although this might interfere with in-network computing approaches. Ways to implement and combine security measures with in-network computing are described in more detail in [I-D.fink-coin-sec-priv].

5. Immersive Devices

5.1. Mobile Application Offloading

NOTE: Will be moved here from
[I-D.draft-sarathchandra-coin-appcentres-03]

5.2. Edge AR/VR

NOTE: Could be transfered from (expired)
[I-D.draft-montpetit-coin-xr-03]

Additional potential sources: [I-D.draft-geng-rtgwg-cfn-req-00]

6. Infrastructure Services

6.1. Distributed AI

NOTE: Will be moved here from
[I-D.draft-sarathchandra-coin-appcentres-03]

6.2. Content Delivery Networks

NOTE: Will be moved here from
[I-D.draft-sarathchandra-coin-appcentres-03]

6.3. CFaaS

NOTE: Will be moved here from
[I-D.draft-sarathchandra-coin-appcentres-03]

7. Taxonomy

NOTE: The taxonomy is intended to generalize characteristics of the different presented use cases and work on it will start once more use cases are added to the draft.

8. IANA Considerations

N/A

9. Conclusion

There are several domains that can profit from COIN.

Industrial scenarios have unique sets of requirements mostly focusing around tight latency constraints with high required bandwidths.

NOTE: Further aspects will be added once more use cases are added to the draft.

10. Informative References

[GLEBKE] Glebke, R., Henze, M., Wehrle, K., Niemietz, P., Trauth, D., Mattfeld MBA, P., and T. Bergs, "A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems", Proceedings of the 52nd Hawaii International Conference on System Sciences, DOI 10.24251/hicss.2019.871, 2019, <<https://doi.org/10.24251/hicss.2019.871>>.

[I-D.draft-geng-rtgwg-cfn-req-00]
Geng, L. and P. Willis, "Compute First Networking (CFN) Scenarios and Requirements", Work in Progress, Internet-Draft, draft-geng-rtgwg-cfn-req-00, 4 November 2019, <<http://www.ietf.org/internet-drafts/draft-geng-rtgwg-cfn-req-00.txt>>.

[I-D.draft-montpetit-coin-xr-03]
Montpetit, M., "In Network Computing Enablers for Extended Reality", Work in Progress, Internet-Draft, draft-montpetit-coin-xr-03, 8 July 2019, <<http://www.ietf.org/internet-drafts/draft-montpetit-coin-xr-03.txt>>.

- [I-D.draft-sarathchandra-coin-appcentres-03]
Trossen, D., Sarathchandra, C., and M. Boniface, "In-Network Computing for App-Centric Micro-Services", Work in Progress, Internet-Draft, draft-sarathchandra-coin-appcentres-03, 23 October 2020, <<http://www.ietf.org/internet-drafts/draft-sarathchandra-coin-appcentres-03.txt>>.
- [I-D.fink-coin-sec-priv]
Fink, I. and K. Wehrle, "Enhancing Security and Privacy with In-Network Computing", Work in Progress, Internet-Draft, draft-fink-coin-sec-priv-01, 8 September 2020, <<http://www.ietf.org/internet-drafts/draft-fink-coin-sec-priv-01.txt>>.
- [I-D.mcbride-edge-data-discovery-overview]
McBride, M., Kutscher, D., Schooler, E., Bernardos, C., Lopez, D., and X. Foy, "Edge Data Discovery for COIN", Work in Progress, Internet-Draft, draft-mcbride-edge-data-discovery-overview-05, 1 November 2020, <<http://www.ietf.org/internet-drafts/draft-mcbride-edge-data-discovery-overview-05.txt>>.
- [PENNEKAMP]
Pennekamp, J., Henze, M., Schmidt, S., Niemietz, P., Fey, M., Trauth, D., Bergs, T., Brecher, C., and K. Wehrle, "Dataflow Challenges in an Internet of Production: A Security & Privacy Perspective", Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy - CPS-SPC'19, DOI 10.1145/3338499.3357357, 2019, <<https://doi.org/10.1145/3338499.3357357>>.
- [RUETH]
Rueth, J., Glebke, R., Wehrle, K., Causevic, V., and S. Hirche, "Towards In-Network Industrial Feedback Control", Proceedings of the 2018 Morning Workshop on In-Network Computing - NetCompute '18, DOI 10.1145/3229591.3229592, 2018, <<https://doi.org/10.1145/3229591.3229592>>.
- [SAPIO]
Sapio, A., "Scaling Distributed Machine Learning with In-Network Aggregation", 2019, <<https://arxiv.org/abs/1903.06701>>.
- [TSN]
IEEE, ., "Time-Sensitive Networking (TSN) Task Group", 2019, <<https://1.ieee802.org/tsn/>>.
- [VESTIN]
Vestin, J., Kassler, A., and J. Akerberg, "FastReact: In-Network Control and Caching for Industrial Control Networks using Programmable Data Planes", 2018 IEEE 23rd

International Conference on Emerging Technologies and
Factory Automation (ETFA), DOI 10.1109/etfa.2018.8502456,
September 2018,
<<https://doi.org/10.1109/etfa.2018.8502456>>.

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: January 9, 2020

D. Kutscher
University of Applied Sciences Emden/Leer
T. Kaerkkainen
J. Ott
Technical University Muenchen
July 08, 2019

Directions for Computing in the Network
draft-kutscher-coinrg-dir-00

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 January 9, 2020.

Copyright 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 (<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	4
3.2. Packet Processing	5
3.3. Computing in the Network	6
3.4. Elements for Computing in the Network	8
4. Research Challenges	10
4.1. Categorization of Different Use Cases for Computing in the Network	10
4.2. Networking and Remote-Method-Invocation Abstractions	10
4.3. Transport Abstractions	12
4.4. Programming Abstractions	13
4.5. Security, Privacy, Trust Model	14
4.6. Failure Handling, Debugging, Management	15
5. Acknowledgements	15
6. Informative References	15
Authors' Addresses	16

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 (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.

Microservices can be seen as a (light-weight) extension of the cloud computing model (application logic in containers and orchestrators for resource allocation and other management functions), leveraging more light-weight 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 - not for general Computing in the Network purposes.

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 wildly 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/Procedure Invocation 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.

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 mentioned security problems.

Programmable Data Plane approach such as P4 are providing abstractions of different types of network switch hardware (NPUs, CPUs, FPGA, PISA) from a switch/network programming perspective. 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 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.

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 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.

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;
- o function off-loading 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:

- 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 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 \cite{crowd-computing} or mist computing \cite{mist-computing} 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 "decentrally" provided 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, in-network computing 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.

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. 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.

4.1. 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.

4.2. 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 function identify 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").

4.3. 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 bears 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; 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] [I-D.kutscher-icnrg-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?
- 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?

4.4. 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 amount 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?

4.5. 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?
- 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).

4.6. 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?

5. Acknowledgements

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

6. 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>>.
- [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.
- [I-D.kutscher-icnrg-rice] Krol, M., Habak, K., Oran, D., Kutscher, D., and I. Psaras, "Remote Method Invocation in ICN", draft-kutscher-icnrg-rice-00 (work in progress), October 2018.
- [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 - ICN '18, DOI 10.1145/3267955.3267956, 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 - HotNets-XVI, DOI 10.1145/3152434.3152461, 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.

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

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

- [CROWD] Murray, D., Yoneki, E., Crowcroft, J., and S. Hand, "The case for crowd computing", Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds - MobiHeld '10, DOI 10.1145/1851322.1851334, 2010.
- [EDGESURVEY] Mach et al, P., "Mobile Edge Computing -- A Survey on Architecture and Computation Offloading", 2017, <<https://ieeexplore.ieee.org/document/7879258>>.
- [FLINK] Katsifodimos, A. and S. Schelter, "Apache Flink: Stream Analytics at Scale", 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), DOI 10.1109/ic2ew.2016.56, April 2016.
- [FOGEDGE] Salaht, F., Desprez, F., and A. Lebre, "An Overview of Service Placement Problem in Fog and Edge Computing", ACM Computing Surveys Vol. 53, pp. 1-35, DOI 10.1145/3391196, July 2020.
- [I-D.trossen-sfc-name-based-sff] Trossen, D., Purkayastha, D., and A. Rahman, "Name-Based Service Function Forwarder (nSFF) component within SFC framework", draft-trossen-sfc-name-based-sff-07 (work in progress), May 2019.
- [MEC] ETSI, ., "Multi-access Edge Computing (MEC)", 2020, <<https://www.etsi.org/technologies/multi-access-edge-computing>>.
- [MEC5G] Sami Kekki et al, ., "MEC in 5G Networks", 2018, <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf>.
- [MIST] Barik, R., Dubey, A., Tripathi, A., Pratik, T., Sasane, S., Lenka, R., Dubey, H., Mankodiya, K., and V. Kumar, "Mist Data: Leveraging Mist Computing for Secure and Scalable Architecture for Smart and Connected Health", Procedia Computer Science Vol. 125, pp. 647-653, DOI 10.1016/j.procs.2017.12.083, 2018.
- [RAY] Moritz et al, P., "Ray -- A Distributed Framework for Emerging AI Applications", 2018, <<http://dl.acm.org/citation.cfm?id=3291168.3291210>>.

- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RICE] KrA^3l, M., Habak, K., Oran, D., Kutscher, D., and I. Psaras, "RICE", Proceedings of the 5th ACM Conference on Information-Centric Networking, DOI 10.1145/3267955.3267956, September 2018.
- [SAPIO] Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come", Proceedings of the 16th ACM Workshop on Hot Topics in Networks, DOI 10.1145/3152434.3152461, November 2017.
- [SPARROW] Ousterhout, K., Wendell, P., Zaharia, M., and I. Stoica, "Sparrow", Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13, DOI 10.1145/2517349.2522716, 2013.

8.2. URIs

- [1] <https://akka.io/>
- [2] <https://flink.apache.org/>
- [3] <https://doc.akka.io/docs/akka/2.3/scala/remoting.html>
- [4] <https://doc.akka.io/docs/akka/current/general/addressing.html>
- [5] <https://doc.akka.io/docs/akka/current/typed/routers.html>
- [6] <https://doc.akka.io/docs/akka/current/typed/cluster.html>
- [7] <https://zookeeper.apache.org/>

Authors' Addresses

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

Email: ietf@dkutscher.net

Teemu 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

Computing in Network Research Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

P. Liu
L. Liu
China Mobile
July 8, 2019

Requirement of Computing in network
draft-liu-coinrg-requirement-00

Abstract

New technology such as IOT, edge computing, etc. propose the requirement of computing in network, so convergence of network and computing has become a trend. This document points out the requirements of computing according to the development of new Industry, and analyzes the new architecture including precision and Intelligence of the network.

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 9, 2020.

Copyright 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

(<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	2
2.1. New Architecture	2
2.2. Precision	3
2.3. Ubiquitous Intelligence	3
3. Conclusion	4
4. Security Considerations	4
5. IANA Considerations	4
6. References	4
6.1. Normative References	4
6.2. Informative References	4
Authors' Addresses	4

1. Overview

Intelligence in the whole industry has a huge demand for computing, which put forwards immense challenges to cloud computing and network. Network Architecture is undergoing a transformation towards cloud and service, providing rapid response and flexible deployment for different industries. Precision and intelligence will be the important direction of next generation network development.

2. Requirements of Network

Next generation network needs new architecture, to building precision network and new ubiquitous intelligence capability.

2.1. New Architecture

New Architecture needs to integrate computing into a part of the network, so the computing capability could extend as the network. It also needs to use the AI technology to enable network the capabilities of self-generating and self-optimizing. On the one hand, the network driven by users and services could be self-generating, self-constructing, self-configuring and self-management on demand. On the other hand, the capability and service of network could be iterated, optimized and evolved automatically.

2.2. Precision

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

In the face of time sensitive services such as industrial Internet and telemedicine, 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 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.

Besides those, how to precisely distribute network computing power 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 to dynamic match the computing power. So the user experience, utilization rate of computing resources and the network efficiency can be optimum.

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.

2.3. Ubiquitous Intelligence

Ubiquitous Intelligence refers to use AI in the whole network, which could realize the programmability of network, flexible scheduling, heterogeneous computing resources and Atomized AI algorithm.

Traditional IP-based addressing and TCP/TLS session-based network model are difficult to play dynamic, micro-service, ubiquitous computing advantages, also can't guarantee the maximum computing efficiency. Function-based addressing distributes the application solution of Server side into the cloud platform. So the client only needs to care about the computing function itself, but not about the computing resources such as server, virtual machine, container and so on, so as to realize the function as a service.

Traditional IP network and users can not configure each other, the network can not provide customized services according to users' needs, and the terminal also lacks the ability to perceive the necessary network state. So it is necessary to open network interface to express user's needs and realize network programmability. The network can configure parameters according to users' needs, and users can transfer requirements based on network capabilities/states, thus effectively supporting future business requirements.

3. Conclusion

Under the new business requirements, the existing network technology can not meet the needs of transmission, calculation and efficiency. Precision and intelligence are the key directions in the evolution of network computing convergence. The new network needs to make the connecting reach everywhere, calculation distribute everywhere and Intelligence support everywhere.

4. Security Considerations

TBD.

5. IANA Considerations

TBD.

6. References

6.1. Normative References

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

6.2. Informative References

[IEEE802.1Qbu]
IEEE, "Frame Preemption", 2015.

[IEEE802.1Qbv]
IEEE, "Enhancements for Scheduled Traffic", 2016.

Authors' Addresses

Peng Liu
China Mobile
Beijing 100053
China

Email: liupengyjy@chinamobile.com

Liang Geng
China Mobile
Beijing 100053
China

Email: gengliang@chinamobile.com

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

COIN
INTERNET-DRAFT
Intended Status: Informational
Expires: November 3, 2019

C. Sarathchandra
D. Trossen
InterDigital Europe, Ltd.
M. Boniface
University of Southampton
May 2, 2019

In-Network Computing for App-Centric Micro-Services
draft-sarathchandra-coin-appcentres-00

Abstract

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

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	Terminology	4
3.	Use Cases	4
3.1	Mobile Application Function Offloading	4
3.2	Collaborative Gaming	6
4	Enabling Technologies	6
4.1	Application Packaging	6
4.2	Service Deployment	7
4.3	Service Routing	8
4.4	Service Pinning	8
4.5	State Synchronisation	9
5	Security Considerations	9
6	IANA Considerations	9
7	Conclusion	9
8	References	9
8.1	Normative References	9
8.2	Informative References	10
	Authors' Addresses	11

1 Introduction

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

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

Interpreting any resource rich networked computing (and storage) capability not just as a pico or micro-data centre, but as an application-centric execution data centre (AppCentre), allows distributed execution of micro-services. These micro-services may then be deployed on the most appropriate AppCentre (edge/fog/cloud resources) to satisfy requirements under varying constraints. In addition, the high degree of distribution of application and data partitions, and compute resources offered by the execution environment decentralizes control between multiple cooperating parties (multi-technology, multi-domain, multi-ownership environments).

The emergence of AppCentres will democratise infrastructure and service provision to anyone with compute resources, while app-centric computing provides a truly pervasive user experience. Moreover, this will lead to new forms of application interactions and experiences based on cooperative AppCentres (pico-micro and large cloud data centres), in which applications are being designed, (micro-services) dynamically composed and executed.

2 Terminology

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

3. Use Cases

3.1 Mobile Application Function Offloading

The App-centric model increases the ubiquity of computing elements available for execution of application functions. Most functions can be categorised into any of three, "receiving", "processing" and "displaying" function groups. Partitioning an application into micro-services allows for denoting the application as a sequence of receiving->processing->displaying, for a flexible composition and distributed execution. Any device may realize one or more of the micro-services of an application and expose them to the execution environment. When the micro-service sequence is executed on a single device, the outcome is what you see today as applications running on mobile devices. However, if any of the three functions are terminated on the device (e.g., for optimising the experience), the execution of the rest of the functions may be moved to other suitable devices which have exposed the corresponding micro-services to the environment. The result of the latter is flexible mobile function offloading, for possible reduction of power consumption (e.g., offloading CPU intensive process functions to a remote server) or for improved device capabilities (e.g., moving display functions to a nearby smart TV).

The above scenario can be exemplified in an immersive gaming application, where a single user plays a game using a VR headset. The headset hosts functions that "display" frames to the user, as well as the functions for VR content processing and frame rendering combining with input data received from sensors in the VR headset. Once this application is partitioned into micro-services and deployed in an app-centric execution environment, only the "display" micro-service is left in the headset, while the compute intensive real-time VR content processing micro-services can be offloaded to a nearby resource rich home PC, for a better execution (faster and possibly higher resolution generation).

Figure 1 shows one realisation of the above scenario, where a 'DPR app' running on a mobile device (containing the partitioned Display(D), Process(P) and Receive(R) micro services) over an SDN network. The packaged applications are made available through a localised 'playstore server'. The application installation is realized as a 'service deployment' process (Section 4.2.), combining the local app installation with a distributed micro-service

deployment (and orchestration) on most suitable AppCentres ('processing server').

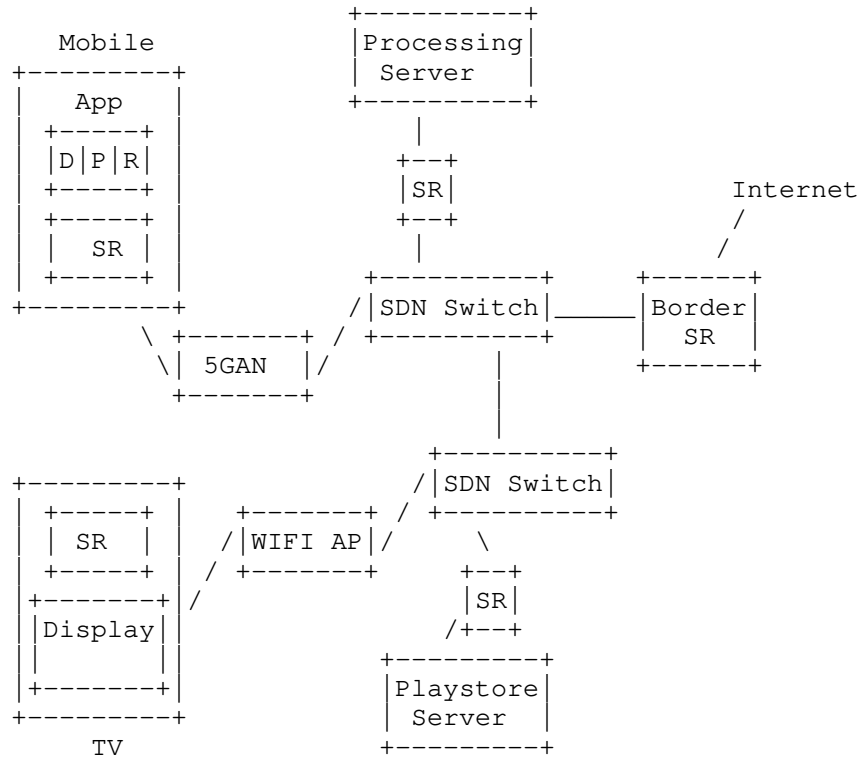


Figure 1: Application Function Offloading Example

Such localized deployment could, for instance, be provided by a visiting site, such as a hotel or a theme park. Once the 'processing' micro-service is terminated on the mobile device, the 'service routing' (SR) elements in the network (Section 4.3.) route requests to the previously deployed 'processing' micro-service running on the 'processing server' AppCentre over an existing SDN network.

Any app-centric execution environment MUST provide means for dynamically choosing the best possible micro-service sequence (i.e., chaining of micro-services) for a given application experience. Any solution SHOULD also provide methods for choosing and/or deploying the best possible instances of micro-services in the app-centric execution environment, for service routing, and for pinning the resources to the corresponding micro-services.

3.2 Collaborative Gaming

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

The same example in previous section can be envisaged from a multi-player gaming scenario. Here the micro-services that need to be executed cooperatively are executed in a localised and synchronised manner for player coordination and synchronizing interaction and state between collaborating players.

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

4 Enabling Technologies

4.1 Application Packaging

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

Decomposing micro-services of an application, and then executing them on peer execution points in AppCentresS (e.g., on an app-centric serverless runtime [SRVLESS]) can be done with design-time planning. Micro-service decomposition process involves, defining clear boundaries of the micro-service (e.g., using wrapper classes for handling input/output requests), which could be done by the application developer at design-time (e.g., through Android app packaging by including, as part of the asset directory, a service orchestration template [TOSCA] that describes the decomposed micro-services). Likewise, the peer execution points could be 'known' to the application (e.g., using well-known and fixed peer execution

points on AppCentreS) and incorporated with the micro-services by the developer at design-time.

Existing programming frameworks address decomposition and execution of applications centering around other aspects such as concurrency [ERLANG]. Application elements can be profiled using various techniques such as dynamic program analysis or dwarf application benchmarks. Such techniques can also be used for identifying and then defining micro-service boundaries at runtime (e.g., identifying a slice of an application that use a specific set of instructions and using the borders of which for defining the micro-service boundaries). Moreover, mechanisms for governance and discovery can be employed for 'unknown' peer execution points on AppCentreS with distributed loci of control.

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

4.2 Service Deployment

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

A combination of application installation procedure and orchestrated service deployment can be achieved by utilizing the application packaging with integrated service deployment templates described in Section 4.1 such that the application installation procedure on the

installing device is being extended to not only install the local application package but also extract the service deployment template for orchestrating with the localized infrastructure, using, for instance, REST APIs for submitting the template to the orchestrator.

4.3 Service Routing

Service routing within a combined compute and network infrastructure that will enable true end-to-end experiences across distributed application execution points provisioned on distant cloud, edge and device-centric resources (e.g., using ICN/name-based routing methods), is a key aspect of app-centric micro-service execution. Once the micro-services are packaged and deployed in such highly distributed micro-data centres, the routing mechanisms will ensure efficient information exchange (e.g., for satisfying application requirements) between corresponding micro-services within the multi-X execution environment.

Routing becomes a problem of routing the micro-service requests, not just packets, as done through IP. Traditionally, the combination of the Domain Naming Service (DNS) and IP routing has been used for this purpose. However, the advent of virtualisation with use cases such as those outlined above have made it challenging to further rely on the DNS. This is mainly down to the long delay in updating DNS entries to 'point' to the right micro-service instances. If one was to use the DNS, would be updating the DNS entries at a high rate, caused by the diversity of trigger, e.g., through movement. DNS has not been designed for such frequent update, rendering it useless for such highly dynamic applications. With many edge scenarios in the VR/AR space demanding interactivity and being latency-sensitive, efficient routing will be key to any solution.

Various ongoing work on service request forwarding [nSFF] with the service function chaining [RFC7665] framework as well as name-based routing [ICN5G][ICN4G] address some aspects described above. However, further extensions to those need to be considered supporting an app-centric model.

4.4 Service Pinning

Allocating the right resources to the right micro-services is a fundamental task when executing micro-services on such highly distributed app-centric micro-data centres (e.g., resource management in cloud [CLOUDFED]), particularly in the light of volatile resource availability as well as concurrent and highly dynamic resource access. Once the specific set of micro-services of an application has been identified, during the lifetime of the application, requirements (e.g., QoS) must be ensured by the execution environment. Therefore,

all micro-data centres and the execution environment will realize mechanisms for ensuring the utilization of specific resources within a pool of resources (i.e., resources in all app-centric micro-data centres), for a specific set of micro-services belonging to one application, while also ensuring integrity in the wider system.

4.5 State Synchronisation

Given the highly distributed nature of app-centric micro-services, their state exchange and synchronisation is a very crucial aspect for ensuring in-application and system wide consistency. Mechanisms that ensure consistency will ensure that data is synchronised with different spatial, temporal and relational data within a given time period.

5 Security Considerations

N/A

6 IANA Considerations

N/A

7 Conclusion

This draft positions the evolution of data centres as one of becoming execution centres for the app-centric experiences provided today by smartphones. With the proliferation of data centres closer to the end user in the form of edge-based micro data centres, we believe that app-centric experiences will ultimately be executed across those many, highly distributed execution points that this increasingly rich edge environment will provide. Although a number of activities are currently underway to address some of the challenges for realizing such AppCentre evolution, we believe that the proposed COIN research group will provide a suitable forum to drive forward the remaining research and its dissemination into working systems and the necessary standardization of key aspects and protocols.

8 References

8.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7665] Halpern, J., Ed., and C. Pignataro, Ed., "Service Function

Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.

8.2 Informative References

- [MSERVICE1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195–216). Springer, Cham.
- [MSERVICE2] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3), 42–52.
- [SRVLESS] C. Cicconetti, M. Conti and A. Passarella, "An Architectural Framework for Serverless Edge Computing: Design and Emulation Tools," 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, 2018, pp. 48–55. doi: 10.1109/CloudCom2018.2018.00024
- [TOSCA] Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. OASIS Standard. <<http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>>.
- [ERLANG] Armstrong, Joe, et al. "Concurrent programming in ERLANG." (1993). [SCOMPOSE] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations", *ACM Computing Surveys*, 46(4):1–34, Mar. 2014
- [SCOMPOSE] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations", *ACM Computing Surveys*, 46(4):1–34, Mar. 2014
- [SDEPLOY1] Lu, H., Shtern, M., Simmons, B., Smit, M., & Litoiu, M. (2013, June). Pattern-based deployment service for next generation clouds. In 2013 IEEE Ninth World Congress on Services (pp. 464–471). IEEE.
- [SDEPLOY2] Eilam, T., Elder, M., Konstantinou, A. V., & Snible, E.

(2011, May). Pattern-based composite application deployment. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops (pp. 217-224). IEEE.

- [nSFF] Trossen, D., Purkayastha, D., Rahman, A., "Name-Based Service Function Forwarder (nSFF) component within SFC framework", <<https://datatracker.ietf.org/doc/draft-trossen-sfc-name-based-sff>> (work in progress), April 2019.
- [ICN5G] Ravindran, R., Suthar, P., Trossen, D., Wang, C., White, G., "Enabling ICN in 3GPP's 5G NextGen Core Architecture", <<https://tools.ietf.org/html/draft-ravi-icnrg-5gc-icn-03>> (work in progress), March 2019.
- [ICN4G] Suthar, P., Jangam, Ed., Trossen, D., Ravindran, R., "Native Deployment of ICN in LTE, 4G Mobile Networks", <<https://tools.ietf.org/html/draft-irtf-icnrg-icn-lte-4g-03>> (work in progress), March 2019.
- [CLOUDFED] M. Liaqat, V. Chang, A. Gani, S. Hafizah Ab Hamid, M. Toseef, U. Shoaib, R. Liaqat Ali, "Federated cloud resource management: Review and discussion", Elsevier Journal of Network and Computer Applications, 2017.

Authors' Addresses

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

Email: Chathura.Sarathchandra@InterDigital.com

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

Email: Dirk.Trossen@InterDigital.com

INTERNET DRAFT

App-Centric Micro-Services

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

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

COIN
INTERNET-DRAFT
Intended Status: Informational
Expires: July 26, 2021

D. Trossen
Huawei
C. Sarathchandra
InterDigital Inc.
M. Boniface
University of Southampton
January 26, 2021

In-Network Computing for App-Centric Micro-Services
draft-sarathchandra-coin-appcentres-04

Abstract

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

We outline the key enabling technologies that needs to be provided for such evolution to be realized, including references to ongoing standardization efforts in key areas.

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	Terminology	4
3	Use Cases	4
4	Requirements	4
5	Enabling Technologies	5
5.1	Application Packaging	5
5.2	Service Deployment	7
5.3	Compute Inter-Connection at Layer 2	7
5.4	Service Routing	8
5.5	Constraint-based Forwarding Decisions	9
5.6	Collective Communication	10
5.7	State Synchronization	11
5.8	Dynamic Contracts	11
6	Overview of Relevant Standardization Efforts	11
7	Security Considerations	12
8	IANA Considerations	12
9	Conclusion	12
10	References	13
10.1	Normative References	13
10.2	Informative References	13
	Authors' Addresses	16

1 Introduction

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

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

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

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

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

2 Terminology

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

3 Use Cases

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

Originally being described in more detail in this draft, use cases are now gathered and described in more detail in [COIN-usecases], following a common taxonomy for their description. Specifically, the use cases for immersive devices and infrastructure services have guided the following requirements and technology selection, although this draft also applies to a number of industrial use cases as well. For more detail on those use cases overall, we refer the reader to [COIN-usecases].

4 Requirements

The following requirements are derived from the use cases in Section 5 and 6 in [COIN-usecases], serving as a guidance for the following discussions on enabling technologies in Section 5 of this document.

Req 1 - Service Routing: Any app-centric execution environment MUST provide means for routing of service requests between resources in the distributed environment.

Req 2 - Constraint-based Forwarding Decisions: Any app-centric execution environment MUST provide means for dynamically choosing the best possible micro-service sequence (i.e., chaining of micro-services) for a given application experience. Means for discovering suitable micro-service SHOULD be provided.

- Req 3 - Flow Affinity: Any app-centric execution environment **MUST** provide means for pinning the execution of a specific micro-service to a specific resource instance in the distributed environment.
- Req 4 - Deployment: Any app-centric execution environment **SHOULD** provide means for packaging micro-services for deployments in distributed networked computing environments. The packaging **SHOULD** include any constraints regarding the deployment of service instances in specific network locations or compute resources. Such packaging **SHOULD** conform to existing application deployment models, such as mobile application packaging, TOSCA orchestration templates or tar balls or combinations thereof.
- Req 5 - Synchronization: Any app-centric execution environment **MUST** provide means for real-time synchronization and consistency of distributed application states.
- Req 6 - Generic Invocation: Any app-centric execution environment **MUST** provide support for app/micro-service specific invocation protocols.
- Req 7 - Collective Communication: Any app-centric execution environment **SHOULD** utilize Layer 2 multicast transmission capabilities for responses to concurrent service requests.
- Req 8 - Orchestration: Any app-specific execution environment **SHOULD** expose means to specify the requirements for the tenant-specific compute fabric being utilized for the app execution. Any app-specific execution environment **SHOULD** allow for dynamic integration of compute resources into the compute fabric being utilized for the app execution; those resources include, but are not limited to, end user provided resources. Any app-specific execution environment **MUST** provide means to optimize the inter-connection of compute resources, including those dynamically added and removed during the provisioning of the tenant-specific compute fabric. Any app-specific execution environment **MUST** provide means for ensuring availability and usage of resources is accounted for.

5 Enabling Technologies

We now discuss a number of enabling technologies that address the requirements set out in Section 4.

5.1 Application Packaging

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

Decomposing micro-services of an application, and then executing them on peer execution points in AppCentreS (e.g., on an app-centric serverless runtime [SRVLESS]) can be done with design-time planning. Micro-service decomposition process involves, defining clear boundaries of the micro-service (e.g., using wrapper classes for handling input/output requests), which could be done by the application developer at design-time (e.g., through Android app packaging by including, as part of the asset directory, a service orchestration template [TOSCA] that describes the decomposed micro-services). Likewise, the peer execution points could be 'known' to the application (e.g., using well-known and fixed peer execution points on AppCentreS) and incorporated with the micro-services by the developer at design-time.

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

Therefore, with this app-centric model, application packaging can be done at runtime by constructing micro-service chains for satisfying

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

5.2 Service Deployment

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

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

The concept of 'intent-based networking' [IB_CONC] has been the focus of studies in the Network Management RG, allowing for declaratively stating the goals that a network shall meet. In relation to service deployment, intent-based concepts may be useful for the placement of service endpoints in a distributed environment under given service-specific constraints, e.g., on HW constraints for the execution of service endpoints or similar. This could also link into conveying service-specific constraints for the forwarding of information, as discussed in the following Section 5.5.

5.3 Compute Inter-Connection at Layer 2

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

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

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

5.4 Service Routing

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

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

of the request rather than individually.

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

Various ongoing work on service request forwarding [RFC8677] with the service function chaining [RFC7665] framework as well as name-based routing [ICN5G][ICN4G][ICNIP] addresses some aspects described above albeit with a focus on HTTP as the main invocation protocol. Extensions will be required to support other invocation protocols, such as GRPC or MPI (for distributed AI use cases). Proposals such as those in [DYN-CAST] suggest extensions to the IP anycast scheme to enable the flexible routing of service requests to one or more service instances. Common to those proposals is the use of a semantic identifier, often a service identifier akin to a URL, in the routing decision within the network.

Efforts existed in the IRTF, in the form of the Routing RG [RRG], to specifically study aspects of routing. The RRG concluded its work in 2014, but its possible revival has been suggested in ongoing discussions on routing evolution [FIPE] as a forum to study semantic-rich routing approaches.

5.5 Constraint-based Forwarding Decisions

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

for ensuring the utilization of specific resources within a pool of resources for a specific set of micro-services belonging to one application, while also ensuring integrity of the wider system. Application-layer solution frameworks, such as those developed as part of the Alto WG [ALTO], can be used for utilizing app/service-specific constraints.

In relation to the service routing capability, realized below the application layer and discussed in the previous sub-section, constraints may need to be introduced into the forwarding decisions for service requests. Such constraints will likely go beyond network load and latency, as often applied in scenarios such as load balancing in CDNs and as used in solutions such as [RFC7868]. Instead, those constraints could be app/service-specific and will need a suitable representation for the use within network nodes that are forwarding service requests, as also outlined in [DYN-CAST]. Moreover, individual router decisions (e.g., realized through matching operations such as min/max/equal over a constraint representation) may be coordinated to achieve a distribution of service requests among many service instances, effectively realizing a service scheduling capability in the network, optimized around service-specific constraints, not unlike many existing data centre service switching schemes.

As discussed already in Section 5.2, managing the constraints (for controlling the forwarding behaviour) may be linked into the concepts of intent-based networking [IB_CONC] to declaratively describe the goals of the forwarding or steering of traffic, while specific signaling protocols will need to be used to convey the actual constraints as well as the operations performed on them in order to fulfil the stated intent (or goals).

5.6 Collective Communication

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

Solutions at Layer 2 have been discussed in [ICNIP], enabling the delivery of service requests over a Layer2 forwarding solution. The

solution in [BIER-MC] utilizes the capabilities introduced by the BIER multicast overlay [BIER] to form such spontaneous multicast relations. Both approaches, however, are limited to the reachability of the respective transport technology, i.e., the Layer 2 or BIER overlay. Solutions over Layer 3 are currently limited to long-lived IP multicast groups or will need to rely on application-level solutions, mapping the group communication to replicated unicast forwarding operations at the network layer, such as done in the message passing interface [MPI], leading to significant inefficiencies through high peak-to-average ratios for the required transport network deployments.

5.7 State Synchronization

Given the highly distributed nature of app-centric micro-services, their state exchange and synchronization is a very crucial aspect for ensuring in-application and system wide consistency. Efforts such as those in [GAIA-X] aim at developing solutions for application areas such as distributed storage and data repositories. For this, mechanisms that ensure consistency will ensure that data is synchronized with different spatial, temporal and relational data within a given time period. From the perspective of support through in-network compute capabilities, such as provided through technologies like P4, it is important to consider what system and protocol support is required to utilize such in-network capabilities.

5.8 Dynamic Contracts

NOTE: left for future revision

6 Overview of Relevant Standardization Efforts

Requirement	Standardization Efforts
1-Service Routing	former Routing RG [RRG], possibly re-instated Dyncast [DYN-CAST] APN BoF [APN]
2-Constraint based Fwd Decision	Dyncast [DYN-CAST] EIGRP [RFC7868] Alto WG [ALTO] Intent-based Networking [IB_CONC]
3-Flow Affinity	Dyncast [DYN-CAST]

4-Deployment	ETSI NFV MANO Intent-based Networking [IB_CONC]
5-Synchroni- zation	GAIA-X [GAIA-X]
6-Generic invocation	Internet Services over ICN [ICNIP]
7-Collective Communication	BIER WG [BIER] Internet Services over ICN (ICNIP) Multicast for HTTP over BIER [BIER-MC]
8-Orchestr.	3GPP 5GLAN [SA2-5GLAN] and ETSI MANO

Figure 1: Mapping of Requirements to Standardization Efforts

7 Security Considerations

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

8 IANA Considerations

N/A

9 Conclusion

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

Based on the requirements relevant to those key technologies, derived

from the COIN use cases, we have further provided an evaluation of ongoing and related efforts in the relevant areas of study. We believe that this analysis can be useful for positioning work discussed and pursued in COIN against those ongoing efforts. Furthermore, it may guide those interested in the respective key technologies to create appropriate linkages to those ongoing efforts elsewhere.

10 References

10.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.
- [RFC7665] Halpern, J., Ed., and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

10.2 Informative References

- [MSERVICE1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering* (pp. 195–216). Springer, Cham.
- [MSERVICE2] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3), 42–52.
- [SRVLESS] C. Cicconetti, M. Conti and A. Passarella, "An Architectural Framework for Serverless Edge Computing: Design and Emulation Tools," 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, 2018, pp. 48–55. doi: 10.1109/CloudCom2018.2018.00024
- [TOSCA] Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. OASIS Standard. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>.
- [ERLANG] Armstrong, Joe, et al. "Concurrent programming in ERLANG." (1993).

- [SCOMPOSE] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations", ACM Computing Surveys, 46(4):1-34, Mar. 2014
- [SDEPLOY1] Lu, H., Shtern, M., Simmons, B., Smit, M., & Litoiu, M. (2013, June). Pattern-based deployment service for next generation clouds. In 2013 IEEE Ninth World Congress on Services (pp. 464-471). IEEE.
- [SDEPLOY2] Eilam, T., Elder, M., Konstantinou, A. V., & Snible, E. (2011, May). Pattern-based composite application deployment. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops (pp. 217-224). IEEE.
- [RFC8677] Trossen, D., Purkayastha, D., Rahman, A., "Name-Based Service Function Forwarder (nSFF) Component within a Service Function Chaining (SFC) Framework", RFC 8677, November 2019.
- [ICN5G] Ravindran, R., Suthar, P., Trossen, D., Wang, C., White, G., "Enabling ICN in 3GPP's 5G NextGen Core Architecture", <https://www.ietf.org/archive/id/draft-irtf-icnrg-5gc-icn-04>, (work in progress), January 2021.
- [ICN4G] Suthar, P., Jangam, Ed., Trossen, D., Ravindran, R., "Native Deployment of ICN in LTE, 4G Mobile Networks", <https://tools.ietf.org/html/draft-irtf-icnrg-icn-lte-4g-08>, (work in progress), January 2021.
- [CLOUDFED] M. Liaqat, V. Chang, A. Gani, S. Hafizah Ab Hamid, M. Toseef, U. Shoaib, R. Liaqat Ali, "Federated cloud resource management: Review and discussion", Elsevier Journal of Network and Computer Applications, 2017.
- [GRPC] High performance open source universal RPC framework, <https://grpc.io/>
- [MPI] A. Vishnu, C. Siegel, J. Daily, "Distributed TensorFlow with MPI", <https://arxiv.org/pdf/1603.02339.pdf>
- [FCDN] M. Al-Naday, M. J. Reed, J. Riihijarvi, D. Trossen, N. Thomos, M. Al-Khalidi, "fCDN: A Flexible and Efficient CDN Infrastructure without DNS Redirection of Content Reflection", <https://arxiv.org/pdf/1803.00876.pdf>
- [DYN-CAST] P. Liu, P. Willis, D. Trossen, "Dynamic-Anycast (Dyncast) Use Cases and Problem Statement",

<https://tools.ietf.org/html/draft-liu-rtgwg-dyncast-ps-usecases-00>, (work in progress), January 2021

[SA2-5GLAN] 3gpp-5glan, "SP-181129, Work Item Description, Vertical_LAN(SA2), 5GS Enhanced Support of Vertical and LAN Services", 3GPP,
http://www.3gpp.org/ftp/tsg_sa/TSG_SA/Docs/SP-181120.zip

[COIN-usecases] I. Kunze, K. Wehrle, D. Trossen, "Use Cases for In-Network Computing", <https://tools.ietf.org/html/draft-kunze-coin-industrial-use-cases-04>, (work in progress), January 2021.

[APN] Application-Aware Networking (APN), IETF BoF,
<https://datatracker.ietf.org/group/apn/about/> (work in progress), January 2021.

[RFC7868] D. Davage et al. , "Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)", RFC 7868, May 2016,
<https://tools.ietf.org/html/rfc7868>

[ALTO] Application-Layer Traffic Optimization, IETF Working Group, <https://datatracker.ietf.org/wg/alto/about/>, January 2021

[GAIA-X] Gaia-X, "GAIA-X: A Federated Data Infrastructure for Europe", accessed January 2021, <https://www.data-infrastructure.eu/GAIA-X/Navigation/EN/Home/home.html>, January 2021

[ICNIP] D. Trossen, S. Robitzsch, M. Reed, M. Al-Naday, J. Riihijarvi, "Internet Services over ICN in 5G LAN Environments", <https://tools.ietf.org/html/draft-trossen-icnrg-internet-icn-5glan-04>, (work in progress), January 2021

[BIER-MC] D. Trossen, A. Rahman, C. Wang, T. Eckert, "Applicability of BIER Multicast Overlay for Adaptive Streaming Services", <https://tools.ietf.org/html/draft-ietf-bier-multicast-http-response-05>, (work in progress), January 2021

[BIER] Bit Indexed Explicit Replication, IETF Working Group,
<https://datatracker.ietf.org/wg/bier/about/>, January 2021

[RRG] Routing RG (concluded), IRTF Research Group,
<https://trac.ietf.org/trac/irtf/wiki/RoutingResearchGroup>, accessed January 2021

[FIPE] Future Internet Protocol Evolution (FIPE) side meeting,
<https://github.com/FIPE-Study/IETF109-Side-Meeting-FIPE>,
November 2020

[IB_CONC] A. Clemm, L. Ciavaglia, L. Granville, J. Tantsura,
"Intent-Based Networking - Concepts and Definitions",
<https://datatracker.ietf.org/doc/draft-irtf-nmrg-ibn-concepts-definitions/>, (work in progress), September 2020

Authors' Addresses

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

Email: Dirk.Trossen@Huawei.com

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

Email: Chathura.Sarathchandra@InterDigital.com

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

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