

T2T Research Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

J. Hong
Y-G. Hong
ETRI
X. de Foy
InterDigital Communications
M. Kovatsch
Huawei Technologies Duesseldorf GmbH
E. Schooler
Intel
D. Kutscher
University of Applied Sciences Emden/Leer
July 08, 2019

Problem Statement of IoT integrated with Edge Computing
draft-hong-t2trg-iot-edge-computing-00

Abstract

This document describes new challenges such as strict latency, uplink cost, uninterrupted services, privacy and security, for IoT services originated from the IoT environmental changes. In order to address those new challenges, the integration of Edge computing and IoT has been emerged as a promising solution. This document describes the concept of IoT integrated with Edge computing as well as the state-of-the-art of IoT Edge computing. It also proposes an architecture of IoT Edge computing. The direction of Edge computing for IoT should be discussed in the IETF/IRTF.

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	3
2. Conventions and Terminology	3
3. Background	4
3.1. Internet of Things (IoT)	4
3.2. Cloud computing	4
3.3. Edge computing	5
4. New challenges of IoT	5
4.1. Strict Latency and Jitter	5
4.2. Uplink Cost	6
4.3. Uninterrupted Services	6
4.4. Privacy and Security	6
5. IoT integrated with Edge Computing	7
5.1. IoT Data in Edge Computing	7
5.1.1. Data Storage	8
5.1.2. Data Processing	8
5.1.3. Data Analyzing	8
5.2. IoT Device Management in Edge Computing	9
6. Architecture of IoT integrated with Edge Computing	9
7. State-of-the-art of IoT Edge Computing	11
7.1. Common aspects of IoT edge computing service platforms	11
7.2. Use Cases of IoT Edge Computing	12
8. Security Considerations	14
9. Acknowledgements	14
10. References	14
10.1. Normative References	14
10.2. Informative References	14
Appendix A. Overview of the IoT Edge Computing	17
A.1. Open Source Projects	17
A.1.1. Gateway/CPE Platforms	17
A.1.2. Edge Cloud Management Platforms	18
A.1.3. Related Projects	19

A.2. Products	19
A.2.1. IoT Gateways	19
A.2.2. Edge Cloud Platforms	20
A.3. Standards Initiatives	20
A.3.1. ETSI Multi-access Edge Computing	20
A.3.2. Edge Computing Support in 3GPP	21
A.3.3. OpenFog Consortium	22
A.3.4. Related Standards	22
A.4. Research Projects	22
A.4.1. Named Function Networking	22
A.4.2. 5G-CORAL	23
A.4.3. FLAME	23
Authors' Addresses	24

1. Introduction

Nowadays, most IoT services are based on Cloud computing since it can provide virtually unlimited storage and processing power. The integration of IoT with Cloud computing brings many advantages such as flexibility, efficiency, and ability to store and use data.

However, the IoT environment is changing in such a way that vast amounts of data are created at edge/local networks and about a half of data is stored, processed, analyzed and acted upon close to the data producer. Thus, emerging IoT services introduce new challenges that cannot be addressed by today's centralized Cloud computing models alone.

In this document, we describe new challenges for emerging IoT services such as strict latency, uplink cost, uninterrupted services, privacy and security due to the IoT environmental changes.

In order to address those new challenges for IoT services, the integration of Edge computing with IoT has been emerged as a promising solution. In this document, we describe the concept of IoT integrated with Edge computing as well as the state-of-the-art of IoT Edge computing and propose an architecture of IoT Edge computing. The purpose of this document is to bring up the issues of Edge computing for IoT services in IETF/IRTF.

2. Conventions and 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 [RFC2119].

3. Background

3.1. Internet of Things (IoT)

Since the phrase 'Internet of Things (IoT)' was coined by Kevin Ashton in 1999 working on Radio-frequency identification (RFID) technology at the Auto-ID Center of the Massachusetts Institute of Technology (MIT) [Ashton], the concept of IoT has been that things connected to the Internet can send and receive information collected by sensors without human intervention, where things are various embedded systems such as home appliances, mobile equipment, wearable devices, etc. IoT has become one of the notable innovations playing an important role in our daily lives [Lin]. IoT is generally characterized by real world small things that are widely distributed but have limited storage and processing power, which involve concerns regarding reliability, performance, security, and privacy.

3.2. Cloud computing

Cloud computing have been defined in [NIST]: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Cloud computing has been a predominant technology which has virtually unlimited capacity in terms of storage and processing power. The availability of virtually unlimited storage and processing capabilities at low cost enabled the realization of a new computing model, in which virtualized resources can be leased in an on-demand fashion, being provided as general utilities. Companies like Amazon, Google, Facebook, etc. widely adopted this paradigm for delivering services over the Internet, gaining both economical and technical benefits [Botta].

Now with IoT, we will reach the era of post-Clouds where unprecedented volume and variety of data will be generated by things at edge/local networks and many applications will be deployed on the edge networks to consume these IoT data. Some of the applications may need very short response times, some may contain personal data, and others may generate vast amounts of data. Today's Cloud based service models are not suitable for these applications.

It is predicted that by 2019, 45% of the data created in IoT will be stored, processed, analyzed and acted close to, or at the edge of the network and about 50 billion devices will connect to the Internet by 2020 [Evans]. So, moving all data from edge/local networks to the cloud data center may not be an efficient way anymore to process vast amounts of data.

In Cloud computing, users traditionally only consumed IoT data through Cloud services. Now, however, users are also producing IoT data with their mobile devices. This change requires more functionality at edge/local networks [Shi].

3.3. Edge computing

Edge computing is a new paradigm in which substantial computing and storage resources are placed at the Internet's edge in close proximity to mobile devices or sensors so that computing happens near data sources [Mahadev]. It works on both downstream data on behalf of cloud services and upstream data on behalf of IoT services. An edge device is any computing or networking resource residing between data sources and cloud-based datacenters. In Edge computing, the end device not only consumes data but also produces data. And at the network edge, devices not only request services and information from the cloud but also handle computing tasks including processing, storage, caching, and load balancing on data sent to and from the cloud [Shi].

The definition of Edge computing from ISO is 'Form of distributed computing in which significant processing and data storage takes place on nodes which are at the edge of the network' [ISO_TR]. And the similar concept of Fog computing from Open Fog Consortium is 'A horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum' [OpenFog]. Based on these definitions, we can summarize a general philosophy of Edge computing as "Distribute the required functions close to users and data".

4. New challenges of IoT

As the IoT is maturing, systems are converging, deployments are growing, and IoT technology is used with more and more demanding applications such as industrial, automotive, or healthcare. This leads to new challenges for the IoT. In particular, the amount of data created at the edge is expected to be vast. Industrial machines such as laser cutters already produce over 1 terabyte per hour, the same applies for autonomous cars [NVIDIA]. 90% of IoT data is expected to be stored, processed, analyzed, and acted upon close to the source [Kelly], as Cloud Computing models alone cannot address the new challenges [Chiang].

4.1. Strict Latency and Jitter

Many industrial control systems, such as manufacturing systems, smart grids, oil and gas systems, etc., often require stringent end-to-end latency between the sensor and control node. While some IoT

applications may require latency below a few tens of milliseconds [Weiner], industrial robots and motion control systems have use cases for cycle times in the order of microseconds [_60802]. An important aspect for real-time communications is not only the latency, but also guarantees for jitter. This means control packets need to arrive with as little variation as possible with a strict deadline. Given the best-effort characteristics of the Internet, this challenge is virtually impossible to address with a pure cloud model, when also taking the further challenges into account.

4.2. Uplink Cost

Many IoT deployments are not challenged by a constrained network bandwidth to the cloud. The fifth generation mobile networks (5G) and Wi-Fi 6 both theoretically top out at 10 gigabits per second (i.e., 4.5 terabyte per hour), which enables high-bandwidth uplinks. However, the resulting cost for high-bandwidth connectivity to upload all data to the cloud is unjustifiable and impractical for most IoT applications.

4.3. Uninterrupted Services

Many IoT devices such as sensors, data collectors, actuators, controllers, etc. have very limited hardware resources and cannot rely solely on their limited resources to meet all their computing and/or storage needs. They require reliable, uninterrupted services to augment their capabilities in order to fulfill their application tasks. This is hard and partly impossible to achieve with cloud services for systems such as vehicles, drones, or oil rigs that have intermittent network connectivity.

4.4. Privacy and Security

When IoT services are deployed at home, personal information can be learned from detected usage data. For example, one can extract information about employment, family status, age, and income by analyzing smart meter data [ENERGY]. Policy makers started to provide frameworks that limit the usage of personal data and put strict requirements on data controllers and processors. However, data stored indefinitely in the cloud also increases the risk of data leakage, for instance, through attacks on rich targets.

Industrial systems are often argued to not have privacy implications, as no personal data is gathered. Yet data from such systems is often highly classified, as one might be able to infer trade secrets such as the setup of production lines. Hence, the owner of these systems are generally reluctant to upload related IoT to the cloud.

5. IoT integrated with Edge Computing

As described in section 4, there are new challenges for supporting emerging IoT services and Edge computing is one of the candidates to satisfy these challenges. The motivation for IoT Edge computing was discussed at an Edge computing discussion in IETF/IRTF meetings as follows: [IETF_Edge]

- o Delay-sensitive
- o High-volume
- o Trust-sensitive
- o (Intermittently) disconnected
- o Energy-challenged
- o Costly to transmit

As we described at previous sections, the above motivation for IoT Edge computing could directly be benefits of Edge computing in the IoT environment. The above motivation for IoT Edge computing is mainly related to IoT data and other motivation for IoT Edge computing can exist as other aspects of networking and communication.

In spite of its benefits, Edge computing in IoT services has challenges such as programmability, naming, data abstraction, service management, privacy and security and optimization metrics.

Edge computing can support IoT services independently of Cloud computing. However, Edge computing is increasingly connected to Cloud computing in most IoT systems for processing and storing data. Thus, the relationship of Edge Computing to Cloud Computing is also another challenge of Edge Computing in IoT [ISO_TR].

5.1. IoT Data in Edge Computing

As an aspect of IoT, Edge computing can provide many capabilities for IoT services because IoT systems are based on sensors and actuator devices in edge area and IoT data generated from sensors and actuator devices are gathered through a gateway [ISO_TR]. Besides on IoT data, other functions such as computing, control and network functions are also very remarkable to support IoT services. In this document, we will first concentrate on IoT data's aspect since the benefit of Edge computing with IoT data is very big in use cases.

5.1.1. Data Storage

As tremendous IoT sensors, IoT actuators, and IoT devices are connected to the Internet, IoT data volume from these things are expected to increase explosively. And it is expected that much of this high volume of IoT data is produced and/or consumed within edge/local networks, not to traverse through cloud networks. Until now, most IoT data generated by IoT things is transferred and accumulated in a remote server and storage of IoT data in a remote server is expensive in transmission and storage. To mitigate the cost of transmission and storage, it is required to divide IoT data into two types of data; one is stored in edge/local networks and the other is stored in cloud networks. The effect of Edge computing is revealed with the handling IoT data in edge/local networks.

5.1.2. Data Processing

Until now, most network equipment such as routers, gateways, and switches just forward data delivered from other network devices without reading or modifying the content. In end-to-end communication, data is acknowledged and proceed at a final corresponding node. This is a typical usage of cloud computing and a client-server communication. But, in the IoT environment, some IoT data will be transferred to a cloud network and some will be delivered to an edge node. The main reason of this separation is to provide real-time processing and security enhancement in IoT. Although there are many new technologies to reduce the delay and transmission time, it is not easy to guarantee real-time processing. The typical use case of this requirement is industrial Internet and smart factory. Even though there are also several solutions to provide security in IoT, the more basic rule is not to expose the privacy data to public networks. If we separate IoT data into private and non-private data, and keep private data within an edge/local network not to expose them in a public network, the security and privacy in IoT can be addressed by the separation.

5.1.3. Data Analyzing

If it is possible to separate IoT data in edge/local networks and cloud networks, Edge computing can do more functions with IoT data in edge/local networks. Because Edge computing has the capabilities to handle IoT data in edge/local networks, it is also possible to analyze IoT data to provide enhanced IoT services such as intelligence. To analyze IoT data in an edge/local network, it is required to have comparatively processing performance and this requirement is not obstacle to deploy Edge computing due to the development of H/W and S/W.

5.2. IoT Device Management in Edge Computing

If we consider new challenges of IoT services, not only the big volume of IoT data but also the massive number of IoT things can be a critical problem. Even though, we acknowledge this future problem, the Internet architecture originally has the capability of scalability and it will mitigate scalability issue in the IoT environment. But, we cannot estimate the number of IoT things in the future and we cannot guarantee the Internet architecture still sustain the scalability issue in the IoT environment. Edge computing will separate the scalability domain into edge/local networks and outside network (e.g., cloud networks) and this separation of scalability domain can provide more efficient way to tackle the massive number of IoT things.

Because Edge computing can handle IoT data in an edge area and store the IoT data in an edge node, and proceed IoT data if it is needed, it can also separate the management domain into two parts. Edge Computing can concentrate on management of IoT things in an edge area and cooperate with the management of other outside networks.

6. Architecture of IoT integrated with Edge Computing

When we consider the implementation and deployment of Edge computing, it can be mainly referred to an IoT Gateway. The role of an IoT Gateway is to provide multiple accesses to the heterogeneous IoT devices/sensors, handling IoT data and delivering the IoT data to the final destinations such as cloud networks. Similar to an IoT Gateway, an Edge computing architecture as an edge computing node provides downside connectivity to IoT sensors and devices (southbound connectivity) and upside connectivity to cloud networks (northbound connectivity). Also, the architecture provides the function of data storage. Beside these functions, the Edge computing architecture should provide the computing functions, such as data processing, data analyzing, and additional function of intelligence.

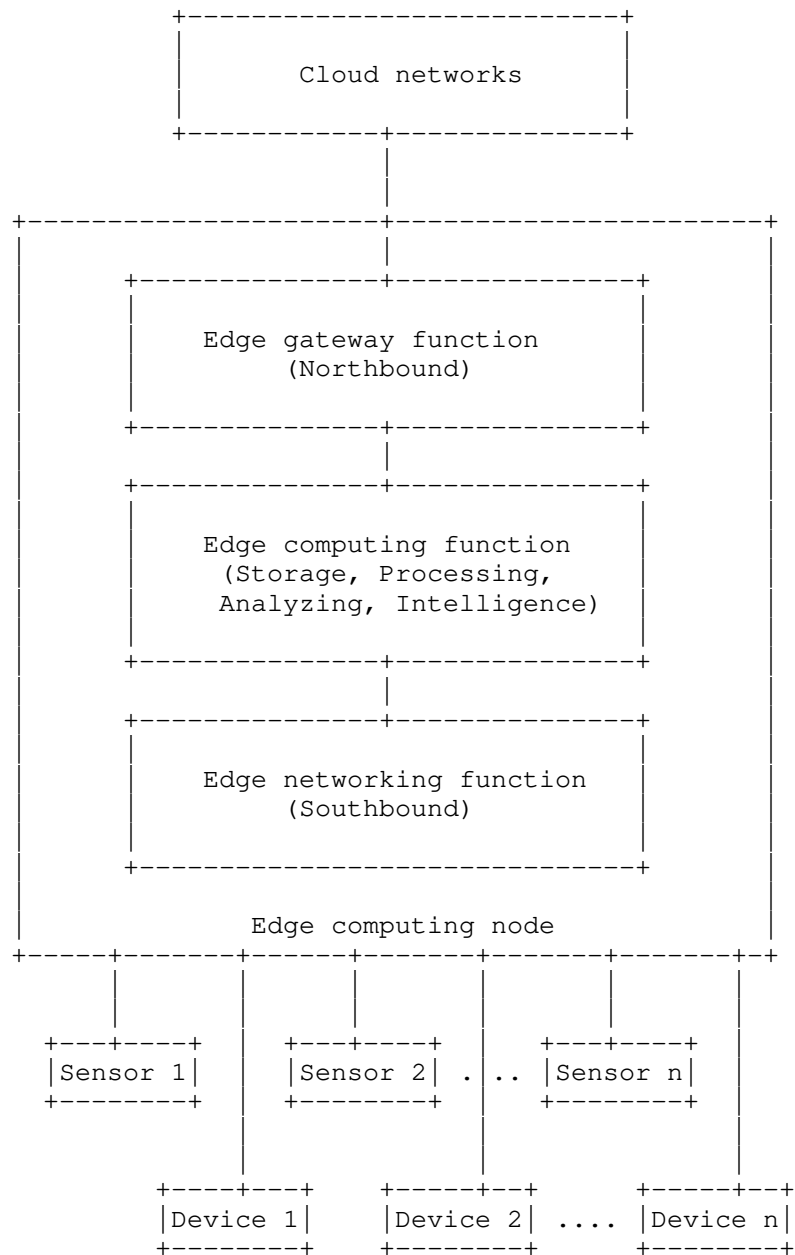


Figure 1: Architecture of IoT integrated with Edge computing

It is expected that the Edge computing architecture will play an important role to deploy new IoT services with integration to big data and AI services.

7. State-of-the-art of IoT Edge Computing

7.1. Common aspects of IoT edge computing service platforms

This section provides an overview of today's IoT Edge Computing field, based on a limited review of standards, research, open-source and proprietary products in Appendix A. Common aspects of IoT edge computing service platforms are summarized here:

Computing devices: IoT gateways (Appendix A.2.1, Appendix A.1.1) represent a common class of IoT edge computing products, where the gateway is providing a local service on customer premises, and is remotely managed through a cloud service. IoT communication protocols are typically used between IoT devices and the gateway, including CoAP, MQTT and many specialized IoT protocols, while the gateway communicates with the distant cloud using typically HTTP and WebSocket.

Virtualization platforms enable the deployment of virtual edge computing functions, including IoT gateway software, on servers in the mobile network infrastructure (at base station and concentration points), in edge datacenters (in central offices) or regional datacenters located near central offices.

End devices as computing devices are envisioned in fog architecture and research projects, but are not commonly used as such today.

Service models: Physical or virtual IoT gateways can host application programs built using an SDK.

Edge cloud system operators host their customers' applications VMs or containers on servers located in or near access networks. These application have access to edge service APIs. For example, mobile network services include radio network information, location, bandwidth management.

In a cloud-like service model, service providers consume low-level edge platform APIs and offer high-level APIs to their own customers' applications. This cloud-like model can be offered as an edge cloud service, or as an hybrid cloud service covering edge and distant cloud.

Management: Life cycle management of services and applications on physical IoT gateways is often cloud-based. Edge cloud management platforms and products (Appendix A.1.2, Appendix A.2.2) adapt cloud management technologies (e.g. kubernetes) to the edge cloud, i.e. to smaller, distributed computing devices running outside a controlled data center. Services and application life-cycle is typically using a NFV-like management and orchestration model.

Communication services: The platform typically includes services to advertise or consume APIs, and enables communicating with local and remote endpoints. The service platform is typically extensible by edge applications, since they can advertise an API that other edge applications can consume. IoT communication services include protocols translation, analytics and transcoding. Communication between edge computing devices is enabled in tiered deployments or distributed deployments.

Storage models: An edge cloud platform may enable pass-through without storage, local storage (e.g. on IoT gateways). Some edge cloud platforms use a distributed form of storage, e.g. an ICN network or a distributed storage platform. External storage, e.g. on databases in distant or local IT cloud, is typically used for filtered data deemed worthy of long term storage, or in some cases for all data, for example when required for regulatory reasons.

Computing models: Stateful computing is supported on platforms hosting native programs, VMs or containers. Stateless computing is supported on platforms providing a "serverless computing" service (a.k.a. function-as-a-service), or on systems based on named function networking.

Network traffic patterns: Network traffic is typically high volume uplink with throttling by edge computing devices (or deferred to off-peak hours or using physical shipping); and downlink for control and software updates.

7.2. Use Cases of IoT Edge Computing

Smart Constructions: In traditional construction domain, there are many heavy equipment and machineries and dangerous elements. Even though human pay attention to risk elements, it is not easy to avoid them. If some accidents are happened in a construction site, it causes a loss of lives and property. Thus, there have been many trials in a construction area to protect lives and property. Measurements of noise, vibration, and gas in a construction area are recorded on a remote server and reported to an inspector. Today, data produced by such measurements is collected by a gateway in a construction area and transferred to a

remote server. This incurs transmission cost, e.g. over a LTE connection, and storage cost, e.g. when using Amazon Web Services. When an inspector wants to investigate some accidents, he checks the information stored in a server. If we deploy Edge computing in a construction area, the sensor data can be processed and analyzed in a gateway located within or near a construction area. And with the help of a statistical analysis or machine learning technologies, we can predict future accidents in advance and this prediction can be used as an alarm in a construction area and a notification to an inspector. To determine the exact cause of some accident, not only sensor data but also audio and video data are transferred to a remote server or cloud networks. In this case, the data volume of audio and video is quite big and the cost of transmission can be a problem. If Edge computing can predict the time of accident, it can reduce the data volume of transmission; in general period, it can transmit the audio and video data with a low resolution/degree and in emergent period, it transmits the audio and video data with a high resolution/degree. By adjusting the resolution/degree of audio and video data, it can reduce transmission cost significantly.

Smart Grid: In future smart cities, Smart grids will be critical in ensuring availability and efficiency for energy saving and control in city-wide electricity management. Edge computing is expected to play a significant role in those systems to improve transmission efficiency of electricity, react and restore for power disturbances, reduce operation cost, reuse renewable energy effectively, save energy of electricity for future usage, and so on. In addition, Edge computing can help monitoring power generation and power demands, and making electrical energy storage decisions in the Smart grid system.

Smart Water System: The Water system is one of the most important aspects for building smart city. Effective use of water, and cost-effective and environment-friendly treatment of water are critical for water control and management. This can be facilitated by Edge computing in Smart water systems, to help monitor water consumption, transportation, prediction of future water use, and so on. For example, water harvesting and ground water monitoring will be supported from Edge computing. Also, a Smart water system is able to analyze collected information related to water control and management, control the reduction of water losses and improve the city water system through Edge computing.

Smart Buildings: [TBA]

Smart Cities: [TBA]

Connected Vehicles: [TBA]

8. Security Considerations

[TBA]

9. Acknowledgements

The authors would like to thank Joo-Sang Youn and Akbak Rahman for their valuable comments and suggestions on this document.

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>>.

10.2. Informative References

- [Ashton] Ashton, K., "That Internet of Things thing", RFID J. vol. 22, no. 7, pp. 97-114, 2009.
- [Lin] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications", IEEE Internet of Things J. vol. 4, no. 5, pp. 1125-1142, Oct. 2017.
- [NIST] Mell, P. and T. Grance, "The NIST definition of Cloud computing", Natl. Inst. Stand. Technol 53 (6), pp. 50, 2009.
- [Botta] Botta, A., Donato, W., Persico, V., and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey", Future Gener. Comput. Syst. 56, pp. 684-700, 2016.
- [Evans] Evans, D., "The Internet of Things: How the next evolution of the Internet is changing everything", CISCO White Paper vol. 1, pp. 1-11, 2011.
- [Shi] Shi, W., Cao, J., Zhang, Q., Li, Y., and L. Xu, "Edge computing: vision and challenges", IEEE Internet of Things J. vol. 3, no. 5, pp. 637-646, Oct. 2016.

- [Mahadev] Satyanarayanan, M., "The Emergence of Edge Computing", Computer vol. 50, no. 1, pp. 30-39, Jan. 2017.
- [Chiang] Chiang, M. and T. Zhang, "Fog and IoT: An overview of research opportunities", IEEE Internet Things J. vol. 3, no. 6, pp. 854-864, Dec. 2016.
- [Weiner] Weiner, M., Jorgovanovic, M., Sahai, A., and B. Nikolic, "Design of a low-latency, high-reliability wireless communication system for control applications", IEEE Int. Conf. Commun. (ICC) Sydney, NSW, Australia, pp. 3829-3835, 2014.
- [Kelly] Kelly, R., "Internet of Things Data to Top 1.6 Zettabytes by 2022", <https://campustechnology.com/articles/2015/04/15/internet-of-thingsdata-to-top-1-6-zettabytes-by-2020.aspx>, April 2016.
- [ISO_TR] "Information Technology - Cloud Computing - Edge Computing Landscape", ISO/IEC TR 23188, April 2018.
- [OpenFog] "OpenFog Reference Architecture for Fog Computing", OpenFog Consortium, Feb. 2017.
- [IETF_Edge] Kutscher, D. and E. Schooler, "IoT Edge Computing Discussion @ IETF-98", slides-99-t2trg-edge-computing-summary-of-chicago-discussion-and-ideas-for-next-steps-00, Mar. 2017.
- [ETSI_MEC_03] ETSI, "Mobile Edge Computing (MEC); Framework and Reference Architecture", ETSI GS 003, 2019, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf>.
- [ETSI_MEC_02] ETSI, "Multi-access Edge Computing (MEC); Phase 2: Use Cases and Requirements", ETSI GS 002, 2016, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/02.01.01_60/gs_MEC002v020101p.pdf>.
- [_3GPP.23.501] 3GPP, "System Architecture for the 5G System", 3GPP TS 23.501, 2019, <<http://www.3gpp.org/ftp/Specs/html-info/23501.htm>>.

- [ETSI_MEC_WP_28]
ETSI, "MEC in 5G networks", White Paper , June 2018,
<[https://www.etsi.org/images/files/ETSIWhitePapers/
etsi_wp28_mec_in_5G_FINAL.pdf](https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf)>.
- [Linux_Foundation_Edge]
Linux Foundation, "Linux Foundation Edge", Portal , 2019,
<<https://www.lfedge.org/>>.
- [StarlingX]
OpenStack Foundation, "StarlingX", Portal , 2019,
<<https://www.starlingx.io/>>.
- [Sifalakis]
Sifalakis, M., Kohler, B., Scherb, C., and C. Tschudin,
"An Information Centric Network for Computing the
Distribution of Computations", Proceedings of the 1st
international conference on Information-centric
networking INC '14, 2014.
- [FLAME]
Horizon 2020 Programme, "FLAME Project", Portal , 2019,
<<https://www.ict-flame.eu/>>.
- [POINT]
Horizon 2020 Programme, "IP Over ICN - the better IP
(POINT) Project", Portal , 2019,
<<https://www.point-h2020.eu/>>.
- [_5G-CORAL]
Horizon 2020 Programme, "5G Convergent Virtualised Radio
Access Network Living at the Edge (5G-CORAL) Project",
Portal , 2019, <<http://5g-coral.eu/>>.
- [OpenEdgeComputing]
"Open Edge Computing", Portal , 2019,
<<http://openedgecomputing.org/>>.
- [IEEE-1934]
IEEE, "FOG - Fog Computing and Networking Architecture
Framework", Portal , 2019,
<<https://standards.ieee.org/standard/1934-2018.html>>.
- [NVIDIA]
Grzywaczewski, A., "Training AI for Self-Driving Vehicles:
the Challenge of Scale", NVIDIA Developer Blog , October
2017, <[https://devblogs.nvidia.com/
training-self-driving-vehicles-challenge-scale/](https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/)>.

- [_60802] IEEE 802, "Use Cases IEC/IEEE 60802 V1.3", IEC/IEEE 60802 , September 2018, <<http://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0818-v13.pdf>>.
- [ENERGY] Beckel, C., Sadamori, L., Staake, T., and S. Santini, "Revealing Household Characteristics from Smart Meter Data", Energy vol. 78, pp. 397-410, December 2014, <<https://www.vs.inf.ethz.ch/publ/papers/beckel-2014-energy.pdf>>.

Appendix A. Overview of the IoT Edge Computing

This list of initiatives, projects and products aim to provide an overview of the IoT Edge Computing. Our goal is to be representative rather than exhaustive. Please help us complete this overview by communicating with us about entries we have missed.

A.1. Open Source Projects

A.1.1. Gateway/CPE Platforms

EdgeX Foundry, Home Edge, Edge Virtualization Engine are Linux Foundation projects ([Linux_Foundation_Edge]) aiming to provide a platform for edge computing devices. Such an open source platform can, for example, host proprietary programs currently run on IoT gateway products (Appendix A.2). EdgeX Foundry develops an edge computing framework running on the IoT gateway. Home Edge develops an edge computing framework especially dedicated to home computing devices, controlling home appliances, sensors, etc., and enabling AI applications, especially distributed and parallel machine learning. The Edge Virtualization Engine (EVE) project develops a virtualization platform (for VMs and containers) designed to run outside of the datacenter, in an edge network; EVE is deployed on bare-metal hardware.

Computing devices: Hardware support for EdgeX and EVE is similar: they support x86 and ARM-based computing devices; A typical target can be a Linux Raspberry Pi with 1GB RAM, 64bit CPU, 32GB storage.

Service platform: EdgeX uses a micro-service architecture. Micro-services on the gateway are connected together, and to outside applications, through REST, or messaging technologies such as MQTT, AMQP and 0MQ. The gateway can communicate with external backend applications or other gateways (north-south in tiered deployments or east-west in more distributed deployments). Gateway-device communication can use a wide range of IoT protocols. "Export services" enable on-gateway and off-gateway

clients to register as recipient for data from devices. Core services are microservices that deal with persisting data from devices or alternatively "streaming" device data through, without persistence (core data service); managing information about the IoT devices, including their sensors, how to communicate with them, etc. (metadata service); and actual communication with IoT devices, on behalf of other on-gateway or off-gateway services (command service). A rule engine provides an API to register actions in response to conditions typically including an IoT device ID, sensor values to check, thresholds, etc. The scheduling micro service deals with organizing the removal of data persisted on the gateway. Alerts and notifications microservice can be used to dispatch alert/notifications from internal or external sources to interested consumers including backend servers, or human operators through email or SMS.

Edge cloud applications: Target applications for EdgeX include industrial IoT (e.g. IoT sensor data and actuator control mixed with augmented reality application for technicians). Home Edge focuses on smart home use cases, including using AI lifestyle and safety applications.

A.1.2. Edge Cloud Management Platforms

This set of open-source projects setup and manage clouds of individual edge computing devices. StarlingX ([StarlingX]) extends OpenStack to provide virtualization platform management for edge clouds, which are distributed (in the range of 100 compute devices), secure and highly available. Akraino Edge Stack, another project from the Linux Foundation Edge [Linux_Foundation_Edge], has a wider scope of developing a management platform adapted for the edge (e.g., covering 1000 plus locations), aiming for zero-touch provisioning, and zero-touch lifecycle management.

Computing devices: Compute devices are typically Linux-based application servers or more constrained devices.

Service platform: StarlingX adds new management services to OpenStack by leveraging building blocks such as Ceph for distributed storage, Kubernetes for orchestration. The new services are for management of configuration (enabling auto-discovery and configuration), faults, hosts (enabling host failure detection and auto-recovery), services (providing high availability through service redundancy and multi-path communication) and software (enabling updates).

Edge cloud applications: An edge computing platform may support a wide range of use cases. E.g., autonomous vehicles, industrial

automation and robotics, cloud RAN, metering and monitoring, mobile HD video, content delivery, healthcare imaging and diagnostics, caching and surveillance, augmented/virtual reality, small cell services for high density locations (stadiums), universal CPE applications, retail.

A.1.3. Related Projects

Open Edge Computing ([OpenEdgeComputing]) is an initiative from universities, manufacturers, infrastructure providers and operators, enabling efficiently offloading cloudlets (VMs) to the edge. Computing devices are typically powerful, well-connected servers located in mobile networks (e.g. collocated with base stations or aggregation sites). The service platform is built on top of OpenStack++, an extension of OpenStack to support cloudlets. This project is mentioned here as a related project because of its edge computing focus, and potential for some IoT use cases. Nevertheless, its primary use cases are typically non-IoT related, such as offloading processing-intensive applications from a mobile device to the edge.

A.2. Products

A.2.1. IoT Gateways

Multiple products are marketed as IoT gateways (Amazon Greengrass, Microsoft Azure IoT Edge, Google Cloud IoT Core, and gateway solutions from Bosh and Siemens). They are typically composed of a software frameworks that can run on a wide range of IoT gateway hardware devices to provide local support for cloud services, as well as some other local IoT gateway features such as relaying communication and caching content. Remote cloud is both used for management of the IoT gateways, and for hosting customer application components. Some IoT gateway products (Amazon Snowball) have a primary purpose of storing edge data on premises, to enable physically moving this data into the cloud without incurring digital data transfer cost.

Computing devices: Typical computing devices run Linux, Windows or a Real-Time OS over an ARM or x86 architecture. The level of service support on the computing device can range from low-level packages giving maximum control to embedded developers, to high-level SDKs. Typical requirements can start at 1GHz and 128MB RAM, e.g. ranging from Raspberry Pi to a server-level appliance.

Service platform: IoT gateways can provide a range of service including: running stateless functions; routing messages between connected IoT devices (using a wide range of IoT protocols);

caching data; enabling some form of synchronization between IoT devices; authenticating and encrypting device data. Association between IoT devices and gateway based can require a device certificate.

Edge cloud applications: Pre-processing of IoT data for later processing in the Cloud is a major driver. Use cases include industrial automation, farming, etc.

A.2.2. Edge Cloud Platforms

Services such as MobileEdgeX provide a platform for application developers to deploy software (e.g. as software containers) on edge networks.

Computing devices: Bare metal and virtual servers provided by mobile network operators are used as computing devices.

Service platform: The service platform provides end device location service, using GPS data obtained from platform software deployed in end devices, correlated with location information obtained from the mobile network. The service platform manages the deployment of application instances (containers) on servers close to end devices, using a declarative specification of optimal location from the application provider.

Edge cloud applications: Use cases include autonomous mobility, asset management, AI-based systems (e.g. quality inspection, assistance systems, safety and security cameras) and privacy-preserving video processing. There are also non-IoT use cases such as augmented reality and gaming.

A.3. Standards Initiatives

A.3.1. ETSI Multi-access Edge Computing

The ETSI MEC industry standardization group develops specifications that enable efficient and seamless integration of applications from vendors, service providers, and 3rd parties across multi-vendor MEC platforms ([ETSI_MEC_03]). Basic principles followed include: leveraging NFV infrastructure; being compliant with 3GPP systems; focusing on orchestration, MEC services, applications and platforms. Phase 1 (2015-2016) focused on basic platform services. Phase 2 (2017-2019) focuses on: supporting non-3GPP radio access technologies, especially WiFi; supporting a distributed, multi-operator and multi-vendor architecture; supporting non-VM based virtualization such as containers and PaaS.

Computing devices: Computing devices are typically application servers, attached to an eNodeB or at a higher level of aggregation point, and provide service to end users.

Service platform: The mobile edge platform offers an environment where the mobile edge applications can discover, advertise, consume and offer mobile edge services. The platform can provide certain native services such as radio network information, location, bandwidth management etc. The platform manager is responsible for managing the life cycle of applications including informing the mobile edge orchestrator of relevant application related events, managing the application rules and requirements including service authorizations, traffic rules, DNS configuration.

Edge cloud applications: Some of the use cases for MEC ([ETSI_MEC_02]) are IoT-related, including: security and safety (face recognition and monitoring), sensor data monitoring, active device location (e.g., crowd management), low latency vehicle-to-infrastructure and vehicle-to-vehicle (V2X, e.g., hazard warnings), video production and delivery, camera as a service.

A.3.2. Edge Computing Support in 3GPP

The 3GPP standards organization included edge computing support in 5G [_3GPP.23.501]. Integration of MEC and 5G systems has been studied in ETSI as well [ETSI_MEC_WP_28].

Computing devices: From 3GPP standpoint, a mobile device may access any computing device located in a local data network, i.e. traffic is steered towards the local data network where the computing device is located.

Service platform: An external party may influence steering, QoS and charging of traffic towards the computing device. Session and service continuity can ensure that edge service is maintained when a client device moves. The network supports multiple-anchor connections, which makes it possible to connect a client device to both a local and a remote data network. The client device can be made aware of the availability of a local area data network, based on its location.

Edge cloud applications: Edge cloud applications in 3GPP can help support the major use cases envisioned for 5G, including massive IoT and V2X.

A.3.3. OpenFog Consortium

The OpenFog Consortium (now part of the Industrial Internet Consortium) aims to standardize industrial IoT, fog and edge computing. It produced a reference architecture for the Fog ([OpenFog]), which has been published as IEEE standard P1934 in 2018.

Computing devices: Fog nodes include computational, networking, storage and acceleration elements. This includes nodes collocated with sensors and actuators, roadside or mobile nodes involved in V2X connectivity. Fog nodes should be programmable and may support multi-tenancy. Fog computing devices must employ a hardware-based immutable root of trust, i.e. a trusted hardware component which receives control at power-on.

Service platform: The service platform is structured around "pillars" including: security end-to-end, scalability by adding internal components or adding more fog nodes, openness in term of discovery of/by other nodes and networks, autonomy from centralized clouds (for discovery, orchestration and management, security and operation) and hierarchical organization of fog nodes.

Edge cloud applications: Major use cases include smart cars and traffic control, visual security and surveillance, smart cities.

A.3.4. Related Standards

The IEEE Fog Computing and Networking Architecture Framework Working Group [IEEE-1934] published the OpenFog architecture as an IEEE document, and plan to do further work on taxonomy, architecture framework, and compliance guidelines.

A.4. Research Projects

A.4.1. Named Function Networking

Named Function Networking ([Sifalakis]) is a research project that aims to extend ICN concepts (especially named data networking) to have the network orchestrate computation. Interests are sent for a combination of function and argument names, instead of using the content name in NDN.

Computing devices: NFN-capable switches are collocated with computing devices.

Service platform: NFN enables accessing static data and dynamic computation results in one data-oriented framework, thus

benefiting from usual ICN features such as data authenticity and caching, as well as enabling the network to perform various optimizations, e.g. moving data, code or both closer to requesters. NFN also enables secure access to individual elements within Named Data Objects, e.g. for filtering or aggregation.

Edge cloud applications: Use cases include some form of MapReduce operations and service chaining. NDN, on which NFN is based, has been studied in the context of IoT, where it can provide local trust management and rendezvous service.

A.4.2. 5G-CORAL

The 5G-CORAL project ([_5G-CORAL]) aims to enable convergence of access across multiple RATs using Fog computing, using for this purpose an Edge and Fog Computing System (EFS).

Computing devices: Computing devices used in 5G-CORAL include cloud and central data center servers, edge data center servers, and fixed or mobile "Fog Computing Devices", which can be computing devices located in vehicles or factories, e.g. IoT gateways, mobile phones, cyber-physical devices, etc.

Service platform: 5G-CORAL architecture is based on an integrated virtualized edge and fog computing system (EFS), that aims to be flexible, scalable and interoperable with other domains including transport (fronthaul, backhaul), core and clouds. An Orchestration and Control System (OCS) enables automatic discovery of heterogeneous, multiple-owner resources, and federate them into a unified hosting environment. OCS monitors resource usage to guarantee service levels. Finally, OCS also includes orchestration and life cycle functions, including live migration and scaling. Applications (user and third-party) both inside and outside the EFS subscribe to EFS services through APIs, with emphasis on IoT and cyber-physical functionalities.

Edge cloud applications: EFS-hosted services include analytics obtained from IoT gateways (e.g. LORA or eNodeB gateways), context information services from RATs, transport (fronthaul and backhaul) and core networks. EFS-hosted functions include network performance acceleration functions, virtualized C-RAN functions for access nodes and possible end user devices.

A.4.3. FLAME

The FLAME project ([FLAME]) aims to improve performance of interactive media systems while keeping infrastructure costs low. It builds over virtualization technologies such as XOS, OpenStack and

ONOS/ODL to offer a programmable media service platform. FLAME leverages IP-over-ICN technology developed through earlier projects including POINT ([POINT]).

Computing devices: The FLAME platform provides a service layer on top of an infrastructure platform, which can include cloud servers as well as computing devices collocated with WiFi access points.

Service platform: The FLAME platform can be seen as an edge + cloud computing platform with a use case focus on media dissemination, although the basic platform can be suitable for micro-services in general. The computing platform is comprised of: computing devices, an infrastructure platform (XOS, OpenStack, ONOS/ODL), NFV-MANO components (orchestrator, virtual infrastructure manager) and FLAME platform core services (PCE, network access point, surrogate manager).

Edge cloud applications: IoT use cases include public safety, such as supporting body-worn camera for police and social workers. As opposed to other multi-media applications that are also envisioned (pre-processing, user reporting, curation...), where a typical goal is to curate content early at the edge, to reduce expected high data volume, public safety use cases are typically about implementing triggers at the edge: everything needs to be kept anyway, to be available in case of an audit. Content is stored offline during off peak-hours delivery. For privacy and data volume concerns, triggers for, e.g., alerting police, cannot be performed in the cloud and should be performed as close to the data source as possible.

Authors' Addresses

Jungha Hong
ETRI
218 Gajeong-ro, Yuseung-Gu
Daejeon 34129
Korea

Email: jhong@etri.re.kr

Yong-Geun Hong
ETRI
218 Gajeong-ro, Yuseung-Gu
Daejeon 34129
Korea

Email: yghong@etri.re.kr

Xavier de Foy
InterDigital Communications, LLC
1000 Sherbrooke West
Montreal H3A 3G4
Canada

Email: Xavier.Defoy@InterDigital.com

Matthias Kovatsch
Huawei Technologies Duesseldorf GmbH
Riesstr. 25 C // 3.OG
Munich 80992
Germany

Email: matthias.kovatsch@huawei.com

Eve Schooler
Intel

Email: eve.m.schooler@intel.com

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

Email: ietf@dkutscher.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 14 January 2021

J. Hong
Y-G. Hong
ETRI
X. de Foy
InterDigital Communications, LLC
M. Kovatsch
Huawei Technologies Duesseldorf GmbH
E. Schooler
Intel
D. Kutscher
University of Applied Sciences Emden/Leer
13 July 2020

IoT Edge Challenges and Functions
draft-hong-t2trg-iot-edge-computing-05

Abstract

Many IoT applications have requirements that cannot be met by the traditional Cloud (aka cloud computing). These include time sensitivity, data volume, uplink cost, operation in the face of intermittent services, privacy and security. As a result, the IoT is driving the Internet toward Edge computing. This document outlines the requirements of the emerging IoT Edge and its challenges. It presents a general model, and major components of the IoT Edge, with the goal to provide a common base for future discussions in T2TRG and other IRTF and IETF groups.

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 14 January 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	3
2. Background	3
2.1. Internet of Things (IoT)	3
2.2. Cloud Computing	4
2.3. Edge Computing	4
2.4. Example of IoT Edge Computing Use Cases	6
2.4.1. Smart Construction	6
2.4.2. Smart Grid	6
2.4.3. Smart Water System	7
3. IoT Challenges Leading Towards Edge Computing	7
3.1. Time Sensitivity	7
3.2. Uplink Cost	8
3.3. Resilience to Intermittent Services	8
3.4. Privacy and Security	8
4. IoT Edge Computing Functions	9
4.1. Overview of IoT Edge Computing Today	9
4.2. General Model	10
4.3. OAM Components	14
4.3.1. Virtualization Management	14
4.3.2. Resource Discovery and Authentication	15
4.3.3. Edge Organization and Federation	15
4.4. Functional Components	16
4.4.1. External APIs	16
4.4.2. Communication Brokering	16
4.4.3. In-Network Computation	17
4.4.4. Edge Caching	18
4.4.5. Other Services	19
4.5. Application Components	19
4.5.1. IoT End Devices Management	19
4.5.2. Data Management	19
4.6. Simulation and Emulation Environments	20
5. Security Considerations	20

6. Acknowledgment	21
7. Informative References	21
Authors' Addresses	26

1. Introduction

Currently, many IoT services leverage the Cloud, since it can provide virtually unlimited storage and processing power. The reliance of IoT on back-end cloud computing brings additional advantages such as flexibility and efficiency. Today's IoT systems are fairly static with respect to integrating and supporting computation. It's not that there is no computation, but systems are often limited to static configurations (edge gateways, cloud services).

However, IoT devices are creating vast amounts of data at the network edge. To meet IoT use case requirements, that data increasingly is being stored, processed, analyzed, and acted upon close to the data producers. These requirements include time sensitivity, data volume, uplink cost, resiliency in the face of intermittent connectivity, privacy, and security, which cannot be addressed by today's centralized cloud computing. These requirements suggest a more flexible way to distribute computing (and storage) and to integrate it in the edge-cloud continuum. We will refer to this integration of edge computing and IoT as "IoT edge computing". Our draft describes background, uses cases, challenges, and presents system models and functional components.

2. Background

2.1. Internet of Things (IoT)

Since the term "Internet of Things" (IoT) was coined by Kevin Ashton in 1999 working on Radio-Frequency Identification (RFID) technology [Ashton], the concept of IoT has evolved. It now reflects a vision of connecting the physical world to the virtual world of computers using (wireless) networks over which Things can send and receive information without human intervention. Recently, the term has become more literal by actually connecting Things to the Internet and converging on Internet and Web technology.

Things are usually embedded systems of various kinds, such as home appliances, mobile equipment, wearable devices, etc. Things are widely distributed, but typically have limited storage and processing power, which raise concerns regarding reliability, performance, energy consumption, security, and privacy [Lin]. This limited storage and processing power leads to complementing IoT with cloud computing.

2.2. Cloud Computing

Cloud computing has been defined in [NIST]: "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Cloud computing has become a predominant technology that offers virtually unlimited capacity in terms of storage and processing power, at low cost. This offering enabled the realization of a new computing model, in which virtualized resources can be leased in an on-demand fashion, being provided as general utilities. Companies like Amazon, Google, Facebook, etc. widely adopted this paradigm for delivering services over the Internet, gaining both economical and technical benefits [Botta].

Today, an unprecedented volume and variety of data is generated by Things and applications deployed in edge networks consume this data. Some of these applications may need very short response times, some may access personal data, while others may generate vast amounts of data. Today's cloud-based service models are not suitable for these applications, which can instead leverage edge computing.

2.3. Edge Computing

Edge computing, in some settings also referred to as fog computing, is a new paradigm in which substantial computing and storage resources are placed at the edge of the Internet, that is, in close proximity to mobile devices, sensors, actuators, or machines. Edge computing happens near data sources [Mahadev], or closer (topologically, physically, in term of latency, etc.) to where decisions or interactions with the physical world are happening. It processes both downstream data, e.g. originated from cloud services, and upstream data, e.g. originated from end devices or network elements. The term fog computing usually represents the notion of a multi-tiered edge computing, that is, several layers of compute infrastructure between the end devices and cloud services.

An edge device is any computing or networking resource residing between data sources and cloud-based datacenters. In edge computing, end devices not only consume data, but also produce data. And at the network edge, devices not only request services and information from the Cloud, but also handle computing tasks including processing, storage, caching, and load balancing on data sent to and from the Cloud [Shi]. This does not preclude end devices from hosting computation themselves when possible, independently or as part of a distributed edge computing platform (this is also referred to as Mist Computing).

Several standards defining organization and industry forums have provided definitions of edge and fog computing:

- * ISO defines edge computing as a "form of distributed computing in which significant processing and data storage takes place on nodes which are at the edge of the network" [ISO_TR].
- * ETSI defines multi-access edge computing as a "system which provides an IT service environment and cloud-computing capabilities at the edge of an access network which contains one or more type of access technology, and in close proximity to its users" [ETSI_MEC_01].
- * The Industrial Internet Consortium (formerly OpenFog) defines fog computing as "a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum" [OpenFog].

Based on these definitions, we can summarize a general philosophy of edge computing as to distribute the required functions close to users and data, while the difference to classic local systems is the usage of management and orchestration features adopted from cloud computing.

Actors from various industries approach edge computing using different terms and reference models, although in practice these approaches are not incompatible and may integrate with each other:

- * The telecommunication industry tends to use a model where edge computing services are deployed over NFV infrastructure at aggregation points, or in proximity to the user equipment (e.g., gNodeBs) [ETSI_MEC_03].
- * Enterprise and campus solutions often interpret edge computing as an "edge cloud", that is, a smaller data center directly connected to the local network (often referred to as "on-premise").
- * The automation industry defines the edge as the connection point between IT from OT (Operational Technology). Hence, here edge computing sometimes refers to applying IT solutions to OT problems such as analytics, more flexible user interfaces, or simply having more compute power than an automation controller.

2.4. Example of IoT Edge Computing Use Cases

IoT edge computing can be used in home, industry, grid, healthcare, city, transportation, agriculture, and/or education scenarios. We discuss here only a few examples of such use cases, to point out differentiating requirements.

2.4.1. Smart Construction

In traditional construction domain, heavy equipment and machinery pose risks to humans and property. Thus, there have been many attempts to deploy technology to protect lives and property in construction sites. For example, measurements of noise, vibration, and gas can be recorded and reported to an inspector. Today, data produced by such measurements is collected by a local gateway and transferred to a remote cloud server. This incurs transmission costs, e.g., over a LTE connection, and storage costs, e.g., when using Amazon Web Services. When an inspector needs to investigate an incident, he checks the information stored on the cloud server.

To determine the exact cause of an incident, sensor data including audio and video are transferred to a remote server. In this case, audio and video data volume is typically very large and the cost of transmission can be an issue. By leveraging IoT edge computing, sensor data can be processed and analyzed on a gateway located within or near a construction site. And with the help of statistical analysis or machine learning technologies, we can predict future incidents in advance and trigger an on-site alarm. Furthermore, predicting the time of an incident can help reducing significantly the volume and cost of transmitted data, by transmitting video at high resolution during critical periods, while otherwise using a lower resolution.

2.4.2. Smart Grid

In future smart city scenarios, the Smart Grid will be critical in ensuring highly available/efficient energy control in city-wide electricity management. Edge computing is expected to play a significant role in those systems to improve transmission efficiency of electricity; to react and restore power after a disturbance; to reduce operation costs and reuse renewable energy effectively, since these operations involve local decision making. In addition, edge computing can help monitoring power generation and power demand, and making local electrical energy storage decisions in the smart grid system.

2.4.3. Smart Water System

The water system is one of the most important aspects of a city. Effective use of water, and cost-effective and environment-friendly water treatment are critical aspects of this system. Edge computing can help with monitoring water consumption and transport, and with predicting future water usage level. Examples of application include: water harvesting, ground water monitoring, locally analyzing collected information related to water control and management to limit water losses.

3. IoT Challenges Leading Towards Edge Computing

This section describes challenges met by IoT, that are motivating the adoption of edge computing for IoT. Those are distinct from research challenges applicable to IoT edge computing, some of which will be mentioned in Section 4.3.

IoT technology is used with more and more demanding applications, e.g. in industrial, automotive or healthcare domains, leading to new challenges. For example, industrial machines such as laser cutters already produce over 1 terabyte per hour, and similar amounts can be generated in autonomous cars [NVIDIA]. 90% of IoT data is expected to be stored, processed, analyzed, and acted upon close to the source [Kelly], as cloud computing models alone cannot address the new challenges [Chiang].

Below we discuss IoT use case requirements that are moving cloud capabilities to be more proximate and more distributed and disaggregated.

3.1. Time Sensitivity

Many industrial control systems, such as manufacturing systems, smart grids, oil and gas systems, etc., often require stringent end-to-end latency between the sensor and control node. While some IoT applications may require latency below a few tens of milliseconds [Weiner], industrial robots and motion control systems have use cases for cycle times in the order of microseconds [_60802]. In some cases speed-of-light limitations may simply prevent a solution based on remote cloud, however it is not the only challenge relative to time sensitivity. An important aspect for real-time communications is not only the latency, but also guarantees for jitter. This means control packets need to arrive with as little variation as possible and within a strict deadline. Given the best-effort characteristics of the Internet this challenge is virtually impossible to address, without using end-to-end guarantees for individual message delivery and continuous data flows.

3.2. Uplink Cost

Many IoT deployments are not challenged by a constrained network bandwidth to the Cloud. The fifth generation mobile networks (5G) and Wi-Fi 6 both theoretically top out at 10 gigabits per second (i.e., 4.5 terabyte per hour), which enables high-bandwidth uplinks. However, the resulting cost for high-bandwidth connectivity to upload all data to the Cloud is unjustifiable and impractical for most IoT applications. In some settings, e.g. in aeronautical communication, higher communication costs reduce the amount of data that can be practically uploaded even further.

3.3. Resilience to Intermittent Services

Many IoT devices such as sensors, data collectors, actuators, controllers, etc. have very limited hardware resources and cannot rely solely on their limited resources to meet all their computing and/or storage needs. They require reliable, uninterrupted or resilient services to augment their capabilities in order to fulfill their application tasks. This is hard and partly impossible to achieve with cloud services for systems such as vehicles, drones, or oil rigs that have intermittent network connectivity. The dual is also true, a cloud back-end might want to have a reading of the device even if it's currently asleep.

3.4. Privacy and Security

When IoT services are deployed at home, personal information can be learned from detected usage data. For example, one can extract information about employment, family status, age, and income by analyzing smart meter data [ENERGY]. Policy makers started to provide frameworks that limit the usage of personal data and put strict requirements on data controllers and processors. However, data stored indefinitely in the Cloud also increases the risk of data leakage, for instance, through attacks on rich targets.

Industrial systems are often argued to not have privacy implications, as no personal data is gathered. Yet data from such systems is often highly sensitive, as one might be able to infer trade secrets such as the setup of production lines. Hence, the owner of these systems are generally reluctant to upload IoT data to the Cloud.

Furthermore, passive observers can perform traffic analysis on the device-to-cloud path. Hiding traffic patterns associated with sensor networks can therefore be another requirement for edge computing.

4. IoT Edge Computing Functions

In this section we first look at the current state of IoT edge computing Section 4.1, and then define a general system model Section 4.2. This provides context for IoT edge computing functions, which are listed in Section 4.3.

4.1. Overview of IoT Edge Computing Today

This section provides an overview of today's IoT edge computing field, based on a limited review of standards, research, open-source and proprietary products in [I-D-defoy-t2trg-iot-edge-computing-background].

IoT gateways, both open-source (such as EdgeX Foundry or Home Edge) and proprietary (such as Amazon Greengrass, Microsoft Azure IoT Edge, Google Cloud IoT Core, and gateways from Bosh, Siemens), represent a common class of IoT edge computing products, where the gateway is providing a local service on customer premises, and is remotely managed through a cloud service. IoT communication protocols are typically used between IoT devices and the gateway, including CoAP, MQTT and many specialized IoT protocols (such as OPC UA and DDS in the Industrial IoT space), while the gateway communicates with the distant cloud typically using HTTPS. Virtualization platforms enable the deployment of virtual edge computing functions (as VMs, application containers, etc.), including IoT gateway software, on servers in the mobile network infrastructure (at base station and concentration points), in edge datacenters (in central offices) or regional datacenters located near central offices. End devices are envisioned to become computing devices in forward looking projects, but are not commonly used as such today.

Physical or virtual IoT gateways can host application programs, which are typically built using an SDK to access local services through a programmatic API. Edge cloud system operators host their customers' applications VMs or containers on servers located in or near access networks, which can implement local edge services. For example, mobile networks can provide edge services for radio network information, location and bandwidth management.

Life cycle management of services and applications on physical IoT gateways is often cloud-based. Edge cloud management platforms and products (such as StarlingX, Akraino Edge Stack, Mobile EdgeX) adapt cloud management technologies (e.g., Kubernetes) to the edge cloud, i.e., to smaller, distributed computing devices running outside a controlled data center. Services and application life-cycle is typically using a NFV-like management and orchestration model.

The platform typically includes services to advertise or consume APIs (e.g., Mpl interface in ETSI MEC supports service discovery and communication), and enables communicating with local and remote endpoints (e.g., message routing function in IoT gateways). The service platform is typically extensible by edge applications, since they can advertise an API that other edge applications can consume. IoT communication services include protocols translation, analytics and transcoding. Communication between edge computing devices is enabled in tiered deployments or distributed deployments.

An edge cloud platform may enable pass-through without storage or local storage (e.g., on IoT gateways). Some edge cloud platforms use a distributed form of storage such as an ICN network (e.g., NFN nodes can store data in NDN) or a distributed storage platform (e.g., Ceph). External storage, e.g., on databases in distant or local IT cloud, is typically used for filtered data deemed worthy of long term storage, although in some case it may be for all data, for example when required for regulatory reasons.

Stateful computing is supported on platforms hosting native programs, VMs or containers. Stateless computing is supported on platforms providing a "serverless computing" service (a.k.a. function-as-a-service), or on systems based on named function networking.

In many IoT use cases, a typical network usage pattern is high volume uplink with some form of traffic reduction enabled by processing over edge computing devices. Alternatives to traffic reduction include deferred transmission (to off-peak hours or using physical shipping). Downlink traffic includes application control and software updates. Other, downlink-heavy traffic patterns are not excluded but are more often associated with non-IoT usage (e.g., video CDNs).

4.2. General Model

Edge computing is expected to play an important role in deploying new IoT services integrated with Big Data and AI, enabled by flexible in-network computing platforms. Although there are lots of approaches to edge computing, we attempt to lay out a general model and list associated logical functions in this section. In practice, this model can map to different architectures, such as:

- * A single IoT gateway, or a hierarchy of IoT gateways, typically connected to the cloud (e.g., to extend the traditionally cloud-based management of IoT devices and data to the edge). A common role of an IoT Gateway is to provide access to an heterogeneous set of IoT devices/sensors; handle IoT data; and deliver IoT data to its final destination in a cloud network. Whereas an IoT gateway needs interactions with cloud like as conventional cloud computing, it can also operate independently.
- * A set of distributed computing nodes, e.g., embedded in switches, routers, edge cloud servers or mobile devices. Some IoT end devices can have enough computing capabilities to participate in such distributed systems due to advances in hardware technology. In this model, edge computing nodes can collaborate with each other to share their resources.

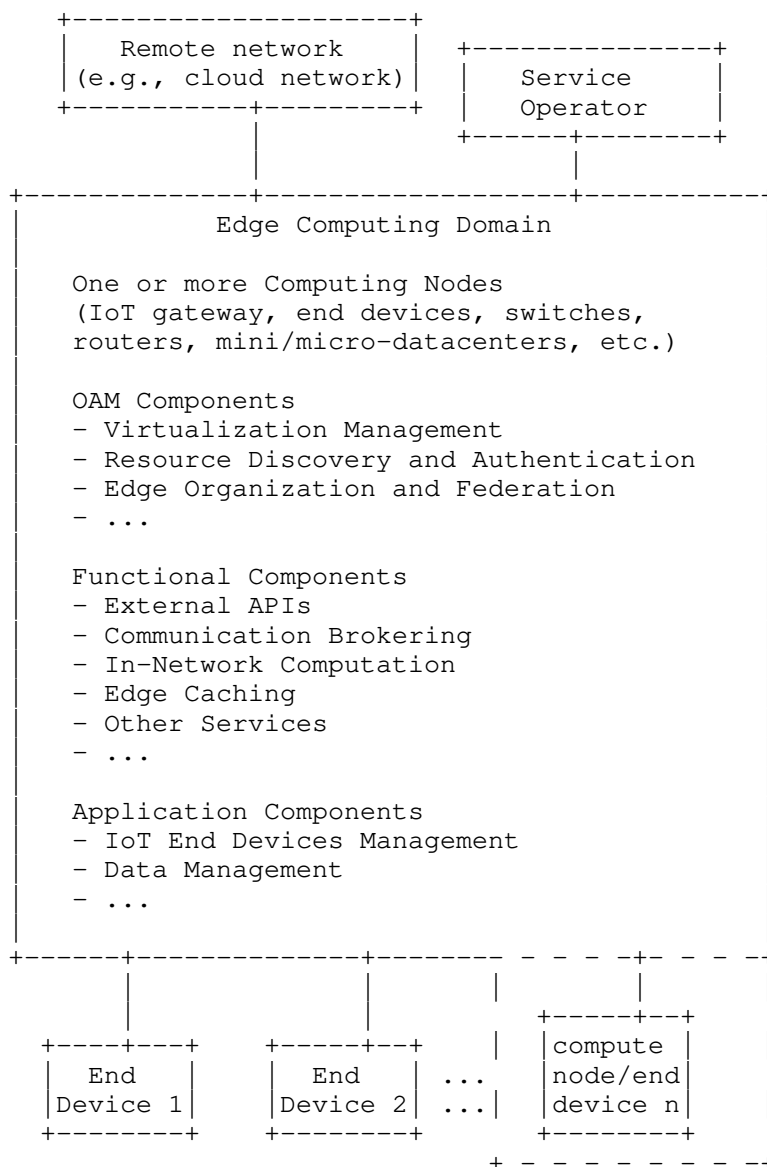


Figure 1: Model of IoT Edge Computing

In the above model, the edge computing domain is interconnected with IoT end devices (southbound connectivity) and possibly with a remote/cloud network (northbound connectivity), and with a service operator's system. Edge computing nodes provide multiple logical functions, or components, which may not all be present in a given

system. They may be implemented in a centralized or distributed fashion, in the edge network, or through some interworking between the edge network and a remote cloud network.

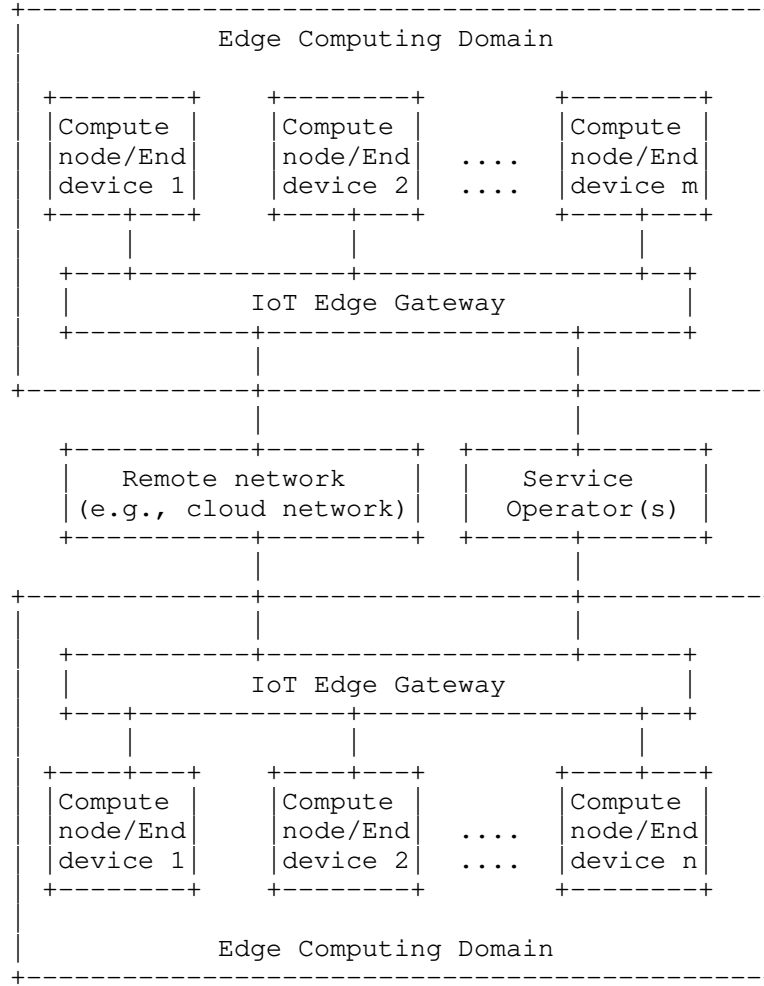


Figure 2: Example: Machine Learning over a Distributed IoT Edge Computing System

In the above example of system, the edge computing domain is composed of IoT edge gateways and IoT end devices which are also used as computing nodes. Edge computing domains are connected with a remote/cloud network, and with their respective service operator's system. IoT end devices/computing nodes provide logical functions, as part of a distributed machine learning application. The processing

capabilities in IoT end devices being limited, they require the support of other nodes: the training process for AI services is executed at IoT edge gateways or cloud networks and the prediction (inference) service is executed in the IoT end devices.

We now attempt to enumerate major edge computing domain components. They are here loosely organized into OAM, functional and application components, with the understanding that the distinction between these classes may not always be clear, depending on actual system architectures. Some representative research challenges are associated with those functions. We used input from co-authors, IRTF attendees and some comprehensive reviews of the field ([Yousefpour], [Zhang2], [Khan]).

4.3. OAM Components

Edge computing OAM goes beyond the network-related OAM functions listed in [RFC6291]. Besides infrastructure (network, storage and computing resources), edge computing systems can also include computing environments (for VMs, software containers, functions), IoT end devices, data and code.

Operation related functions include performance monitoring for service level agreement measurement; fault management and provisioning for links, nodes, compute and storage resources, platforms and services. Administration covers network/compute/storage resources, platforms and services discovery, configuration and planning. Management covers monitoring and diagnostics of failures, as well as means to minimize their occurrence and take corrective actions. This may include software updates management, high service availability through redundancy and multipath communication. Centralized (e.g., SDN) and decentralized management systems can be used.

We further detail a few OAM components.

4.3.1. Virtualization Management

Some IoT edge computing systems make use of virtualized (compute, storage and networking) resources, which need to be allocated and configured. This function is covered to a large extent by ETSI NFV and MEC standards activities. Projects such as [LFEDGE-EVE] further cover virtualization and its management into distributed edge computing settings.

Related challenges include:

- * Minimizing virtual function instantiation time and resource usage

- * Integration of edge computing with virtualized radio networks (Fog RAN) [I-D.bernardos-sfc-fog-ran] and with 5G access networks
- * Handling of multi-tenancy with regards to limited resources at the network edge

4.3.2. Resource Discovery and Authentication

Discovery and authentication may target platforms, infrastructure resources, such as compute, network and storage, but also other resources such as IoT end devices, sensors, data, code units, services, applications or users interacting with the system. Broker-based solutions can be used, e.g. using an IoT gateway as broker to discover IoT resources. Today, centralized gateway-based systems rely, for device authentication, on the installation of a secret on IoT end devices and on computing devices (e.g., a device certificate stored in a hardware security module).

Related challenges include:

- * Discovery, authentication and trust establishment between end devices, compute nodes and platforms, with regards to concerns such as mobility, heterogeneity, scale, multiple trust domains, constrained devices, anonymity and traceability
- * Intermittent connectivity to the Internet, preventing relying on a third-party authority [Echeverria]
- * Resiliency to failures [Harchol], denial of service attacks, easier physical access for attackers

4.3.3. Edge Organization and Federation

In a distributed system context, once edge devices have discovered and authenticated each other, they can be organized, or self-organize, into hierarchies or clusters. Organization may range from centralized to peer-to-peer. Such groups can also form federations with other edge or remote clouds.

Related challenges include:

- * Sharing resources in multi-vendor/operator scenarios, with a goal to optimize criteria such as profit [Anglano], resource usage, latency or energy consumption

- * Support for scaling, and enabling fault-tolerance or self-healing [Jeong]. Besides using hierarchical organization to cope with scaling, another available and possibly complementary mechanism is multicast ([RFC7390] [I-D.ietf-core-oscore-groupcomm])
- * Capacity planning, placement of infrastructure nodes to minimize delay [Fan], cost, energy, etc.
- * Incentives for participation, e.g. in peer-to-peer federation schemes

4.4. Functional Components

4.4.1. External APIs

An IoT edge cloud may provide a northbound data plane or management plane interface to a remote network, e.g., a cloud, home or enterprise network. This interface does not exist in standalone (local-only) scenarios. To support such an interface when it exists, an edge computing component needs to expose an API, deal with authentication and authorization, support secure communication.

An IoT edge cloud may provide an API or interface to local or mobile users, for example to provide access to services and applications, or to manage data published by local/mobile devices.

Related challenges include:

- * Defining edge computing abstractions suitable for users and cloud systems to interact with edge computing systems. In one example, this interaction can be based on the PaaS model [Yanguil]

4.4.2. Communication Brokering

A typical function of IoT edge computing is to facilitate communication with IoT end devices: for example, enable clients to register as recipients for data from devices, as well as forwarding/routing of traffic to or from IoT end devices, enabling various data discovery and redistribution patterns, e.g., north-south with clouds, east-west with other edge devices [I-D.mcbride-edge-data-discovery-overview]. Another related aspect is dispatching of alerts and notifications to interested consumers both inside and outside of the edge computing domain. Protocol translation, analytics and transcoding may also be performed when necessary.

Communication brokering may be centralized in some systems, e.g., using a hub-and-spoke message broker, or distributed like with message buses, possibly in a layered bus approach. Distributed systems may leverage direct communication between end devices, over device-to-device links. A broker can ensure communication reliability, traceability, and in some cases transaction management.

Related challenges include:

- * Enabling secure and resilient communication between IoT end devices and remote cloud, e.g. through multipath support

4.4.3. In-Network Computation

A core function of IoT edge computing is to enable computation offloading, i.e., to perform computation on an edge node on behalf of a device or user, but also to orchestrate computation (in a centralized or distributed manner) and manage applications lifecycle. Support for in-network computation may vary in term of capability, e.g., computing nodes can host virtual machines, software containers, software actors or unikernels able run stateful or stateless code, or a rule engine providing an API to register actions in response to conditions such as IoT device ID, sensor values to check, thresholds, etc.

QoS can be provided in some systems through the combination of network QoS (e.g., traffic engineering or wireless resource scheduling) and compute/storage resource allocations. For example in some systems a bandwidth manager service can be exposed to enable allocation of bandwidth to/from an edge computing application instance.

Related challenges include:

- * (Computation placement) Selecting, in a centralized or distributed/peer-to-peer manner, an appropriate compute device based on available resources, location of data input and data sinks, compute node properties, etc., and with varying goals including for example end-to-end latency, privacy, high availability, energy conservation, network efficiency (e.g. using load balancing techniques to avoid congestion)
- * Onboarding code on a platform or compute device, and invoking remote code execution, possibly as part of a distributed programming model and with respect to similar concerns of latency, privacy, etc. These operations should deal with heterogeneous compute nodes [Schafer], and may in some cases also support end devices as compute nodes

- * Adapting Quality of Results (QoR) for applications where a perfect result is not necessary [Li]
- * Assisted or automatic partitioning of code [I-D.sarathchandra-coin-appcentres]
- * Supporting computation across trust domains, e.g. verifying computation results
- * Relocating an instance from one compute node to another, while maintaining a given service level.
- * Session continuity when communicating with end devices that are mobile, possibly at high speed (e.g. in vehicular scenarios)
- * Defining, managing and verifying SLAs for edge computing systems. Pricing is a related challenge

4.4.4. Edge Caching

A purpose of local caching may be to enable local data processing (e.g., pre-processing or analysis), or to enable delayed virtual or physical shipping. A responsibility of the edge caching component is to manage data persistence, e.g., to schedule removal of data when it is no longer needed. Another aspect of this component may be to authenticate and encrypt data. It can for example take the form of a distributed storage system.

Related challenges include

- * (Cache and data placement) Using cache positioning and data placement strategies to minimize data retrieval delay [Liu], energy consumption. Caches may be positioned in the access network infrastructure or may be on end devices using device-to-device communication
- * Maintaining data consistency, freshness and privacy in systems that are distributed, constrained and dynamic (e.g. due to end devices and computing nodes churn or mobility). For example, age of information [Yates], a performance metric that captures the timeliness of information from a sender (e.g. an IoT device), can be exposed to networks to enable tradeoffs in this problem space

4.4.5. Other Services

Data generated by IoT devices and associated information obtained from the access network may be used to provide high level services such as end device location, radio network information and bandwidth management.

4.5. Application Components

IoT edge computing can host applications such as the ones mentioned in Section 2.4. While describing components of individual applications is out of our scope, some of those applications share similar functions, such as IoT end device management, data management, described below.

4.5.1. IoT End Devices Management

IoT end device management includes managing information about the IoT devices, including their sensors, how to communicate with them, etc. Edge computing addresses the scalability challenges from the massive number of IoT end devices by separating the scalability domain into edge/local networks and remote network.

Challenges listed in Section 4.3.2 may be applicable to IoT end devices management as well.

4.5.2. Data Management

Data storage and processing at the edge is a major aspect of IoT edge computing, directly addressing high level IoT challenges listed in Section 3. Data analysis such as performed in AI/ML tasks performed at the edge may benefit from specialized hardware support on computing nodes.

Related challenges include:

- * Addressing concerns on resource usage, security and privacy when sharing, discovering or managing data. For example by presenting data in views composed of an aggregation of related data [Zhang], protecting data communication between authenticated peers [Basudan], classifying data (e.g., in terms of privacy, importance, validity, etc.), compressing data
- * Data driven programming models [Renart], e.g. event-based, including handling of naming and data abstractions

- * Addressing concerns such as limited resources, privacy, dynamic and heterogeneous environment, to deploy machine learning at the edge. For example, making machine learning more lightweight and distributed, supporting shorter training time and simplified models, and supporting models that can be compressed for efficient communication [Murshed]
- * While edge computing can support IoT services independently of cloud computing, it can also be connected to cloud computing. Thus, the relationship of IoT edge computing to cloud computing, with regard to data management, is another potential challenge [ISO_TR]

4.6. Simulation and Emulation Environments

IoT Edge Computing brings new challenges to simulation and emulation tools used by researchers and developers. A varied set of applications, network and computing technologies can coexist in a distributed system, which make modelling difficult. Scale, mobility and resource management are additional challenges [SimulatingFog].

Tools include simulators, where simplified application logic runs on top of a fog network model, and emulators, where actual applications can be deployed, typically in software containers, over a cloud infrastructure (e.g. Docker, Kubernetes) itself running over a network emulating edge network conditions such as variable delays, throughput and mobility events. To gain in scale, emulated and simulated systems can be used together in hybrid federation-based approaches [PseudoDynamicTesting], while to gain in realism physical devices can be interconnected with emulated systems. Examples of related work and platforms include the publicly accessible MEC sandbox work recently initiated in ETSI [ETSI_Sandbox], and open source simulators and emulators ([AdvantEDGE] emulator and tools cited in [SimulatingFog]).

5. Security Considerations

As discussed in Section 4.3.2, authentication and trust (between computing nodes, management nodes, end devices) can be challenging as scale, mobility and heterogeneity increase. The sometimes disconnected nature of edge resources can prevent relying on a third-party authority. Distributed edge computing is exposed to issues with reliability and denial of service attacks. Personal or proprietary IoT data leakage is also a major threat, especially due to the distributed nature of the systems (Section 4.5.2).

However, edge computing also brings solutions in the security space: maintaining privacy by computing sensitive data closer to data generators is a major use case for IoT edge computing. An edge cloud can be used to take actions based on sensitive data, or anonymizing, aggregating or compressing data prior to transmitting to a remote cloud server. Edge computing communication brokering functions can also be used to secure communication between edge and cloud networks.

6. Acknowledgment

The authors would like to thank Joo-Sang Youn, Akbar Rahman, Michel Roy, Robert Gazda, Rute Sofia, Thomas Fossati and Chonggang Wang for their valuable comments and suggestions on this document.

7. Informative References

- [AdvantEDGE] "Mobile Edge Emulation Platform", Source Code Repository , 2020, <<https://github.com/InterDigitalInc/AdvantEDGE>>.
- [Anglano] Anglano, C., Canonico, M., Castagno, P., Guazzzone, M., and M. Sereno, "A game-theoretic approach to coalition formation in fog provider federations", IEEE Third International Conference on Fog and Mobile Edge Computing (FMEC), pages 123-130 , 2018.
- [Ashton] Ashton, K., "That Internet of Things thing", RFID J. vol. 22, no. 7, pp. 97-114 , 2009.
- [Basudan] Basudan, S., Lin, X., and K. Sankaranarayanan, "A privacy-preserving vehicular crowdsensing-based road surface condition monitoring system using fog computing", IEEE Internet of Things Journal, 4(3):772-782 , 2017.
- [Botta] Botta, A., Donato, W., Persico, V., and A. Pescapé, "Integration of Cloud Computing and Internet of Things: A survey", Future Gener. Comput. Syst., vol. 56, pp. 684-700 , 2016.
- [Chiang] Chiang, M. and T. Zhang, "Fog and IoT: An overview of research opportunities", IEEE Internet Things J., vol. 3, no. 6, pp. 854-864 , 2016.
- [Echeverria] Echeverria, S., Klinedinst, D., Williams, K., and G. A Lewis, "Establishing trusted identities in disconnected edge environments", IEEE/ACM Symposium Edge Computing (SEC), pages 51-63. , 2016.

- [ENERGY] Beckel, C., Sadamori, L., Staake, T., and S. Santini, "Revealing Household Characteristics from Smart Meter Data", *Energy*, vol. 78, pp. 397-410 , 2014.
- [ETSI_MEC_01] ETSI, ., "Multi-access Edge Computing (MEC); Terminology", ETSI GS 001 , 2019, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/001/02.01.01_60/gs_MEC001v020101p.pdf>.
- [ETSI_MEC_03] ETSI, ., "Mobile Edge Computing (MEC); Framework and Reference Architecture", ETSI GS 003 , 2019, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf>.
- [ETSI_Sandbox] "Multi-access Edge Computing (MEC) MEC Sandbox Work Item", Portal , 2020, <https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=57671>.
- [Fan] Fan, Q. and N. Ansari, "Cost aware cloudlet placement for big data processing at the edge", *IEEE International Conference on Communications (ICC)*, pages 1-6 , 2017.
- [Harchol] Harchol, Y., Mushtaq, A., McCauley, J., Panda, A., and S. Shenker, "Cessna: Resilient edge-computing", *Workshop on Mobile Edge Communications*, pages 1-6. ACM , 2018.
- [I-D-defoy-t2trg-iot-edge-computing-background] de Foy, X., Hong, J., Hong, Y., Kovatsch, M., Schooler, E., and D. Kutscher, "Machine learning at the network edge: A survey", *draft-defoy-t2trg-iot-edge-computing-background-00* , 2020, <<http://www.ietf.org/internet-drafts/draft-defoy-t2trg-iot-edge-computing-background-00.txt>>.
- [I-D.bernardos-sfc-fog-ran] Bernardos, C., Rahman, A., and A. Mourad, "Service Function Chaining Use Cases in Fog RAN", *Work in Progress, Internet-Draft, draft-bernardos-sfc-fog-ran-07*, 11 March 2020, <<http://www.ietf.org/internet-drafts/draft-bernardos-sfc-fog-ran-07.txt>>.

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP", Work
in Progress, Internet-Draft, draft-ietf-core-oscore-
groupcomm-09, 23 June 2020, <[http://www.ietf.org/internet-
drafts/draft-ietf-core-oscore-groupcomm-09.txt](http://www.ietf.org/internet-drafts/draft-ietf-core-oscore-groupcomm-09.txt)>.
- [I-D.mcbride-edge-data-discovery-overview]
McBride, M., Kutscher, D., Schooler, E., and C. Bernardos,
"Edge Data Discovery for COIN", Work in Progress,
Internet-Draft, draft-mcbride-edge-data-discovery-
overview-03, 29 January 2020, <[http://www.ietf.org/
internet-drafts/draft-mcbride-edge-data-discovery-
overview-03.txt](http://www.ietf.org/internet-drafts/draft-mcbride-edge-data-discovery-overview-03.txt)>.
- [I-D.sarathchandra-coin-appcentres]
Sarathchandra, C., Trossen, D., and M. Boniface, "In-
Network Computing for App-Centric Micro-Services", Work in
Progress, Internet-Draft, draft-sarathchandra-coin-
appcentres-02, 28 February 2020, <[http://www.ietf.org/
internet-drafts/draft-sarathchandra-coin-appcentres-
02.txt](http://www.ietf.org/internet-drafts/draft-sarathchandra-coin-appcentres-02.txt)>.
- [ISO_TR] "Information Technology - Cloud Computing - Edge Computing
Landscape", ISO/IEC TR 23188 , 2018.
- [Jeong] Jeong, T., Chung, J., Hong, J.W., and S. Ha, "Towards a
distributed computing framework for fog", IEEE Fog World
Congress (FWC), pages 1-6 , 2017.
- [Kelly] Kelly, R., "Internet of Things Data to Top 1.6 Zettabytes
by 2022", 2015,
<[https://campustechnology.com/articles/2015/04/15/
internet-of-things-data-to-top-1-6-zettabytes-by-
2020.aspx](https://campustechnology.com/articles/2015/04/15/internet-of-things-data-to-top-1-6-zettabytes-by-2020.aspx)>.
- [Khan] Khan, L.U., Yaqoob, I., Tran, N.H., Kazmi, S.M.A., Dang,
T.N., and C.S. Hong, "Edge Computing Enabled Smart Cities:
A Comprehensive Survey", arXiv:1909.08747 , 2019.
- [LFEDGE-EVE]
Linux Foundation, ., "Project Edge Virtualization Engine
(EVE)", Portal , 2020,
<<https://www.lfedge.org/projects/eve>>.

- [Li] Li, Y., Chen, Y., Lan, T., and G. Venkataramani, "Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization", IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 1261-1270 , 2017.
- [Lin] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications", IEEE Internet of Things J., vol. 4, no. 5, pp. 1125-1142 , 2017.
- [Liu] Liu, J., Bai, B., Zhang, J., and K.B. Letaief, "Cache placement in fog-rans: From centralized to distributed algorithms", IEEE Transactions on Wireless Communications, 16(11):7039-7051 , 2017.
- [Mahadev] Satyanarayanan, M., "The Emergence of Edge Computing", Computer, vol. 50, no. 1, pp. 30-39 , 2017.
- [Murshed] Murshed, M., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., and F. Hussain, "Machine learning at the network edge: A survey", arXiv:1908.00080 , 2019.
- [NIST] Mell, P. and T. Grance, "The NIST definition of Cloud Computing", Natl. Inst. Stand. Technol, vol. 53, no. 6, p. 50 , 2009.
- [NVIDIA] Grzywaczewski, A., "Training AI for Self-Driving Vehicles: the Challenge of Scale", NVIDIA Developer Blog , 2017, <<https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/>>.
- [OpenFog] "OpenFog Reference Architecture for Fog Computing", OpenFog Consortium , 2017.
- [PseudoDynamicTesting] Ficco, M., Esposito, C., Xiang, Y., and F. Palmieri, "Pseudo-Dynamic Testing of Realistic Edge-Fog Cloud Ecosystems", IEEE Communications Magazine, Nov. 2017 , 2017.
- [Renart] Renart, E.G., Diaz-Montes, J., and M. Parashar, "Data-driven stream processing at the edge", IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pages 31-40 , 2017.

- [RFC6291] Andersson, L., van Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", BCP 161, RFC 6291, DOI 10.17487/RFC6291, June 2011, <<https://www.rfc-editor.org/info/rfc6291>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [Schafer] Schafer, D., Edinger, J., VanSyckel, S., Paluska, J.M., and C. Becker, "Tasklets: Overcoming Heterogeneity in Distributed Computing Systems", IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW), Nara, pp. 156-161 , 2016.
- [Shi] Shi, W., Cao, J., Zhang, Q., Li, Y., and L. Xu, "Edge computing: vision and challenges", IEEE Internet of Things J., vol. 3, no. 5, pp. 637-646 , 2016.
- [SimulatingFog] Svorobej, S. and . al, "Simulating Fog and Edge Computing Scenarios: An Overview and Research Challenges", MPDI Future Internet 2019 , 2019.
- [Weiner] Weiner, M., Jorgovanovic, M., Sahai, A., and B. Nikolic, "Design of a low-latency, high-reliability wireless communication system for control applications", IEEE Int. Conf. Commun. (ICC), Sydney, NSW, Australia, pp. 3829-3835 , 2014.
- [Yangui] Yangui, S., Ravindran, P., Bibani, O., H Glitho, R., Ben Hadj-Alouane, N., Morrow, M.J., and P.A. Polakos, "A platform as-a-service for hybrid cloud/fog environments", IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 1-7 , 2016.
- [Yates] Yates, R.D. and S.K. Kaul, "The Age of Information: Real-Time Status Updating by Multiple Sources", IEEE Transactions on Information Theory, vol. 65, no. 3, pp. 1807-1827 , 2019.

[Yousefpour]

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and J.P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey", Journal of Systems Architecture, vol. 98, pp. 289-330 , 2019.

[Zhang]

Zhang, Q., Zhang, X., Zhang, Q., Shi, W., and H. Zhong, "Firework: Big data sharing and processing in collaborative edge environment", Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pages 20-25 , 2016.

[Zhang2]

Zhang, J., Chen, B., Zhao, Y., Cheng, X., and F. Hu, "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues", IEEE Access, vol. 6, pp. 18209-18237 , 2018.

[_60802]

IEC/IEEE, ., "Use Cases IEC/IEEE 60802 V1.3", IEC/IEEE 60802 , 2018, <<http://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0818-v13.pdf>>.

Authors' Addresses

Jungha Hong
ETRI
218 Gajeong-ro, Yuseung-Gu
Daejeon

Email: jhong@etri.re.kr

Yong-Geun Hong
ETRI
218 Gajeong-ro, Yuseung-Gu
Daejeon

Email: yghong@etri.re.kr

Xavier de Foy
InterDigital Communications, LLC
1000 Sherbrooke West
Montreal H3A 3G4
Canada

Email: xavier.defoy@interdigital.com

Matthias Kovatsch
Huawei Technologies Duesseldorf GmbH
Riesstr. 25 C // 3.OG
80992 Munich
Germany

Email: ietf@kovatsch.net

Eve Schooler
Intel
2200 Mission College Blvd.
Santa Clara, CA, 95054-1537
United States of America

Email: eve.m.schooler@intel.com

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

Email: ietf@dkutscher.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 30 August 2022

A. Keranen
Ericsson
M. Kovatsch
Huawei Technologies
K. Hartke
26 February 2022

Guidance on RESTful Design for Internet of Things Systems
draft-irtf-t2trg-rest-iot-09

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style. This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG).

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 30 August 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Basics	7
3.1. Architecture	8
3.2. System design	10
3.3. Uniform Resource Identifiers (URIs)	11
3.4. Representations	12
3.5. HTTP/CoAP Methods	13
3.5.1. GET	14
3.5.2. POST	14
3.5.3. PUT	15
3.5.4. DELETE	15
3.5.5. FETCH	15
3.5.6. PATCH	16
3.6. HTTP/CoAP Status/Response Codes	16
4. REST Constraints	16
4.1. Client-Server	17
4.2. Stateless	17
4.3. Cache	18
4.4. Uniform Interface	18
4.5. Layered System	19
4.6. Code-on-Demand	20
5. Hypermedia-driven Applications	20
5.1. Motivation	21
5.2. Knowledge	21
5.3. Interaction	22
5.4. Hypermedia-driven Design Guidance	22
6. Design Patterns	23
6.1. Collections	23
6.2. Calling a Procedure	23
6.2.1. Instantly Returning Procedures	24
6.2.2. Long-running Procedures	24
6.2.3. Conversion	24
6.2.4. Events as State	25
6.3. Server Push	26
7. Security Considerations	27
8. Acknowledgement	28
9. References	28
9.1. Normative References	28
9.2. Informative References	31
Authors' Addresses	34

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems. At its core is a set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability. When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7230] and CoAP [RFC7252]. Since RESTful APIs are often lightweight and enable loose coupling of system components, they are a good fit for various Internet of Things (IoT) applications, which in general aim at interconnecting the physical world with the virtual world. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information.

Design of a good RESTful IoT system has naturally many commonalities with other Web systems. Compared to other systems, the key characteristics of many RESTful IoT systems include:

- * accommodating for constrained devices [RFC7228], so with IoT, REST is not only used for scaling out (large number of clients on a Web server), but also for scaling down (efficient server on constrained node, e.g., in energy consumption or implementation complexity)
- * facilitating efficient transfer over (often) constrained networks and lightweight processing in constrained nodes through compact and simple data formats
- * minimizing or preferably avoiding the need for human interaction through machine-understandable data formats and interaction patterns
- * enabling the system to evolve gradually in the field, as the usually large number of endpoints can not be updated simultaneously
- * having endpoints that are both clients and servers

2. Terminology

This section explains selected terminology that is commonly used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228]. Terminology on modeling of Things and their affordances (Properties, Actions, and Events) was taken from [I-D.ietf-asdf-sdf].

Action: An affordance that can potentially be used to perform a named operation on a Thing.

Action Result: A representation sent as a response by a server that does not represent resource state, but the result of the interaction with the originally addressed resource.

Affordance: An element of an interface offered for interaction, defining its possible uses or making clear how it can or should be used. The term is used here for the digital interfaces of a Thing only; it might also have physical affordances such as buttons, dials, and displays.

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses; it therefore has the initiative to interact. In RESTful IoT systems it is common for nodes to have more than one role (i.e., to be both server and client; see Section 3.1).

Client State: The state kept by a client between requests. This typically includes the currently processed representation, the set of active requests, the history of requests, bookmarks (URIs stored for later retrieval), and application-specific state (e.g., local variables). (Note that this is called "Application State" in [REST], which has some ambiguity in modern (IoT) systems where resources are highly dynamic and the overall state of the distributed application (i.e., application state) is reflected in the union of all Client States and Resource States of all clients and servers involved.)

Content Type: A string that carries the media type plus potential parameters for the representation format such as "text/plain; charset=UTF-8".

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource. The most common forms of content negotiation are Proactive Content Negotiation and Reactive Content Negotiation.

Dereference: To use an access mechanism (e.g., HTTP or CoAP) to interact with the resource of a URI.

Dereferenceable URI: A URI that can be dereferenced, i.e., interaction with the identified resource is possible. Not all HTTP or CoAP URIs are dereferenceable, e.g., when the target resource does not exist.

Event: An affordance that can potentially be used to (recurrently) obtain information about what happened to a Thing, e.g., through server push.

Form: A hypermedia control that enables a client to construct more complex requests, e.g., to change the state of a resource or perform specific queries.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself or to service the response from a cache in order to reduce response time, network bandwidth, and energy consumption.

Gateway: A reverse proxy that provides an interface to a non-RESTful system such as legacy systems or alternative technologies such as Bluetooth Attribute Profile (ATT) or Generic Attribute Profile (GATT). See also "Reverse Proxy".

Hypermedia Control: Information provided by a server on how to use its RESTful API; usually a URI and instructions on how to dereference it for a specific interaction. Hypermedia Controls are the serialized/encoded affordances of hypermedia systems.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the client state.

Link Relation Type: An identifier that describes how the link target resource relates to the current resource (see [RFC8288]).

Media Type: An IANA-registered string such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted and how it is encoded.

Method: An operation associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the expressed preference of the client. For example, an IoT application could send a request that prefers to accept the media type "application/senml+json".

Property: An affordance that can potentially be used to read, write, and/or observe state on a Thing.

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata (e.g., available media types) included in the response.

Representation: A serialization that represents the current or intended state of a resource and that can be transferred between client and server. REST requires representations to be self-describing, meaning that there must be metadata that allows peers to understand which representation format is used. Depending on the protocol needs and capabilities, there can be additional metadata that is transmitted along with the representation.

Representation Format: A set of rules for serializing resource state. On the Web, the most prevalent representation format is HTML. Other common formats include plain text and formats based on JSON [RFC8259], XML, or RDF. Within IoT systems, often compact formats based on JSON, CBOR [RFC8949], and EXI [W3C.REC-exi-20110310] are used.

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource: An item of interest identified by a URI. Anything that

can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Resource State: A model of the possible states of a resource that is expressed in supported representation formats. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model, e.g., business logic implemented on the server side such as sampling a sensor.

Resource Type: An identifier that annotates the application- semantics of a resource (see Section 3.1 of [RFC6690]).

Reverse Proxy: An intermediary that appears as a server towards the client but satisfies the requests by forwarding them to the actual server (possibly via one or more other intermediaries). A reverse proxy is often used to encapsulate legacy services, to improve server performance through caching, and to enable load balancing across multiple machines.

Safe Method: A method that does not result in any state change on the origin server when applied to a resource.

Server: A node that listens for requests, performs the requested operation, and sends responses back to the clients. In RESTful IoT systems it is common for nodes to have more than one role (i.e., to be both server and client; see Section 3.1).

Thing: A physical item that is made available in the Internet of Things, thereby enabling digital interaction with the physical world for humans, services, and/or other Things.

Transfer protocols: In particular in the IoT domain, protocols above the transport layer that are used to transfer data objects and provide semantics for operations on the data.

Transfer layer: Re-usable part of the application layer used to transfer the application specific data items using a standard set of methods that can fulfill application-specific operations.

Uniform Resource Identifier (URI): A global identifier for resources. See Section 3.3 for more details.

3. Basics

3.1. Architecture

The components of a RESTful system are assigned one or both of two roles: client or server. Note that the terms "client" and "server" refer only to the roles that the nodes assume for a particular message exchange. The same node might act as a client in some communications and a server in others. Classic user agents (e.g., Web browsers) are always in the client role and have the initiative to issue requests. Origin servers always have the server role and govern over the resources they host. Simple IoT devices, such as sensors and actuators, are commonly acting as servers and exposing their physical world interaction capabilities (e.g., temperature measurement or door lock control capability) as resources.

Which resources exist and how they can be used is expressed by the servers in so-called affordances, which is metadata that can be included in responses (e.g., the initial response from a well-known resource) or be made available out of band (e.g., through a W3C Thing Description document [W3C-TD] from a directory). In RESTful systems, affordances are encoded as hypermedia controls of which exist two types: links that allow to navigate between resources and forms that enable clients to formulate more complex requests (e.g., to modify a resource or perform a query).

A typical IoT system client can be a cloud service that retrieves data from the sensors and commands the actuators based on the sensor information. Alternatively an IoT data storage system could work as a server where IoT sensor devices send their data in client role.

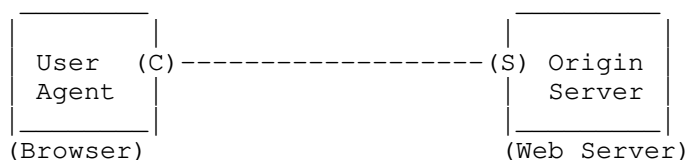


Figure 1: Client-Server Communication

Intermediaries (such as forward proxies, reverse proxies, and gateways) implement both roles, but only forward requests to other intermediaries or origin servers. They can also translate requests to different protocols, for instance, as CoAP-HTTP cross-proxies [RFC8075].

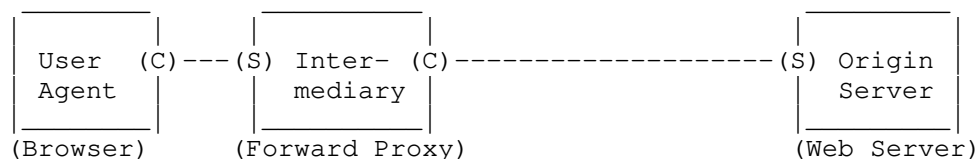


Figure 2: Communication with Forward Proxy

Reverse proxies are usually imposed by the origin server. In addition to the features of a forward proxy, they can also provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. In this case, reverse proxies are usually called gateways. This property is enabled by the Layered System constraint of REST, which says that a client cannot see beyond the server it is connected to (i.e., it is left unaware of the protocol/paradigm change).

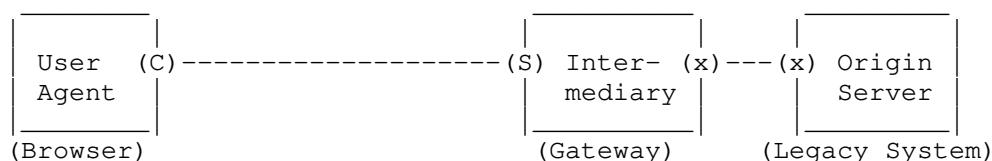


Figure 3: Communication with Reverse Proxy

Nodes in IoT systems often implement both roles. Unlike intermediaries, however, they can take the initiative as a client (e.g., to register with a directory, such as CoRE Resource Directory [I-D.ietf-core-resource-directory], or to interact with another Thing) and act as origin server at the same time (e.g., to serve sensor values or provide an actuator interface).

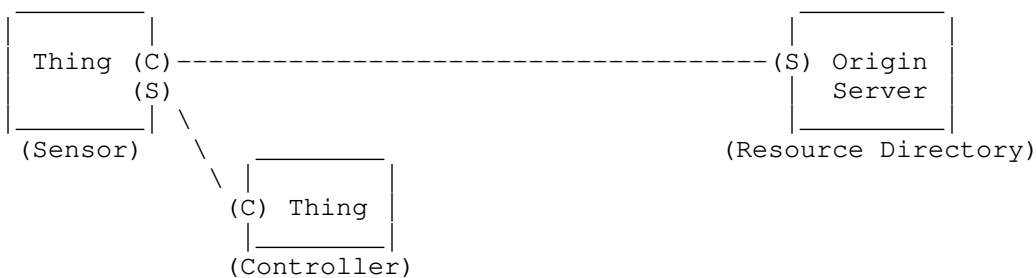


Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the application as distributed state and assigning it to the different components (i.e., clients and servers). The secondary effort is then selecting or designing the necessary representation formats to exchange information and enable interaction between the components through resources.

How clients can navigate through the resource space and modify state to achieve their goals is encoded in hypermedia controls, that is, links and forms within the representations. The concept behind hypermedia controls is to provide machine-understandable "affordances" [HCI], which refer to the perceived and actual properties of a Thing and determine how it could possibly be used. A physical door may have a door knob as affordance, indicating that the door can be opened by twisting the knob; a keyhole may indicate that it can be locked. For Things in the IoT, these affordances may be serialized as two hypermedia forms, which include semantic identifiers from a controlled vocabulary (e.g., schema.org) and the instructions on how to formulate the requests for opening and locking, respectively. Overall, this allows to realize a Uniform Interface (see Section 4.4), which enables loose coupling between clients and servers.

Hypermedia controls span a kind of state machine where the nodes are resources (or action results) and the transitions are links or forms. Clients run this distributed state machine (i.e., the application) by retrieving representations, processing the data, and following the included links and/or submitting forms to modify remote state. This is usually done by retrieving the current state, modifying the state on the client side, and transferring the new state to the server in the form of new representations -- rather than calling a service and modifying the state on the server side.

Client state encompasses the current state of the described state machine and the possible next transitions derived from the hypermedia controls within the currently processed representation. Furthermore, clients can have part of the state of the distributed application in local variables.

Resource state includes the more persistent data of an application (i.e., independent of individual clients). This can be static data such as device descriptions, persistent data such as system configurations, but also dynamic data such as the current value of a sensor on a Thing.

In the design, it is important to distinguish between "client state" and "resource state", and keep them separate. Following the Stateless constraint, the client state must be kept only on clients. That is, there is no establishment of shared information about past and future interactions between client and server (usually called a session). On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient and scalable, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances (cf. load balancing). For IoT systems, this constraint lowers the memory requirements for server implementations, which is particularly important for constrained servers (e.g., sensor nodes) and servers serving large amount of clients (e.g., Resource Directory).

3.3. Uniform Resource Identifiers (URIs)

An important aspect of RESTful API design is to model the system as a set of resources, which potentially can be created and/or deleted dynamically and whose state can be retrieved and/or modified.

Uniform Resource Identifiers (URIs) are used to indicate resources for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

foo://example.com:8042/over/there?name=ferret#nose

scheme authority path query fragment

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "https" scheme. A hostname (e.g., a fully qualified domain name) or an IP address literal, potentially followed by a transport layer port number, are usually used for the authority component. The path and query contain data to identify a resource within the scope of the scheme-dependent naming authority (i.e., "http://www.example.org/" is a different authority than "https://www.example.org"). The fragment allows referring to some portion of the resource, such as a Record in a SenML Pack (Section 9 of [RFC8428]). However, fragments are processed only at client side

and not sent on the wire. [RFC8820] provides more details on URI design and ownership with best current practices for establishing URI structures, conventions, and formats.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS, [RFC5246] for TLS 1.2 and [RFC8446] for TLS 1.3). (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the protocols HTTP for "http" and "https" and CoAP for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the remainder of the URI is identical, these are two different resources, in two distinct namespaces.

Some schemes are for URIs with main purpose as identifiers, and hence are not dereferenceable, e.g., the "urn" scheme can be used to construct unique names in registered namespaces. In particular the "urn:dev" URI [RFC9039] details multiple ways for generating and representing endpoint identifiers of IoT devices.

The query parameters can be used to parameterize the resource. For example, a GET request may use query parameters to request the server to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not used consistently. Whether the order of the query parameters matters in URIs is unspecified; they can be re-ordered, for instance by proxies. Therefore, applications should not rely on their order; see Section 3.3.4 of [RFC6943] for more details.

Due to the relatively complex processing rules and text representation format, URI handling can be difficult to implement correctly in constrained devices. Constrained Resource Identifiers [I-D.ietf-core-href] provide a CBOR-based format of URIs that is better suited also for resource constrained IoT devices.

3.4. Representations

Clients can retrieve the resource state from a server or manipulate resource state on the (origin) server by transferring resource representations. Resource representations must have metadata that identifies the representation format used, so the representations can be interpreted correctly. This is usually a simple string such as the IANA-registered Internet Media Types. Typical media types for IoT systems include:

- * "text/plain" for simple UTF-8 text

- * "application/octet-stream" for arbitrary binary data
- * "application/json" for the JSON format [RFC8259]
- * "application/cbor" for CBOR [RFC8949]
- * "application/exi" for EXI [W3C.REC-exi-20110310]
- * "application/link-format" for CoRE Link Format [RFC6690]
- * "application/senml+json" and "application/senml+cbor" for Sensor Measurement Lists (SenML) data [RFC8428]

A full list of registered Internet Media Types is available at the IANA registry [IANA-media-types] and numerical identifiers for media types, parameters, and content codings registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

The terms "media type", "content type" (media type plus potential parameters), and "content format" (short identifier of content type and content coding, abbreviated for historical reasons "ct") are often used when referring to representation formats used with CoAP. The differences between these terms are discussed in more detail in [I-D.bormann-core-media-content-type-format].

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. As part of the Uniform Interface constraint, each method can have certain properties that give guarantees to clients.

Safe methods do not cause any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource. Thus, it is always safe for a client to retrieve a representation without affecting server-side state.

Idempotent methods can be applied multiple times to the same resource while causing the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same state for the resource. However, the response from the server can be different when the same idempotent method is used multiple times. For example when DELETE is used twice on an existing resource, the first request would remove the association and return success acknowledgement whereas the second request would likely result in error response due to non-existing resource.

The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource, while the origin server must ensure that there are no side effects on the resource state. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field (with HTTP) or Location-Path and/or Location-Query Options (with CoAP) that provide an identifier for the resource created. The server also includes a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent. A PUT request applied to the target resource can have side effects on other resources.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.5.5. FETCH

The CoAP-specific FETCH method [RFC8132] requests a representation of a resource parameterized by a representation enclosed in the request.

The fundamental difference between the GET and FETCH methods is that the request parameters are included as the payload of a FETCH request, while in a GET request they're typically part of the query string of the request URI.

The FETCH method is safe and idempotent.

3.5.6. PATCH

The PATCH method [RFC5789] [RFC8132] requests that a set of changes described in the request entity be applied to the target resource.

The PATCH method is not safe nor idempotent.

The CoAP-specific iPATCH method is a variant of the PATCH method that is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" with HTTP or "4.04" with CoAP) where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed. Codes starting with 2 indicate a successful request. Codes starting with 3 indicate redirection; further action is needed to complete the request. Codes starting with 4 and 5 indicate errors. The codes starting with 4 mean client error (e.g., bad syntax in the request) whereas codes starting with 5 mean server error; there was no apparent problem with the request, but server was not able to fulfill the request.

Responses may be stored in a cache to satisfy future, equivalent requests. HTTP and CoAP use two different patterns to decide what responses are cacheable. In HTTP, the cacheability of a response depends on the request method (e.g., responses returned in reply to a GET request are cacheable). In CoAP, the cacheability of a response depends on the response code (e.g., responses with code 2.04 are cacheable). This difference also leads to slightly different semantics for the codes starting with 2; for example, CoAP does not have a 2.00 response code whereas 200 ("OK") is commonly used with HTTP.

4. REST Constraints

The REST architectural style defines a set of constraints for the system design. When all constraints are applied correctly, REST enables architectural properties of key interest [REST]:

- * Performance

- * Scalability
- * Reliability
- * Simplicity
- * Modifiability
- * Visibility
- * Portability

The following subsections briefly summarize the REST constraints and explain how they enable the listed properties.

4.1. Client-Server

As explained in the Architecture section, RESTful system components have clear roles in every interaction. Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.

This improves simplicity and visibility (also for digital forensics), as it is clear which component started an interaction. Furthermore, it improves modifiability through a clear separation of concerns.

In IoT systems, endpoints often assume both roles of client and (origin) server simultaneously. When an IoT device has initiative (because there is a user, e.g., pressing a button, or installed rules/policies), it acts as a client. When a device offers a service, it is in server role.

4.2. Stateless

The Stateless constraint requires messages to be self-contained. They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state.

This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions. Furthermore, the Stateless constraint enables caching.

For IoT, the scaling properties of REST become particularly important. Note that being self-contained does not necessarily mean that all information has to be inlined. Constrained IoT devices may

choose to externalize metadata and hypermedia controls using Web linking, so that only the dynamic content needs to be sent and the static content such as schemas or controls can be cached.

4.3. Cache

This constraint requires responses to have implicit or explicit cache-control metadata. This enables clients and intermediary to store responses and re-use them to locally answer future requests. The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.

Cache improves performance, as less data needs to be transferred and response times can be reduced significantly. Needing fewer transfers also improves scalability, as origin servers can be protected from too many requests. Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

Caching usually only makes sense when the data is used by multiple participants. In IoT systems, however, it might make sense to cache also individual data to protect constrained devices and networks from frequent requests of data that does not change often. Security often hinders the ability to cache responses. For IoT systems, object security [RFC8613] may be preferable over transport layer security, as it enables intermediaries to cache responses while preserving security.

4.4. Uniform Interface

All RESTful APIs use the same, uniform interface independent of the application. This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state -- which applies to all applications.

In contrast, in a service-oriented RPC approach, all required ways to modify state need to be modeled explicitly in the interface resulting in a large set of methods -- which differs from application to application. Moreover, it is also likely that different parties come up with different ways how to modify state, including the naming of the procedures, while the state within an application is a bit easier to agree on.

A REST interface is fully defined by:

- * URIs to identify resources
- * representation formats to represent and manipulate resource state
- * self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)
- * hypermedia controls within representations

The concept of hypermedia controls is also known as HATEOAS: Hypermedia As The Engine Of Application State. The origin server embeds controls for the interface into its representations and thereby informs the client about possible next requests. The most used control for RESTful systems today is Web Linking [RFC8288]. Hypermedia forms are more powerful controls that describe how to construct more complex requests, including representations to modify resource state.

While this is the most complex constraints (in particular the hypermedia controls), it improves many key properties. It improves simplicity, as uniform interfaces are easier to understand. The self-descriptive messages improve visibility. The limitation to a known set of representation formats fosters portability. Most of all, however, this constraint is the key to modifiability, as hypermedia-driven, uniform interfaces allow clients and servers to evolve independently, and hence enable a system to evolve.

For a large number of IoT applications, the hypermedia controls are mainly used for the discovery of resources, as they often serve sensor data. Such resources are "dead ends", as they usually do not link any further and only have one form of interaction: fetching the sensor value. For IoT, the critical parts of the Uniform Interface constraint are the descriptions of messages and representation formats used. Simply using, for instance, "application/json" does not help machine clients to understand the semantics of the representation. Yet defining very precise media types limits the re-usability and interoperability. Representation formats such as SenML [RFC8428] try to find a good trade-off between precision and re-usability. Another approach is to combine a generic format such as JSON with syntactic as well as semantic annotations (see [I-D.handrews-json-schema-validation] and [W3C-TD], resp.).

4.5. Layered System

This constraint enforces that a client cannot see beyond the server with which it is interacting.

A layered system is easier to modify, as topology changes become transparent. Furthermore, this helps scalability, as intermediaries such as load balancers can be introduced without changing the client side. The clean separation of concerns helps with simplicity.

IoT systems greatly benefit from this constraint, as it allows to effectively shield constrained devices behind intermediaries and is also the basis for gateways, which are used to integrate other (IoT) ecosystems.

4.6. Code-on-Demand

This principle enables origin servers to ship code to clients.

Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format). It also improves performance, as the server can provide code for local pre-processing before transferring the data.

As of today, code-on-demand has not been explored much in IoT systems. Aspects to consider are that either one or both nodes are constrained and might not have the resources to host or dynamically fetch and execute such code. Moreover, the origin server often has no understanding of the actual application a mashup client realizes. Still, code-on-demand can be useful for small polyfills, e.g., to decode payloads, and potentially other features in the future.

5. Hypermedia-driven Applications

Hypermedia-driven applications take advantage of hypermedia controls, i.e., links and forms, which are embedded in representations or response message headers. A hypermedia client is a client that is capable of processing these hypermedia controls. Hypermedia links can be used to give additional information about a resource representation (e.g., the source URI of the representation) or pointing to other resources. The forms can be used to describe the structure of the data that can be sent (e.g., with a POST or PUT method) to a server, or how a data retrieval (e.g., GET) request for a resource should be formed. In a hypermedia-driven application the client interacts with the server using only the hypermedia controls, instead of selecting methods and/or constructing URIs based on out-of-band information, such as API documentation. The Constrained RESTful Application Language (CoRAL) [I-D.ietf-core-coral] provides a hypermedia-format that is suitable for constrained IoT environments.

5.1. Motivation

The advantage of this approach is increased evolvability and extensibility. This is important in scenarios where servers exhibit a range of feature variations, where it's expensive to keep evolving client knowledge and server knowledge in sync all the time, or where there are many different client and server implementations. Hypermedia controls serve as indicators in capability negotiation. In particular, they describe available resources and possible operations on these resources using links and forms, respectively.

There are multiple reasons why a server might introduce new links or forms:

- * The server implements a newer version of the application. Older clients ignore the new links and forms, while newer clients are able to take advantage of the new features by following the new links and submitting the new forms.
- * The server offers links and forms depending on the current state. The server can tell the client which operations are currently valid and thus help the client navigate the application state machine. The client does not have to have knowledge which operations are allowed in the current state or make a request just to find out that the operation is not valid.
- * The server offers links and forms depending on the client's access control rights. If the client is unauthorized to perform a certain operation, then the server can simply omit the links and forms for that operation.

5.2. Knowledge

A client needs to have knowledge of a couple of things for successful interaction with a server. This includes what resources are available, what representations of resource states are available, what each representation describes, how to retrieve a representation, what state changing operations on a resource are possible, how to perform these operations, and so on.

Some part of this knowledge, such as how to retrieve the representation of a resource state, is typically hard-coded in the client software. For other parts, a choice can often be made between hard-coding the knowledge or acquiring it on-demand. The key to success in either case is the use of in-band information for identifying the knowledge that is required. This enables the client to verify that it has all the required knowledge or to acquire missing knowledge on-demand.

A hypermedia-driven application typically uses the following identifiers:

- * URI schemes that identify communication protocols,
- * Internet Media Types that identify representation formats,
- * link relation types or resource types that identify link semantics,
- * form relation types that identify form semantics,
- * variable names that identify the semantics of variables in templated links, and
- * form field names that identify the semantics of form fields in forms.

The knowledge about these identifiers as well as matching implementations have to be shared a priori in a RESTful system.

5.3. Interaction

A client begins interacting with an application through a GET request on an entry point URI. The entry point URI is the only URI a client is expected to know before interacting with an application. From there, the client is expected to make all requests by following links and submitting forms that are provided in previous responses. The entry point URI can be obtained, for example, by manual configuration or some discovery process (e.g., DNS-SD [RFC6763] or Resource Directory [I-D.ietf-core-resource-directory]). For Constrained RESTful environments `"/.well-known/core"` relative URI is defined as a default entry point for requesting the links hosted by servers with known or discovered addresses [RFC6690].

5.4. Hypermedia-driven Design Guidance

Assuming self-describing representation formats (i.e., human-readable with carefully chosen terms or processable by a formatting tool) and a client supporting the URI scheme used, a good rule of thumb for a good hypermedia-driven design is the following: A developer should only need an entry point URI to drive the application. All further information how to navigate through the application (links) and how to construct more complex requests (forms) are published by the server(s). There must be no need for additional, out-of-band information (e.g., API specification).

For machines, a well-chosen set of information needs to be shared a priori to agree on machine-understandable semantics. Agreeing on the exact semantics of terms for relation types and data elements will of course also help the developer. [I-D.hartke-core-apps] proposes a convention for specifying the set of information in a structured way.

6. Design Patterns

Certain kinds of design problems are often recurring in variety of domains, and often re-usable design patterns can be applied to them. Also, some interactions with a RESTful IoT system are straightforward to design; a classic example of reading a temperature from a thermometer device is almost always implemented as a GET request to a resource that represents the current value of the thermometer. However, certain interactions, for example data conversions or event handling, do not have as straightforward and well established ways to represent the logic with resources and REST methods.

The following sections describe how common design problems such as different interactions can be modeled with REST and what are the benefits of different approaches.

6.1. Collections

A common pattern in RESTful systems across different domains is the collection. A collection can be used to combine multiple resources together by providing resources that consist of set of (often partial) representations of resources, called items, and links to resources. The collection resource also defines hypermedia controls for managing and searching the items in the collection.

Examples of the collection pattern in RESTful IoT systems are the CoRE Resource Directory [I-D.ietf-core-resource-directory], CoAP pub/sub broker [I-D.ietf-core-coap-pubsub], and resource discovery via ".well-known/core". Collection+JSON [CollectionJSON] is an example of a generic collection Media Type.

6.2. Calling a Procedure

To modify resource state, clients usually use GET to retrieve a representation from the server, modify that locally, and transfer the resulting state back to the server with a PUT (see Section 4.4). Sometimes, however, the state can only be modified on the server side, for instance, because representations would be too large to transfer or part of the required information shall not be accessible to clients. In this case, resource state is modified by calling a procedure (or "function"). This is usually modeled with a POST request, as this method leaves the behavior semantics completely to

the server. Procedure calls can be divided into two different classes based on how long they are expected to execute: "instantly" returning and long-running.

6.2.1. Instantly Returning Procedures

When the procedure can return within the expected response time of the system, the result can be directly returned in the response. The result can either be actual content or just a confirmation that the call was successful. In either case, the response does not contain a representation of the resource, but a so-called action result. Action results can still have hypermedia controls to provide the possible transitions in the application state machine.

6.2.2. Long-running Procedures

When the procedure takes longer than the expected response time of the system, or even longer than the response timeout, it is a good pattern to create a new resource to track the "task" execution. The server would respond instantly with a "Created" status (HTTP code 201 or CoAP 2.01) and indicate the location of the task resource in the corresponding header field (or CoAP option) or as a link in the action result. The created resource can be used to monitor the progress, to potentially modify queued tasks or cancel tasks, and to eventually retrieve the result.

Monitoring information would be modeled as state of the task resource, and hence be retrievable as representation. The result -- when available -- can be embedded in the representation or given as a link to another sub-resource. Modifying tasks can be modeled with forms that either update sub-resources via PUT or do a partial write using PATCH or POST. Canceling a task would be modeled with a form that uses DELETE to remove the task resource.

6.2.3. Conversion

A conversion service is a good example where REST resources need to behave more like a procedure call. The knowledge of converting from one representation to another is located only at the server to relieve clients from high processing or storing lots of data. There are different approaches that all depend on the particular conversion problem.

As mentioned in the previous sections, POST request are a good way to model functionality that does not necessarily affect resource state. When the input data for the conversion is small and the conversion result is deterministic, however, it can be better to use a GET request with the input data in the URI query part. The query is

parameterizing the conversion resource, so that it acts like a look-up table. The benefit is that results can be cached also for HTTP (where responses to POST are not cacheable). In CoAP, cacheability depends on the response code, so that also a response to a POST request can be made cacheable through a 2.05 Content code.

When the input data is large or has a binary encoding, it is better to use POST requests with a proper Media Type for the input representation. A POST request is also more suitable, when the result is time-dependent and the latest result is expected (e.g., exchange rates).

6.2.4. Events as State

In event-centric paradigms such as pub/sub, events are usually represented by an incoming message that might even be identical for each occurrence. Since the messages are queued, the receiver is aware of each occurrence of the event and can react accordingly. For instance, in an event-centric system, ringing a doorbell would result in a message being sent that represents the event that it was rung.

In resource-oriented paradigms such as REST, messages usually carry the current state of the remote resource, independent from the changes (i.e., events) that have lead to that state. In a naive yet natural design, a doorbell could be modeled as a resource that can have the states unpressed and pressed. There are, however, a few issues with this approach. Polling (i.e., periodically retrieving) the doorbell resource state is not a good option, as the client is highly unlikely to be able to observe all the changes in the pressed state with any realistic polling interval. When using CoAP Observe with Confirmable notifications, the server will usually send two notifications for the event that the doorbell was pressed: notification for changing from unpressed to pressed and another one for changing back to unpressed. If the time between the state changes is very short, the server might drop the first notification, as Observe only guarantees eventual consistency (see Section 1.3 of [RFC7641]).

The solution is to pick a state model that fits better to the application. In the case of the doorbell -- and many other event-driven resources -- the solution could be a counter that counts how often the bell was pressed. The corresponding action is taken each time the client observes a change in the received representation. In the case of a network outage, this could lead to a ringing sound long after the bell was rung. Also including a timestamp of the last counter increment in the state can help to suppress ringing a sound when the event has become obsolete. Another solution would be to change the client/server roles of the doorbell button and the ringer, as described in Section 6.3.

6.3. Server Push

Overall, a universal mechanism for server push, that is, change-of-state notifications and stand-alone event notifications, is still an open issue that is being discussed in the Thing-to-Thing Research Group. It is connected to the state-event duality problem and custody transfer, that is, the transfer of the responsibility that a message (e.g., event) is delivered successfully.

A proficient mechanism for change-of-state notifications is currently only available for CoAP: Observing resources [RFC7641]. The CoAP Observe mechanism offers eventual consistency, which guarantees "that if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state". It intrinsically deals with the challenges of lossy networks, where notifications might be lost, and constrained networks, where there might not be enough bandwidth to propagate all changes.

For stand-alone event notifications, that is, where every single notification contains an identifiable event that must not be lost, observing resources is not a good fit. A better strategy is to model each event as a new resource, whose existence is notified through change-of-state notifications of an index resource (cf. Collection pattern). Large numbers of events will cause the notification to grow large, as it needs to contain a large number of Web links. Block-wise transfers [RFC7959] can help here. When the links are ordered by freshness of the events, the first block can already contain all links to new events. Then, observers do not need to retrieve the remaining blocks from the server, but only the representations of the new event resources.

An alternative pattern is to exploit the dual roles of IoT devices, in particular when using CoAP: they are usually client and server at the same time. An endpoint interested in observing the events would subscribe to them by registering a callback URI at the origin server,

e.g., using a POST request with the URI or a hypermedia document in the payload, and receiving the location of a temporary "subscription resource" as handle in the response. The origin server would then publish events by sending requests containing the event data to the observer's callback URI; here POST can be used to add events to a collection located at the callback URI or PUT can be used when the event data is a new state that shall replace the outdated state at the callback URI. The cancellation can be modeled through deleting the subscription resource. This pattern makes the origin server responsible for delivering the event notifications. This goes beyond retransmissions of messages; the origin server is usually supposed to queue all undelivered events and to retry until successful delivery or explicit cancellation. In HTTP, this pattern is known as REST Hooks.

Methods for configuring server push and notification conditions with CoAP are provided by the CoRE Dynamic Resource Linking specification [I-D.ietf-core-dynlink].

In HTTP, there exist a number of workarounds to enable server push, e.g., long polling and streaming [RFC6202] or server-sent events [W3C.REC-html5-20141028]. In IoT systems, long polling can introduce a considerable overhead, as the request has to be repeated for each notification. Streaming and server-sent events (the latter is actually an evolution of the former) are more efficient, as only one request is sent. However, there is only one response header and subsequent notifications can only have content. Individual status and metadata needs to be included in the content message. This reduces HTTP again to a pure transport, as its status signaling and metadata capabilities cannot be used.

7. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. We assume that system designers apply classic Web security on top of the basic RESTful guidance given in this document. Thus, security protocols and considerations from related specifications apply to RESTful IoT design. These include:

- * Transport Layer Security (TLS): [RFC8446], [RFC5246], and [RFC6347]
- * Internet X.509 Public Key Infrastructure: [RFC5280]
- * HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.
- * CoAP security: Section 11 of [RFC7252]

- * URI security: Section 7 of [RFC3986]

IoT-specific security is active area of standardization at the time of writing. First finalized specifications include:

- * (D)TLS Profiles for the Internet of Things: [RFC7925]
- * CBOR Object Signing and Encryption (COSE) [RFC8152]
- * CBOR Web Token [RFC8392]
- * Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs) [RFC8747]
- * Object Security for Constrained RESTful Environments (OSCORE) [RFC8613]
- * Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework [I-D.ietf-ace-oauth-authz]
- * ACE profiles for DTLS [I-D.ietf-ace-dtls-authorize] and OSCORE [I-D.ietf-ace-oscore-profile]

Further IoT security considerations are available in [RFC8576].

8. Acknowledgement

The authors would like to thank Mike Amundsen, Heidi-Maria Back, Carsten Bormann, Tero Kauppinen, Michael Koster, Mert Ocak, Robby Simpson, Ravi Subramaniam, Dave Thaler, Niklas Widell, and Erik Wilde for the reviews and feedback.

9. References

9.1. Normative References

- [I-D.ietf-core-coral]
Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-04, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-coral-04.txt>>.

[I-D.ietf-core-dynlink]

Koster, M. and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", Work in Progress, Internet-Draft, draft-ietf-core-dynlink-14, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-dynlink-14.txt>>.

[I-D.ietf-core-href]

Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-09, 15 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-href-09.txt>>.

[I-D.ietf-core-resource-directory]

Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt>>.

[REST]

Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.

[RFC3986]

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.

[RFC5246]

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.

[RFC5280]

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.

[RFC6202]

Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <<https://www.rfc-editor.org/info/rfc6202>>.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <<https://www.rfc-editor.org/info/rfc9039>>.

[W3C.REC-exi-20110310] Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, 10 March 2011, <<https://www.w3.org/TR/2011/REC-exi-20110310>>.

[W3C.REC-html5-20141028] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, 28 October 2014, <<https://www.w3.org/TR/2014/REC-html5-20141028>>.

9.2. Informative References

[CollectionJSON] Amundsen, M., "Collection+JSON - Document Format", February 2013, <<http://amundsen.com/media-types/collection/format/>>.

[HCI] Interaction Design Foundation, "The Encyclopedia of Human-Computer Interaction", 2nd Ed., 2013, <<https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed>>.

[I-D.bormann-core-media-content-type-format] Bormann, C. and H. Birkholz, "On Media-Types, Content-Types, and related terminology", Work in Progress, Internet-Draft, draft-bormann-core-media-content-type-format-04, 22 February 2021, <<https://www.ietf.org/archive/id/draft-bormann-core-media-content-type-format-04.txt>>.

[I-D.handrews-json-schema-validation] Wright, A., Andrews, H., and B. Hutton, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", Work in Progress, Internet-Draft, draft-handrews-json-schema-validation-02, 17 September 2019, <<https://www.ietf.org/archive/id/draft-handrews-json-schema-validation-02.txt>>.

[I-D.hartke-core-apps]

Hartke, K., "CoRE Applications", Work in Progress, Internet-Draft, draft-hartke-core-apps-08, 22 October 2018, <<https://www.ietf.org/archive/id/draft-hartke-core-apps-08.txt>>.

[I-D.ietf-ace-dtls-authorize]

Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

[I-D.ietf-asdf-sdf]

Koster, M. and C. Bormann, "Semantic Definition Format (SDF) for Data and Interactions of Things", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-10, 16 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-asdf-sdf-10.txt>>.

[I-D.ietf-core-coap-pubsub]

Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>>.

- [IANA-CoAP-media]
"CoAP Content-Formats", n.d.,
<<http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>>.
- [IANA-media-types]
"Media Types", n.d., <<http://www.iana.org/assignments/media-types/media-types.xhtml>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<https://www.rfc-editor.org/info/rfc5789>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.

- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.
- [RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <<https://www.rfc-editor.org/info/rfc8576>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <<https://www.rfc-editor.org/info/rfc8820>>.
- [W3C-TD] Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., and M. Kovatsch, "Web of Things (WoT) Thing Description", April 2020, <<https://www.w3.org/TR/wot-thing-description/>>.

Authors' Addresses

Ari Keranen
Ericsson
FI-02420 Jorvas
Finland
Email: ari.keranen@ericsson.com

Matthias Kovatsch
Huawei Technologies
Riesstr. 25
D-80992 Munich
Germany
Email: matthias.kovatsch@huawei.com

Klaus Hartke
Email: hartke@projectcool.de

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 24, 2020

I. Petrov, Ed.
Acklio
July 23, 2019

YANG Object Universal Parsing Interface
draft-petrov-t2trg-youpi-00

Abstract

YANG Object Universal Parsing Interface (YOUPI) specification describes generic way to encode and decode binary data based on a YANG model for use of constrained devices. YOUPI is a generic mechanism designed for great flexibility, so that it can be adapted for any of the constrained devices.

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 24, 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
1.1. Terminology	3
2. YOUPI	3
2.1. YANG extentions	3
2.2. Position	4
2.2.1. Bit positions	4
2.2.2. Cursor	5
2.2.3. Absolute position	5
2.2.4. Relative position	5
2.3. FieldIndex	5
2.4. Condition	6
2.5. Multiplier	6
2.6. Offset	6
2.7. Units-subject	6
2.8. Lists	6
3. Security Considerations	7
4. IANA Considerations	7
5. Normative References	7
Author's Address	7

1. Introduction

A huge number of very constraint IoT devices are expected to be coming to the market. They are very constraint in terms of the MTU (sometimes as small as 10b per message). As they are expected to be running for many years without the need for external energy, energy efficiency which is directly linked to the size of the payloads that need to be sent, is also very important. For those devices JSON and even CBOR formats might be too wasteful in terms of payload size. The reality of the ecosystem is that currently a great number of applications use proprietary binary formats for exchanging information. A significant problem exists if those systems are to be interacting in a purely M2M fashion. While there are a number of possibilities to resolve those issues, due to the constraints it is mandatory to have a way to extract and encode information from/to the binary payload and be able to annotate it with semantic metadata.

While binary formats can be rather complicated to parse and sometimes even context dependent (some entity needs to keep context in order to parse a message), for most cases a simple description format could be sufficient.

A good solution should not be bounded to the output format. It should be a data modeling language like YANG [RFC7950] that simply describes the structure of the obtained data and that allows different serialization formats afterwards.

1.1. Terminology

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

2. YOUPI

YOUPI provides a number of yang extentions as defined in Section 2.1. Thanks to that additional information in the YANG definitions, it is possible to decode binary data and then transform it to a different easier to parse format like JSON, XML or CBOR. Additionally it defines extensions that allow meta information to be added so that JSON-LD is generated. This draft is not describing how the data is formatted as JSON or other format. For information how this could be done, please refer to RESTCONF, NETCONF or CORECONF.

The opposite process is also possible - generating binary packets from parsed data that comes from JSON or other format.

2.1. YANG extentions

The definitions of the YANG extensions.

```
<CODE BEGINS> file "petrov-youpi-file@2019-07-22.yang"
module youpi {
    namespace "http://ackl.io/youpi";

    prefix "youpi";

    organization
        "Acklio";

    contact
        "Ivaylo Petrov
        <mailto:ivaylo@ackl.io>";

    description
        "This module defines the extentions used by youpi.";

    revision 2019-07-22 {
        description "Initial revision.";
    }

    /**
     *
```

```
* Extension for Binary data to CBOR mapping.
*
**/
extension position {
    argument object;
}

extension fieldIndex {
    argument object;
}

extension condition {
    argument object;
}

extension multiplier {
    argument object;
}

extension offset {
    argument object;
}

extension units-subject {
    argument object;
}

extension js {
    argument object;
}
}
```

2.2. Position

Information about which bits need to be used in order to find the value of a field.

2.2.1. Bit positions

If the position is not present or is empty, the value contains 0 bits and has a default value of 0 (or equivalent for the given type). Could be useful if a field needs to be the result of arithmetic operations from different fields.

It is possible to have a single bit read by giving only its value in the position extension.

If continuous bits need to be used to obtain the value of a given field, this can be achieved using the ".." syntax. For example "0..3" means bits 0, 1, 2 and 3.

If non-continuous bits need to be used, one can use the concatenation of bit ranges using the "|" operator. For example "0..1 | 3".

2.2.2. Cursor

Starts at 0 and changes with each read to the last bit index that was read. Used in Section 2.2.4 to determine where the read will start from. Section 2.2.3 is not affected by it, but changes its value.

2.2.3. Absolute position

The default one if no keyword is used. Alternatively "absolute" keyword can be provided to explicitly request such position.

Example:

```
leaf temp {
    type uint8;
    default -19;
    description "The temperature";
    youpi:position "0..6";
}
```

2.2.4. Relative position

Example:

```
leaf temp {
    type uint8;
    default -19;
    description "The temperature";
    youpi:position "1..7";
}
```

This means that the value starts 1 bit after the current cursor and will read up to 7 bits after the current cursor position, including that 7th bit.

2.3. FieldIndex

Can be used to change the order in which fields are processed. By default the order in which fields appear in the document is the order in which they are processed.

2.4. Condition

Inside a choice statement, the condition extension gives information based on what value the choice will be decided.

For example considering that there is a value "mode" with the value of btn inside the model

```
choice data {
  case _btn {
    container button-data {
      ...
    }
  }
  case _temp {
    container temperature-data {
      ...
    }
  }
  youpi:condition "mode";
}
```

Then the button-data container will be used to parse the data.

2.5. Multiplier

A value or another field by which a given field needs to be multiplied before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in Section 2.6).

2.6. Offset

A value or another field to which a given field needs to be added before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in Section 2.6).

2.7. Units-subject

Meta information used to compute JSON-LD.

2.8. Lists

TBD

3. Security Considerations

The YANG file should be valid.

Segmentation faults might result from invalid data being provided with a given YANG model.

Resource exhaustion can be looked for.

4. IANA Considerations

This document registers a YANG model.

5. 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>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

Author's Address

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 7, 2020

I. Petrov, Ed.
Acklio
November 04, 2019

YANG Object Universal Parsing Interface
draft-petrov-t2trg-youpi-01

Abstract

YANG Object Universal Parsing Interface (YOUPI) specification describes generic way to encode and decode binary data based on a YANG model for use of constrained devices. YOUPI is a generic mechanism designed for great flexibility, so that it can be adapted for any of the constrained devices.

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 May 7, 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
1.1. Terminology	3
2. YOUPI	3
2.1. YANG extentions	3
2.2. Position	5
2.2.1. Bit positions	5
2.2.2. Cursor	5
2.2.3. Absolute position	5
2.2.4. Relative position	5
2.3. FieldIndex	6
2.4. Multiplier	6
2.5. Offset	6
2.6. Units-subject	6
2.7. Data definitions	6
2.7.1. Supported built-in type	6
2.7.2. Leafs	7
2.7.3. Type min/max values	7
2.7.4. Type fraction digits	7
2.7.5. Containers	8
2.7.6. Condition	8
2.7.7. Lists	9
2.7.8. Enumerations as mappings	10
2.7.9. Groupings	10
2.7.10. Typedefs	10
3. Security Considerations	10
4. IANA Considerations	11
Acknowledgements	11
Contributors	11
7. Normative References	11
Appendix A. Complete examples	11
Author's Address	11

1. Introduction

A huge number of very constraint IoT devices are expected to be coming to the market. They are very constraint in terms of the MTU (sometimes as small as 10b per message). As they are expected to be running for many years without the need for external energy, energy efficiency which is directly linked to the size of the payloads that need to be sent, is also very important. For those devices JSON and even CBOR formats might be too wasteful in terms of payload size. The reality of the ecosystem is that currently a great number of applications use proprietary binary formats for exchanging information. A significant problem exists if those systems are to be interacting in a purely M2M fashion. While there are a number of possibilities to resolve those issues, due to the constraints it is

mandatory to have a way to extract and encode information from/to the binary payload and be able to annotate it with semantic metadata.

While binary formats can be rather complicated to parse and sometimes even context dependent (some entity needs to keep context in order to parse a message), for most cases a simple description format could be sufficient.

A good solution should not be bounded to the output format. It should be a data modeling language like YANG [RFC7950] that simply describes the structure of the obtained data and that allows different serialization formats afterwards.

1.1. Terminology

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

2. YOUPI

YOUPI provides a number of yang extentions as defined in Section 2.1. Thanks to that additional information in the YANG definitions, it is possible to decode binary data and then transform it to a different easier to parse format like JSON, XML or CBOR. Additionally it defines extensions that allow meta information to be added so that JSON-LD is generated. This draft is not describing how the data is formatted as JSON or other format. For information how this could be done, please refer to RESTCONF, NETCONF or CORECONF.

The opposite process is also possible - generating binary packets from parsed data that comes from JSON or other format.

2.1. YANG extentions

The definitions of the YANG extensions.

```
<CODE BEGINS> file "petrov-youpi-file@2019-07-22.yang"
module youpi {
    namespace "http://ackl.io/youpi";

    prefix "youpi";

    organization
        "Acklio";
```

```
contact
  "Ivaylo Petrov
  <mailto:ivaylo@ackl.io>";

description
  "This module defines the extentions used by youpi.";

revision 2019-07-22 {
  description "Initial revision.";
}

/**
 *
 * Extension for Binary data to CBOR mapping.
 *
 */
extension position {
  argument object;
}

extension fieldIndex {
  argument object;
}

extension condition {
  argument object;
}

extension multiplier {
  argument object;
}

extension offset {
  argument object;
}

extension units-subject {
  argument object;
}

extension js {
  argument object;
}
}
```

2.2. Position

Information about which bits need to be used in order to find the value of a field.

2.2.1. Bit positions

If the position is not present or is empty, the value contains 0 bits and has a default value of 0 (or equivalent for the given type). Could be useful if a field needs to be the result of arithmetic operations from different fields.

It is possible to have a single bit read by giving only its value in the position extension.

If continuous bits need to be used to obtain the value of a given field, this can be achieved using the ".." syntax. For example "0..3" means bits 0, 1, 2 and 3.

If non-continuous bits need to be used, one can use the concatenation of bit ranges using the "|" operator. For example "0..1 | 3".

2.2.2. Cursor

Starts at 0 and changes with each read to the last bit index that was read. Used in Section 2.2.4 to determine where the read will start from. Section 2.2.3 is not affected by it, but changes its value.

2.2.3. Absolute position

The default one if no keyword is used. Alternatively "absolute" keyword can be provided to explicitly request such position.

Example:

```
leaf temp {  
    type uint8;  
    default -19;  
    description "The temperature";  
    youpi:position "0..6";  
}
```

2.2.4. Relative position

Example:

```
leaf temp {  
    type uint8;  
    default -19;  
    description "The temperature";  
    youpi:position "relative 1..7";  
}
```

This means that the value starts 1 bit after the current cursor and will read up to 7 bits after the current cursor position, including that 7th bit.

2.3. FieldIndex

Can be used to change the order in which fields are processed. By default the order in which fields appear in the document is the order in which they are processed.

2.4. Multiplier

A value or another field by which a given field needs to be multiplied before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in Section 2.5).

2.5. Offset

A value or another field to which a given field needs to be added before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in Section 2.5).

2.6. Units-subject

Meta information used to compute JSON-LD.

2.7. Data definitions

2.7.1. Supported built-in type

- o binary
- o enumeration
- o int8
- o int16
- o int32

- o int64
- o string
- o uint8
- o uint16
- o uint32
- o uint64

2.7.2. Leafs

Simple fields like integers and strings are represented by leafs in YOUPI.

2.7.3. Type min/max values

"range" attribute can be used for giving a "min"/"max" acceptable value for a type. If the value is outside of the defined range, it is silently excluded from the final result.

Example:

```
typedef temp {  
    type int8 {  
        range "-20 .. 107";  
    }  
}
```

2.7.4. Type fraction digits

It is possible to specify how many fraction digits are expected for a value to have.

Example:

```
leaf temp {  
    type decimal64 {  
        fraction-digits 2;  
    }  
}
```

2.7.5. Containers

Complex fields like objects are represented by containers in YOUPI.

2.7.6. Condition

2.7.6.1. Choice

Inside a choice statement, the condition extension gives information based on what value the choice will be decided.

For example considering that there is a value "mode" with the value of btn inside the model

```
leaf mode {  
    ...  
}  
choice data {  
    case _btn {  
        container button-data {  
            ...  
        }  
    }  
    case _temp {  
        container temperature-data {  
            ...  
        }  
    }  
    youpi:condition "../mode";  
}
```

Then the button-data container will be used to parse the data.

2.7.6.2. When

With when statement it is possible to link the presence of some piece of data to a value of another field. For example it is possible to have button-data or temperature-data depending of the value of the mode field.

```
container button-data {  
    when "../mode[.=1]"  
    ...  
}  
container temperature-data {  
    when "../mode[.=2]"  
    ...  
}
```

2.7.7. Lists

List statements are supported and they generate an array of a given composite type.

2.7.7.1. With explicit length

A list of minimum and maximum temperatures can be defined as:

```
leaf temperature-len {  
    type int32;  
}  
  
list temperatures {  
    youpi:length "../temperatures-len";  
    leaf min-value {  
        type int32;  
    }  
    leaf max-value {  
        type int32;  
    }  
}
```

2.7.7.2. Until the end of input

The list as defined in Section 2.7.7.1 can omit the length extension statement if all the remaining bytes in the payload are part of the list.

2.7.7.3. Until a specific value

The list as defined in Section 2.7.7.1 can also omit the length if it has a defined key and if it only has one leaf or container in the list apart from the key and it is a subject to when statement that defines a stop value for the key.

```
list temperatures {  
    key option-id;  
    leaf option-id {  
        type int32;  
    }  
    container value {  
        when "../option-id[!=0xffffffff]";  
        ...  
    }  
}
```

2.7.8. Enumerations as mappings

Enumerations can be used inside a typedef in order to restrict a field only to a set of acceptable values or in order to accomplish mapping between some values and other values (for example 0 stands for "temperature", 1 stands for "humidity", etc).

Example:

```
typedef mode-type {  
    type enumeration {  
        enum temp {  
            value 0;  
        }  
        enum humidity {  
            value 1;  
        }  
        enum light {  
            value 2;  
        }  
        ...  
    }  
    ...  
}
```

2.7.9. Groupings

Groupings can be used for better reuse of definitions. They don't affect the generated output.

2.7.10. Typedefs

Typedefs can be used to provide extra information about the type of a field, including semantic information about it.

3. Security Considerations

The YANG file should be valid.

Segmentation faults might result from invalid data being provided with a given YANG model.

Resource exhaustion can be looked for.

4. IANA Considerations

This document registers a YANG model.

Acknowledgements

Contributors

7. 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>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

Appendix A. Complete examples

Author's Address

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io