KuVS-Fachgespräch Fog Computing 2018

Boris Koldehofe, Andreas Reinhardt, Stefan Schulte (Eds.)

boris.koldehofe@kom.tu-darmstadt.de, reinhardt@ieee.org, s.schulte@infosys.tuwien.ac.at

Technical Report Vienna, Darmstadt, Mar. 7, 2018

https://www.kuvs.de/fg/fogcomputing/

Editors

Boris Koldehofe
Multimedia Communications Lab
TU Darmstadt
Rundeturmstr. 10
64283 Darmstadt
Germany
boris.koldehofe@kom.tu-darmstadt.de

Andreas Reinhardt
TU Clausthal
Institut für Informatik
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld
Germany
reinhardt@ieee.org

Stefan Schulte
TU Wien, Distributed Systems Group
1040 Wien, Argentinierstrasse 8/194-2
Austria
s.schulte@infosys.tuwien.ac.at

PREFACE

Fog computing is a recent computing paradigm which brings well-known principles from cloud computing to the edge of the network. This includes (i) on-demand provisioning of computational resources, (ii) rapid elasticity, (iii) unified management interfaces, and (iv) virtualization. Apart from incorporating basic principles from cloud computing, fog computing is also based on concepts from the fields of mobile cloud and in-network processing.

Through the application of fog computing principles, it is possible to provide Internet of Things (IoT)-based computational resources in a similar vein as virtual machines and software containers are offered on the cloud. The result is a virtualized computing infrastructure that spans cloud data centers, various intermediary nodes such as routers and gateways, and IoT devices. Fog computing has recently gained much interest by the research community, especially since it allows to (pre-) process or filter IoT data on-site instead of sending data to the cloud.

Since fog computing addresses several research topics which are closely linked to the "Kommunikation und Verteilte Systeme" (KuVS) community, we are happy that the KuVS board accepted our proposal for a "Fachgespräch" on this very recent research topic. In general, Fachgespräche provide a forum for (young) researchers to exchange ideas in an informal setting. The goal is to provide the opportunity to identify potential collaborations by presenting early work and research ideas.

We are happy that – despite being the inaugural edition of this Fachgespräch – we have received a number of high-quality submissions. This shows the interest by the KuVS community regarding this topic.

Nine submissions have been selected for presentation at the Fachgespräch on March 8th and 9th, 2018. Within their paper *Fog Computing: Current Research and Future Challenges*, Julien Gedeon et al. present an overview of research questions in the field, with a focus on future challenges. Olena Skarlat et al. present a concrete testbed for fog computing in their paper *FogFrame: IoT Service Deployment and Execution in the Fog*. Sevil Dräxler et al. analyze cloud services with regard to resource demands in their paper *Towards Predicting Resource Demands and Performance of Distributed Cloud Services*. Peter Danielis et al. present a potential use case for fog computing in their paper *A Distributed Protocol for Crowd Counting in Urban Environments*. Manisha Lutra et al. present an approach to apply Complex Event Processing in the fog or in the cloud in their paper *Adaptive Complex Event Processing over Fog-Cloud Infrastructure Supporting Transitions*. An approach to provide data processing using heterogeneous IoT devices is presented by Aditya Raj and Andreas Reinhardt in *CLAP: Cooperative Locality-Aware Data Processing in Heterogeneous Fog Environments*. Zoltan Mann provides insights on *Data Protection in Fog Computing through Monitoring and Adaptation*, with a special focus on adaptive approaches to data protection. In their paper *Migrating IoT Processing to Fog Gateways*, Daniel Happ et al. apply on-site gateways for sensor data processing. Stefan Geissler and Thomas Zinner present an approach to pipeline processing, which helps to overcome device heterogeneity in their paper *TableVisor 2.0: Hardware-independent Multi Table Processing*.

In addition to these talks, there were two keynotes: Ruben Mayer gave insights on *Fog Computing: On the Road to an Open Infrastructure for the Internet of Things*, and Lasse Lehmann from AGT International gave an industrial keynote.

We'd like to thank the local organization team at TU Darmstadt for their help with setting up the Fachgespräch. Also, we thank the *Profile Area Internet and Digitization* at TU Darmstadt and the *DFG Collaborative Research Center Multi-Mechanisms Adaptation for the Future Internet (MAKI)* for supporting this event. We'd like to thank our industrial sponsors AGT International and Ascora GmbH. Through their help, it was possible to organize the Fachgespräch with reasonable registration fees for all participants. Last but not least, we thank the keynote speakers Ruben Mayer and Lasse Lehmann.

Vienna, Darmstadt, Mar. 7, 2018
Boris Koldehofe, Andreas Reinhardt, Stefan Schulte

CONTENTS

Fog Computing: Current Research and Future Challenges

Julien Gedeon, Jens Heuschkel, Lin Wang, Max Mühlhäuser
Telecooperation Lab, Technische Universität Darmstadt
Email: {gedeon, heuschkel, wang, max}@tk.tu-darmstadt.de

***Abstract*—Acknowledging the shortcomings of cloud computing, recent research efforts have been devoted to fog computing. Motivated by a rapidly increasing number of devices at the extreme edge of the network that imply the need for timely and local processing, fog computing offers a promising solution to move computational capabilities closer to the data generated by those devices. In this vision paper, we summarize these current research efforts, describe applications where fog computing is beneficial and identify future challenges that remain open to bring fog computing to a breakthrough.**

I. Introduction

Cloud computing infrastructures are the predominant way to store data and perform computations today. Cloud computing offers powerful and reliable infrastructures that are scalable and accessible via flexible pay-as-you-go models. However, with the increasing number of resource-constrained mobile devices at the edge of the network (e.g., mobile phones, connected cars, internet of things (IoT) sensors) that need to offload data and computations, cloud computing creates network bottlenecks. Future applications running on these kinds of devices require ultra-fast processing of data, e.g., for augmented reality applications or real-time event detection. Today's cloud computing infrastructures are unable to fulfill these requirements. Therefore, we can observe a trend in research to move computations away from the cloud and closer to the data sources and their consumers.

Fog computing[1] [3], [4], [5] is a promising research direction in this domain. Compared to the cloud, fog computing offers proximate, small-scale resources that can be instantiated dynamically. Fog infrastructures are located between (mobile) end devices and the cloud in an intermediate layer, as depicted in Figure 1. Most often, this intermediate layer represents the access network (e.g., Wifi routers or cellular base stations) and network middleboxes. Because of this, compared to cloud computing, fog computing can provide context awareness and a better support for user mobility. It is important to note that fog computing infrastructures are heterogeneous, i.e., fog computing infrastructures can be hosted on different kinds of physical devices. Table I summarizes the differences between cloud computing and fog computing.

[1]A similar concept is edge computing [1], [2]. In this paper, we use the term fog computing in general to denote infrastructures that are close to the mobile end devices.

Table I
Comparison between Cloud Computing and Fog Computing

	Cloud Computing	Fog Computing
Proximity	low	high
Latency	high	low
Geo-distribution	locally clustered	widespread
Infrastructure	centralized datacenters	decentralized cloudlets
Heterogeneity	low	high
Deployment	fixed, static	dynamic, opportunistic
Virtualization	heavyweight (e.g., VMs)	lightweight (e.g., containers)
Connections	long-thin	short-fat
Access	through core network	typically via 1-hop wireless
Mobility support	limited	yes
Context Awareness	no	yes

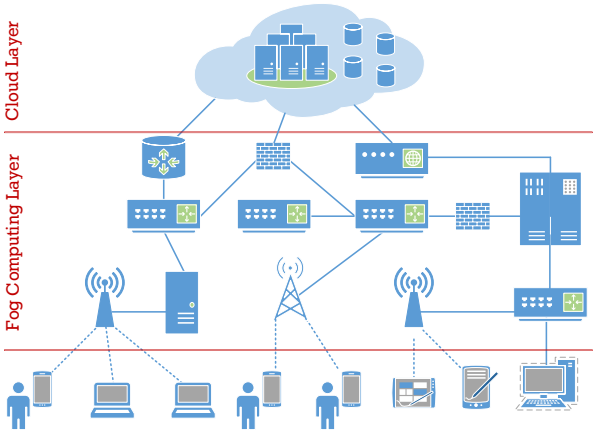


Figure 1. Fog Computing Architecture

In the remainder of this paper, we outline current research topics in fog computing, its possible applications and future challenges.

II. Survey of Current Fog Computing Research

In this section, we review current research efforts in two different domains related to fog computing.

A. Offloading of Data and Computations

Mobile device are inherently constrained in terms of computing power and battery lifetime and, thus, there is the need to offload complex computation to more powerful infrastructures. With potentially many fog locations available, the question arises where to place resources and how to allocate them. Making good placement decisions most often requires detailed

knowledge wrt. the performance of the offloading system. Meurisch et al. [6] have investigated the probing of unknown services, i.e., how to support the decision on where to offload without detailed prior knowledge of the target system.

While most of the work focuses on the offloading of computations, data offloading is another issue worth considering. Instead of storing data in distant cloud centers, we envision storing data close to where it is used. This is especially relevant if multiple users or applications reuse or share the same data. Of course, this means the storage decision needs to be made in consideration of the current context in which the data is captured. Gedeon et al. present a framework for Android devices that enables context-aware micro-storage of data [7].

B. Fog Computing Infrastructures

Fog computing can be realized on different—mostly already existing—physical infrastructures. One possibility is to collocate computational capacities on the radio access network (RAN), e.g., cellular base stations. Besides implementing fog computing on the RAN, some research also studied the use of privately owned Wifi routers as fog computing devices, either to perform computations [8] or as a mechanism to facilitate service discovery [9]. This is motivated by the fact that these devices are ubiquitously present and often underutilized. Several initiatives already promote free Wifi access (e.g., *freifunk*[2] in Germany). We believe that an open computing ecosystem is the next logical step. Of course, this ecosystem requires new programming models. As an example, Hong et al. [10] have suggested *MobileFog*, a lightweight programming model targeting IoT applications in fog computing.

C. Holistic Resource Management

Achieving efficient operation of fog computing systems is critical, as fog resources are not as abundant as in mega data centers. Ideally, fog nodes should be able to offer compute or storage resources to any user in close proximity through an open and standardized mechanism, which allows a set of fog nodes in the same geographic region to form a shared resource pool. With the help of lightweight virtualization technologies, resources will be allocated holistically at a fine granularity (per user) subject to quality of service and system-wide optimization goals.

Compared to resource management in cloud data centers, resource management in fog computing is more challenging due to the fact that fog nodes are more heterogeneous and uncertainties are imposed by multiple factors such as user mobility. While it is still not available yet, a general centralized framework for holistic fog resource management is envisioned. Based on this assumption, a handful of works have been carried out for fog resource allocation and job scheduling [11], [12], [13], [14], [15], [16], [17]. Jia et al. [12] study the load balancing among multiple fog clouds. Tong et al. [13] discuss workload placement for delay minimization in a hierarchical fog computing architecture. Wang et al. [14] focus on stochastic frameworks for optimizing dynamic workload migration based on Markov Decision Processes (MDPs). Recently, Tan et al. [15] studied online job dispatching and scheduling in fog clouds. Wang et al. investigate online mobility-oblivious resource allocation for fog computing [16] and also develop a service entity placement strategy for social virtual reality applications in the fog environment [17].

[2]https://freifunk.net/

III. Applications for Fog Computing

In this section, we turn our attention to different use cases where we consider fog computing to be beneficial.

A. Internet of Things

The IoT [18] is predicted to grow to billions devices in the upcoming years. According to a recent study by BGC, the predicted market size for the IoT is to reach 267 billion dollars by the year 2020.[3] Comprised of small-scale sensors and actuators, data produced by IoT devices is often consumed only locally. Each of these devices will be delivering massive amounts of data to be used in real-time analytics, event detection or complex event processing. If we consider high-volume data like video, it is obvious that this does not scale. Fog computing however provides the possibility to scale the IoT to a huge number of devices by offering proximate processing of IoT data [4], [19].

B. Smart Cities

An especially useful and palpable usage scenario for fog computing can be found in the vision of *Smart Cities* [20], where urban areas are augmented to provide services to their citizens. This requires to process a multitude of sensor data and distribute it to different actuators. An example is smart traffic management. In this vision, traffic lights would not be programmed statically but adapt their cycle based on different types of data as input. Among others, the data may be provided by inductive loops, video cameras mounted above busy intersections and third-party applications that notify about events, e.g., accidents that have occurred. A comprehensive survey on the implications of fog computing for smart cities can be found in [21]. With the future development of connected cars and autonomous vehicles, quick processing of data to recognize ambient events becomes even more important.

Fog Computing is also interesting for the scenario of emergency response and in crises situations, where other communication infrastructures have become unavailable. In such a scenario, opportunistic infrastructures like smart lamp posts or locally deployed cloudlets can be used to provide disaster relief services [22], [23].

C. Augmented Reality & Virtual Reality

Recently, augmented reality (AR) and virtual reality (VR) applications have gained attention both in research and consumer products. These new classes of applications require ultra fast processing of data, i.e., mostly real-time analysis of video streams. Because even small delays have a considerable

[3]https://www.forbes.com/sites/louiscolumbus/2017/01/29/internet-of-things-market-to-reach-267b-by-2020/

impact on the perceived quality of service, cloud offloading cannot be used. As an example, Ha et al. [24] develop a cognitive assistance application using Google Glasses that allows real time scene interpretation by offloading the computations to VM-based cloudlets. It is worth mentioning that AR and VR applications have other specific challenges related to previously mentioned research, such as the placement of services [17].

IV. Fog-supporting technologies

In this section, we outline how the emerging technologies of lightweight virtualization via cloudlets and SDN/NFV can support fog computing.

A. Cloudlets and Lightweight Virtualization

One prominent concept that has been proposed to contrast cloud computing are *cloudlets* [25], which are micro clouds located at the edge of the network. Cloudlets therefore can run on a variety of devices, including the ones with constrained resources such as network routers. Fog computing requires new lightweight virtualization techniques in order to provide quick provisioning and migration of services on heterogeneous resources. The latter is motivated by the high mobility of users at the edge of the network. In this domain, a lot of research has investigated the use of container-based virtualization, such as *Docker*[4], focusing on migration [26] or adapting Docker for the provisioning of resources at the edge of the network [27]. Recently, library operating systems such as unikernels[5] have received quite some attention as an alternative to the VM-based virtualization technology. Unikernels can be very help in edge computing due to the fact that they are lightweight and are much more secure than containers.

B. SDN and NVF

Software Defined Networking (SDN) splits up the data plane and control plane of networks and is a technology that has recently received a lot of attention. In the context of fog computing, SDN can be leveraged to facilitate the management and organization of networks. Instead of configuring every device individually, a set of rules can be managed and installed by a centralized controller software, leveraging a (potentially) global view on the network. Hence, it is possible to plan and optimize the network traffic even in big and complex systems like the core network of Internet Service Providers (ISPs) [28] by executing these rules on the forwarding devices.

Along with SDN, network function virtualization (NFV) plays a big role in upcoming ISP networks. Virtualization of network functions gives the well-known cloud advantages, like dynamic scaling and better cost efficiency, since special hardware is replaced through cheap commercial off-the-shelf (COTS) servers where virtual network functions can be orchestrated within seconds. However, the usage of COTS servers comes at a price since they arise as bottleneck for 1) network-intensive tasks (simple operations on a huge amount of packets) and 2) compute-intensive tasks (complex functions on a set of data) [29]. To solve this problem, researchers can leverage domain-specific hardware, such as field-programmable gate arrays (FPGAs), to accelerate network functions. Since FPGAs are a very limited resource, they have to be organized in a flexible way. Therefore, Noback et al. proposed a dynamic scheduling scheme for leveraging FGPAs to speed up crucial network functions in an optimized manner [30].

Applications typically choose TCP as the transport protocol, resulting in a big potential for SDN to optimize the network protocol stack by bringing application requirements, transport protocol and link layer in harmony. Heuschkel et al. [31] expanded the SDN paradigm to end devices in order to enable a dynamic control for network protocols. This network protocol virtualization (NPV) approach decouples the applications from specific network protocols, delegating the choice of network protocols to a management instance and enabling the cross-layer optimization of application requirements with the given network environment and available transport layer protocols. To centralize the approach and to give a global view for an end-to-end optimization, Heuschkel et al. [32] proposed an OpenFlow-inspired protocol to communicate management commands, rules and network monitoring information to the end devices. Along with the SDN integration, the NPV approach enables small network functions on end devices, placed as additional layer in the network protocol stack. With these pieces in place, NPV adapts the features of SDN and NFV for end devices, and thus, uses the network stack in a dynamic and optimizable way.

V. Future Challenges

Despite the recent efforts in research outlined before, many challenges still remain open. In particular, we identify the following challenges for future research:

Migration of Data and Applications. Users and devices in most fog computing scenarios are highly mobile. Hence, the services required should also follow these dynamics. This requires the migration of application and data instances across different fog instances. While today this is done mostly reactively, we envision doing this proactively based on the predicted mobility and access patterns of users and data.

Orchestration and Seamless Interplay Between Fog and Cloud. While fog computing undoubtedly offers several benefits and meaningful use cases, we still need cloud infrastructures to persistently store and batch process big data. Data gathered at the edge of the network might be both interesting for the immediate processing as close as possible, but also for cloud applications. Orchestration between cloud and fog services therefore is necessary.

Business Models. The term fog computing was initially coined by Cisco in order to promote their IOx platform. From this history we can see the importance of fog computing as a business opportunity for manufacturers of networking hardware, who can rent out parts of their devices' capabilities

[4]https://www.docker.com/

[5]http://unikernel.org

for general-purpose computing. However, this requires new business and pricing models that capture the cooperative nature of fog computing, as migration of data and services needs to be feasible across the domains of different stakeholders. This becomes even more important when users are competing for resources.

Security and Privacy. If we envision executing container-based applications from different application providers on existing infrastructures, this has obvious security implications, such as third party applications maliciously interfering with the infrastructure. Therefore, strong sandboxing/isolation mechanisms are required if we expect fog computing to be an open ecosystem. Several other security challenges in fog computing have been outlined by Stojmenovic et al. [33]. Because fog computing processes the data close to where it originates, it offers the possibility to employ privacy-preserving mechanisms early in the processing chain. Imagine for instance a video camera stream that is fed as a raw data source to an application that detects the presence of objects on a city street. For this particular application, the faces of pedestrians and license plates of cars are not required and are a threat to one's privacy. Fog computing would allow the blurring of these elements before forwarding the data stream to applications. Application models that can give these kinds of privacy are not yet present in today's fog landscape.

VI. Conclusion

In this paper, we have outlined the current research efforts towards fog computing. We further described applications where fog computing can complement existing cloud infrastructures and identified future challenges that need to be addressed for the widespread adoption of fog computing.

Acknowledgement

This work has been funded by the German Research Foundation (DFG) as part of the Collaborative Research Center (CRC) 1053 - MAKI.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *Proc. INFOCOM Workshop*, 2014, pp. 346–351.

[3] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," *Proc. MCC Workshop*, pp. 13–16, 2012.

[5] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog Computing: Platform and Applications," in *Proc. HotWeb*, 2015, pp. 73–78.

[6] C. Meurisch, J. Gedeon, T. A. B. Nguyen, F. Kaup, and M. Mühlhäuser, "Decision support for computational offloading by probing unknown services," in *Proc. ICCCN*, July 2017, pp. 1–9.

[7] J. Gedeon, N. Himmelmann, P. Felka, F. Herrlich, M. Stein, and M. Mühlhäuser, "vStore: A Context-Aware Framework for Mobile Micro-Storage at the Edge," in *Proc. MobiCASE*, 2018, pp. 1–18.

[8] C. Meurisch, A. Seeliger, B. Schmidt, I. Schweizer, F. Kaup, and M. Mühlhäuser, "Upgrading wireless home routers for enabling large-scale deployment of cloudlets," in *Proc. MobiCASE*, 2015, pp. 12–29.

[9] J. Gedeon, C. Meurisch, D. Bhat, M. Stein, L. Wang, and M. Mühlhäuser, "Router-based brokering for surrogate discovery in edge computing," in *Proc. ICDCS Workshops*, 2017, pp. 145–150.

[10] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proc. MCC Workshop*, 2013, pp. 15–20.

[11] L. Wang, L. Jiao, D. Kliazovich, and P. Bouvry, "Reconciling task assignment and scheduling in mobile edge clouds," in *Proc. ICNP*, 2016, pp. 1–6.

[12] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *Proc. INFOCOM*, 2016.

[13] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. INFOCOM*, 2016.

[14] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. S. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Networking*, 2015.

[15] H. Tan, Z. Han, X.-A. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. INFOCOM*, 2017.

[16] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Proc. ICDCS*, June 2017, pp. 1281–1290.

[17] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. INFOCOM*, 2018.

[18] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, 2014.

[19] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug 2016.

[20] J. M. Schleicher, M. Vögler, S. Dustdar, and C. Inzinger, "Enabling a smart city application ecosystem: Requirements and architectural aspects," *IEEE Internet Computing*, vol. 20, no. 2, pp. 58–65, 2016.

[21] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 32:1–32:43, Jun. 2017.

[22] C. Meurisch, T. A. B. Nguyen, J. Gedeon, F. Konhauser, M. Schmittner, S. Niemczyk, S. Wullkotte, and M. Mühlhäuser, "Upgrading wireless home routers as emergency cloudlet and secure DTN communication bridge," in *Proc. ICCCN*, 2017, pp. 1–2.

[23] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, Oct 2013.

[24] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. MobiSys*, 2014, pp. 68–81.

[25] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[26] L. Ma, S.Yi, and Q. Li, "Efficient Service Handoff Across Edge Servers via Docker Container Migration," in *Proc. SEC*, 2017.

[27] P. Liu, D. Willis, and S. Banerjee, "Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge," in *Proc. SEC*, 2016, pp. 1–13.

[28] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking : A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14 – 76, 2015.

[29] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform handling and abstraction of nfv hardware accelerators," *IEEE Network*, vol. 29, no. 3, pp. 22–29, 2015.

[30] L. Nobach, B. Rudolph, and D. Hausheer, "Benefits of conditional FPGA provisioning for virtualized network functions," in *Proc. Netsys*, 2017, pp. 1–6.

[31] J. Heuschkel, I. Schweizer, and M. Mühlhäuser, "Virtualstack: A framework for protocol stack virtualization at the edge," in *Proc. LCN*, 2015, pp. 386–389.

[32] J. Heuschkel, M. Stein, L. Wang, and M. Mühlhäuser, "Beyond the core: Enabling software-defined control at the network edge," in *Proc. Netsys*, 2017, pp. 1–6.

[33] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of fog computing and its security issues," *Concurr. Comput.: Pract. Exper.*, vol. 28, no. 10, pp. 2991–3005, 2016.

FogFrame: IoT Service Deployment and Execution in the Fog

Olena Skarlat*, Kevin Bachmann‡, and Stefan Schulte*
Distributed Systems Group, TU Wien, Austria
*Email: {o.skarlat, s.schulte}@infosys.tuwien.ac.at
‡ kevin.bachmann@gmx.at

***Abstract*—Despite existing theoretical foundations, the adoption of fog computing is still at its very beginning. A particular research challenge is the combination of decentralized data processing needed for Internet of Things (IoT) services with the benefits of fog computing. In this paper, we consider fog computing as an umbrella paradigm that comprises three levels of resources in the network: IoT devices, edge and cloud resources. These resources have become a foundation for a fog landscape. In this work, we address questions of how to virtualize resources in the fog landscape, how to control the fog landscape, and how to deploy and execute services in the fog landscape. To address these questions, we present the architecture and implementation details of a fog computing framework, called *FogFrame*.**

I. Introduction

The support of decentralized processing of data on Internet of Things (IoT) devices in combination with the benefits of cloud technologies and virtualization has been identified as a promising approach to reduce communication overheads and data transfer times in the IoT [1]. For this, it is necessary to move parts of the computational and storage resources, needed to execute IoT services, closer to the edge of the network [2]. Hence, a coordinated control over the IoT landscape and virtualization of IoT resources has to be established. We follow the fog computing definition of the OpenFog consortium [3] where fog computing operates in *cloud-to-thing-continuum*. To establish such a coordinated control over the cloud-to-thing continuum, it is necessary to develop distributed execution environment which aims to integrate available edge and cloud resources into a fog landscape, provide necessary resource virtualization mechanisms, manage resources of the volatile fog landscape, and optimize their consumption. Therefore, in this work, we identify and answer the following research questions: (i) What are the mechanisms to provide virtualization of IoT resources? (ii) What are the methodologies and tools to realize the software environment that manages the fog landscape and executes IoT services? (iii) How to optimize fog resource provisioning?

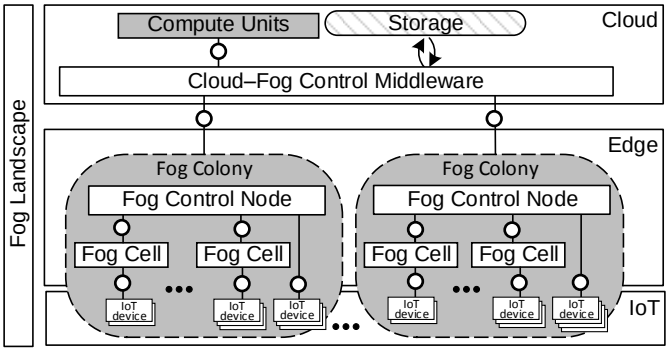
We answer these questions by means of a fog computing framework called *FogFrame* [4]. This framework aims to become a testbed for researchers giving the opportunity to implement and evaluate various policies and mechanisms in the real-world fog landscape based on Raspberry Pi units. The framework provides mechanisms to execute services and control the fog landscape, to place, deploy and migrate services, and to reconfigure the fog landscape infrastructure, optimize resource provisioning. With regard to resource provisioning, in FogFrame several approaches have been designed and implemented to (i) analyze resource utilization within the fog landscape, (ii) create a service placement plan to allocate resources for services, and (iii) perform infrastructural changes and monitoring in the fog landscape.

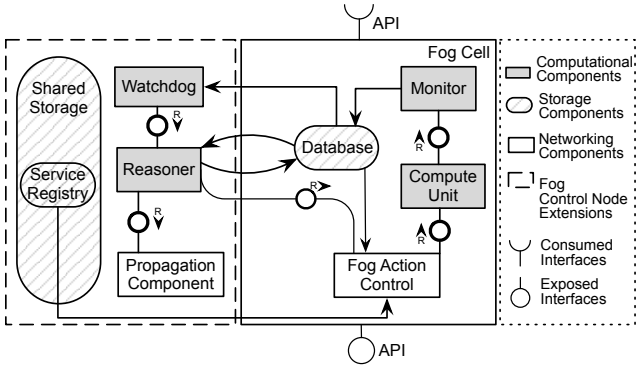
The remainder of this paper is organized as follows: First, we explain a resource model of a fog landscape in Section II and FogFrame system architecture in Section III. Next, in Section IV, we discuss the service deployment issues related to the heterogeneous nature of the fog landscape. We shortly elaborate on the state-of-the-art work in the area of fog computing frameworks in Section V. Finally, in Section VI we conclude the paper and highlight our future work.

II. Architecture of a fog landscape

To discuss a resource model of a fog landscape, we use the notion of *fog colonies* (Fig. 1). At the bottom of a fog colony, there are 'thin' IoT devices, e.g., sensors and actuators. These IoT devices are connected to 'fat' edge devices, which have computational power and execute services in the fog colony. We call such 'fat' edge devices *fog cells*. In each colony there is exactly one head fog cell called *fog control node*, which controls and orchestrates fog cells in its colony and performs dynamic service placement. Such a hierarchical construct of edge devices forms a fog colony. Fog colonies are interconnected between each other via their corresponding fog control nodes. The communication layer between a fog landscape and the cloud is provided by a *cloud-fog control middleware*. Every fog colony connects to the cloud-fog control middleware via its fog control node.

Based on this concept of fog colonies, we are able to orchestrate fog cells and to provide optimal resource provisioning and service placement approaches, i.e., a solution on how to place services on virtualized resources in a fog landscape. For this, we formalized an optimization problem that maximizes the utilization of existing resources in the fog and adheres to the Quality of Service (QoS) parameters of services. To solve the proposed optimization problem, we apply different approaches, namely the exact optimization method and its approximation through a greedy first-fit heuristic and a genetic algorithm. Also, we compared the results to a classical approach that neglects fog resources and runs all services in a centralized cloud.





At first, we simulated such a fog landscape by the means of modeling frameworks CloudSim [5] and iFogSim [6] and evaluated the model with the capabilities of those modeling frameworks [7], [8], [9] with regard to deployment times of applications, adherence to QoS parameters, utilization of resources, and cost of execution. Afterwards, the architecture and functionalities of a fog landscape have been implemented in a fog computing framework, called FogFrame [4].

III. Fog Computing Framework

In this section, we discuss the architecture and implementation details of FogFrame. Service deployment and execution have to be performed both in fog colonies and cloud resources. We support a hierarchy of fog colonies by the means of the cloud-fog control middleware. As was mentioned before, the further layers of the hierarchy are fog control nodes, fog cells, and finally IoT devices at the very bottom of the hierarchy (see Fig. 1).

The *cloud-fog control middleware* is responsible for the execution of applications in the cloud and support of all connected fog colonies. Such support is performed continuously or on-demand, depending on system events, e.g., if new fog cells appear, or in order to recover after faults of fog cells. It has to be noted that the cloud-fog control middleware can overrule fog control nodes in fog colonies, but the latter may also act autonomously in case no middleware is available. The cloud-fog control middleware can either be instantiated in a cloud VM, or any device at the edge of the network, which has an established connection with fog colonies and the cloud provider. The cloud-fog control middleware has to be supplemented with valid authentication credentials, e.g., in the case of using AWS services, the AWS credentials has to be provided.

Fog cells are software components which serve as access points allowing to control and monitor the underlying IoT devices, i.e., sensors and actuators. The architecture of a fog cell is presented on the right hand side of Fig. 2. The *database* stores data about received requests, current system utilization and monitoring data. We use a Redis database, which has to be instantiated in a Docker container on the same host as the fog cell. The *fog action control* performs actions according to the service placement plan produced by the fog control node, e.g., to deploy and start a particular service. The *compute unit* provides the actual computational resources for the deployment and execution of services. The *monitor* observes service executions. Fog cells expose REST APIs for data transfer and control actions, i.e., instantiating, starting, stopping, and deleting services.

To enable a fog cell in a fog colony, the fog cell has to be supplemented with a property file containing: (i) the IP address and port of the cloud-fog control middleware to be able to request pairing with other devices in the fog landscape; (ii) the IP address and port of a fallback control node as a fallback mechanism in the case when the cloud-fog control middleware is unavailable; (iii) its own IP address to enable communication in the network; (iv) coordinates of the fog cell in a fog landscape to enable pairing with the closest fog control node; and (iv) service types which the fog cell can execute. In future work, obtaining and changing these properties can be automated.

Fog control nodes are fog cells with extended functionality which support and control their own fog colonies. In contrast to fog cells, fog control nodes receive requests for execution of IoT applications from users. The fog control node is able to propagate requests for service placement to the cloud-fog control middleware or to other fog colonies. For this, fog control nodes use reasoning mechanisms for service placement and resource allocation. On the left-hand side of Fig. 2, the extensions needed for fog control nodes are shown. The *reasoner* implements the service placement and resource provisioning policies. In the reasoner, several different approaches are realized, i.e., a greedy first-fit heuristic algorithm, an exact optimization algorithm, and a genetic algorithm. These algorithms are based on the optimization problem which we described in our former works in details [7], [8], [9]. If the considered fog colony does not provide enough resources for service execution, requests are propagated to other fog colonies by the *propagation component*. The *watchdog* monitors the utilization of the fog colony. This data is also part of an input for the reasoner to perform calculations. The *service registry* stores service implementations and enables the fog action control to search for services and to deploy them on

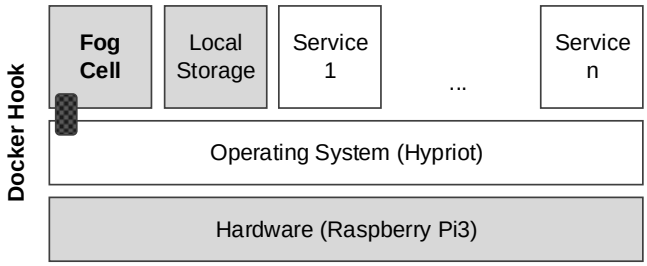


Fig. 3. Fog cell deployment

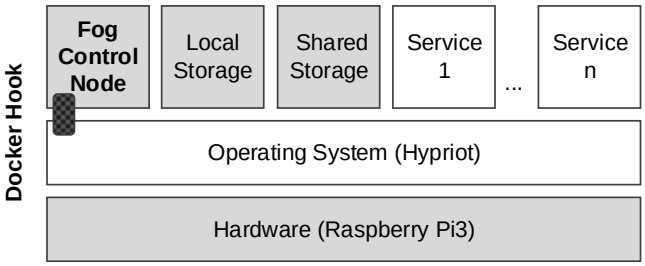


Fig. 4. Fog control node deployment

suitable fog cells. The service registry is located in the storage unit of control nodes, since storing service implementations is resource-consuming. For the service registry, we use the shared Redis data storage, running in an own Docker container on the host device of a fog control node. The *database* additionally stores service placement plans produced by the reasoner. Similarly to a fog cell, we use again a Redis database, which is instantiated in a Docker container on the same host as the fog control node. The *fog action control* performs the service placement according to the plan produced by the reasoner. Apart from properties of a fog cell, properties of a fog control node include a location range, which is reported to the cloud-fog middleware to enable calculation of the closest parent fog control node for fog cells entering the fog landscape.

IV. Service deployment in the fog

During the implementation of FogFrame, we tackled a problem of how to deploy services in the heterogeneous environment of a fog landscape, i.e., in fog colonies and in the cloud. In this section, we present our findings.

Deployment in fog colonies. Services are executed either in fog colonies, or in the cloud. Because fog colonies and cloud resources are different in processor and system architecture, service deployment in fog colonies differs from service deployment in the cloud. In our work, to create fog colonies we use Raspberry Pi units with the ARM processor architecture and a Hypriot operating system. In order to deploy services, Docker containers are used. Therefore, the base Docker images of services to be executed in fog colonies have to be compatible with the ARM processor architecture. These images have to be stored in a repository which is shared in a fog colony between fog cells and a fog control node of the fog colony, or between several fog colonies. The fog cell and fog control node are applications running inside own Docker containers in the Docker runtime of a Raspberry Pi which is provided by the host operating system of the fog device. To make it possible for fog cells and fog control nodes to deploy, start, and stop further Docker containers on the host device, we make use of a the *Docker hook* (see Fig. 3 and 4) [4]. This Docker hook resolves a problem of instantiating other Docker containers on the Docker runtime of the host device from inside the Docker containers of the running fog cells and fog control nodes.

Deployment in the cloud. Service deployment in the cloud is different from the approach presented above. Services are also deployed in Docker containers, but on VMs which need to be deployed and managed. VMs in the cloud have a different processor architecture compared to Raspberry Pi units. Each cloud VM has a CoreOS[1] operating system with a Docker runtime preconfigured. Docker images have to be based on different base images compared to those images running in fog colonies, and have to be stored in a centralized repository, e.g., Docker Hub[2]. When a service request is propagated to the cloud-fog control middleware from a fog colony and there is no deployed VM in the cloud, the cloud service leases and starts a new VM. If a VM is already running, the Docker container which corresponds to the necessary service of the service request is deployed on that VM. The containers are deployed on a cloud VM until a certain limit of containers is reached to ensure the stability of the environment. If there is no free space for another container, a new VM is leased. After execution, when some containers are stopped, and the VM is running with no load, the VM is stopped and the cloud resources are released.

V. Related Work

Being a relatively new topic, a lot of research work in the field of fog computing is based on simulation environments. Therefore, in this section, we cover the existing real-world prototypes and testbeds for service deployment and execution in fog computing.

An IoT ecosystem for elastic stream processing called VISP is proposed in the work of Hochreiner et al. [10]. It allows users to create complex network topologies. VISP analyzes these network topologies and performs constraint-based service placement. Compared to our work, this framework is limited to using cloud resourced. Another framework called DIANE is proposed in the work of Vögler et al. [11]. The framework deploys topologies for IoT applications dynamically and monitors the deployment infrastructure. While in DIANE, only rule-based algorithm is used for resource provisioning, in our work we formulate a concrete optimization problem.

[1]https://coreos.com/
[2]https://hub.docker.com/

de Brito et al. [12] proposes a mechanism to orchestrate microservices in the fog. The architecture of the prototype consists of a fog orchestration agent and for orchestrator. The fog orchestration agent resembles our notion of fog cells, and the fog orchestrator has the same purpose as a fog control node. The prototype is implemented by the means of Docker Swarm and OpenMTC M2M Framework in the Fraunhofer FOKUS IoT testbed. Compared to their work, we consider concrete communication mechanisms in a fog landscape.

In the work of Brogi et al. [13], the prototype FogTorch is presented which is a tool for resource provisioning. The specification of an infrastructure, definition of an application to be deployed, binding of fog devices and deployment policy are provided as an input to FogTorch. After applying various algorithms, FogTorch outputs eligible deployment plans. In our work, we consider multiple fog colonies, and formalize the optimization problem to place services between those colonies.

Vandeputte et al. [14] introduce a ranking-based 'service probe' to assess effectiveness of services. Those probes are accounted for in the constraints of optimization problems. This approach is evaluated by the means of a FUSION framework which is a testbed that provides flexible testing of approaches on the computational nodes of various hardware generations. In our work, we have different reasoning mechanisms, and evaluate them based on a reals-world testbed.

In our previous works [7], [8], we have modeled and simulated a fog landscape by the means of CloudSim and iFogSim. As presented in this paper, we have implemented out model and the functonalities of a fog landscape in the fog computing framework FogFrame. The framework is applied as a foundation for our work on resource provisioning and service placement mechanisms, infrastructure replanning policies, and fault tolerance mechanisms.

VI. Conclusion

In this paper, we considered open issues in fog computing dealing with virtualization of fog resources, a flexible and dynamic software environment to execute services, and service placement and execution in a fog landscape. These issues are addressed by the fog computing framework FogFrame. The framework is built upon light-weight technologies and loosely-coupled components to establish a stable and fault-tolerant distributed system.

We have identified that service execution in a fog landscape in practice requires compatibility with according computational environments, i.e., edge and cloud resources. To enable service execution in both these environments, different container images have to be used, as well as two different service repositories have to be established, i.e., shared service repository for fog colonies, and a separate repository containing images of services executable in the cloud. These aspects are also addressed in FogFrame.

In the future, we will continue to work with the fog computing framework. The architecture can be enhanced by fault tolerance mechanisms to account for mobility in the fog landscape. Another aspect of our future work is the systematic observation of the fog landscape to obtain real-world network data to evaluate the behavior of resource provisioning approaches.

Acknowledgment

This paper is supported by TU Wien research funds. This work is partially supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066).

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, ser. Studies in Computational Intelligence. Springer, March 2014, vol. 546, pp. 169–186.

[2] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications," in *Internet of Things: Principles and Paradigms*. Morgan Kaufmann, 2016, ch. 4, pp. 61–75.

[3] "OpenFog Reference Architecture for Fog Computing," https://www.openfogconsortium.org/ra/, Feb. 2017.

[4] K. Bachmann, "Design and Implementation of a Fog Computing Framework," Master's thesis, Vienna University of Technology (TU Wien), Vienna, Austria, 2017.

[5] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, pp. 23–50, 2011.

[6] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing Environments," *Softw. Pract. Exp.*, vol. 47, pp. 1275–1296, 2017.

[7] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource Provisioning for IoT Services in the Fog," in *9th IEEE Int. Conf. on Service Oriented Computing and Applications (SOCA 2016)*. Hong Kong, China: IEEE, 2016, pp. 32–39.

[8] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware Fog Service Placement," in *1st IEEE Int. Conf. on Fog and Edge Computing (ICFEC 2017)*. Madrid, Spain: IEEE, 2017, pp. 89–96.

[9] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *SOCA J.*, pp. 1–17, 2017.

[10] C. Hochreiner, M. Vögler, P. Waibel, and S. Dustdar, "VISP: An Ecosystem for Elastic Data Stream Processing for the Internet of Things," in *2016 IEEE 20th Int. Enterprise Distributed Object Computing Conf. (EDOC)*. Vienna, Austia: IEEE, 2016, pp. 1–11.

[11] M. Vögler, J. Schleicher, C. Inzinger, and S. Dustdar, "Optimizing Elastic IoT Application Deployments," *Trans. Serv. Comput.*, vol. PP, no. 99, pp. 1–14, 2016.

[12] M. S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keilsa, and F. Schreiner, "A Service Orchestration Architecture for Fog-enabled Infrastructures," in *2nd Int. Conf. on Fog and Mobile Edge Computing (FMEC'17)*. Valencia, Spain: IEEE, 2017, pp. 127–132.

[13] A. Brogi and S. Forti, "QoS-aware Deployment of IoT Applications Through the Fog," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–8, May 2017.

[14] F. Vandeputte, L. Vermoesen, D. Griffin, T. K. Phan, M. Rio, P. Simoens, P. Smet, D. Bursztynowski, F. Schamel, and M. Franke, "Evaluator Services for Optimised Service Placement in Distributed Heterogeneous Cloud Infrastructures," in *24th European Conf. on Networks and Communications (EuCNC)*. Paris, France: IEEE, June 2015, pp. 439–444.

Towards Predicting Resource Demands and Performance of Distributed Cloud Services

Sevil Dräxler, Manuel Peuster, Marvin Illian, Holger Karl
Paderborn University, Paderborn, Germany
{sevil.draexler, manuel.peuster, millian, holger.karl}@uni-paderborn.de

***Abstract*—Understanding the behavior of distributed cloud service components in different load situations is important for efficient and automatic management and orchestration of these services. For this purpose and for practical research in distributed cloud computing in general, there is need for benchmarks and experimental data. In this paper, we describe our experiments for characterizing the relationship between resource demands of application components and the expected performance of applications. We present initial results for predicting the interdependence between resource demands and performance characteristics using support vector regression and polynomial regression models. The data gathered from our experiments is publicly available.**

I. Introduction

Distributed cloud services like Internet video streaming consist of several components like encoders, caches or content servers. These services run on top of an infrastructure, typically managed by an orchestration system that places, deploys, and scales the service and its components. For example, to ensure that the users are served with an acceptable latency, the orchestration system must instantiate the right number of instances for service components in the right locations in the underlying network. The right amount of resources needs to be calculated and allocated to each instance so that the fluctuating amount of load can be handled without violating any service-level agreements or exceeding the capacity constraints of the underlying network. To do so, the orchestration system needs knowledge about the distributed cloud service, typically provided by a service's *descriptors*.

Existing descriptors for service components rely on the knowledge of the component developer to provide the exact and specific amount of resources required to handle a given load. Service components can then be deployed with the requested amount of resources and scaled out/in, e.g., upon reaching pre-defined thresholds. This approach can result in over-/under-estimating the required resources and lead to sub-optimal states for both the service and the underlying network.

In previous work [1], we have proposed the more flexible *service template embedding* approach for optimizing the scaling and placement of services in a single decision step. In this approach, a service template describes the required components of a distributed cloud service and the desired connection patterns among them. Additionally, the template specifies the resource demands of each service component as a function of the load it needs to handle. The load can be characterized, for example, in terms of the data rate on each incoming connection point of the service component. Using such a service template and based on the current load, the components are scaled out and embedded into the network.

To use the flexibility offered by the service template embedding approach, it is important to specify the relationship between the resource demands of service components (e.g., CPU, memory) and the values of performance metrics of interest for each service (e.g., frame rate, video resolution). This is cumbersome to do for a developer. By integrating an automatic profiling step [2], [3] in the management and orchestration systems, these relationships can be identified and used for optimizing the placement and scaling of the services, as well as many other applications, without the service developers having to understand and estimate them.

Such an automatic profiling system could be validated by testing it against well-known services with different performance characteristics. However, there is a pervasive lack of publicly available benchmarks and experimental data for components of distributed cloud services. To overcome this lack, we have set up a testbed to characterize such relationships in a video streaming scenario, which is a common application deployed on distributed networks. We present some results from analyzing the data from these experiments. We will use this data to demonstrate the feasibility of our service template embedding approach; but it should also be useful for other research activities in this field. The data we have gathered from our experiments is publicly available [4].

After an overview of existing work for profiling service components (Section II), we describe our experimental setup (Section III) and introduce our prediction models based on polynomial regression and support vector regression (Section IV).

II. Related Work

In this section, we give an overview of related approaches to the two aspects of our work, namely, predicting resource demands for achieving a certain performance and estimating the achievable performance under a given resource configuration.

Gmach et al. [5] determine the minimum resource needs to run a certain workload, based on the distribution of historic traces in a data center setup. Rasoolzadeh et al. [6] focus on optimizing energy consumption by estimating and adjusting the CPU cycles required for video encoding. Fan and Wang [7] compare responsiveness of web servers under a heavy request load, in a bare-metal environment. In contrast

TABLE I
VIDEOS USED AS INPUT FOR THE ENCODING PROCESS

Name	Frames/s	Mb/s	Characteristics
bunny [11]	60	3.826	vibrant colors, balanced pace
docu 1 [12]	25	14.333	faded colors, grainy, fixed camera
docu 2 [13]	29.97	3.453	balanced colors, balanced pace
game [14]	60	5.4533	vibrant colors, fast
noise [15]	25	13.173	monochrome, no recognizable pattern

to these attempts, our work focuses on characterizing resource requirements to fulfill a certain target performance.

Xu et al. [8] model the relationship between throughput and response time of a web server, for request rates below the peak load. In their experiments, they vary the memory allocation and the number of CPU time slices before potentially being preempted. Do et al. [9] focus on predicting the performance of applications on virtual machines (VMs). Given a target performance, their models determine whether or not it can be achieved on a certain system. Giannakopoulos et al. [10] approximate service performance, given a certain hardware configuration. They use neural networks and linear regression and approximate the performance using a Gaussian distribution. We also approximate and verify functions that take available resources as input and yield possible values for certain performance metrics. Additionally, we use SVR as a machine learning technique and polynomial regression.

In a previous related work [2], we have analyzed the practical requirements of a profiling system to generate performance behavior information, which can be used to support resource allocation decisions for virtual network functions in service chains. This is a work in progress and we are working on integrating the profiling information into service orchestration, to optimize flexible placement and scaling decisions.

III. METHODOLOGY

We have done test runs using a video encoder function and a cache function. Because of space limitations, we only show a subset of results from analyzing the video encoder.

We have performed our experiments on top of an OpenStack testbed. The OpenStack controller and the video encoder function were deployed on two different physical machines to avoid resource allocation conflicts among different functionalities. We have deployed the video encoding function as a virtual machine running *ffmpeg* for the actual encoding and *ffserver* for distributing the video stream. We have collected CPU and memory utilization data once per second using the Unix tool *ps*. It is a lightweight tool that provides the percentage of used CPU time and memory utilization for each process.

As the resource demands during the encoding process might vary based on the video, we have used multiple videos with different frames per seconds, average data rates, and visual characteristics, as described in Table I. Each video has a resolution of 1920x1080 and 60 seconds duration.

We used the MP4 format for the source videos, with the video stream encoded in H.264 format, the standard video codec for the distribution of video content over Internet. Data rate of the videos including the audio ranges from 628 Kb/s to 101,628 Kb/s. We used the encoder *x264*, which requires a preset value for the trade-off between video quality and encoding time. We used the value *medium* for videos with low bit rates and good visual quality.

For each video file, described in Table I, we altered the following variables, resulting in a total of 10,500 test runs:

- Resolution: 426x240, 640x360, 854x480, 1280x720, 1920x1080
- Frame rates: 24, 30, 40, 50, 60
- Bit rates: 500 Kb/s to 10500 Kb/s in steps of 500
- Number of vCPUs: 1 to 4

We present the results of these experiments, including analysis of CPU utilization, memory consumption, and achievable frame rates in the following sections. To analyze the data, we have used support vector regression (SVR) [16] and polynomial regression (PR), as two fundamentally different methods with different advantages and limitations.

PR can be applied to linear and non-linear problems by adjusting the degree of the polynomial. SVR is a machine learning technique which works for both linear and non-linear problems. SVR can be applied reliably to avoid producing over-fitted functions. PR requires manual examination of the results to decrease the degree of the polynomial in case of over-fitting. When using SVR, only the so-called support vectors, which are a subset of the input data, have influence on the model. Therefore, wrong parameters can lead to results that ignore important input data points. To avoid these effects, careful parameter tuning is necessary for both approaches.

The general goal of the SVR approach is to find a function that gives estimated values with a deviation of at most ϵ from the actual training data points. To relax the requirements, slight errors are also allowed in the model. These errors are penalized using a value C, which defines the trade-off between the allowed deviations and the complexity of the model. We have tested and validated our models using different values for these parameters, to get balanced values for our predictions.

We have evaluated the quality of our models using the mean squared error (MSE). The error is a nonnegative value and represents how much predictions of a model deviate from the actual values they should predict. We have additionally evaluated the visual plots of the models to make sure they are not over- or under-fitting and adapted them if necessary.

IV. ANALYSIS

In this section, we present examples of the prediction models we have developed. A complete report of the models is available as a bachelor's thesis [17].

We used SVR- and PR-based models for predicting the minimum required vCPUs. The training data we have used for creating the SVR-based model consists of the test runs where the frame rate was never below the targeted frame rate, using the minimum number of vCPUs among all such observations. We have set $C{=}10^6$ and $\epsilon{=}10^{-12}$ after testing a wide range of values for these parameters and evaluating them based on MSE

and their visual representations. Figure 1(a) shows a plot for predicting the number of vCPUs based on bit rate, resolution, and frame rate. To be able to visualize this 4-dimensional relationship, here we have fixed the bit rate to 5500 Kb/s.

For the PR-based approach, we have tested degrees 0 to 10 for the polynomial. While degree 7 gave the lowest mean squared error, the plots suggest a highly over-fitted model using this degree. Figure 1(b) shows the PR model (for bit rate 5500 Kb/s) using the following 1st-degree polynomial, which resulted in the best trade-off between MSE value and observable over-fitting. $c(b, r, f)$ represents the number of vCPUs, given bit rate b, resolution r, and frame rate f[1]. Black dots represent the measured data points.

$$c(b, r, f) = 3.29 \cdot r + 1.77 \cdot f + 1.10 \cdot b - 0.96$$

Comparing the mean squared errors and different plots of the SVR and PR models, the SVR-based approach gives better predictions for the minimum number of required vCPUs.

We have also developed additional SVR and PR models to predict the minimum number of required vCPUs c based on the target bit rate b, resolution r, or frame rate f individually. We omit the corresponding plots due to space constraints and present only the corresponding functions as follows.

$$\begin{aligned} c(b) &= 15.12 \cdot b^5 - 50.31 \cdot b^4 + 62.12 \cdot b^3 - 35.20 \cdot b^2 \\ &\quad + 9.58 \cdot b + 0.10 \\ c(f) &= -0.017 \cdot f^2 + 0.93 \cdot f + 1.70 \\ c(r) &= -111.57 \cdot r^2 + 35.55 \cdot r + 0.73 \end{aligned}$$

Predicting and allocating the right amount of memory to the video encoding and streaming VM ensures that no data is swapped into the hard drive and eliminates this performance-limiting factor for the video streaming application. Similar to the prediction models for the number of required vCPUs, we have developed SVR and PR models for predicting the maximum amount of required memory.

As training data for the SVR-based prediction model, we have taken the used memory in test runs where the minimum number of vCPUs are used and no violation of the target values for bit rate, resolution, and frame rate has occurred for all video files. Figure 2(a) shows the SVR-based model with C=10 and ϵ=10, for bit rate of 5500 Kb/s. With these values we could observe the lowest MSE. Figure 2(b) shows the PR-based model using a 1st-degree polynomial, which resulted in the best trade-off between the MSE and the over-fitting detectable in different plots. In this function, $m(b, r, f)$ represents the maximum required memory (in MB) to achieve a given bit rate b, resolution r, and frame rate f.

[1]All coefficients in this and all following functions have been rounded to 2 decimal points. b is given as Kb/s, r as height of the video in pixels assuming a 16:9 aspect ratio, and f as frames/s. Moreover, we have divided the values of these parameters by powers of 10 in our experiments, such that all values are between 0 and 1, to avoid computational errors we were observing in our SVR models using the actual values.

$$m(b, r, f) = 472.54 \cdot r + 196.77 \cdot f + 18.38 \cdot b - 130.51$$

The SVR-based model predicts the maximum required memory with a lower mean squared error than the PR-based model. However, examining different plots for both models shows that the predictions from both models cover the measured data points similarly well in most cases.

Using similar approaches, we have developed prediction models for the achievable frame rate based on bit rate, resolution, and the number of available vCPUs. In our test runs, we have observed seconds where no frames were encoded, e.g., because of encoding delays. To prevent having these values distorting the prediction model, we have calculated the frame rate per configuration with a confidence interval at 95 % confidence. For this, we have taken all values for frame rate $f(r, b, c)$ over all encoding processes using all values for resolution (r), bit rate (b), number of vCPUs (c), and all video files. Instead of the actual minimum value for observed frame rate, we have used the lower bound of the confidence interval as the minimum observed frame rate for training our model.

Figure 3(a) shows a plot for the SVR-based prediction, using $C = 10^8$ and $\epsilon = 10^{-7}$, for bit rate 5500 Kb/s. Figure 3(b) shows the PR-based prediction using a 4th-degree polynomial function. The function has 36 coefficients, so we omit it in the paper. As shown in the plots, both approaches give only partially reasonable predictions and cannot adapt to all values. The reason for this could be the large diversity in the observed values for different metrics. The model might be improved by analyzing different subsets of the gathered data individually, for example, by separating the data gathered with different number of vCPUs.

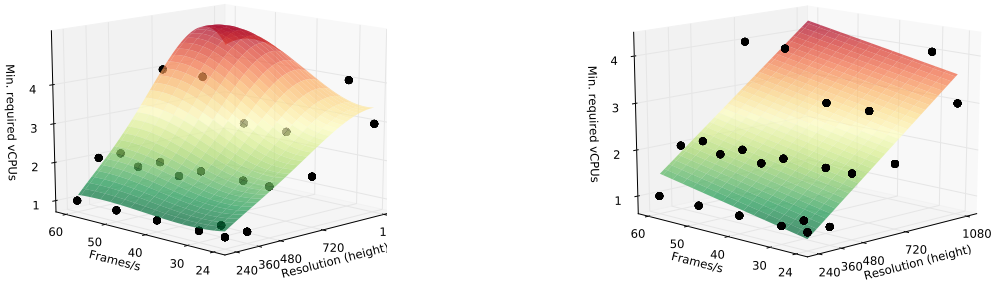
V. Conclusion and Future Work

This work is an attempt to provide some insights into the performance and resource demands of components of a video streaming service. We have done experiments using a video encoding and streaming function, as well as a cache function under different resource constraints and different performance targets. Our models show the feasibility of characterizing the resource demands and performance metric values of distributed cloud service components. Our analysis shows that these relationships are non-trivial even for simple applications, reinforcing the need for experimental data.

As our models are based on data from specific hardware settings, a more generic model can be obtained by normalizing the results based on experiments in different environments.

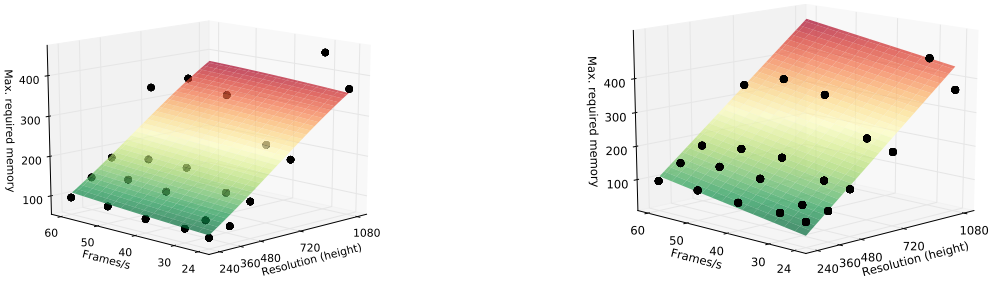
Acknowledgments

This work has been supported by the 5G-PICTURE project, co-funded by the European Commission 5G-PPP Programme under grant number 762057. This work has been performed in the framework of the German Research Foundation (DFG) within the Collaborative Research Center On-The-Fly Computing (SFB 901).



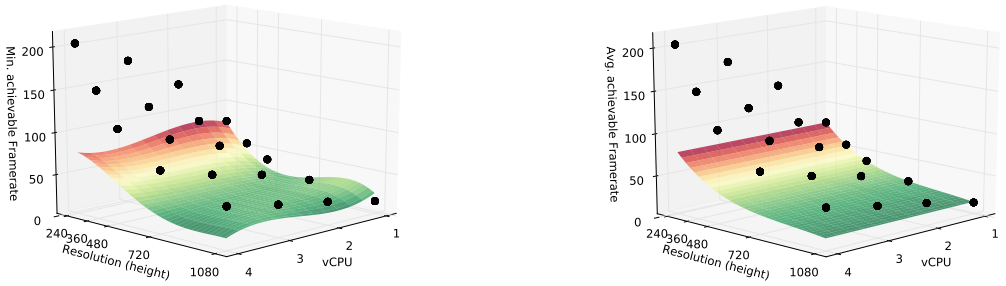
(a) SVR approach (b) PR approach

Fig. 1. Prediction of required vCPUs based on bit rate, resolution, and frame rate, shown for bit rate of 5500 Kb/s.



(a) SVR approach (b) PR approach

Fig. 2. Prediction of required memory based on bit rate, resolution, and frame rate, shown for bit rate of 5500 Kb/s.



(a) SVR approach (b) PR approach

Fig. 3. Prediction of achievable frame rate based on bit rate, resolution, and number of vCPUs, shown for bit rate of 5500 Kb/s.

REFERENCES

[1] S. Dräxler, H. Karl, and Z. Á. Mann, "Joint optimization of scaling and placement of virtual network services," in *IEEE/ACM CCGrid*, 2017.

[2] M. Peuster and H. Karl, "Understand your chains: Towards performance profile-based network service management," in *IEEE EWSDN*, 2016.

[3] ——, "Profile your chains, not functions: Automated network service profiling in devops environments," in *IEEE NFV-SDN*, 2017.

[4] "Data from the experiments in this paper." [Online]. Available: https://uni-paderborn.sciebo.de/index.php/s/G9q2hmUNg4n8LEg

[5] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Capacity management and demand prediction for next generation data centers," in *IEEE ICWS*, 2007.

[6] S. Rasoolzadeh, M. Saedpanah, and M. R. Hashemi, "Estimating application workload using hardware performance counters in real-time video encoding," in *7th Int. Symposium on Telecommunications (IST)*, 2014.

[7] Q. Fan and Q. Wang, "Performance comparison of web servers with different architectures: A case study using high concurrency workload," in *IEEE HotWeb*, 2015.

[8] X. Xu, T. Xu, Y. Yin, and J. Wan, "Performance evaluation model of web servers based on response time," in *IEEE Conf. Anthology*, 2013.

[9] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *IEEE CLOUD*, 2011.

[10] I. Giannakopoulos, D. Tsoumakos, N. Papailiou, and N. Koziris, "Panic: Modeling application performance over virtualized resources," in *IEEE Int. Conf. on Cloud Engineering*, 2015.

[11] [Online]. Available: https://peach.blender.org/download/

[12] [Online]. Available: http://www.imdb.com/title/tt2608732/

[13] [Online]. Available: http://www.imdb.com/title/tt3996164/

[14] [Online]. Available: https://www.youtube.com/watch?v=ERTIWNfx93w

[15] [Online]. Available: https://www.youtube.com/watch?v=DH0BQtwEAsMw

[16] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[17] M. Illian, "Prediction of resource requirements and performance of virtualised network functions in a video streaming context," Bachelor's Thesis, Paderborn University, 2017.

A Distributed Protocol for Crowd Counting in Urban Environments

Peter Danielis*, Sylvia T. Kouyoumdjieva† and Gunnar Karlsson†
*Faculty of Computer Science and Electrical Engineering, University of Rostock, Germany
†ACCESS Linnaeus Center, School of Electrical Engineering, KTH Royal Institute of Technology, Sweden
E-mail: peter.danielis@uni-rostock.de,{stkou,gk}@kth.se

***Abstract*—Monitoring, control, and estimation of spontaneous crowd formations in cities, e.g., during open-air festivals or rush hour, are necessary actions to be taken by city administration. The most common way to implement these measures is via installation of observation cameras for the purpose of crowd counting. However, cameras are more suitable for intensive surveillance of restricted hot spots during well-defined time periods than for long-term monitoring of big areas. In this work, we present a fully distributed crowd counting protocol for urban environments with high crowd densities. It relies on nodes equipped with mobile phones roaming around in an urban area and participating in the distributed crowd estimation. Each node collects crowd size estimates from other participants in the system whenever in communication range and immediately integrates these estimates into a local estimate. The objective of the proposed protocol is to produce a precise mapping of the local estimate to the real value. We evaluate the proposed protocol via simulations of realistic mobility models. We demonstrate that in dense environments the local estimate does not deviate by more than 7%.**

I. Introduction

Crowds can occur in urban environments during city festivals, sport events, rush hours, or in shopping areas. As a result, monitoring, control, and estimation of crowd densities and sizes are crucial for city administration, especially for managing disaster situations. In this work, we investigate the use case of *crowd counting* with particular focus on the estimation of the size of dense crowds with hundreds of people. Crowd counting can be applied for the purpose of urban planning and is particularly useful if global knowledge about the crowd size is unknown, e.g., inside a subway station. Nowadays, observations cameras are often deployed for crowd counting. However, the application of cameras imposes several disadvantages. The achievable image quality depends on the environment, in which cameras are deployed, and privacy issues are raised. Alternative solutions execute crowd counting based on measurements obtained from central instances or acquired by devices carried by humans.

In this work, we introduce a fully distributed protocol for distributed crowd counting in urban environment with moving pedestrians. It obtains information required for the accurate crowd estimation through mobile device-to-device communication between the mobile phones of pedestrians and hence does not need additional infrastructure. The proposed protocol enables the calculation of a local estimate of the *current* crowd size in an area. Note that people in a crowd are often only *temporarily connected* since solely a few are in direct communication range of each other at a point in time. Moreover, a crowd is subject to *churn*, i.e., pedestrians arrive steadily in an area, linger there for some time, and eventually exit. Consequently, the pedestrians have to refresh their local estimate of the crowd size continually to consider the churn. They update their local estimates immediately as soon as they are in range of another pedestrian by exchanging crowd size information through any wireless communication interface on their mobile phones.

In this extended abstract, we present selected results based on a previously published work [1]. It was shown in [1] that for densities above 0.1 nodes/m^2, the distributed crowd counting protocol quickly converges to precise estimates under realistic mobility. We describe the fully distributed protocol for crowd counting, which is able to operate in urban areas with moving pedestrians who continually enter and leave the area. We evaluate it by means of simulations with realistic mobility models. In dense urban scenarios, distributed crowd counting is able to determine precise estimates and achieves an accuracy of at least 93%. As opposed to the state-of-art, our protocol is able to correctly estimate the crowd size even in highly dynamic scenarios.

II. The Distributed Protocol for Crowd Counting

The proposed protocol works fully decentralized in that each pedestrian determines a precise estimate of the people in a crowd. Moving pedestrians, further referred to as nodes, therefore continually broadcast data containing all nodes they have got to know about (shared nodes) and who have exited the area (left nodes) and thereby the data is distributed epidemically. Received data of other nodes is merged with own data in order to compute the current crowd size from the difference of the number of shared and left nodes. Since nodes do not only arrive in an area but also leave it after some time, each node needs to announce its intention to exit once it enters a specific boundary region of the area. To avoid considering the same nodes more often than once, shared and left nodes can be kept in bit vectors, in which a bit position denotes a particular node. All bit positions except the one referring to the own ID are initialized with 0s. Bit positions are set to 1 once knowledge about shared or left node is received from another node.

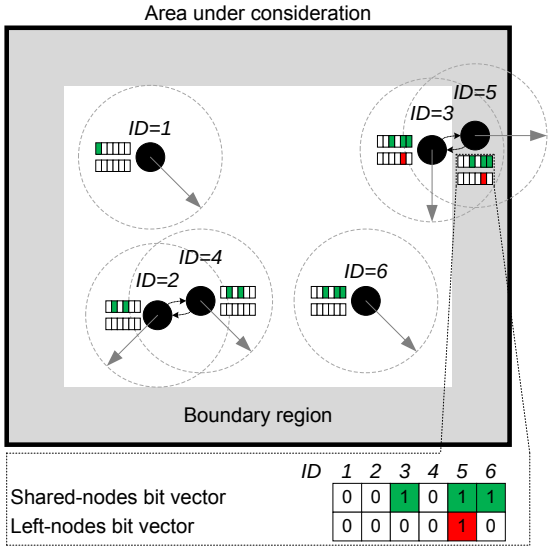


Fig. 1. An example of the execution of the proposed protocol for six moving nodes.

In order to explain how our protocol works, we give an example of six moving nodes in Fig. 1. If a node arrives in an area, it enters himself into its shared-nodes vector while its left-nodes vector remains empty. For instance, nodes 2 and 4 as well as nodes 3 and 5, have already exchanged their vectors and hence identically estimate the crowd size. Moreover, node 6 has already exchanged its vector with node 3 and 5 before node 5 announces its intention to leave.

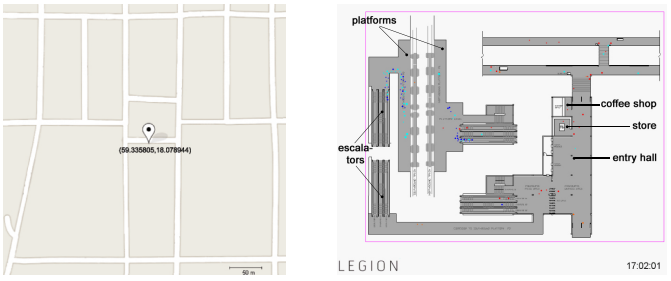
Nevertheless, [2] shows that keeping crowd size estimates in a binary bit vector does not scale well due to the fact that the size of the bit vector grows linearly with the number of people in a crowd. To solve this issue, we utilize *D-GAP compression* [3], which has been shown to be beneficial in [4]. Bit vectors are therefore actually represented by two D-GAP vectors. To preserve the privacy of a node, the position, in which its information is stored in the D-GAP vectors, is computed by a hash function like MD5, e.g., from the MAC address of a node's mobile phone.

III. Evaluation Scenarios

In this section, we describe the realistic pedestrian scenario, the simulation setup, and the performance metric. Mobility scenarios are characterized by an arrival rate λ, with which nodes arrive in an area A. They stay in the area with a lifetime T before they exit.

A. Realistic pedestrian mobility

To model pedestrian mobility in a realistic way, we make use of the Walkers traces [5] captured in the commercial simulator Legion Studio [6]. Its multi-agent pedestrian model relies on analytical and empirical models, which have been calibrated by measurement studies. Each simulation run produces a trace file, which comprises a snapshot of the positions of all nodes every 600 ms. Fig. 2(a) and 2(b) depict the realistic environments considered in our evaluation: an outdoor urban scenario, modeling a city square in central Stockholm, Sweden



(a) (b)

Fig. 2. Urban scenarios: (a) a square in downtown Stockholm, Östermalm, and (b) a two-level subway station.

(which we later denote as the Östermalm scenario), and an indoor scenario, recreating a two-level subway station. The Östermalm scenario comprises a grid of interconnected streets. Fourteen passages connect the observed area to surrounding places. The active area, i.e., the total surface of the streets, is 5872 m^2. The nodes are continually moving and thus the scenario exhibits high mobility. The Subway station has train platforms connected via escalators to an entry level. Nodes arrive on foot from any of five entries, or when a subway arrives at the platform. The subway arrivals lead to burstiness in the node arrivals and departures. Nodes gather while waiting for a subway at one of the platforms, or while having a rest in the store or the coffee shop at the entry level. The active area is 1921 m^2.

B. Simulation setup

In our evaluation scenarios, for the sake of simplicity we suppose that all nodes carry mobile phones and all are taking part in the crowd counting. Scenarios, in which not all people participate or are malicious, are out of scope and left for future work. For the evaluation, we make use of an opportunistic content distribution system in the OMNeT++ modeling framework MiXiM [7]. Each simulation run is executed in synchronous rounds of 600 ms which corresponds to the granularity of the mobility traces. Nodes broadcast their shared-nodes and left-nodes vector at the start of each round. To reduce collisions on the wireless medium, the broadcast transmission of each node in each round is distributed uniformly at random $\mathcal{U}(0, 500)$ ms. The transmission range is set to 10 m.

C. Performance metric

We evaluate our results by means of the accuracy Δ as performance metric. The deviation is a measure of the accuracy of the proposed protocol, i.e., it demonstrates how close the local estimate of a node is to the real value. Let $\hat{x}$ be the local estimate and x the actual real value. Then, the deviation is calculated as:

$$\Delta = \left| \frac{\hat{x} - x}{x} \right| \tag{1}$$

IV. Selected Simulation Results

In this section, we evaluate the accuracy of the proposed protocol with respect to different arrival rates. We first present

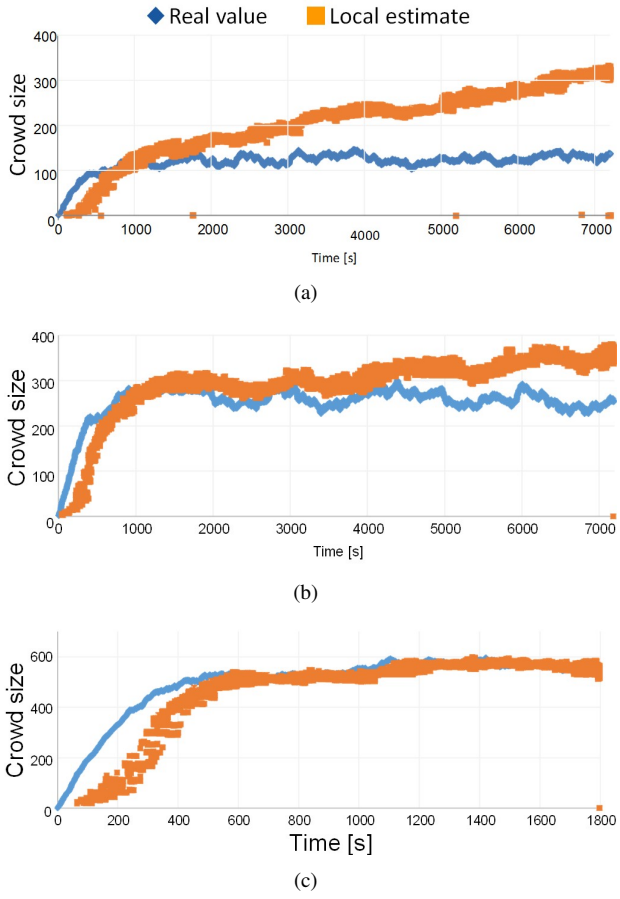


Fig. 3. Local crowd size estimates of all nodes vs. real value for the Östermalm scenario: (a) λ = 0.42 nodes/s, (b) λ = 0.98 nodes/s, and (c) λ = 2.1 nodes/s.

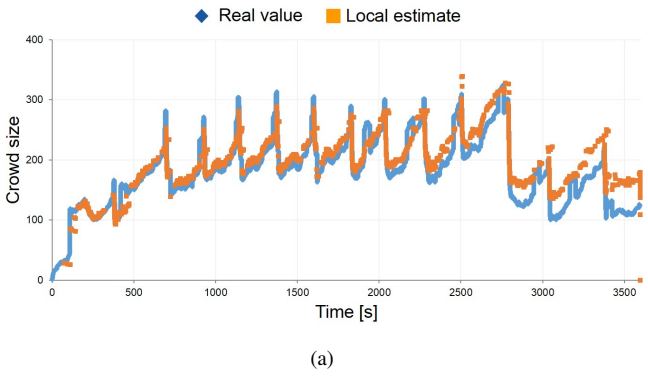


Fig. 4. Local crowd size estimates of all nodes vs. real value for the Subway scenario.

selected results for the Östermalm scenario. Figure 3 depicts the local estimates vs. real value over time for $\lambda = \{0.42, 0.98, 2.1\}$ nodes/s with average node numbers of $\{137, 312, 614\}$ nodes. The results confirm that in sparser scenarios, Figure 3(a) and 3(b), the crowd estimate is often overestimated due to poor information spreading. However, as λ increases, the local estimate approaches the real value leading to an accuracy of 95% in Figure 3(c) (λ = 2.1 nodes/s, average node number = 614 nodes).

Finally in Figure 4, we show the accuracy in the Subway scenario with an average node number of 205 nodes. We would like to point out that the scenario is more challenging than the Östermalm scenario due to the bursty nature of node arrivals and departures. The results reveal that the proposed protocol is able to correctly estimate the crowd size in the area with deviations from the real value of no more than 7%.

V. Related Work

The three different techniques that can be used for crowd counting comprise image-based, device-free non-image based, and device-based non-image based. Our approach falls into the latter category since nodes carry a mobile phone and hence we compare our protocol to device-based non-image based techniques.

In [8] and [9], a system is described, in which users voluntarily scan the environment for discoverable Bluetooth devices with their mobile phones to analyze crowd conditions in urban environments. The authors acknowledge that their solution needs to recruit volunteers who regularly move around the city and leave investigations on the volunteers' number and mobility patterns for future work. A system using audio tones is proposed in [10]. Mobile devices receive audio frequencies from other devices and broadcast a corresponding bit pattern as crowd size estimate. In an experimental setup consisting of 25 Android devices, crowd size estimates with 90% accuracy are obtained while the accuracy decreases to 50% in other scenarios. Moreover, the counting latency increases tremendously when the number of device increases, which makes the system unsuitable for large-scale scenarios. An approach to compute concurrent estimates in dense wireless networks with up to 100 nodes is proposed in [11]. This solution uses the rendezvous time to capture the density of the environment. Although mobility is taken into account, the evaluation setup solely comprises a few moving nodes. The work of Khan et al. suggests using the acoustic and motion sensors of mobile phones to determine the number of people in a group [12]. However, their solution is only suitable for small groups of up to 10 people. Our protocol however scales well with an increasing number of nodes in an area and can estimate crowds of different sizes. Pajevic et al. propose an approach to estimate the size of a (mobile) crowd by epidemically broadcasting small messages among users [13]. They investigate the performance of their approach using a stochastic model. However, the model is unable to capture changes in scenarios with bursty arrivals and departures such as a subway scenario. In contrast to this work, our protocol can also determine the size of highly dynamic crowds.

Crowd counting can also be realized by distributed aggregation protocols with a counting objective [14]. However, such protocols either perform poorly in dynamic environments or they lack accuracy. In contrast to these protocols, our protocol is able to achieve accurate estimates in highly dynamic environments.

VI. Conclusion

Monitoring, control, and estimation of spontaneous crowd formations in urban environments, e.g., during open-air festivals or rush hours, are important measures for city administration. In this work, we introduced a fully distributed protocol for crowd counting via device-to-device communication, which is able to operate in urban environments with moving nodes and churn. Each mobile device carried by a pedestrian collects crowd size estimates from other participants in the system whenever in direct communication range and immediately integrates these estimates into its own local estimate. We evaluated the proposed protocol via simulations of realistic mobility models. We demonstrated that for sufficiently dense scenarios, distributed crowd counting produces precise estimates. When the density is sufficient, we showed that the proposed protocol is able to produce a local estimate with at least 93%. Moreover, we presented that it provides a scalable solution for crowd counting for both indoor and outdoor scenarios. As part of our future work, we plan to evaluate other scenarios in a realistic testbed.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG) (research fellowship, GZ: DA 1687/2-1) and Stockholm County Council (SLL) (research grant LS2016-1423) for their financial support.

References

[1] P. Danielis, S. T. Kouyoumdjieva, and G. Karlsson, "Urbancount: Mobile crowd counting in urban environments," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Oct 2017, pp. 640–648.

[2] D. Chronopoulos, "Extreme chaos: Flexible and efficient all-to-all data aggregation for wireless sensor networks," Ph.D. dissertation, TU Delft, Delft University of Technology, 2016.

[3] A. Kuznetsov, "D-gap compression," 2002. [Online]. Available: http://bmagic.sourceforge.net/dGap.html

[4] P. Danielis, S. T. Kouyoumdjieva, and G. Karlsson, "Divote: A distributed voting protocol for mobile device-to-device communication," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 01, Sept 2016, pp. 69–77.

[5] S. T. Kouyoumdjieva, Ó. R. Helgason, and G. Karlsson, "CRAWDAD data set kth/walkers (v. 2014-05-05)," Downloaded from http://crawdad.org/kth/walkers/, May 2014.

[6] "Legion Studio," http://www.legion.com/.

[7] Ó. R. Helgason and K. V. Jónsson, "Opportunistic networking in OMNeT++," in *Proc. SIMUTools, OMNeT++ workshop*, 2008.

[8] J. Weppner and P. Lukowicz, "Collaborative crowd density estimation with mobile phones," *Proc. of ACM PhoneSense*, 2011.

[9] J. Weppner, P. Lukowicz, U. Blanke, and G. Tröster, "Participatory bluetooth scans serving as urban crowd probes," *IEEE Sensors Journal*, vol. 14, no. 12, pp. 4196–4206, Dec 2014.

[10] P. G. Kannan, S. P. Venkatagiri, M. C. Chan, A. L. Ananda, and L.-S. Peh, "Low cost crowd counting using audio tones," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM, 2012, pp. 155–168.

[11] M. Cattani, M. Zuniga, A. Loukas, and K. Langendoen, "Lightweight neighborhood cardinality estimation in dynamic wireless networks," in *Proceedings of the 13th international symposium on Information processing in sensor networks*. IEEE Press, 2014, pp. 179–189.

[12] M. A. A. H. Khan, H. M. S. Hossain, and N. Roy, "Sensepresence: Infrastructure-less occupancy detection for opportunistic sensing applications," in *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, vol. 2, June 2015, pp. 56–61.

[13] L. Pajevic and G. Karlsson, "Characterizing opportunistic communication with churn for crowd-counting," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, June 2015, pp. 1–6.

[14] L. Nyers and M. Jelasity, "A comparative study of spanning tree and gossip protocols for aggregation," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4091–4106, 2015, cpe.3549.

Adaptive Complex Event Processing over Fog-Cloud Infrastructure Supporting Transitions

Manisha Luthra
Technical University of Darmstadt
Multimedia Communication Lab (KOM)
Darmstadt, Germany
manisha.luthra@KOM.tu-darmstadt.de

Boris Koldehofe
Technical University of Darmstadt
Multimedia Communication Lab (KOM)
Darmstadt, Germany
boris.koldehofe@KOM.tu-darmstadt.de

Ralf Steinmetz
Technical University of Darmstadt
Multimedia Communication Lab (KOM)
Darmstadt, Germany
ralf.steinmetz@KOM.tu-darmstadt.de

***Abstract*—Fog Computing is an emerging trend that can enable profound applications in the Internet of Things (IoT) arena. The IoT applications typically deliver vital information from multiple sources to the end-users in the form of notifications e.g., heart status in health monitoring. Complex Event Processing (CEP) is a powerful paradigm that bridges this gap from raw sensor data to meaningful information. But, IoT applications involves a wide distribution of heterogeneous devices that are highly dynamic (e.g. mobile nodes). This poses a strict need for a highly adaptive system. Consequently, first we propose a TCEP system that allows in-network processing of CEP operator graph on the fog-cloud infrastructure. Secondly, we describe how such a system can benefit from *transitions* between different CEP algorithms (mechanisms) to overcome the heterogeneity and dynamics of the fog-cloud infrastructure. This leads us to important research questions related to transition that are presented and addressed in this research work.**

***Index Terms*—Fog Computing, Internet of Things (IoT), Complex Event Processing (CEP)**

I. Introduction

Nowadays, there is a growing surge of Internet of Things (IoT) applications like smart cities, health, industry (Industrie 5.0) etc. [1] The IoT applications connect a multitude of devices over an interoperable communication network, exchanging enormous amount of information about themselves and their surroundings. Complex Event Processing (CEP) is an important paradigm to extract meaningful information from disparate sources in real-time. Typically, a CEP system consists of *event producer(s)* generating low-level data streams e.g., sensor data that are to be processed and correlated. The *event consumer(s)* or the IoT application *end-users* can specify the events of interest as a continuous query with the system. The query is transformed into an *operator graph*, where *operators* are the semantic units of query e.g., joins, filter, windows etc. The *event broker(s)* perform distributed in-network query (operator graph) processing, in order to determine the events of interest. The CEP system then notifies the events of interest to the event consumer(s). In this way, CEP provides a powerful way to deliver meaningful information to the end user for IoT applications.

However, the emerging wave of IoT applications e.g., in *Smart City* environment are demanding in one or more performance objectives, stringent latency requirement, network bandwidth constraints, mobility support and location-awareness. *Fog-computing* [2] is an emerging platform that provides some of the aforementioned characteristics. It extends the cloud-computing paradigm to bring computation towards the edge near to the end-users (consumers). Big cloud providers like Amazon and Google have recently launched fog locations, Amazon CloudFront[1] and Google Cloud CDN[2] respectively, that enable new breed of such applications in gaming, e-commerce, social media etc.

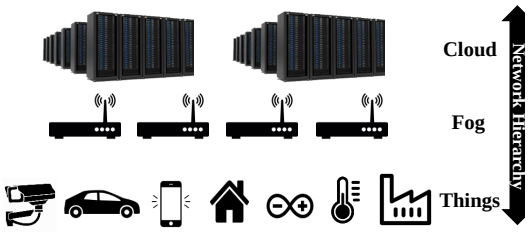


Fig. 1. Fog-Cloud infrastructure for deployment of IoT applications.

It is often pointed out that a federation of cloud and fog can allow wider range of applications where latency sensitive operators can be placed at the fog, while compute intensive operators at the cloud. An example of such a network hierarchy is illustrated in Fig. 1 [3].

Such a diffused infrastructure is suitable to support large-scale distributed CEP systems enabling deployment of IoT applications. For instance, to perform in-network continuous *query* processing over the fog-cloud infrastructure. However, there are two main challenges in this: 1) presence of heterogeneous infrastructure as illustrated in Fig. 1 by the three layers (things, fog and cloud) ranging from smartphones, vehicles to switches and routers to data center, 2) dynamic streaming environment consisting of: a) *data streams* produced at varying data rates, b) *devices* that can be mobile and c) fluctuating properties of the *communication network* e.g., bandwidth. Most importantly, the dynamic environment influences performance objectives of a large number of users. The distributed CEP system must take into account these challenges to deliver the desired performance e.g., low latency to an IoT application. Yet, state of the art distributed CEP (DCEP) systems fall short in their support towards highly

[1]https://aws.amazon.com/cloudfront/ [Accessed January 2018]
[2]https://cloud.google.com/cdn/docs/locations [Accessed January 2018]

dynamic nature of event consumers, producers and brokers. Considering mobility, there exists some approaches [3] that allow producers and consumers to be mobile, but connected to a fixed or quasi-stationary broker network. Additionally, they are restricted in their flexibility of providing mechanisms at run-time, that could deal with the dynamic nature of the environment. Some authors proposed elasticity in DCEP systems to deal with varying workload, but lack support for higher mobility [4].

To this end, we propose a concept for a highly adaptive DCEP system over a fog-cloud infrastructure that supports strong dynamics of streaming environment (viz. producers, consumers and brokers) for IoT applications. This is accomplished by runtime adaptation, aka. *transition* [5] between distributed CEP mechanisms. The transition to a *new* DCEP mechanism must fulfil varying and conflicting performance objectives from a large number of users. The ultimate goal is to provide methods to enable adaptation between DCEP mechanisms to deal with a dynamic streaming environment and performance objectives. Towards this, we investigate the following research questions for a Transitive-CEP (TCEP) system:

1) What are the potential mechanisms in a DCEP system that can benefit from transitions?
2) How can we specify transitions in a DCEP system?
3) How, when and who triggers the transitions in a TCEP system?
4) How to enable uninterrupted user experience using TCEP system?

We elaborate further on the problem based on a case study on Smart City Environment in Section II, where flexibility in performance objectives is desired. Afterwards, we enumerate important challenges identified in the case study, that influences the execution of an CEP *operator graph*, in Section III. Next, we identify key transitions in DCEP mechanisms for operator graph and overlay network in Section IV. Finally, we conclude and give next steps for our fog-cloud assisted Transitive-CEP (TCEP) system in Section V.

II. Case Study: Smart City Environment

According to Cisco[3], the *Smart-city* market is estimated to generate a revenue of hundred billion dollars by 2025. Running projects on the smart cities like European Smart Cities[4], highlight the increasing significance of key industry and service sectors in this domain including Smart Mobility, Smart Health, Smart Home, and other value added services. While the IoT space offers an end-to-end solution, our focus in this work is to enable real-time processing of IoT data using DCEP.

Current IoT architectures rely on either of the two extremes for data processing – cloud, or fog [2]. On the one hand, sending all the data to the cloud for high capacity storage and computation, e.g., video surveillance for traffic monitoring. On the other hand, time-critical applications like autonomous driving rely on local computation (on the sensors), particularly the Things layer in Figure 1. Still, IoT applications e.g., in the Smart City scenario pose challenges that are not solved by the two extreme architectures. For instance, routing data to the cloud can take several hundred milliseconds to react, that can lead to life critical situation, e.g., for future autonomous car to car communication. On the other side, some IoT devices are resource constrained, hence processing data locally can also be complex as well as costly. For instance, widely used Raspberry Pi devices used for multiple purposes comes with 1.2 GHz processor and 1 GB RAM. Such devices can only be used for pre-processing primitive events. Another concern is that sending data over a communication network to a cloud server consumes much energy and bandwidth. This implies a complex trade-off, particularly if local nodes are battery-powered. In this paper, we address these limitations of current IoT architectures by flexible and adaptive DCEP enabling in-network *query* processing guided by transitions (cf. Section IV). Therefore, we focus on three important questions: 1) *where*, 2) *how* to perform processing and 3) how placement of processing operators *impact performance objectives* of IoT end users.

A. Traffic Control

According to INRIX 2017 Global Traffic Scorecard[5], European drivers spent over 91 hours in congestion last year. The traffic congestion continues to rise, if it is left unchecked. A multitude of sensors in smart cities such as smart cameras, environmental sensors like audio, radars, induction loops and GPS sensors on smart cars can be used to derive insightful information about traffic. For example, notifications can be delivered to the drivers regarding the traffic flow, congested roads, unobstructed roads, or warning about road condition and accidents. DCEP allows this correlation of sensor data from the multiple data sources to derive higher level information such as the status of traffic congestion. This is performed by processing the information inside the network at multiple devices (e.g., fog or cloud nodes). We look into three possibilities of distributed CEP: 1) distributed local/fog CEP, 2) distributed cloud CEP or 3) distributed fog-cloud CEP.

Although, devices such as smartphones operate on high-speed processors with clock frequencies upto 1 GHz, processing big video streams from traffic monitoring cameras, locally on these low-powered sources is not resource efficient. A typical traffic monitoring camera captures at a resolution $>= \ 320 \times 240$ pixels and frame rate $10fps$ i.e. 768,000 pixels/data points per second, approximately. Sending all of this data to a cloud server imposes a significant cost. On the other side, processing lower level sensor data from vehicular sensors can be performed at local/fog nodes. Thus, trade off between performing DCEP locally or at cloud must be considered.

[3]The city of the future: Smart and connected, from http://www.cisco.com/web/tomorrow-starts-here/connectedcities/preview.html [Accessed January 2018]

[4]http://www.smart-cities.eu/ [Accessed January 2018]

[5]http://inrix.com/scorecard/ [Accessed January 2018]

The detection of complex event like traffic congestion is time and location dependent. It is not mindful to send a notification for traffic congestion when it has been cleared (time), or for a road where the user is not travelling (location). Thus, for an accurate and efficient decision, the system must be time and location-aware, but also be aware of other environmental factors (context-aware) that are significant for controlling traffic congestion. For example, it is important to note that the traffic conditions are not the same the entire day, as they are during rush hours. Therefore, it is very important to reconfigure DCEP mechanisms at runtime in accord with the traffic conditions and the need of end-users. For instance, for emergency services the notifications must be delivered undelayed, in contrast to a normal user.

B. Smart Health Monitoring

With the commence of IoT, there has been a growing number of devices that allows efficient health monitoring, including fitbit, body cardio scale, blood pressure (BP) monitor, Kito+ and many more. DCEP allows to collaboratively process the information from this variety of sensors to gain meaningful insights on one's health, e.g., heart status. However, to make full use of these vital sensors, information must be processed quickly and efficiently. The power of cloud can be used to process the information quickly, but transferring data to cloud for processing is time consuming, and the delays caused might lead to life critical situations like a heart stroke. Besides, cloud computing can also be a source of data privacy concerns. The privacy can be protected if the data is processed locally, either within the sensor, the body or the house/hospital where it was produced. Since user tend to be mobile, once the data leaves the private sphere, sophisticated DCEP mechanisms must be deployed to guarantee protection of privacy. On the other hand, prevalent insights can be obtained by data collection and batch processing offered by cloud, e.g, heart status over an year.

Vital signs from various sensors can be used to predict individual's health status e.g., by means of tools like Early Warning Score (EWS) system. It is a manual tool widely used in hospitals to track the condition of patients. It involves measuring five physiological parameters namely heart rate, systolic BP, breath rate, SPO_2 and body temperature and assigning them score between 0 and 3, with a lower score meaning a better condition [2]. IoT-enabled wearables for health monitoring can be used to empower systems like EWS, to continuously track and predict individual's health in an automated way. However, the system alone faces open issues that must be addressed [2]. Environmental factors influences the vital signs like heart rate etc., which must be considered to reach to a more realistic solution. For instance, a heart rate of 120 beats per minute would be an alarming sign for a patient who is sleeping, while it can be completely normal for who is exercising. If not considered, such characteristics can drastically decrease the performance of the system in terms of accuracy and precision. Corresponding DCEP mechanisms are needed to adjust the scores in order to avoid false alarms as well as guaranteeing protection of privacy in such systems. To summarize, there is a strong need to reconsider DCEP mechanisms to cater the needs of different end users of IoT applications in accord with the environmental context.

III. Challenges

In this section, we summarize the challenges brought by IoT applications studied in the aforementioned case study examples of traffic control and health monitoring.

Dynamic streaming environment. a) Data streams produced by IoT devices arrive at varying rates and volume, but also with a constantly changing quality (or certainty). E.g., sensor data produced by some devices can be noisy and erroneous, while by some devices can be more accurate. Besides, b) the streamed data has to be processed by user devices that perhaps can be resource and memory constrained (cf. Section II-A). In addition, due to mobility some devices might become unavailable for processing. Furthermore, c) the communication network that connects the IoT devices, possess highly fluctuating properties e.g., time varying latency, bandwidth, etc., that might have an influence on performance.

Varying performance objectives. As pointed out earlier in our case study, the dynamic streaming environment causes changes in the performance objectives of the IoT application. For instance, latency requirement of an emergency service (ambulance) for traffic notification will be significantly urgent than of a normal user. Similarly, in health monitoring the environment or user context (like location and activity) impact the performance perceived by the end user – privacy and accuracy, respectively.

Heterogeneity of infrastructure. The diffused infrastructure of fog-cloud itself is a challenge because of network and system heterogeneity. Some nodes might be geographically located far away, while some could be near. Because of this, there can be huge latencies between some nodes. Thus, TCEP system must prepare for proper coordination and planning of execution, i.e. where to process an operator – on cloud or fog node, split an operator to fog and/or cloud or perform parallel processing at both of them. The decision varies based on the adaptive selection of CEP mechanisms.

IV. Mechanism Transition in DCEP

A. Need for Transition in DCEP mechanisms

In the view of the aforementioned challenges, there is a strong need of run-time adaptation of the underlying DCEP mechanisms. Current systems neglect that the DCEP mechanism performs well only under the given environmental conditions, the respective assumptions and performance objectives. However, if dynamics are introduced into the environment, the system's performance objective might not be met. For instance, higher workload might render system unreliable and inefficient, as observed in the case study presented in Section II. Besides, performance needs of large number of users might vary significantly e.g., for emergency service in traffic control and exercising habits of different users in health monitoring. Thus, a DCEP system must adapt its mechanisms at runtime subject to the performance objectives of the end-users. In our

existing work [5], we show that in this case a *transition* is well suited. We extend this work, to show our hypothesis that transition in CEP mechanisms e.g., operator placement (cf. Section IV-B) could aid us in run-time fulfilment of performance objectives for dynamic user environment.

B. Mechanism: Overlay and operator graph transitions

DCEP system detects events of interest by distributed in-network processing of *operator graph* over the fog-cloud infrastructure. To do this, an appropriate selection of a node to deploy an CEP operator is performed, i.e. well known as *operator placement* problem. The operators are assigned to the nodes such that the performance objectives specified by the system are achieved. The performance objective are such that, they best satisfy the end-user's requirements. Once the operators are placed on the nodes, the *operator graph* is processed in a CEP *overlay* called an *operator network* [3]. It has been proven in the literature, that an optimal assignment of an operator to a node is a NP-complete problem. Thus, many heuristics and approximation algorithms are proposed for operator placement [6]. Although, each algorithm provides a solution to optimally place an operator to a node, they make different assumptions for the respective problem. The design characteristics of the placement are based on these assumptions and the performance objective of the application. The dynamics in the environment as discussed before (cf. Section III) influences the assumptions, performance objectives and hence choice of the placement mechanism.

Centralized vs. Decentralized Algorithms. The placement algorithms are characterized based on: 1) *how* the placement decisions are made and 2) *whether* the decisions require centralized knowledge about the environment [6]. The centralized algorithms have central knowledge about the environment e.g., on network state, workload and the resources, whereas the decentralized algorithms make decision on placement based on local knowledge. Apparently, centralized algorithms suffer from scalability issues while decentralized algorithms not. The event workload on the system in the IoT applications vary significantly, as specified in our case study. Thus, transitions between centralized and decentralized algorithms are to be explored, to ensure that the CEP overlay is not under-utilized but also not over-utilized. Scalability can be provided in a distributed CEP system in two ways: 1) vertical scalability (scale up), 2) horizontal scalability (scale out). Scaling up means to add/remove resources to/from the nodes, while scaling out means to add/remove nodes in/from the CEP overlay network. Techniques like parallel processing (scale up) or load partitioning and balancing (scale out) can be used to adapt to decentralized algorithm and thereby satisfying the workload requirements. Recently, there is an increasing interest in providing such adaptations by using auto-scaling strategies [4]. This work could be a start point to further analyze the CEP overlay transitions.

Static vs. Dynamic Operator Network. Mobility is one of the major causes of dynamics in the IoT applications. Decentralized algorithms can efficiently respond to such dynamic changes at runtime. This requirement raises important issues like efficient operator migration. It refers to efficient movement of an operator from one node to another to optimally satisfy a performance objective, in response to changes in the environment e.g., mobility. However, operator migration is costly, especially for stateful operators [3]. Thus, transition between mechanisms for static vs dynamic networks are investigated to avoid additional costs and to provide an efficient and seamless transition. For static or quasi-static networks it is recommended to avoid operator migration, as it is very expensive. On the other hand, for extremely dynamic operator network, migration is crucial. Existing work [3], [5] looks into this problem but not for mobile operator network. Additionally, methods for cost of operator migration are yet undiscovered. Moreover, existing approaches for operator placement and migration satisfy the *same* performance objective i.e. statically specified at the design time. However, there are situations in an IoT environment where the performance objective changes as illustrated in the case study. For this reason, transitions in the identified CEP mechanisms are prevalent.

V. Preliminary Conclusion and Future Work

In this paper, we presented important research questions motivating the need for transitions between different existing CEP mechanisms in a highly adaptive and context-aware CEP system. We presented a case study on Smart City environment for two different applications, traffic congestion control and smart health monitoring. Several scenarios (traffic control and health) are identified where transitions in CEP mechanisms can be beneficial with assistance of a fog-cloud architecture. Finally, important CEP mechanisms are identified for transitions corresponding to the situations in our case study. Intuitively, transitions are costly and therefore in the near future, we will look into its cost and benefits w.r.t. aforementioned scenarios. Furthermore, important research questions such as how, when and who triggers the transition are to be investigated.

Acknowledgement: This work has been co-funded by the German Research Foundation (DFG) as part of the project C2 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] P. Samulat, *Die Digitalisierung der Welt: Wie das Industrielle Internet der Dinge aus Produkten Services macht*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017, pp. 103–124.

[2] M. A. Rahmani, L.-S. P. Preden, and A. Jantsch, *Fog Computing in the Internet of Things*. Springer International Publishing, 2018.

[3] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, and U. Ramachandran, "MCEP: A Mobility-Aware Complex Event Processing System," *ACM Transactions on Internet Technology (TOIT)*, vol. 14, no. 1, pp. 1–24, 2014.

[4] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer, "Latency-aware elastic scaling for distributed data stream processing systems," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. ACM, 2014, pp. 13–22.

[5] P. Weisenburger, M. Luthra, B. Koldehofe, and G. Salvaneschi, "Quality-aware runtime adaptation in complex event processing," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '17, 2017, pp. 140–151.

[6] G. T. Lakshmanan, Y. Li, and R. Strom, "Placement strategies for internet-scale data stream systems," *IEEE Internet Computing*, vol. 12, no. 6, pp. 50–60, 2008.

CLAP: Cooperative Locality-Aware Data Processing in Heterogeneous Fog Environments

Aditya Raj
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
aditya.raj@tu-clausthal.de

Andreas Reinhardt
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
reinhardt@ieee.org

***Abstract*—Wireless Sensor Networks deployed in recent years often share the commonality of relying on homogeneous hardware platforms. The upcoming vision of the Internet of Things, however, is strongly based on the co-existence of embedded devices manufactured and operated by different stakeholders. Thus, device heterogeneity will become an omnipresent characteristic of future sensor deployments, particularly within the scope of fog computing. In this paper we present CLAP, a collaborative data processing approach that exploits device heterogeneity for collaborative data processing instead of trying to mitigate its negative effects. Through simulations, we demonstrate that CLAP, in conjunction with a data collection protocol, can effectively reduce the volume of traffic and thus cater to energy savings and consequently prolonged network runtimes.**

I. Introduction

In the past decade, Wireless Sensor Networks (*WSN*s) have shown tremendous potential for application domains like disaster relief, healthcare, or intelligent buildings [1]. Based on low-power embedded sensing systems with a limited energy budget and low computational power, WSNs commonly gather information from distributed locations and collect them at devices on the network edge (often referred to as *base stations*, *BS*), for processing. A dedicated family of routing protocols has emerged to accommodate this traffic flow: *Collection protocols*, such as MultiHopLQI [2] or CTP [3]. However, one of the key limitations of data collection WSNs is their concentration of wireless traffic around the base station, leading to a quicker energy depletion of nodes located there. To cater for a more balanced use of energy inside the WSN, in-network processing techniques have been proposed, e.g., distributed outlier detection [4], clustering [5], or aggregation [6]. When executed close to the data collection points, distributed data processing reduces energy demand and wireless traffic alike.

Along with the vision of the upcoming Internet of Things, sensor nodes *(SNs)* can be expected to carry more diverse traffic than traditional WSNs. Moreover, device heterogeneity in terms of computational power, storage capacity, wireless range, or the available energy budget will become omnipresent characteristics. Despite the technological support for advanced in-network processing operations, however, current systems generally lack the option for a seamless data exchange on application level, and do not leverage heterogeneity at all. By presenting CLAP, we lay the foundation for a better utilization of device heterogeneity by allowing heterogeneous SNs to *co-operate* and harness the in-network data processing capabilities of dedicated Processing Devices (*PD*s). In a nutshell, CLAP allows SNs to find PDs in their vicinity, exploit their heterogeneous computational capabilities for in-network processing, and subsequently forward processed data to their destination.

This work builds on a theoretical energy assessment of heterogeneous networks in own prior work [7]. However, we make the following two novel contributions in this paper: (1) We introduce the concept of Cooperative Locality-Aware Data Processing (CLAP), which enables SNs to actively cooperate, and present selected implementation details in Sec. II. (2) We evaluate CLAP and demonstrate its efficacy in two simulated scenarios in Sec. III. We conclude this paper in Sec. IV.

II. Objectives and Approach

Our primary research objective is to assess to what extent the presence of computationally heterogeneous nodes in a WSN deployment can lead to a reduction of communications overhead when decentralized data processing takes place. An example application scenario is shown in Fig. 1. Two data sources (DS_1 and DS_2) are part of a collection tree, rooted at the BS. While DS_1 would normally forward its data via the dashed link towards the sink, the availability of a processing device in its one-hop neighborhood changes this route. Its data are consequently routed to processing device PD_1, where they are processed and only the result is forwarded along the established collection tree. In case of DS_2, the next PD is two hops away. An intermediate node is used as a relay device and configured to forward incoming traffic from DS_2 to PD_2, which in turn uses the underlying collection protocol to forward its processed data to the BS.

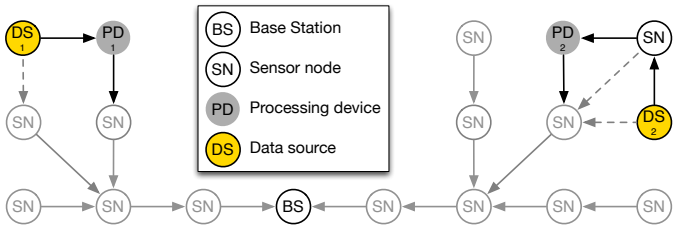


Fig. 1. Two sample application cases for the proposed collaborative data processing approach. Continuous arrows indicate the resultant collection tree, while dashed arrows show the data flow without collaborative data processing.

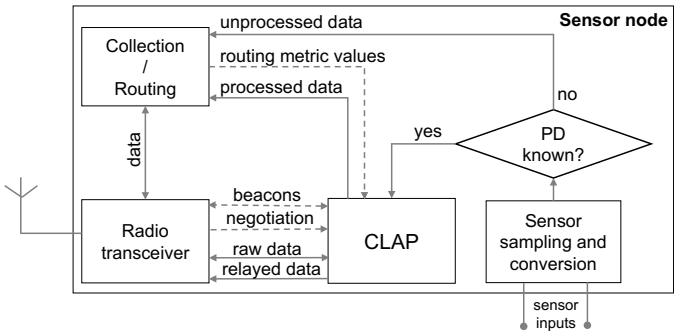


Fig. 2. Sensor nodes' network stack to realize collaborative data processing.

A. Requirements Analysis

To enable heterogeneous devices on the IoT to collaboratively process collected data in their locality, the following fundamental requirements need to be fulfilled:

1) SNs must be able to determine whether any PDs exist within their neighborhood. To avoid excessive deviations from the collection route to the BS and to eliminate the complexity of including an additional routing protocol, CLAP does not consider PDs more than two hops away from a SN.
2) SNs must be able to establish a path to one of the PDs in their vicinity (directly or via an intermediate node) and send their sensor data along this route instead of forwarding them to the BS using the collection tree.

B. Overview of CLAP

A conceptual overview of CLAP's integration with the data flow on a regular sensor node is shown in Fig. 2. If a PD is known and available for collaborative data processing, the CLAP module takes care of data routing and processing. However, if CLAP finds no PD in the node's vicinity, the SN's data flow remains identical to the case of running a collection protocol without CLAP. The functionalities required to accomplish the seamless collaborative in-network processing are detailed in the following subsections.

C. Finding Processing Devices

In order to utilize data processing capabilities of PDs, SNs must become aware of the presence of any PDs in their two-hop neighborhood. CLAP relies on proactive advertising of PDs through beacons, which leads to less announcement traffic in case the number of SNs exceeds the number of PDs in the network. The identification of PDs and their capabilities is accomplished on demand through probing messages to ensure a reliable communication can be established with the PD even when its announcements have been received a while ago.

In CLAP, PDs periodically send advertisement beacons to make their availability known to their neighborhood. Beacons are re-broadcast once by the nodes in the immediate neighborhood of the PD and can thus be assumed to reach all two-hop neighbors. These advertisement beacons contain three fields: the network address of the PD itself, its routing metric value according to the collection protocol (e.g., the ETX to the BS), and the address of a relaying node. The latter field is initially empty, and only populated when the advertisement beacon is re-broadcast by a PD's neighboring device. Knowledge of the PD's routing metric assists in selecting the best PD for the data processing, as detailed in Sec. II-D. However, the routing metric can easily be substituted by other desired quality constraints to improve the PD selection process.

Each node maintains a list of the PDs from which advertisements have been received. Upon reception of an advertisement beacon from a PD or a relaying node, the receiving node updates its list of PDs accordingly. A threshold for the minimum number of beacons received from a PD is being used in order to only enter PDs with stable network links into the list. Stale members are periodically purged from the list.

D. Selecting a Processing Device

In order to choose the processing device for locally collected data, all entries in the PD list are sorted based on their distance to the local device (i.e., hop count) and their routing metric value of the underlying collection protocol. The main reason for using this sorting technique is to ensure that locality is preferred. Relaying nodes to accomplish two-hop transmissions are only used in case no PD exists in the SN's direct neighborhood.

E. Negotiating Processing Data Transfers

Data transfers to a PD are always preceded by a request-reply handshake, initiated by the SN in need of in-network data processing. A request packet is sent to the selected PD, with the network addresses of the data source and the requested PD contained in its payload; if a relaying device is required to transfer data from the source to the PD, its address is also explicitly specified. Besides addressing information, the request message contains the type of the requested processing operation and the temporal duration for which the processing is needed. When a PD receives the request message and is ready to accept the client, it acknowledges the reservation by sending a reply message to the sender. The handshake is completed through a final reply message, confirming the successful resource reservation, which carries the same information fields as the request packet. In case a PD does not respond within a timeout period, the SN proceeds to negotiate collaborative data processing with the next entry in its PD list. Once the negotiations of processing device, duration, and function have been completed, the data transfers can commence.

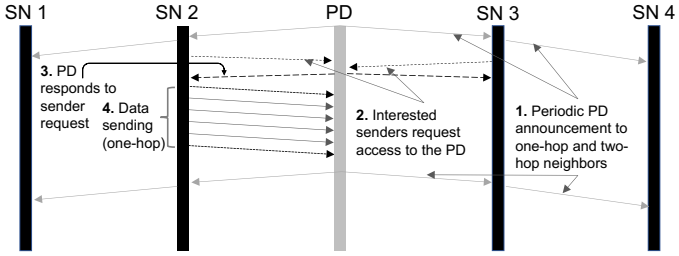
This separate transfer handshake, executed before actual data transmissions, is implemented for three reasons: Firstly, to ensure availability of desired PD to receive and process data. Secondly, to allow the PD to perform computations on the incoming data based on the desired service requested through the handshake messages. Thirdly, to configure an intermediate device (if SN and PD are only within two-hop distance of each other) to relay all following data from the SN to the PD. In case a preferred PD is not ready to process data or cannot offer the requested processing service, the handshake does not complete successfully, and another PD will be approached.

F. Exchange of Application Data

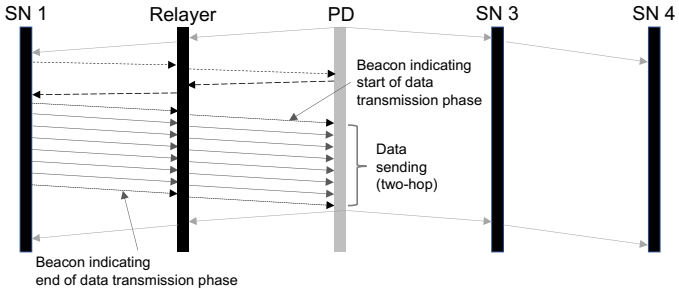
In CLAP, data transmissions between an SN and its selected PD are always preceded by the aforementioned handshake to indicate the start of a data transmission, and terminated by a final delimiting message. By encapsulating data transmissions this way, PDs can schedule the use of their processing resources better. On reception of end-of-transmission packet, the PD performs the requested computation on the data received so far and forwards the resulting packet over the underlying collection tree on behalf of its original sender. CLAP does not explicitly provide flow or congestion control, but is interoperable with transport protocols offering such features.

G. Typical Communication Flow

Figs. 3a and 3b show typical communication flows when running CLAP. At first, the PD's announcements of its processing abilities to its one-hop neighborhood are visible. These messages are then re-broadcast by all immediate neighbors to publicize the PD's existence to its two-hop neighbors. Subsequently, a data processing request is made by SN 2 (in Fig. 3a), or SN 1 (in Fig. 3b), respectively. Depending on the number of simultaneously allowed data processing operations, the PD can individually determine whether to accept a request or not. In the figures, the PD is configured to provide processing services to at most one SN at a time. Consequently, the successful handshake completion message sent to SN 2 in Fig. 3a acts as a trigger to initiate data transmissions, and simultaneously serves as a negative acknowledgment for SN 3; therefore, SN 3 continues sending unprocessed data using the underlying data collection protocol. Data transfer takes place through unicast transmissions (possibly involving a relay node, shown in Fig. 3b) and ends with a delimiter message from the sender.



(a) Typical flow of exchanged messages for one-hop data transmissions.



(b) Typical flow of exchanged messages for two-hop data transmissions.

Fig. 3. Communication flows for one-hop and two-hop collaborative data processing using CLAP.

III. Evaluation

We present our evaluation results after outlining the experimental setup and introducing the network topologies analyzed.

A. Evaluation Setup

Across all of our evaluations, we assume a heterogeneous network setting, in which several SNs and PDs exist, and the BS is the recipient of all data collected on the sensing devices. We use the Collection Tree Protocol (CTP) [3] as the underlying collection protocol because of its reliability and its available implementation for the TinyOS operating system. CTP forms an acyclic tree, based on adaptively updated ETX values, to relay data to the BS. Although the ETX values can be used in CLAP to sort PDs for better performance and reliability, their availability is not mandatory for CLAP's operation.

All following evaluations have been conducted using the COOJA simulator [8], using the *tmote sky* [9] as the SN. The maximum number of clients a data processing device can serve simultaneously has been set to one. This ensures a higher competition for PDs in our analysis.

Heterogeneous processing capabilities in the simulation have been realized by adding a data processing service on a small fraction of the simulated devices. Although the demanded service for data processing by SNs is always a summation of all received values (generated as a 16-bit random payload every 100 ms at every SN), more complex algorithms can be substituted easily. CLAP is agnostic to the choice of processing algorithms offered by PDs, thus we do not investigate different algorithms in this research work.

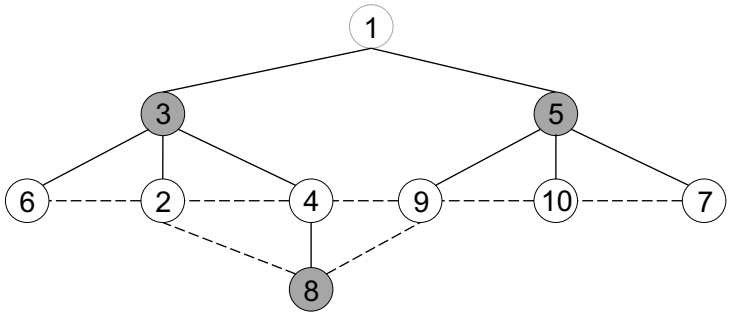
B. Simulated Topologies

We run our simulations in two topologies, shown in Fig. 4, which demonstrate differently expected traffic flows. In both scnearios, the devices with identifiers 3, 5, and 8 are configured as PDs, and node 1 serves as the base station. Continuous lines indicate the tree formed by the collection protocol, whereas dashed lines show additional wireless connectivity. In the tree topology (shown in Fig. 4a), each data generating SN, has at least one PD in its direct neighborhood, and most even find two PDs in their two-hop neighborhoods. In contrast, in the ellipsoid topology in Fig. 4b, each SN has exactly two neighbors in one-hop distance.

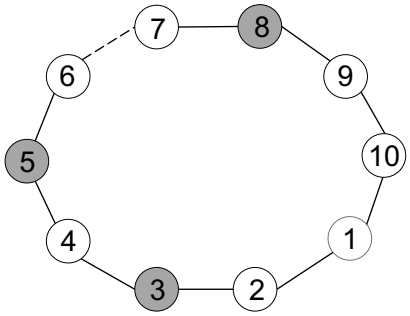
C. Advantages of using CLAP and CTP

Experiments were performed to assess how much messaging overhead can be saved by performing computations on PDs instead of forwarding unprocessed data to the BS. For this, we set up two simulations, one with only CTP is enabled, and another one with both CLAP and CTP enabled. The number of packets received at the base station in 25 minutes of simulation time at different data generation rates is shown in Fig. 5. In all simulations, SNs have been configured to make reservations for data processing operations for durations of 12–20 seconds.

The maximum difference in the number of packets received at the BS can be observed when a high volume of data packets



(a) Tree topology.



(b) Ellipsoid topology.

Fig. 4. Network topologies used in our analyses. The node with identifier 1 always serves as the base station; devices shaded in grey are processing devices.

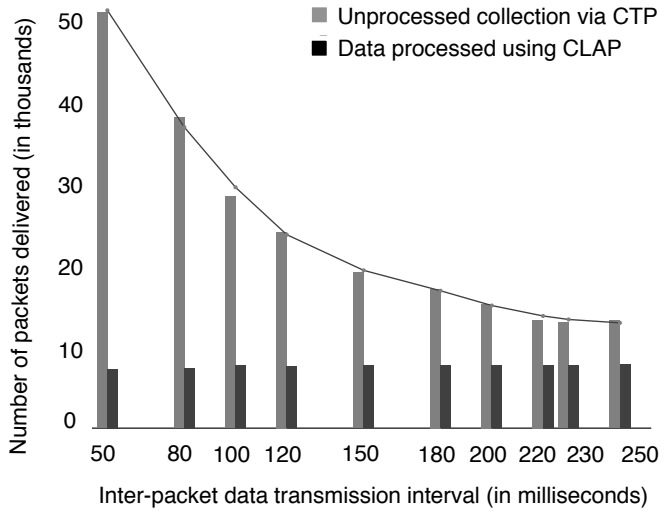


Fig. 5. Comparison of the number of packets delivered to the base station.

are being generated. In fact, more than seven times as many packets are received at the BS when running CTP as compared to the use of CLAP for a 50 ms data transfer interval. While the absolute difference in terms of the number of received packets shrinks with growing inter-packet intervals, collecting unprocessed data via CTP always results in a higher number of messages being delivered to the BS. In contrast, the number of processed data packets is almost constant when CLAP is being used, as collection packets are only emitted whenever a processing operation has terminated. Even though this is the expected result from shifting data processing operations into the network and allowing for their collaborative execution, the reduction of traffic directly translates into energy savings. It thus confirms CLAP's efficacy and its viability to combat energy holes around the BS.

IV. Conclusion

The IoT is growing fast and expected to be composed of billions of heterogeneous systems [10]. However, even when IoT devices use the same wireless communication standards, they suffer from a lack of application-level interoperability. This is especially important in the domain of fog computing, where IoT devices will be interfaced with a broad range of other systems. In this paper we have proposed CLAP as a method to overcome this limitation. CLAP runs on top of any tree-based routing or data collection protocol (such as CTP or RPL) and provides an abstraction from system heterogeneity. Simulations comparing CLAP to a setting without in-network data processing show the high potential of exploiting device heterogeneity to reduce the number of packets delivered to the base station. Even though heterogeneity is often seen as an obstacle, we have shown the potential of exploiting it in data collection sensor network deployments instead of trying to mitigate its effects. CLAP has performed demonstrably well for data aggregation; we consider investigations into other distributed data processing algorithms and the scalability of the algorithm to different topologies as future work.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, 2002.

[2] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "TEP119: Collection," Online: https://github.com/tinyos/tinyos-main/blob/master/doc/txt/tep119.txt, 2006.

[3] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "CTP: An Efficient, Robust, and Reliable Collection Tree Protocol for Wireless Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 10, no. 1, 2013.

[4] J. W. Branch, C. G. amd Boleslaw Szymanski, R. Wolff, and H. Kargupta, "In-network Outlier Detection in Wireless Sensor Networks," *Knowledge and Information Systems*, vol. 34, no. 1, 2013.

[5] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, 2000.

[6] C.-M. Chao and T.-Y. Hsiao, "Design of Structure-free and Energy-balanced Data Aggregation in Wireless Sensor Networks," *Journal of Network and Computer Applications*, vol. 37, 2014.

[7] A. Reinhardt and D. Burgstahler, "Exploiting Platform Heterogeneity in Wireless Sensor Networks by Shifting Resource-Intensive Tasks to Dedicated Processing Nodes," in *Proceedings of the 14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2013.

[8] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN)*, 2006.

[9] Memsic Inc., *TelosB data sheet*. [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

[10] D. Evans, "The Internet of Things – How the Next Evolution of the Internet is Changing Everything," Cisco Internet Business Solutions Group (IBSG) White Paper, 2011.

Data protection in fog computing through monitoring and adaptation

Zoltán Ádám Mann
paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Essen, Germany

***Abstract*—Fog computing has enormous potential to offer increased computational capacity with acceptable latency near the network edge. However, fog computing also introduces many risks relating to the protection of sensitive data, which threaten the practical adoption of the fog computing paradigm. In this paper, the main challenges of data protection in fog computing and potential mitigation techniques are briefly reviewed. We argue that, given the highly dynamic nature of fog computing systems and the negative side-effects of existing data protection techniques, such techniques should be used adaptively, always in accordance with the relevant data protection risks. We sketch an approach to monitor a fog system and activate data protection techniques adaptively, followed by a research agenda to elaborate the details of the proposed approach.**

I. Introduction

Fog computing is the natural next step in the evolution of cloud computing, bringing cloud-like elastic compute capacity to the network edge, near to end user devices [1]. This way, computation-sensitive tasks can be offloaded from the end devices (like mobile phones, wearable devices, or cameras) to fog resources (i.e., compute resources at or near the network edge, e.g., in routers, base stations, or geographically distributed data centers of telecommunication providers). Offloading is advantageous for many applications that require higher computational capacity than what is available in end devices. Compared to offloading compute tasks to a large centralized cloud data center, fog computing has the advantage of considerably lower latency in the data transfers, which is essential for several time-critical applications [2].

Nevertheless, fog computing is also subject to several challenges. In particular, fog computing offers a plethora of opportunities for malicious parties to gain access to, or even manipulate, sensitive information [3]. Some of these threats are inherited from cloud computing, but some are new and specific to fog computing. More importantly, concerns about data protection can significantly hinder the adoption of the fog computing paradigm.

Of course, there are several known security techniques with which the access to sensitive data can be protected. However, the available techniques also have some limitations (e.g., some assume the availability of special hardware) or drawbacks (e.g., overhead). Therefore we argue that security techniques should be applied in an adaptive way. That is, the most appropriate technique should be selected based on the current situation. Adaptations should be carried out at run time, since also the situation may change dynamically at run time. Therefore, the current system state has to be monitored, so that risks concerning data protection can be identified and mitigated on the fly.

The contributions of this paper are as follows:

- A review of data protection challenges in fog computing;
- A proposed framework for adaptive handling of risks relating to data protection;
- Identification of research challenges to realize the proposed concept.

II. Data protection challenges in fog computing

For a more detailed survey of the general field of security and privacy in fog computing, the reader is referred to [4]. Here we only review the most important challenges related to the protection of sensitive information in fog computing.

Just like in cloud computing, users lose control of their data by uploading them to a server that is beyond their control [5]. Thus, the provider operating the given cloud or fog resource may get access to users' confidential data. The provider may also let third parties access the data – intentionally or unintentionally, with or without consent from the user – so that also these third parties may abuse the data. Moreover, because of the intrinsic multi-tenancy of both the cloud computing and fog computing paradigms, other users may also use the same server or fog resource, which might make it possible for those other users to gain unauthorized access to sensitive data. In some cases, it is also possible that users try to get access to confidential data of the provider, for instance to get to know important business secrets about the provider's infrastructure. All these types of attacks are conceivable in both cloud and fog computing.

In addition, there are some specific characteristics of fog computing that make data protection even more challenging than in cloud computing:

- **Reduced physical protection**. While cloud data centers are typically protected by strict physical access control mechanisms (e.g., doors that can be opened only by authorized personnel with their entry cards), fog resources are often deployed "in the wild" where malicious parties can get physical access much more easily. Even more importantly, fog computing is mostly based on wireless networking technologies which may be broken into without physical contact. In contrast, cloud computing

is mostly based on wired networks, which are easier to protect.

- **Less clarity about stakeholders**. In cloud computing, users choose service providers explicitly and deliberately, also giving explicit consent regarding the use of their data. On the other hand, in some fog computing scenarios, a device may use a variety of fog services for offloading computations, without the user of the device – or the data subject about whom the device is collecting data – being aware of the stakeholders that operate those resources or have otherwise access to the resources.
- **Direct access to confidential information**. A device using fog computing resources may leak sensitive personal information even without transferring any data explicitly to the fog resources. For example, since devices prefer to use nearby fog resources, an attacker might be able to determine a user's approximate location or the route of a mobile user based on which fog resources the user's device has connected to, thus violating location privacy [6], [7]. Another example is the violation of usage privacy in smart metering where the information gathered by smart meters reveals usage patterns of electronic devices in a household [8].
- **Scarce resources**. Existing methods for data protection, such as advanced cryptographic protocols or data obfuscation techniques, are often resource-intensive. This, however, is a problem in end devices that have limited computational capacity, limited battery power, and limited network bandwidth.

For these reasons, data protection is a very challenging problem in fog computing.

III. The case for adaptive data protection

Fog computing systems are very dynamic: end devices connect to fog resources and then disconnect, fog resources appear and disappear, wireless network connections get stronger or weaker etc. [9], [10]. With all those changes, also risk levels keep changing. For instance, from the point of view of an end device, risk levels may be low if the device can connect to a known and trusted fog resource, but the risk of data protection issues gets much higher if the connection to the trusted fog resource is lost and an unknown fog resource must be connected instead.

As already mentioned, existing security techniques that can ensure the protection of sensitive data often have downsides. For instance, homomorphic encryption makes it possible to offload computations on encrypted data to an untrusted server. Since the server gets access to the ciphertext only, it cannot abuse the actual data. However, homomorphic encryption introduces a large performance overhead [11]. Another option is the use of secure hardware enclaves, i.e., special hardware enabling the protection of code and data even from attackers with highest operating system privileges [12]. Performing computations in a secure enclave thus shields the data both from co-located applications and even from the operator of the server. However, also the use of secure enclaves incurs some overhead (although lower than in the case of homomorphic encryption) and even more importantly, it presumes the availability of appropriate hardware.

For these reasons, we argue that data protection mechanisms should be applied in an adaptive manner. In other words, data protection mechanisms should be activated only when needed to minimize their negative impact on resource consumption; moreover, from available alternative mechanisms always the most appropriate one should be chosen, taking into account the sensitivity of the data and the current configuration of the fog system.

In the following subsections, we review how adaptive application of data protection mechanisms can be achieved – first in general, and then focusing on the viewpoints of end users and fog service providers, respectively.

A. Enabling adaptations

The fundamental model underlying most adaptive systems is a control loop according to the MAPE (monitor, analyze, plan, execute) principle [13]. This principle can also be applied to the problem of adaptive data protection in fog computing, as follows.

The basis for any decision-making is monitoring. That is, the current configuration of the fog computing system needs to be monitored, including the available resources, the planned computations, the involved data, and any other information that may have an impact on risks (e.g., known vulnerabilities or reputation of stakeholders). Monitoring may be complemented by prediction, e.g. to predict the future availability of wireless network connections or the duration of offloaded tasks [14]. Based on the information provided by monitoring and potentially prediction, an analysis has to be carried out to determine the risks of data protection violation and the possible risk-mitigating actions. The results of the analysis form the input to planning. The aim of planning is to decide which risk-mitigating action(s) to take, based on the impact of the possible actions on both data protection risks and other system goals like performance or costs. Finally, the chosen actions have to be executed.

For implementing the MAPE loop, a model-based approach is advantageous. This means that a model of the fog computing system is maintained at run time in a machine-readable format. Monitoring updates the model so that it remains in sync with reality. Analysis and planning can be performed directly on the model, while execution ensures that modifications performed on the model are also transferred to the real world.

In our previous work, we have proposed a run-time model for reasoning about data protection in cloud systems [15]. This model should be extended and adjusted to capture the necessary entities of fog computing.

B. Adaptation in end devices

In the simplest case, an end device wants to offload some computations to a fog resource. Monitoring and analysis could be used to answer the following questions:

- Can the provider of the fog resource be trusted (e.g., because of previous experience or because of high reputation in publicly available provider evaluations)?
- What security capabilities does the resource offer (e.g., secure hardware enclaves)?
- How sensitive are the data involved in the planned compute task?
- What would be the impact of the available client-side data protection mechanisms, and how critical would that impact be in terms of system goals like performance and energy consumption?

Based on these pieces of information, a sound decision can be made on the action to be taken. For example, if the data are not sensitive and/or the provider is trusted, then the computation can be offloaded without using further data protection mechanisms. Otherwise, if the targeted fog resource features secure hardware enclaves, then the computation should be performed within an enclave. Otherwise, if the resource situation allows it, homomorphic encryption should be used. If none of the above is applicable, then the computation should not be offloaded because the associated risks cannot be effectively mitigated.

There can also be more complicated cases, e.g., if not only an edge device and a fog resource are involved, but additionally a central cloud as well. To keep the analysis and planning manageable, a pattern-based approach can be used, like we suggested previously for cloud computing [16]. In this approach, the system configurations that would lead to unacceptably high risks of data protection violation are captured in the form of so-called risk patterns. If an instance of a risk pattern can be found as a substructure of the run-time model, then the risk is too high. An appropriate adaptation is needed so that the run-time model will not contain any of the identified risk patterns as substructure.

C. Adaptation in fog resources

For a provider of fog resources, the goal is to fulfill the data protection requirements, while serving as many end client devices as possible and also ensuring a smooth and efficient operation [17]. This leads to interesting optimization problems [18]. For example, if a subset of the resources owned by the provider offer secure hardware enclaves, then taking this into account when allocating client requests to resources is a useful lever for keeping costs low while fulfilling data protection requirements. Our previous experience has shown that, with appropriate optimization algorithms, if only a small fraction of resources offer secure hardware enclaves, this can already lead to considerable savings in energy consumption [19].

IV. Research challenges

In this paper, we sketched why adaptive data protection in fog computing is sensible and how it might be achieved. However, to make adaptive data protection in fog computing a reality, several research challenges need to be addressed (the list is not intended to be exhaustive):

- Appropriate models should be devised that incorporate all entities of fog computing deployments (along with their attributes and relations) that are relevant for data protection.
- The underlying trust models and attack models need to be better understood, categorized, formalized, and made available to automated run-time decision-making.
- A catalog of risk patterns needs to be elaborated that capture the relevant risks to data protection in fog computing.
- Appropriate monitoring techniques are necessary to efficiently and effectively monitor fog computing systems.
- Analysis, planning, and optimization algorithms need to be elaborated that can efficiently cope with decision-making on the different layers of fog computing systems (end devices, fog resources, cloud).
- Algorithm efficiency and scalability are crucial to ensure that adaptation works well in real time even under high dynamics.
- Testing, auditing, and verification techniques need to be devised to improve the credibility of data protection solutions in fog computing.
- The interplay among multiple autonomous systems that perform self-adaptations on their own needs to be better understood, including the possible coordination models and emergent behavior.

Moreover, it would be advantageous for fog computing research in general to define and make publicly available some representative examples that can be used to objectively assess and compare different approaches.

Acknowledgment

This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured).

References

[1] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[2] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for IoT services in the fog," in *IEEE 9th International Conference on Service-Oriented Computing and Applications*. IEEE, 2016, pp. 32–39.

[3] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Federated Conference on Computer Science and Information Systems*. IEEE, 2014, pp. 1–8.

[4] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 2015, pp. 685–695.

[5] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, "From security to assurance in the cloud: A survey," *ACM Computing Surveys*, vol. 48, no. 1, 2015, article 2.

[6] Z. Riaz, F. Dürr, and K. Rothermel, "On the privacy of frequently visited user locations," in *17th IEEE International Conference on Mobile Data Management*, vol. 1. IEEE, 2016, pp. 282–291.

[7] T. He, E. N. Ciftcioglu, S. Wang, and K. S. Chan, "Location privacy in mobile edge clouds: A chaff-based approach," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2625–2636, 2017.

[8] A. Reinhardt, F. Englert, and D. Christin, "Averting the privacy risks of smart metering by local data preprocessing," *Pervasive and Mobile Computing*, vol. 16, pp. 171–183, 2015.

[9] A. Aral and I. Brandic, "Quality of service channelling for latency sensitive edge applications," in *IEEE International Conference on Edge Computing*. IEEE, 2017, pp. 166–173.
[10] S. Dräxler, H. Karl, and Z. A. Mann, "Joint optimization of scaling and placement of virtual network services," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2017, pp. 365–370.
[11] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 698–706, 2015.
[12] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security Symposium*, 2016, pp. 857–874.
[13] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[14] G. Orsini, D. Bade, and W. Lamersdorf, "CloudAware: A context-adaptive middleware for mobile edge and cloud computing applications," in *IEEE International Workshops on Foundations and Applications of Self* Systems*. IEEE, 2016, pp. 216–221.
[15] Z. A. Mann, A. Metzger, and S. Schoenen, "Towards a run-time model for data protection in the cloud," in *Modellierung 2018*, I. Schaefer, D. Karagiannis, A. Vogelsang, D. Méndez, and C. Seidl, Eds. Gesellschaft für Informatik e.V., 2018, pp. 71–86.
[16] S. Schoenen, Z. A. Mann, and A. Metzger, "Using risk patterns to identify violations of data protection policies in cloud systems," in *13th International Workshop on Engineering Service-Oriented Applications and Cloud Services*, 2017.
[17] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *IEEE 37th International Conference on Distributed Computing Systems*, 2017, pp. 1281–1290.
[18] Z. A. Mann, *Optimization in computer engineering – Theory and applications*. Scientific Research Publishing, 2011.
[19] Z. A. Mann and A. Metzger, "Optimized cloud deployment of multi-tenant software considering data protection concerns," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 609–618.

Migrating IoT Processing to Fog Gateways

Daniel Happ, Sanjeet Raj Pandey, Vlado Handziski

Technische Universität Berlin, Telecommunication Networks Group (TKN)
{happ, pandey, handziski}@tkn.tu-berlin.de

***Abstract*—Internet-connected sensor devices usually send their data to cloud-based servers for storage, data distribution and processing, although the data is often mainly consumed locally to the source. This creates unnecessary network traffic, increases latency and raises privacy concerns. Fog and edge computing instead propose to migrate some of those functions to the edge of the network. In particular, on premise gateways have the potential to offer more privacy preserving and low-latency local storage and processing capabilities. In this study, we outline our ongoing efforts to combine the benefits of fog and cloud sensor data processing. We present our work-in-progress towards a system that automatically selects the most suitable execution location for processing tasks between cloud and fog. We present a protocol for migration of processing tasks from one system to another without service interruption, and propose a reference architecture. We additionally introduce an analytical cost model that serves as basis for the placement selection and give advice on its parametrization. Finally, we show initial performance results, gathered with an early prototype of the proposed architecture.**

I. Introduction

The Internet of Things (IoT) enables applications that interact with the physical world around us in real-time by embedding sensors, software and network connectivity into physical objects. Additional software components for sensor data distribution, storage, and analytics, are often offered as services by cloud providers [1]. However, cloud computing is not always the most suitable option for offering those services in the IoT context, especially for latency and privacy sensitive applications [2]. Instead of forcing all data through cloud servers that are possibly located far away, fog computing proposes to move some of those services to the edge of the network [3], in our case to existing gateways devices.

We envision the architecture of future IoT systems to have 3 layers, as shown in Figure 1. The device layer consists of sensor and actuator devices, which are severely constrained in terms of processing power, memory and energy. Those ”things” are connected to a nearby gateway which relays their data to the Internet. Fog and cloud layers provide the means to distribute, store and process sensor data. While fog instances are relatively local and thus closer to the end-user, the centralized remote cloud provides ubiquitous and seemingly infinite access to storage and processing.

We expect that all the components use the publish/subscribe pattern to communicate [2], [4]. In this one-to-many pattern, the matching between consumer and producer of data is done by message brokers, which we expect to run on cloud and fog instances. We mainly consider the topic based naming scheme, where the symbolic channel addresses are strings.

We identify gateway devices as a prime candidate for local, low-latency and privacy preserving storage and processing. While sensor devices themselves are very constrained, today's IoT gateway hardware include powerful embedded single-board computers (Raspberry Pis, BeagleBone Blacks, Intel NUCs, home routers, etc.) and smartphones and are powerful enough to take over some of the services the cloud is providing today. Gateways in industrial and home environments are usually mains-powered and not energy constrained. We argue that already deployed gateway hardware has a significant amount of potential processing power and storage available right on premise that is not yet fully utilized. Through leveraging local processing on gateway devices and migrating heavy computation to resourceful cloud servers on demand, the resource constraints of local gateway hardware regarding CPU-, memory-, or network-intensive tasks can be overcome while still benefiting from the locality gateways provide.

In this work, we present our ongoing effort aimed at developing a flexible framework for IoT process relocation. Although the concepts provided here universally apply to process migration in distributed IoT systems using publish/subscribe, we consider the context of gateway to cloud offloading as an example. Our main contributions in this work are threefold: 1) We outline a migration mechanism and a framework for sensor processing tasks. 2) We provide a cost model for processing task migration and give advice on a suitable parametrization. 3) We show preliminary measurements of a research prototype. Those parts of this paper describing the framework and measurements are largely based on already published work [5]. The in-depth analysis of the cost model for task migration has not been published before.

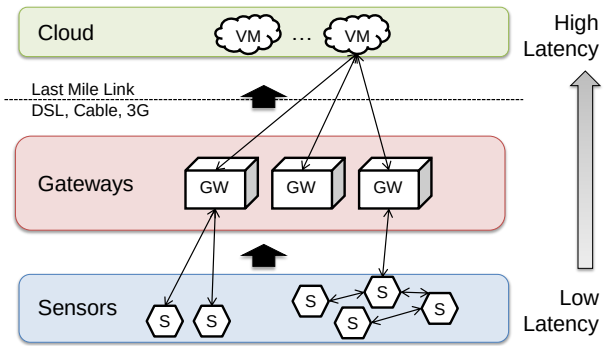


Fig. 1. Overall architecture of a simplified Fog-enabled IoT platform.

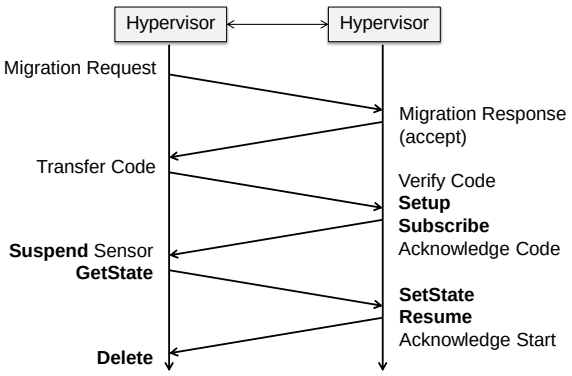


Fig. 2. Migration between two hypervisors.

II. Gateway to Cloud Offloading of Sensor Processing Tasks

In this work, we define a sensor processing task very broadly as an arbitrary computation on a set of input sensor streams that create one or more output streams. Input streams are exclusively obtained by subscribing to a pub/sub system. Likewise, the output is published on one or more output topics. The process can be defined freely by the user; we do not make any assumptions about the function provided.

A process has two parts, the code that defines its function and its state. To enable migration, we need a standard way to snapshot the state and to restore it at the new location. We therefore require each processing task to implement a predefined set of operations, which are given in Table I. The getState method is used to extract the current process state. The setState method is called to set the (initial) state of the task. The state has to be serialize to a byte stream. The specific encoding of the state is up to the task developer, as long as it is serialized to a byte stream.

For starting, suspending and resuming tasks, we use additional operations. Since we do not want to make any assumption on the specific type of task, each one has its own constructor/deconstructor (setup/delete) methods to initialize the process, e.g. with a pub/sub broker. The subscribe method starts to subscribe on every input topic given and fills the relevant queues with sensor data. It does not start processing any data. This is triggered by a call to the resume method, which takes the sequence number of the next value to process. A process is likewise suspended using the suspend call, which returns the next sequence number to process.

The system we envision should dynamically adapt to the current state of each available processing instance and offer migration from cloud to gateway and from gateway to cloud. While not specifically targeted in this work, a gateway to gateway offloading would be possible as well. As a simplified example, we use one gateway and one cloud instance. We envision that the system has a default entry point that stays responsible for the task. In our case, this is the local gateway. Each task is defined by code, initial state and estimates for resource usage. Since different task have different requirements,

TABLE I
Overview of specific processing task methods.

Method	Desciption
setup	called before launch to initialize task
subscribe	called to subscribe to input topics and fill buffers
setState	called to set initial state to representation given
resume	called to start processing
restart	called to restart processing
suspend	called to suspend processing
getState	called to get representation of the process state
delete	called for terminating the processing task

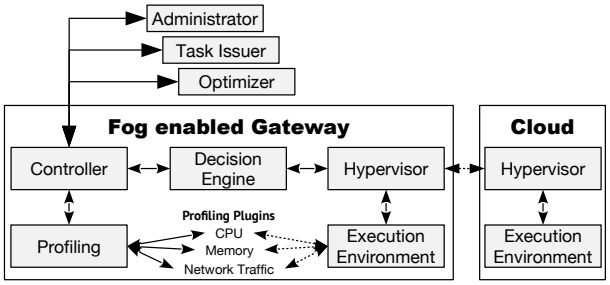


Fig. 3. Simplified gateway to Cloud offloading architecture.

not all tasks are equally suitable for remote execution. The task issuer may thus give additional constraints, e.g. if a task must not be offloaded. The placement of the processing task must comply with the given constraints, as well as the available resources on the gateway. Resource usage is continuously monitored and the corresponding task meta-data updated.

Figure 3 depicts the high-level architecture of the offloading system. Every entity offering execution of processing tasks has a Hypervisor that suspends, resumes and migrates processing tasks. The execution environment is instrumented with profilers that monitor the overall and per task resource usage. The entities that not only support processing of existing tasks but also issuing new processing tasks have a controller that receives the necessary task definitions and relevant metadata. A decision engine decides whether and where to offload the task based on constraints, issued tasks, optimization goal and profiling data.

We further identify three distinct roles to interact with the system: The local administrator defines constraints for the execution environment, e.g. total CPU or memory consumption can be restricted. The task issuer defines the processing function and additional metadata, such as estimates of resource consumption and input and output topics as well as estimated output frequencies. The optimizer sets the optimization goal.

III. Adaptive Offloading Policy

A core part of the presented offloading system is the decision engine, which determines which processes should be offloaded. The problem of assigning processing tasks to processing nodes is related to the knapsack problem and NP-hard. In order to get further insights into the trade-offs of different offloading policies, we formulate the problem of assigning tasks to processing nodes as an optimization problem. We then derive

a greedy strategy and give some pointers for future work in the area.

We assume the simplified case of one fog gateway that can either run processing tasks locally or give them upstream to the next higher layer, which in our case is a cloud backend. We further assume that the cloud can provide infinite resources and the gateway has zero cost for unused resources. We model the system as time discrete with time steps on the creation or destruction of processing tasks and at regular intervals where resource usage is updated and the assignment problem reevaluated. Let us suppose that we have n processing tasks $P_1, P_2, \ldots, P_n$. We further assume a processing task P_i during the time frame t yields an output from the profilers which is characterized as the tuple $\langle cpu_i(t), mem_i(t), rx_{i,loc}(t), rx_{i,rem}(t), tx_{i,loc}(t), tx_{i,rem}(t)\rangle$, where $cpu_i(t)$ is the amount of computing resource required during the time frame (e.g. CPU usage), $mem_i(t)$ is the amount of memory used for the task, $rx_{i,loc}(t)$ and $tx_{i,loc}(t)$ are the number of bytes of traffic received and transmitted locally on the same gateway and $rx_{i,rem}(t)$ and $tx_{i,rem}(t)$ the amount of external traffic to the cloud. When the process is first started, the variables are set to the metadata provided by the task issuer.

We further introduce the decision variable $x_i \in \{0, 1\}$ determining if P_i should be run locally ($x = 0$) or in the remote cloud ($x = 1$). This yields the following function for the monetary cost of the system, which would be one possible optimization goal. The cost function has three parts: The first part describes the bandwidth cost from gateway to cloud. The second and third part are costs for computing power and memory in the cloud respectively. Weight parameters have to be defined according to the cost for CPU, memory and bandwidth of each individual cloud provider. Since we only look at a fixed t, we omit this parameter for clarity:

$$\min_{x_i \in 0,1} (c_{transfer} \cdot \omega_{tr} + c_{memory} \cdot \omega_{mem} + c_{cpu} \cdot \omega_{cpu}) \quad (1)$$

where

$$c_{transfer} = \sum_{i=1}^{n} \quad x_i \qquad (rx_{i,loc} + tx_{i,loc}) \\ +(1 - x_i)(rx_{i,rem} + tx_{i,rem}) \quad (2)$$

$$c_{memory} = \sum_{i=1}^{n} mem_i \cdot x_i \quad (3)$$

$$c_{cpu} = \sum_{i=1}^{n} cpu_i \cdot x_i \quad (4)$$

Additionally, we identify several constraints, namely that processes run locally may in total not exceed a predefined fraction of the available resources:

$$\sum_{i=1}^{n} cpu_i \cdot (1 - x_i) \leq avail_{cpu} \cdot f_{cpu} \quad (5)$$

$$\sum_{i=1}^{n} mem_i \cdot (1 - x_i) \leq avail_{mem} \cdot f_{mem} \quad (6)$$

TABLE II
GOOGLE CUSTOM MACHINE PRICES.

Item	Price ($)	ω
vCPU	0.033174 / vCPU hour	0.210463
Memory	0.004446 / GB hour	0.028207
Bandwidth	0.12 / GB	0.761329

$$\sum_{i=1}^{n} rx_{i,rem} \cdot (1 - x_i) + tx_{i,loc} \cdot x_i \leq avail_{bw\downarrow} * f_{rx} \quad (7)$$

$$\sum_{i=1}^{n} tx_{i,rem} \cdot (1 - x_i) + rx_{i,loc} \cdot x_i \leq avail_{bw\uparrow} * f_{tx} \quad (8)$$

To evaluate different offloading policies, we additionally need to find a suitable parametrization of the aforementioned model, i.e. the weights for the cost parameters. Since these parameters are subjective to the system configuration, we give advice on parametrization with a specific but realistic example. We use prices provided by Google for a "custom" virtual machine types as listed in Table II. We can use those prices to determine the weights ω_{tr}, ω_{mem} and ω_{cpu} by dividing the individual price for one traffic, memory and CPU item by the sum of all prices.

As a first example, this cost function can trivially be transformed into a greedy strategy. First, all processes that were determined to never be offloaded due to provided metadata are copied to the "local" set and resources are allocated accordingly. The algorithm sorts the remaining processes by the specific gain w.r.t. the cost function. It iterates over this sorted list of remaining processes and adds the one with the highest gain to the "local" set and allocates local resources if enough resources are available. That means it effectively determines the process that is saving the most amount of monetary cost when running locally in contrast to running it in the cloud. If that does not yield a possible allocation, the algorithm should stop or reject tasks, e.g. the tasks that where created last.

We plan to use Markov decision processes (MDPs) to determine the solution to our offloading problem in future work. MPDs are discrete time stochastic control processes, which can be used to model decision processes with partly random outcome. Since processing task have event-based input data that shows partly random behavior and random external impact on the system, such as background processing tasks, MPDs seem to be a good fit for the problem at hand. If the state transition function P and the reward function R of an MPD are known, a policy can be computed. Because of the random nature of the amount of input data over a specific timeframe, P and R cannot be expected to be static and known in advance. Our idea is to use reinforcement learning to continuously reestimate the current values for both functions. With this approach, the system would in any state with a relatively low probability "try out" assignments that are expected to give a suboptimal reward to rediscover better solutions in case of an outdated estimate of P and R and update the functions accordingly. As mentioned above, this is part of future studies.

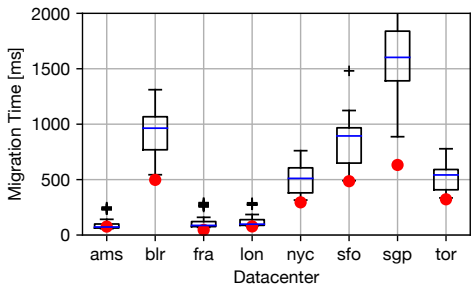


Fig. 4. Migration between two hypervisors.

IV. Evaluating Migration Time

In the previous model we have ignored the cost and time of task migrations. We use a prototype of our system to evaluate this time. The prototype is written in python and uses Message Queue Telemetry Transport (MQTT) as the messaging middleware between all components of the framework and also as the pub/sub system the actual processing tasks uses for input and output data. As a first step, we use the prototype to evaluate the migration time between a gateway node in Berlin, Germany and a cloud-based virtual machine, manifests as a perceived service interruption or lag.

We use a Beaglebone Black as the local gateway and cloud instances in every datacenter of the provider Digital Ocean. We use a processing task that we assume to be realistic to offload, namely a motion detection task that analyses webcam footage. The size of this task is around 3kB. The state of the task is a background image which is used to detect changes and varies in size depending on the image. In our example it is 21kB.

Figure 4 shows the migration times of 64 migrations back and forth along with a red dot marking an estimate calculated from round trip time and throughput measurements, which can be easily done using widely available tools. The migration time is in the worst case under 2.5s even to very remote destinations. Due to the additional overhead and latency introduced by the messaging middleware, the actual average migration time shows to be considerably higher than the prediction, with the estimation being the lower bound of the migration time.

In conclusion, since we do not expect migrations across the globe to happen often, the migration time is small enough to be tolerable by the end-user for most sensor applications; especially for migrations to cloud backends on the same continent, which are well below 200ms. The measurements additionally highlight the need for a prototypical implementation of the system. First, the migration time that is ignored in our model has to be taken into account by future work on the topic. Second, it shows that parameters of an extended model can potentially not be accurately derived from simple estimations but have to be verified by measurements in realistic settings.

V. Related Work

Many mechanisms have been proposed in previous work that address the challenges of seemless offloading, particularly mobile offloading from phones to infrastructure. In the IoT context, existing work is still limited. The authors of [6] describe an integrated fog cloud IoT (IFCIoT) architectural paradigm, including application, analytics, virtualization, reconfiguration, and hardware layer. In [7], the authors propose a virtual machine migration mechanism for fog computing. Aazam and Huh [8] present a resource estimation and pricing model for fog-based IoT, including resource prediction, estimation, reservation, and pricing. We complement this existing work by focusing on the specific context of processing tasks using publish/subscribe. Additionally, previous work often lacks the notion of state, which we deem necessary.

VI. Future Work and Conclusion

In this work, we argue that the full potential of globally interconnected sensor technology will only be fully utilized through increased local sensor data processing. To realize this vision, we present an offloading mechanism and an initial architecture of a generic offloading framework which enables flexible definition of processing tasks as well as constraints and optimization targets. We provide a formal definition of the offloading problem, give advice on the parametrization and give hints on potential solutions. We highlight the need for a prototypical implementation of such a framework and show that processing tasks found in IoT systems can be migrated in a reasonable short time. We further plan to expand our platform to not only take into account the available resources, but to optimize the offloading decision with regard to a wider range of performance targets, such as reduced latency, minimization of energy consumption, or reduced network traffic. We also plan to expand the model and the evaluation in terms of migration time and cost and more realistic use cases.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '15)*. Santa Clara, CA: USENIX Association, Jul. 2015.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[4] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT Platform Requirements with Open Pub/Sub Solutions," *Annals of Telecommunications*, vol. 72, no. 1, pp. 41–52, 2017.

[5] D. Happ and A. Wolisz, "Towards gateway to cloud offloading in iot publish/subscribe systems," in *Second International Conference on Fog and Mobile Edge Computing, FMEC 2017, Valencia, Spain, May 8-11, 2017*, 2017, pp. 101–106.

[6] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot architectural paradigm for future internet of things," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1701.08474

[7] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Nov 2015, pp. 1–8.

[8] M. Aazam and E. N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 687–694.

TableVisor 2.0: Hardware-independent Multi Table Processing

Stefan Geissler*, Thomas Zinner*
*University of Wuerzburg, Germany
Email: {stefan.geissler|zinner}@informatik.uni-wuerzburg.de

I. Introduction

The fog computing paradigm aims to extend the concept of centralized cloud environments by creating a geographically widespread network of distributed compute and network resources at the network edge. The core defining characteristics of fog computing are, among others, low latency and location awareness and a very large number of heterogeneous nodes [1]. The concept thereby promises the flexibility and efficiency of cloud computing while achieving far smaller latencies through geographical distribution of compute and network resources. In order to utilize this concept of highly distributed compute, storage and network nodes most efficiently, the fog computing paradigm leverages the concept of programmable networks by designing edge cloud applications against a standardized network and compute API to realize application specific edge cloud networks, e.g. CloudNets [6] or the Network Cloud [9]. The implementation of this type of dynamically programmable network, however, relies strongly on the concepts of software defined networks (SDN) and network function virtualization (NFV). The application of these paradigms to the network edge consisting of a large number of devices poses several challenges regarding, for example, network function placement [11], network or application performance [17] or device heterogeneity [16]. One of these challenges, the heterogeneity of devices, is discussed in this paper.

The original idea of Software Defined Networking is to control a network consisting of COTS (Commercial Off the Shelf) hardware switches by programming their forwarding behavior using a standardized configuration protocol. To this end, the SDN paradigm aims to separate the control plane from the forwarding plane by moving control elements from the switching hardware towards a dedicated controller. This separation of control and data plane allows a clear separation of concerns among networking components as the controller is only responsible for network management and configuration while the switches handle the forwarding of data plane traffic. This approach not only simplifies network management and configuration tasks, but also enables new and use cases. In the context of fog computing, this approach to network configuration is well suited to provide the means to realize the edge cloud platform required in fog computing [14]. In order to provide the edge cloud platform as a *paas* (platform-as-a-service), the underlying infrastructure needs to be designed with flexibility in mind. Since fog computing environments are expected to consist of a multitude of different, mostly short living, applications, the requirements regarding, for example, latency or bandwidth, are also expected to be diverse. In addition, the hardware infrastructre deployed in these geographically distributed networks will consist of heterogenous devices. This is where the programmability promised by the SDN, and lately the NVF (network function virtualization) concept comes into play. The SDN paradigm, and its accompanying split of control and data plane, promises to enable centralized configuration using standardized configuration protocols like OpenFlow [13]. However, this vision of a unified interface towards the data plane is not always held up in practice. SDN-enabled switches are often shipped with different hardware capabilities and configurations, e.g., with a varying number of flow tables or support for only certain data plane features [4], [12], [5], [10]. Moreover, most SDN switches have highly limited hardware accelerated flow tables.

At the same time, many control plane applications depend on specific data plane capabilities like pushing new headers for VLAN or MPLS. Therefore, missing hardware features, hardware heterogeneity or storage limitations are growing problems for software defined networks. This can be characterized as a *general mismatch* between control plane requirements and data plane capabilities.

The gravity of this problem may in the future be reduced by the deployment of switches with programmable data plane capabilities (e.g. P4 enabled hardware [3]), but can not be solved today, as such switching hardware will not be broadly available for some time. To alleviate this issue, we previously proposed TableVisor [7], [8], a system that addresses the mismatch between control and data plane by aggregating, potentially heterogenous, switching hardware to form a single emulated switch which provides all the capabilities of the underlying hardware devices. Current limitations of TableVisor do not allow the preservation of OpenFlow metadata between physical switches or support for arbitrarily large flow tables.

Hence, in this work we summarize the extensions of the original TableVisor software. The *metadata* feature allows to maintain OpenFlow metadata between switches of the emulated device. The main challenge here is to somehow attach the metadata to a network packet so that the data remains available for all switches involved in the emulated device. The *table extension* feature allows emulation of a single, large flow table by combining the table space of multiple hardware devices. The key challenge here is to ensure consistency while pushing

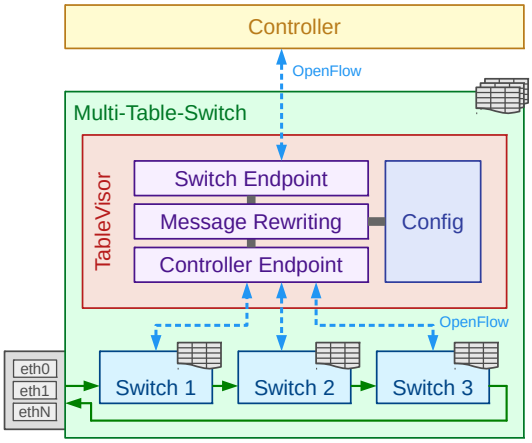


Fig. 1: Logical Layers of TableVisor.

flow rules to multiple switches and to enable the update of flow rules that are distributed among multiple devices.

II. Background

A. Multi Table Processing

The Multi Table Processing (MTP) functionality was introduced into the OpenFlow 1.1 standard and allows switching hardware to perform multiple actions by combining the capabilities of multiple flow tables of a single hardware device. MTP is well suited to alleviate the problem of flow table explosion inherent to various types of SDN applications. This problem commonly occurs as many use cases (e.g. MAC address learning) require independent matches of source and destination addresse and thus require a quadratic number of flow rules to solve a problem that can, using MTP, be solved using a linear number of flow rules.

B. Programmable Data Plane

The Programmable Data Plane (PDP) approach is another relevant trend, not only in the SDN domain, but also regarding the realization of an edge cloud platform infrastructure. The basic idea is to design devices that provide fully customizable packet processing pipelines with hardware accelleration. Common approaches in this direction are P4 [2] or Domino [15]. These approaches enable further use cases in both centralized and distributed cloud environments by providing ASIC-like performance in fully programmable devices.

III. TableVisor 2.0

A. Architecture

Figure 1 shows the logical architecture of the TableVisor software. The TableVisor proxy layer, sitting between the controller and the deployed switching hardware, emulates a multi table switch towards the controller while acting as a regular OpenFlow controller towards the underlying switching hardware.

The **Switch Endpoint** is responsible for establishing a connection to the controller. This layer allows TableVisor to be completely transparent towards the controller and act like a single OpenFlow switch regardless of the number of actual hardware switches connected.

Message Rewriting handles all OpenFlow messages coming from either the controller or one of the hardware switches. Thereby, the content of the messages is adapted in such a way, that regular OpenFlow messages sent by the controller can be distributed to the underlying switching hardware.

Finally, the **Controller Endpoint** allows switches to connect to TableVisor as they would to an OpenFlow controller.

In [7] we presented the *pipeline processing* feature of TableVisor by realizing an MPLS label edge router through the combination of different OpenFlow switches. The core idea behind pipeline processing is to emulate a single multi table device whichs capabilities are only limited by the functionality provided by the involved hardware switches. This provides very high flexibility regarding the capabilities of emulated devices as new capabilities can be added or removed by simply altering the switches involved in the pipeline.

This section describes the two extensions of the TableVisor software presented in this work.

B. OpenFlow Metadata

The first extension described in this work is the new *metadata* feature. This allows to maintain OpenFlow packet metadata between hardware switches of the emulated device. Usually, the meta information is lost as soon as a packet leaves an OpenFlow switch. The transmission of meta information between hardware switches is solved by exploiting packet header fields that are not required for the functionality of TableVisor. Hence, the metadata is not stored in a switch specific register, like during normal OpenFlow operation, but in a packet header field that is not used for the current TableVisor use case. Currently, MAC address fields as well as the VLAN tag can be used to store metadata. This, on the one hand, increases the flexbility of the TableVisor pipeline processing feature, as packets can be handled based on different metadata tags. On the other hand, this feature is required for the *table extension* functionality presented in the following section.

C. Hardware Table Extension

The second feature provided by the current implementation is the *extension of hardware tables*. Since the required TCAM (Ternary Content-Addressable Memory) storage is very complicated and expensive to build, hardware manufacturers limit the provided size in order to provide appliances at competitive prices. Since the flexibility of SDN heavily depends on the number of available hardware flow rules, this is a severe limitation for software defined networks [10].

To overcome this issue, TableVisor supports the combination of multiple hardware switches to emulate a single, large hardware flow table. Thereby, all flow rules of a certain priority are distributed among all switches while higher or lower priority flow rules are stored in the first or last switch of the aggregate respectively. In order to allow for consistency during updates, the switch to hold a flow rule is thereby determined through hashing of the flow mod message. To ensure

that packets are not matched multiple times, the previously described *metadata* feature is used to tag matched packets which are then ignored by all later switches of the emulated flow table.

IV. Conclusion & Outlook

This work presents two extensions to the functionality of TableVisor, a transparent proxy layer software tool that allows pipeline processing and the extension of hardware flow table sizes by using multiple hardware switches.

The goal of TableVisor is to overcome the mismatch between data plane capabilities and control plane requirements present in current software defined networks. Especially in the context of fog computing, these requirements are highly heterogeneous and call for highly flexible solutions to realize the edge computing infrastructure.

In the future it will be interesting to see how it is possible to apply the TableVisor concept with P4 enabled devices. Challenges in this context will be how to compile P4 code into an emulated switch and how to translate P4 code down to involved pipeline switches.

References

[1] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[5] C. Jasson Casey, Andrew Sutton, and Alex Sprintson. tinynbi: Distilling an api from essential openflow abstractions. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 37–42, New York, NY, USA, 2014. ACM.

[6] Anja Feldmann, Manfred Hauswirth, and Volker Markl. Enabling wide area data analytics with collaborative distributed processing pipelines (cdpps). In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1915–1918. IEEE, 2017.

[7] Steffen Gebert, Michael Jarschel, Stefan Herrnleben, Thomas Zinner, and Phuoc Tran-Gia. Table visor: An emulation layer for multi-table open flow switches. In *2015 Fourth European Workshop on Software Defined Networks*, pages 117–118, Sept 2015.

[8] Stefan Geissler, Stefan Herrnleben, Robert Bauer, Steffen Gebert, Thomas Zinner, and Michael Jarschel. Tablevisor 2.0: Towards full-featured, scalable and hardware-independent multi table processing. In *Network Softwarization (NetSoft), 2017 IEEE Conference on*, pages 1–8. IEEE, 2017.

[9] Marco Hoffmann, Michael Jarschel, Rastin Pries, Peter Schneider, Admela Jukan, Wolfgang Bziuk, Steffen Gebert, Thomas Zinner, and Phuoc Tran-Gia. Sdn and nfv as enabler for the distributed network cloud. *Mobile Networks and Applications*, pages 1–8, 2017.

[10] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. *What You Need to Know About SDN Flow Tables*, pages 347–359. Springer International Publishing, Cham, 2015.

[11] Stanislav Lange, Alexej Grigorjew, Thomas Zinner, Phuoc Tran-Gia, and Michael Jarschel. A multi-objective heuristic for the optimization of virtual network function chain placement. In *29th International Teletraffic Congress (ITC)*, 9 2017.

[12] Aggelos Lazaris, Daniel Tahara, Xin Huang, Erran Li, Andreas Voellmy, Y. Richard Yang, and Minlan Yu. Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 199–212, New York, NY, USA, 2014. ACM.

[13] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[14] ETSI Industry Specification Group (ISG) Mobile Edge Computing (MEC). Etsi gs mec 003 v1.1.1: Mobile edge computing (mec); framework and reference architecture. 2016.

[15] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 15–28. ACM, 2016.

[16] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1–8. IEEE, 2014.

[17] Thomas Zinner, Stefan Geissler, Stanislav Lange, Steffen Gebert, Michael Seufert, and Phuoc Tran-Gia. A discrete-time model for optimizing the processing time of virtualized network functions. *Computer Networks*, 125:4–14, 2017.