Transparent Access to 5G Edge Computing Services

Josef Hammer, Philipp Moll, Hermann Hellwagner
Institute of Information Technology
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria
{firstname}.{lastname}@aau.at

***Abstract*—The upcoming 5G telecommunication system is expected to provide high data rates and ultra-low latency to meet the challenging demands of future applications. *Multi-access Edge Computing (MEC)* is a central piece of the solution by providing a cloud computing platform at the edge of the radio access network. In this paper, we argue that edge computing should be invisible to clients and should not require modifications in client applications. A prototype shows how requests to cloud services can be transparently redirected to the closest edge computing host by leveraging Software-Defined Networking, while still allowing to use the cloud as a fallback. In addition, we discuss how to scale our approach for large networks.**

***Index Terms*—5G, Multi-Access Edge Computing, MEC, SDN, Software-Defined Networking**

I. Introduction

Future applications for UAVs, driverless cars, the Industrial Internet, and the Internet of Things (IoT) will demand high bandwidth, high resilience, and/or low latency communication. To meet these requirements, the upcoming 5G communication system will not only facilitate an advanced air interface but also introduce new concepts in the core network – Software-Defined Networking (SDN), Network Function Virtualization (NFV), and, most prominently, *Multi-access Edge Computing (MEC)* [1]. MEC is similar to cloud computing, but happens at the edge of the network, i.e., near the base stations (in 5G terms: gNB; next generation NodeB), and thus is much closer to the client than a conventional cloud server (fig. 1).

The basic idea of distributing processing power at the network edge is also known as edge clouds, cloudlets, and fog computing [2], [3]. Edge Computing is part of a paradigm shift from the current service-agnostic store-and-forward approach towards an intelligent network. Running applications at the edge of the network enables low-latency services but comes at the cost of complexity. In contrast to applications in the cloud,

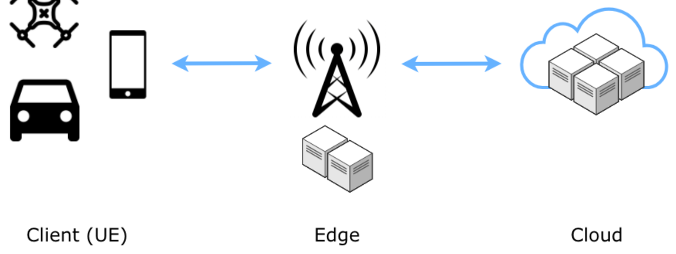

Fig. 1. Edge computing happens near the base station (and the client).

edge applications require advanced location and migration techniques that facilitate both quick and frequent migrations from one edge node to another to support client, i.e., user equipment (UE), mobility. Furthermore, to allow for IoT devices with low processing power, the required intelligence and overhead on the client side must be minimized. Consequently, the intelligence should be moved into the network itself.

The aim of our work is to make edge computing transparent for the UE. The motivation for this goal is twofold: (i) simplify the development of applications that use edge computing and (ii) allow existing applications to use edge computing without any modifications. Just as today's Content Delivery Networks (CDNs) do not require changes to client-side applications such as Internet browsers, edge computing should be invisible to clients. Moreover, the closest edge computing node should be found without querying centralized servers that have the knowledge about the infrastructure.

As an example, consider proprietary sensor networks and surveillance cameras. Instead of uploading all their data to a central server, a much better solution might be to aggregate and filter the data at the edge already. However, those devices might have been built before the era of edge computing, and firmware/software updates might be unreasonable or even impossible. Nevertheless, we want them to benefit from future edge processing capabilities.

Our core idea is to register existing domain names (together with a service port) with mobile edge platform providers. Our system then transparently redirects requests to these cloud services to the corresponding services running in the local edge hosts. If a cloud service is not registered with a given edge platform provider, the original cloud service would be used as a fallback. A prototype was implemented as a module for the OpenFlow POX SDN controller[1], running in a Mininet[2] testbed. Additionally, we present several ideas to improve the performance and scalability of such a system.

II. Related Work

Currently, a lot of research is being conducted to solve the challenges mentioned above. However, so far there are few clear answers and very little is standardized. One area that is particularly lacking is how to address edge services. There is no efficient, standardized naming system for locating edge-computing devices and applications running on them [4]. Such

[1]https://github.com/noxrepo/pox/, last accessed: 2019-03-19
[2]http://mininet.org/, last accessed: 2019-03-19

a naming system would have to handle device mobility, a highly dynamic network topology, privacy and security, while also enabling scalability [5]. Solutions focusing on IoT often follow a centralized approach. Popular frameworks in this area are AWS IoT Greengrass[3], Google Cloud IoT[4], and Microsoft Azure IoT Edge[5]. These solutions have in common that the developers need to provide their own edge computing nodes, as opposed to utilizing the edge nodes provided by the MEC systems of 5G network operators. In general, most proposals favor a solution using SDN and NFV to dynamically route and provision edge services [6]–[8]; however, standardizations are still missing, and concrete implementations are rare.

In [9] Taleb et al. implement their proposed *Follow-me Cloud* using OpenFlow. Recently, in [10] they adapted their system to a *Follow Me Edge-Cloud* that works in the proposed ETSI MEC environment [6]. The goal of the underlying concept is to provide a continuous connection even in case either the user or the service migrates to a different location. This is quite a different and more ambitious goal than ours, and as such requires a more complex system and more routing entries in the OpenFlow tables, making it harder to scale.

Schiller et al. developed CDS-MEC [11], an NFV/SDN-based MEC platform, which provides both edge application provisioning and traffic management. It also focuses on the ETSI MEC Reference Architecture [6] and relies on OpenFlow to redirect UE requests to a nearby edge computing host. Their work primarily addresses LTE; however, the concept should work for 5G as well. While their platform would also allow for transparent edge computing services, it permits a diverse set of redirection rules to be set up by the user; thus, our main goal was not addressed explicitly. We do not target a specific user – it is all about filtering specific cloud services. Furthermore, their focus was on building a MEC platform, and as such it does not address UE migration yet.

In addition to generic OpenFlow switching (as in our proposal), a full solution for LTE is provided in [11], including decapsulating/encapsulating the GPRS Tunneling Protocol packets between the eNB and the Evolved Packet Core (which is not possible yet with today's OpenFlow switches). Another difference is our full integration into an open source SDN controller (POX), which allows for on-demand rule generation and fine-grained timeouts to limit the number of concurrent rules in the switches (see section III). As another benefit, our controller can learn the necessary switch ports automatically, reducing manual configuration.

III. Transparent Edge Services

Our vision is that developers are able to develop edge services as easily as conventional cloud services and do not need to distinguish between developing a cloud or edge service. The parts that shall be executed in the edge can be specified later, either manually, or automatically by applying

[3]https://aws.amazon.com/greengrass/, last accessed: 2019-03-19

[4]https://cloud.google.com/solutions/iot/, last accessed: 2019-03-19

[5]https://azure.microsoft.com/services/iot-edge/, last accessed: 2019-03-19

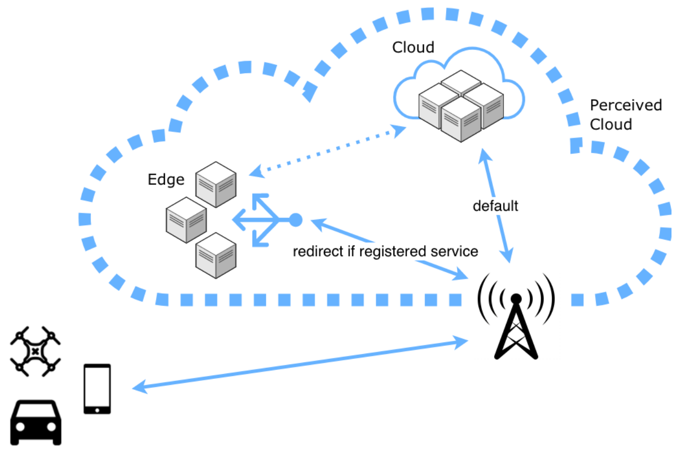

Fig. 2. Perceived cloud vs. real cloud.

machine learning models, or on-the-fly by observing the current network utilization.

Our proposed solution addresses this goal by using SDN. The basic principle is that SDN-enabled switches in the 5G network redirect user requests for registered services (identified by domain name and port) to the closest (as measured by network latency) edge computing node hosting the requested service. The procedure (visualized in fig. 2) works as follows: First, the client requests a service as it would request a conventional cloud service. Then the network intercepts the request and redirects it to the closest available edge node. Finally, the edge node processes the request and responds to the client. As the interception happens automatically by SDN switches in the network, it looks to clients as if the response came from the addressed cloud instance. Note that the edge application itself may utilize the cloud to process the request.

To achieve transparency towards the UE, our solution uses the packet filtering and rewriting capabilities of OpenFlow [12]. Fig. 3 visualizes how our approach makes use of these redirection functionalities. First, the gNB switch matches an incoming packet against the active rules in its flow tables. If a matching flow entry is found, the packet is forwarded accordingly. In case of a miss, the packet is forwarded to the SDN controller as a `PacketIn` message.

The SDN controller (keeping track of the domain-to-IP mappings) compares the destination IP/port combination of the `PacketIn` message against the list of registered edge services. If the request is intended for one of those services, the controller assigns the UE to the closest edge host providing the requested edge application. If no matching entry is found, the packet needs to be forwarded to the cloud according to the default switching rules. In both cases the controller returns a `FlowMod` message containing the `Redirect` instructions for the switch. Additionally, the message contains an *idle timeout* value which causes the redirect entry to be invalidated when no matching packet in the given period is received. This improves the forwarding performance by keeping forwarding tables lean.

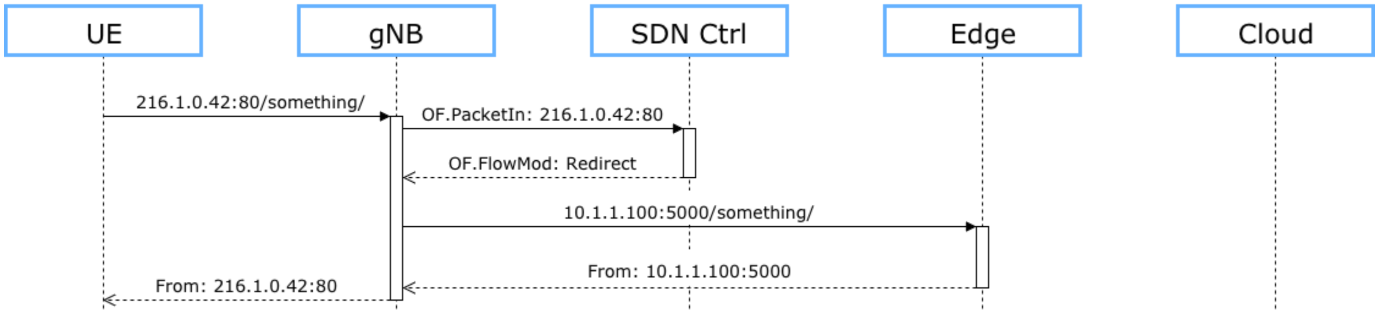

Fig. 3. Routing with a registered service IP: The request is redirected to the closest edge server; transparent to the client (UE). For subsequent requests to the same service, the redirection rule is already known to the gNB; the packet is forwarded directly to the edge host (and not to the controller anymore).

IV. UE Migration

When users move and connect to different base stations (gNBs), the network repeatedly needs to determine new best edge hosts for the given edge application. Fig. 4 shows what happens when a UE migrates from one gNB to another. In this case, the local SDN switch would have no flow entry for the newly arrived UE yet, and thus forward the packet to the SDN controller. Since the SDN controller tracks the attachment point of each UE, it immediately detects the migration and updates the current location of the device.

Depending on the requirements of the edge service, the SDN controller can proceed in case of a UE migration in one of the following ways. It could decide to connect the UE to the previous edge application instance, to migrate the UE to a service running in an edge host located in closer proximity (as measured by network latency), or to use any other host selection strategy based on entirely different criteria. In our prototype, we assume stateless edge applications, and thus forward the UE to the closest edge host that provides the required service. As a result, the request would always be routed such that the lowest possible latency can be achieved. Note that further host selection algorithms using different criteria might be used.

While the concept of transparent edge computing could also be achieved by intercepting and manipulating the corresponding DNS requests, a DNS-based approach fails in combination with UE migration. With each such migration, the UE would have to issue new DNS requests to find out the new optimal edge host. So, while DNS offers the desired separation of service identification and service location, it is usually not used to provide continual updates of the current location of a service.

V. Known Limitations of Our Prototype

While the functional range of edge services is not restricted by our prototype, restrictions for the used transport layer protocol do exist. When a UE needs to be assigned to a different edge host, open TCP connections cannot be migrated. A new connection between the UE and the new edge host has to be established. As a result, the prototype works best for UDP and short-lived TCP connections. We also did not investigate how to deal with secure communication (TLS). Similar to existing approaches used by CDNs, a (suboptimal) solution could be to have the edge application maintain a copy of the private key. We did also not yet integrate a process for registering the cloud services, which shall be redirected to edge applications, with the edge platform operator.

VI. Performance and Scaling Optimizations

Considering that 5G is targeted to support one million IoT devices/km^2, excellent performance and scalability are crucial. Regarding these aspects, we must pay attention to two potential bottlenecks: (i) the number of flow entries per SDN switch and (ii) the number of UEs per SDN controller. To address the number of flow entries per switch, the easiest solution would be to distribute the UEs evenly among multiple switches per gNB. This would immediately reduce the OpenFlow rules belonging to a switch to a fraction of the overall rule set.

Another approach would be to distinguish between switches that process generic traffic and others that process edge-related traffic. As shown in fig. 5, this would lead to two-stage filtering. At the first stage, we could only check whether the destination IP/port combination matches a registered service. If no matching entry is found, the packet is forwarded according to the default switching rules. If, however, the packet matches a registered cloud service at the first stage, it is forwarded to a second switch dedicated solely to edge processing. Only the latter contains all active edge application sessions attached to this gNB. In case of a match (i.e., there exists a rule with the current UE IP and port), the UE request is forwarded to its corresponding edge host and port. In case of a miss, the packet is forwarded to the SDN controller as a `PacketIn` message. The controller then assigns the UE to the closest edge host which provides the given edge application.

As an additional benefit of the separation between regular and edge switches, they could be assigned to different SDN controllers. This would allow better scaling of the latter and thus help mitigate the second bottleneck. The main network controller would only need to handle the separation between regular and edge traffic, while the edge controller would be responsible for assigning UEs to a specific edge. Since the main controller would only handle the separation between regular and edge-related traffic (rules that change only occasionally), instead of using timeouts we could permanently install the flow

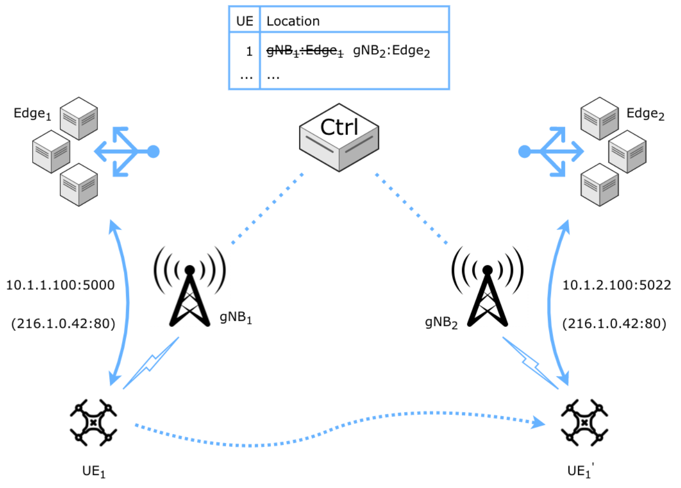

Fig. 4. UE migration (tracked by SDN controller).

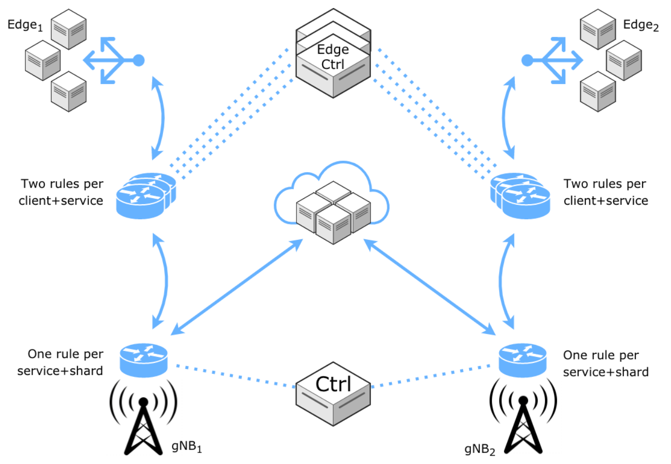
Edge$_1$
Edge Ctrl
Edge$_2$
Two rules per client+service
Two rules per client+service
One rule per service+shard
One rule per service+shard
Ctrl
gNB$_1$
gNB$_2$

Fig. 5. Two-stage filter and sharding of UEs to improve scalability. UE and migration tracking take place in edge SDN controllers only.

entries for registered services in the switches. Outdated rules could be deleted on demand only, e.g., when the DNS lookup reveals a new IP address for one of the registered services.

For additional scalability, the main controller could make use of sharding to distribute the edge connections evenly among multiple edge switches. Those edge switches, in turn, could be assigned to separate edge controllers, further distributing the load (as shown in fig. 5). The distribution could be based, e.g., on the IP address or the MAC address of the UE. Please note that this needs to be based on an attribute that does not change on migration, since a UE must be assigned to exactly one controller to be able to track the current location.

If, however, the goal is to only speed up the filtering without increasing the number of switches, another improved approach could make use of two flow tables (instead of only one) within a switch. The OpenFlow specification [12] allows for multiple flow tables which can be used to match packets in several steps. As in the approach mentioned above, we could do the matching in the same two stages; however, this time within a single switch. Instead of forwarding a packet designated for an edge application to another switch, the rule would continue processing in a follow-up table. This architecture would be flexible enough to allow to start with a single controller and distribute the load to multiple controllers when the need arises.

VII. Conclusion and Future Work

Compared to similar solutions mentioned in section II, our prototype requires fewer flow entries in the switches, leading to better performance and scalability. The current implementation and Mininet testbed provide a solid basis for further research and development. Presently, we are working on a flexible version that allows getting a quantitative comparison of the different scaling optimizations. Our plan is to improve the prototype and to deploy it on a physical testbed.

Our transparent edge service solution reveals an interesting side benefit: If the edge application provider does not have an agreement with an edge platform provider, and thus the service domain has not been registered in the given network, the UE request would be delivered to the targeted cloud service. While this does not have the intended benefits of edge computing, it might be a viable solution in many cases. Furthermore, it allows the cloud service provider to follow the lean startup method and track user demand first and deploy edge applications later or only in networks and regions with enough demand. In general, this incremental deployment approach has proven to be a prerequisite for successful adoption of innovative technologies. By limiting the necessary changes to one edge network at a time, edge application providers and network operators gain a lot of flexibility and freedom.

References

[1] ETSI, “MEC in 5G networks,” *ETSI White Pap. No. 28*, 2018.

[2] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey,” *CoRR*, vol. abs/1808.0, 2018.

[3] P. Mach and Z. Becvar, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things J.*, vol. 3, no. 5, 2016.

[5] W. Shi and S. Dustdar, “The Promise of Edge Computing,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[6] ETSI, “GS MEC 003 - V2.1.1 - Multi-access Edge Computing (MEC); Framework and Reference Architecture,” ETSI, Tech. Rep., 2019.

[7] ——, “GR MEC 018 - V1.1.1 - Mobile Edge Computing (MEC); End to End Mobility Aspects,” ETSI, Tech. Rep., 2017.

[8] B. Blanco, J. O. Fajardo, I. Giannoulakis, E. Kafetzakis, S. Peng, J. Pérez-Romero, I. Trajkovska, P. S. Khodashenas, L. Goratti, M. Paolino, E. Sfakianakis, F. Liberal, and G. Xilouris, “Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN,” *Comput. Stand. Interfaces*, vol. 54, pp. 216–228, 2017.

[9] T. Taleb, P. Hasselmeyer, and F. G. Mir, “Follow-me cloud: An OpenFlow-based implementation,” *Proc. GreenCom-iThings-CPSCom*, pp. 240–245, 2013.

[10] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, “On Enabling 5G Automotive Systems Using Follow Me Edge-Cloud Concept,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5302–5316, 2018.

[11] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun, “CDS-MEC: NFV/SDN-based Application Management for MEC in 5G Systems,” *Comput. Networks*, vol. 135, pp. 96–107, 2018.

[12] Open Networking Foundation, “OpenFlow Switch Specification v1.5.1,” Open Networking Foundation, Tech. Rep., 2015.