Reliable Server Pooling Based Workload Offloading with Mobile Edge Computing: A Proof-of-Concept*

Thomas Dreibholz[a], Somnath Mazumdar[b]

Abstract In recent times, mobile broadband devices have become almost ubiquitous. However, battery-powered devices (such as smartphones), have limitations on energy consumption, computation power and storage space. Cloud computing and Mobile Edge Computing (MEC) can provide low-latency compute and storage services at the vicinity of the user, MEC in particular due to the upcoming 5G networks. However, the complexity lies in how to simply and efficiently realise MEC services, with the auxiliary public (multi-)cloud resources? In this paper, we propose a proof-of-concept for using Reliable Server Pooling (RSerPool) as a light-weight layer of managing resource pools and handling application sessions with these pools. Our approach is simple, efficient, has low overhead and is available as open source. Here, we demonstrate the usefulness of our approach by measuring in a test setup, with a 4G testbed connected to MEC and public multi-cloud resources.

[a]Simula Metropolitan Centre for Digital Engineering
c/o OsloMet – storbyuniversitetet
Pilestredet 52, 0167 Oslo, Norway.
dreibh@simula.no

[b]Department of Digitalization, Copenhagen Business School,
Howitzvej 60, 2000 Frederiksberg, Denmark.
sma.digi@cbs.dk

* This work has been supported by the European Community through the 5G-VINNI project (grant no. 815279) within the H2020-ICT-17-2017 research and innovation program. Parts of this work have also been funded by the Research Council of Norway under project number 208798/F50. The authors would like to thank Ann Edith Wulff Armitstead for her comments.

1 Introduction

The current trend of making large powerful mobile devices (especially smartphones) has become increasingly widespread. Interestingly, such devices lack required computational power as well as proper storage capacity and/or I/O speed, apart from suffering energy-related issues. Overall, executing complex applications with large computational/storage requirements can be delegated to cloud services. It is well-known that latency-tolerant applications are well suited for cloud services, while latency-sensitive applications are suffering in the cloud platform.

It is claimed that Mobile Edge Computing (MEC) [1] in upcoming 5G networks is a way to solve the latency issues, by providing cloud services in data centres nearby the mobile user. However, not much work is available on how to actually realise services for MEC. Reliable Server Pooling (RSerPool) [2,3], is an IETF standard for a light-weight server pooling approach, which initially targeted server redundancy in telephone signalling systems. Current existing literature could be divided into two broad categories. One class of works is primarily focusing on proposing efficient strategies to optimise the respective objective functions, such as latency or energy, by using various algorithms or models, like for instance genetic algorithms [4] and Fuzzy Logic [5], to name a couple. Another category proposes architectures and frameworks for improving the QoS via workload offloading. In [6], a model is proposed to reduce the average response time for mobile users in offloading their workloads to the cloudlets. The primary components of the framework are the cloudlet, software-defined network and the cellular network infrastructure. However, we have not found any testbed-related work based on the Reliable Server Pooling mechanism.

In this paper, we introduce a simple approach for service offloading from mobile devices to MEC resources and even public (multi-)cloud resources [7, 8], depending on availability and workload. The goal of our approach is to combine existing software systems to provide a solution which meets the following goals: simplicity, efficiency, low-overhead, and open-source. To achieve this, we combined and adjusted these components: RSPLIB [2] for RSerPool-based session handling of MEC services; VNF-based Evolved Packet Core (EPC) for the 4G/5G network based on OpenAirInterface [9,10]; Open Source MANO [11] for service orchestration of EPC and MEC systems; and OpenStack for hosting the compute resources for EPC and RSerPool-based MEC services. We have demonstrated the applicability of our proposed approach by a proof-of-concept in a testbed setup. Finally, we also have provided some discussions about this ongoing work related to improving the system.

2 Component Description

In this section, we have provided the required background information of the components which are primarily responsible for managing and orchestrating the MEC as well as network-related services.

2.1 Reliable Server Pooling (RSerPool)

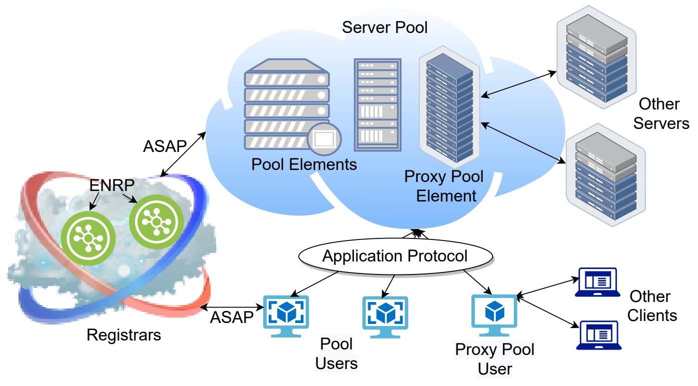

Fig. 1 The RSerPool Architecture

Reliable Server Pooling (RSerPool) [2, 12, 13] was originally motivated by the need for handling server redundancy in telephone signalling systems. However, the problem of handling server redundancy is generic and regularly triggers "reinventing the wheel". RSerPool is therefore a generic, application-independent framework. A particular property of it is to be simple and lightweight, making it suitable also for devices with very limited resources. RSPLIB[2] [2] is the most widespread open-source implementation of RSerPool.

The RSerPool architecture, as described in detail in [2, 3], is depicted in Figure 1. In an RSerPool setup, a number of servers, each providing a certain service, form a pool. Servers of a pool are denoted as Pool Elements (PE). Within its operation scope, a pool is identified by its unique Pool Handle (PH, e.g. a string like "Data Analysis Pool"). The handlespace, which is the set of all pools of an operation scope, is managed by Pool Registrars (PR, also denoted as registrars). Since a single PR would be a single point of failure, RSerPool setups should consist of at least two registrars. They synchronise the handlespace by using the Endpoint haNdlespace Redundancy Protocol (ENRP) [2, 14]. An operation scope is limited to an organisation

[2] RSPLIB: https://www.uni-due.de/~be0001/rserpool/.

or company. Unlike services like the Domain Name System (DNS), RSerPool does not intend to scale to the whole Internet. This significantly simplifies the architecture, making it very light-weight (see also [15]). Nevertheless, pools can be distributed over large geographic areas [2], to achieve a high resilience of services, e.g. to keep a service running in case of an earthquake. Servers can dynamically register to, and deregister from, a pool at a PR of the operation scope, by using the Aggregate Server Access Protocol (ASAP) [2, 16]. The ASAP connection is also used for monitoring the availability of the PE by a keep-alive mechanism [12]. Proxy pool elements can connect "legacy" non-RSerPool servers.

ASAP is also used by clients, denoted as Pool Users (PU) in the context of RSerPool, to access the resources of a pool. They can query a PR of the operation scope to select PE(s). This selection is performed by using a pool-specific pool member selection policy [2,12,17], which is usually just denoted as pool policy. Examples of pool policies are Round Robin (RR) and Least Used (LU). ASAP can also be used between PU and PE, then realising a Session Layer functionality between a PU and a pool. ASAP can then also support the actual Application Layer protocol to handle failovers and help with state synchronisation [2, 12]. Proxy pool users can connect "legacy" non-RSerPool clients to a pool.

2.2 Open Source MANO and the SimulaMet OAI EPC

Network Function Virtualisation (NFV) is a crucial part of 5G networks: Network functionalities can be realised as Network Services (NS), which are composed of Virtual Network Functions (VNF). NSs can then be instantiated as Virtual Machines (VM) in data centres. This allows for a very high flexibility: NSs can dynamically be instantiated when needed and removed when not needed any more. Furthermore, VNF instances can be scaled as needed. However, managing and orchestrating NFV is a complex task. An increasingly popular framework for this purpose is Open Source MANO[3] (OSM) [11]. OSM is the orchestration platform from ETSI. It utilises an underlying Network Function Virtualisation Infrastructure (NFVI) for instantiating the Virtual Deployment Units (VDU) as VMs. A commonly used NVFI is OpenStack, but OSM supports other frameworks as well.

Based on OSM, we developed a VNF for the Enhanced Packet Core (EPC) of OpenAirInterface[4] (OAI), denoted as SimulaMet OAI VNF[5] [9]. In particular, it can be used to easily realise a tailor-made EPC for custom 4G/5G testbed setups. Our EPC (see also Figure 2) consists of four VDUs [9]:

1. *Home Subscriber Server* (HSS) is the central database containing the information about users and their subscriptions. The HSS functionalities

[3] Open Source MANO: https://osm.etsi.org.

[4] OpenAirInterface: https://www.openairinterface.org.

[5] SimulaMet OAI VNF: https://github.com/simula/5gvinni-oai-ns.

include mobility management, session establishment, user authentication and access authorisation. It provides its service to the MME via the S6a interface.

2. *Mobility Management Entity* (MME) handles the procedures of attaching and detaching as well as service requests of *User Equipment* (UE) and eNodeBs. It communicates with eNodeBs over the S1-C interface (particularly using SCTP as Transport Layer protocol), with SPGW-C over the S11 interface, and with HSS over the S6a interface.
3. *Control Plane of the Packet Data Network Gateway* (SPGW-C) provides the control part of a combined Serving Gateway (SGW) and Packet Data Network Gateway (PGW). That is, OAI combines SGW and PGW, but uses *Control and User Plane Separation* (CUPS). The SPGW-C handles control requests from the MME via the S11 interface, and communication with the SPGW-U via the SXab interface.
4. *User Plane of the Packet Data Network Gateway* (SPGW-U) handles the forwarding of user traffic between the *Public Data Network* (PDN) at the SGi interface (i.e. usually the public Internet) and the eNodeB over the S1-U interface. User traffic between eNodeB and SPGW-U is tunnelled via GPRS Tunnelling Protocol (GTP). The setup of user traffic tunnels is controlled by the SPGW-C over the SXab interface.

The configuration flexibility of our VNF can be utilised easily by using NSs, e.g. by adding MEC resources. We will explain the details next in Section 3.

3 Proposed Approach

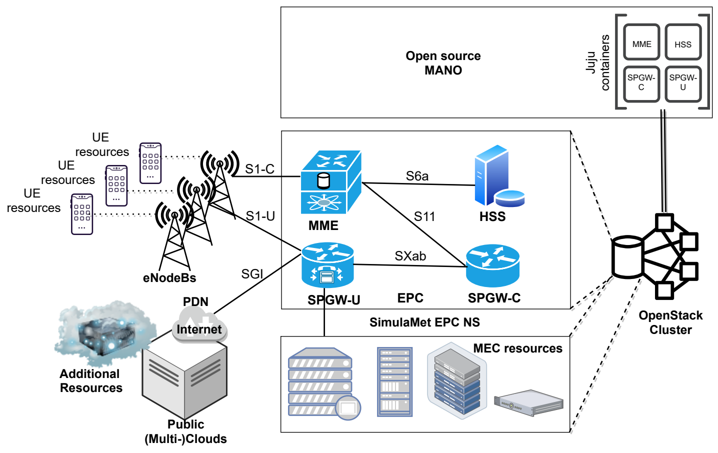

Fig. 2 Testbed Setup with EPC, Open Source MANO and Mobile Edge Computing

Here, we are going to explain our approach to handle the workload offloading challenge. The basic scenario is illustrated in Figure 2: the UEs run a certain application, which is demanding when it comes to computation and/or storage. They have very limited resources. MEC resources to support this application are available nearby the user. Furthermore, it may also be possible to have additional resources in public (multi-)clouds [7,8] somewhere in the Internet. Application examples for workload offloading could be:

- Processing measurement data recorded by the UE, e.g. to apply computation-intensive Machine Learning (ML) algorithms on specialised hardware;
- Performing post-processing and advanced compression of real-time recorded video/audio data as well as storage for further usage by the UE;
- Mining crypto currency for payment of other services, e.g. for allowing the user to read pay-per-view articles of an online journal or newspaper.

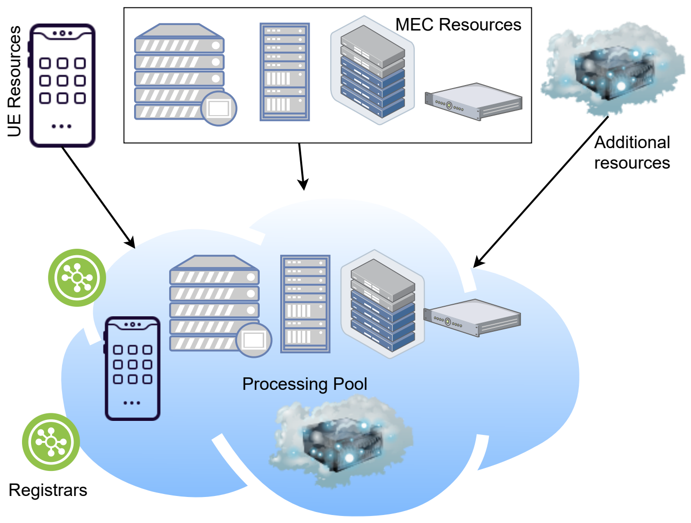

Fig. 3 RSerPool with Mobile Edge Computing and Public (Multi-)Cloud Resources

Our approach to apply RSerPool is illustrated in Figure 3. The resources are added into a pool, here identified by its PH "Processing Pool". That is, the servers in the MEC, and possibly also auxiliary servers in a public (multi-)cloud, become PEs of this pool. From the implementation perspective, it is easy to also start a server instance on the UE itself if the application allows this, due to resource constraints. RSerPool itself is very light-weight, which means it will not add a large management overhead [2, 15]. The PE on the UE would then be part of the pool too. If everything else fails, for instance due to loss of network coverage, it would allow to run the application on the UE device, albeit with reduced performance. The PU-side of the application then just needs to pick a *suitable* PE from the pool (or even use multiple PEs in parallel) to use their service. The choice of a suitable PE is, as described in Subsection 2.1, the goal of the pool policy. It is important to use the right pool policy here, so that the following goals are achieved:

1. Only use the PE on the UE if there is no other choice, e.g. no network coverage or MEC/public (multi-)cloud resources available.
2. Use public (multi-)cloud resources only when they are a suitable choice, e.g. when the MEC resources are highly utilised.
3. Otherwise, use the MEC resources.
4. Apply load balancing.

Least Used (LU) [2, 17] select the PE p where its load L_p is lowest, and round robin or random among PEs with the same lowest load. It will therefore not differentiate between MEC, public (multi-)cloud and UE resources, i.e. not satisfying the first 3 goals. Priority Least Used (PLU) [2] adds a PE-specific load increment constant I_p to LU. That is, PEs are chosen based on the lowest sum $L_p + I_p$. Setting $I_{p_{\text{MEC}}} < I_{p_{\text{PMC}}}$ for all MEC PEs p_{MEC} and public (multi-)cloud PEs p_{PMC}, as well as $I_{p_{\text{UE}}}$=100% for the UE PE. This PLU pool policy setting achieves our goals as described above.

Finally, PRs are needed to manage the handlespace. Clearly, at least one PR needs to run within the core network (e.g. as part of the MEC setup). Another PR can run locally to the additional PEs in the public (multi-)cloud. To allow the UE to run a local PE without network coverage, the UE would need to run its own PR instance. It is light-weight, i.e. having only low memory and CPU requirements, as well.

4 Testbed Description

For our proof-of-concept evaluation in Section 5, we made the following setup: OSM is running "Release EIGHT" in a dedicated VM, connected to an OpenStack setup as NFVI. An NS using the SimulaMet OAI VNF is deployed by OSM in the NFVI. This NFVI is also used for the MEC resources. The User Equipment (UE) is a regular PC. Another regular PC is used as eNodeB. It is running the eNodeB software from OAI, stable version 1.2.2. As Software-Defined Radio (SDR) board, an Ettus B210 connected via USB 3.1 is engaged.

As application, the CalcAppProtocol model from [2, Section 8.3] [12] – and provided as part of RSPLIB [2] – is used: a PE has a given request handling *capacity* given in the abstract unit of calculations/s. An arbitrary application-specific metric for capacity may be mapped to this definition, e.g. CPU operations, processing steps, disk space usage, etc. Each request has a request size, which is the number of calculations consumed by the processing of the request. A PE can process multiple requests simultaneously, following the multi-tasking principle. The user-side performance metric is the handling speed. The total time for handling a request d_{Handling} is defined as the sum of queuing time, start-up time (dequeuing until reception of acceptance acknowledgement) and processing time (acceptance until finish). The *handling speed* (in calculations/s) is defined as: $\text{HandlingSpeed} = \frac{\text{RequestSize}}{d_{\text{Handling}}}$.

In our setup at SimulaMet in Oslo, Norway, we use:

- n PU instances may run on the UE. Each PE generates requests with an average size of 1,000,000 calculations at an average frequency of 10 s (negative exponential distribution for both).
- There is 1 PE on the UE, with a capacity of only 200,000 calculations/s.
- 2 PEs are deployed as MEC resources in OpenStack, each with a capacity of 1,000,000 calculations/s.
- In total, 4 PEs as public multi-cloud resources are deployed, each with a capacity of 1,000,000 calculations/s. These PEs are distributed as multi-cloud using the NorNet Core [18, 19] infrastructure, with each one PE in Longyearbyen, Gjøvik, Tromsø (all Norway) and Haikou (China). That is, there are significant delay differences (see also [20] for details), between around 20 ms within Norway, and more than 300 ms between Norway and China.
- PEs accept up to 4 requests in parallel. When fully loaded, further requests get rejected, and a new PE has to be selected.
- 1 PR on the UE, 1 PR in the MEC cloud, and 1 PR in Stavanger, Norway.

5 Proof-of-Concept Evaluation

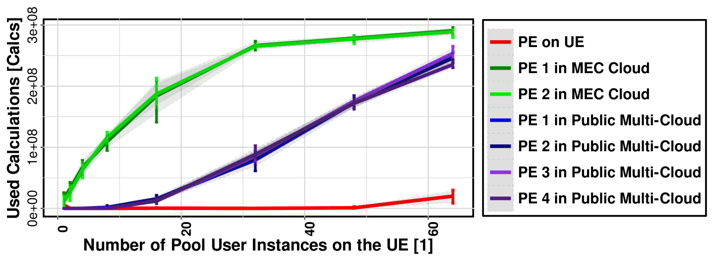

Fig. 4 Used Capacity (in Calculations) of each PE

For our proof-of-concept setup, we have chosen the PLU policy (see Section 3) with I_{PU}=100%, I_{MEC}=10% and I_{PMC}=50%. That is, MEC PEs should be preferred, unless highly loaded. Then, PMC PEs should be used instead. The PE on the UE itself should only be used in case of severe overload (or loss of network connectivity). Each measurement run has been repeated 5 times, with a measurement duration of 300 s. The results show the average of these runs, together with absolute minimum and maximum (thin error bars) as well as 10% and 90% quantiles (thick error bars).

The results for the used capacity on the PEs when increasing the number of PU instances on the UE from 1 to 64 are shown in Figure 4. As expected, under low load, mostly the MEC resources are used (green curves). 1 PU generating requests of 1,000,000 calculations every 10 s utilises the pool of 6 PEs (2×MEC+4×PMC, providing 1,000,000 calculations/s per PE) only by around 1.67% on average. It is clearly visible that only the MEC PEs are used. For 16 PUs, the average pool load is already 26.7%. Then, when sometimes a MEC PE already handles 2 or 3 simultaneous requests, it becomes reasonable to use a PMC PE instead. The pool load increases to around 53.3% for 32 PUs, then leading to an increased usage of the 4 PMC PEs. 48 PUs mean 80% average utilisation, while 64 PUs are overload at around 107%.

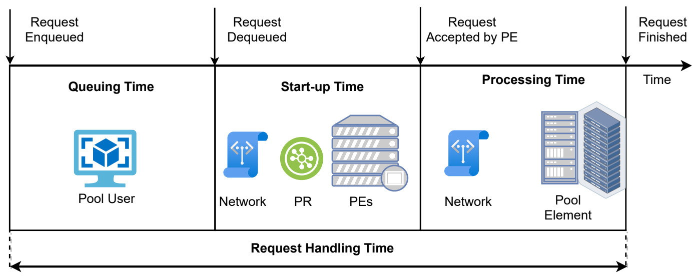

Fig. 5 Queuing Time, Start-up Time and Processing Time

Table 1 Average Per-PU Total Queuing Time

Pool Users	Q_{mean}	Q_{min}	Q_{max}	Q_{Q10}	Q_{Q90}
1	37.51	3.79	106.98	4.16	88.29
2	9.88	1.11	39.18	2.68	17.95
4	9.50	0.42	26.44	0.90	21.98
8	10.32	0.64	29.96	3.88	20.72
16	15.57	0.00	78.19	1.47	33.12
32	31.26	0.00	173.75	4.66	65.87
48	37.90	0.60	341.95	5.97	73.62
64	174.33	1.46	1081.93	27.24	435.95

Figure 1 illustrates the 3 components of the request handling time – queuing, start-up and processing – in detail. Table 1 shows the results for the queuing time, i.e. the total time requests spent in the PU's queue during the measurement time. While it remains low ($\lesssim$38 s) for decent pool loads, it increases significantly – as expected – when the pool is no longer able to process the overload (64 PUs, i.e. 107% load). In this case, the UE PE also gets used.

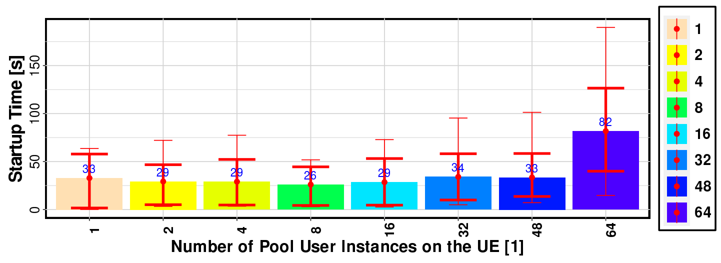

Fig. 6 Average Per-PU Total Start-up Time

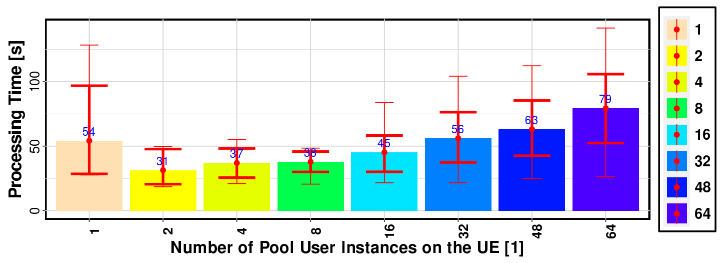

Fig. 7 Average Per-PU Total Processing Time

The corresponding start-up and processing times are displayed in Figure 6 and Figure 7. As expected, the start-up time remains reasonably constant. As long as a PE is not completely full (4 simultaneous requests), it can accept a new request. At 64 PUs, requests get rejected – and another PE selection adds to the start-up time. The processing time remains at a similar level at low to medium pool loads. That is, the PLU pool policy works as intended, realising a load balancing in the pool. Finally, the processing time increases at high load and overload, when capacity becomes a scarce resource.

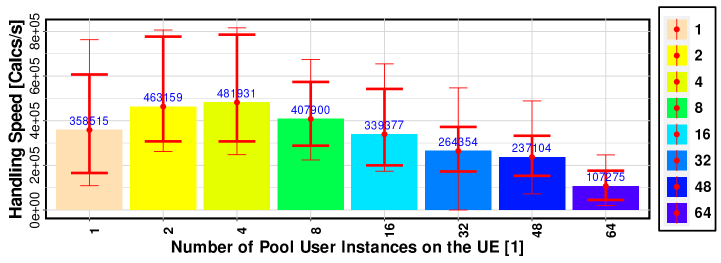

Fig. 8 Average Request Handling Speed (in Calculations/s) at the PU

The average handling speed of the pool, as shown in Figure 8, is reasonable from the user's perspective, as long as the pool is not overloaded. That is, even at 48 PUs running on the UE (which itself would only have a PE capacity of 200,000 calculations/s for *all* requests when disconnected from the pool!), the *per-PU* handling speed is still much more than 218,000 calculations/s.

Hence, our simple proof-of-concept setup achieves the set goals: It is a simple, light-weight, efficient workload offloading from UE into a MEC+PMC system.

6 Conclusions and Future Work

There is a growing demand for offloading workload from mobile devices into (multi-)clouds, because these devices are resource-constrained and battery-powered. The MEC ecosystem aims at providing low-latency communication between user device and cloud service instance or compute device, while using public network services. However, there is a need for a simple, light-weight solution for maintaining sessions with pools of MEC- and (multi-)cloud resources. Reliable Server Pooling (RSerPool) is a standard used for server redundancy and session handling.

In our paper, we propose a simple but efficient approach for using RSerPool as a solution for the workload offloading issue. It is possible to realise an application with light-weight management overhead and low configuration effort by applying useful pool policy configurations. This lightweight property even offers the possibility to run a server instance on the user device itself, to provide local processing – within the device's limits – as a last resort when everything else fails. We presented a simple proof-of-concept evaluation to show the effectiveness of our approach.

As part of future work, it is necessary to evaluate our approach in more detail, in larger setups and different scenarios. It is also useful to integrate the deployment more tightly into Open Source MANO (OSM), to provide the pool element and pool registrar functionalities as part of network services.

References

1. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile Edge Computing – A Key Technology Towards 5G. ETSI White Paper **11**(11) (September 2015) 1–16
2. Dreibholz, T.: Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture. PhD thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems (March 2007)
3. Lei, P., Ong, L., Tüxen, M., Dreibholz, T.: An Overview of Reliable Server Pooling Protocols. Informational RFC 5351, IETF (September 2008)
4. Wang, J., Wu, W., Liao, Z., Sherratt, R.S., Kim, G.J., Alfarraj, O., Alzubi, A., Tolba, A.: A Probability Preferred Priori Offloading Mechanism in Mobile Edge Computing. IEEE Access **8** (2020) 39758–39767

5. Sonmez, C., Ozgovde, A., Ersoy, C.: Fuzzy Workload Orchestration for Edge Computing. IEEE Transactions on Network and Service Management **16**(2) (2019) 769–782
6. Xiang, S., Nirwan, A.: Latency-Aware Workload Offloading in the Cloudlet Network. IEEE Communications Letters **21**(7) (2017) 1481–1484
7. Dreibholz, T., Mazumdar, S., Zahid, F., Taherkordi, A., Gran, E.G.: Mobile Edge as Part of the Multi-Cloud Ecosystem: A Performance Study. In: Proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Lombardia/Italy (February 2019) 59–66
8. Hong, J., Dreibholz, T., Schenkel, J.A., Hu, J.A.: An Overview of Multi-Cloud Computing. In: Proceedings of the 1st International Workshop on Recent Advances for Multi-Clouds and Mobile Edge Computing (M2EC) in conjunction with the 33rd International Conference on Advanced Information Networking and Applications (AINA), Matsue, Shimane/Japan (March 2019) 1055–1068
9. Dreibholz, T.: Flexible 4G/5G Testbed Setup for Mobile Edge Computing using OpenAirInterface and Open Source MANO. In: Proceedings of the 2nd International Workshop on Recent Advances for Multi-Clouds and Mobile Edge Computing (M2EC) in conjunction with the 34th International Conference on Advanced Information Networking and Applications (AINA), Caserta, Campania/Italy (April 2020) 1143–1153
10. Ocampo, A.F., Dreibholz, T., Fida, M.R., Elmokashfi, A.M., Bryhni, H.: Integrating Cloud-RAN with Packet Core as VNF Using Open Source MANO and OpenAirInterface. In: Proceedings of the 45th IEEE Conference on Local Computer Networks (LCN), Sydney, New South Wales/Australia (November 2020)
11. Reid, A., González, A., Armengol, A.E., de Blas, G.G., Xie, M., Grønsund, P., Willis, P., Eardley, P., Salguero, F.J.R.: OSM Scope, Functionality, Operation and Integration Guidelines. White paper, ETSI (June 2019)
12. Dreibholz, T., Rathgeb, E.P.: Overview and Evaluation of the Server Redundancy and Session Failover Mechanisms in the Reliable Server Pooling Framework. International Journal on Advances in Internet Technology (IJAIT) **2**(1) (June 2009) 1–14
13. Dreibholz, T., Zhou, X., Becke, M., Pulinthanath, J., Rathgeb, E.P., Du, W.: On the Security of Reliable Server Pooling Systems. International Journal on Intelligent Information and Database Systems (IJIIDS) **4**(6) (December 2010) 552–578
14. Xie, Q., Stewart, R.R., Stillman, M., Tüxen, M., Silverton, A.J.: Endpoint Handlespace Redundancy Protocol (ENRP). RFC 5353, IETF (September 2008)
15. Dreibholz, T., Rathgeb, E.P.: An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems. SERSC International Journal on Hybrid Information Technology (IJHIT) **1**(2) (April 2008) 17–32
16. Stewart, R.R., Xie, Q., Stillman, M., Tüxen, M.: Aggregate Server Access Protcol (ASAP). RFC 5352, IETF (September 2008)
17. Dreibholz, T., Tüxen, M.: Reliable Server Pooling Policies. RFC 5356, IETF (September 2008)
18. Dreibholz, T.: NorNet at Hainan University in 2021: From Simulations to Real-World Internet Measurements for Multi-Path Transport Research – A Remote Presentation. Keynote Talk at Hainan University, College of Information Science and Technology (CIST) (January 2021)
19. Gran, E.G., Dreibholz, T., Kvalbein, A.: NorNet Core – A Multi-Homed Research Testbed. Computer Networks, Special Issue on Future Internet Testbeds **61** (March 2014) 75–87
20. Dreibholz, T.: HiPerConTracer - A Versatile Tool for IP Connectivity Tracing in Multi-Path Setups. In: Proceedings of the 28th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Hvar, Dalmacija/Croatia (September 2020)