A proof-of-concept 5G mobile gateway with eBPF

A Proof-of-Concept 5G Mobile Gateway with eBPF

Federico Parola
Politecnico di Torino, IT
s252895@studenti.polito.it

Sebastiano Miano
Politecnico di Torino, IT
sebastiano.miano@polito.it

Fulvio Risso
Politecnico di Torino, IT
fulvio.risso@polito.it

ABSTRACT

In this poster we propose the first proof-of-concept open-source implementation of a 5G Mobile Gateway based on eBPF/XDP and present benchmarks that compare its performance with alternative technologies. We show how it outperforms other in-kernel solutions (e.g., OvS) and is comparable with DPDK-based platforms.

CCS CONCEPTS

• **Networks → Programmable networks**; **Middle boxes / network appliances**.

KEYWORDS

Network Functions, eBPF, XDP, 5G Mobile Gateway, 5G User Plane Function

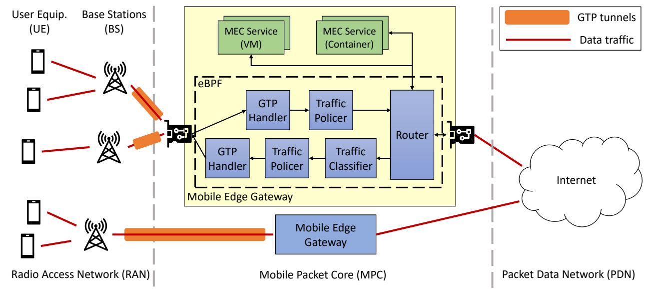

Figure 1: Mobile Gateway prototype architecture.

1 INTRODUCTION

In the context of 5G mobile networks, the Mobile Gateway (MGW) is increasingly located close to the Radio Access Network (RAN), enabling telcos to deploy virtualized network functions and services at close proximity to mobile users, hence benefiting from the reduced latency.

In this scenario, data plane technologies such as DPDK may not be appropriate because they take control of the entire server, leaving no resources to other jobs unless rigid hardware partitioning is enforced. This behaviour would not be acceptable in "mini" data centers that feature a limited number of servers; in this case resources should be dynamically shared between services for the best flexibility. In this scenario, eBPF/XDP can represent a better solution; while its raw performance are inferior to DPDK-based platforms [3], its better integration with vanilla Linux kernel makes it suitable to be used with cloud orchestrators such as Kubernetes.

However, no eBPF/XDP implementations of a MGW exist so far. This originates from the event-driven nature of the eBPF platform, which poses non trivial challenges in the implementation of key components such as shapers/policers, and the difficulties in writing complex data plane services. This poster aims at filling this gap, presenting the first proof-of-concept open-source[1] implementation of a mobile gateway and its preliminary benchmarking. This work confirms the feasibility of a MGW in eBPF/XDP and shows that the performance of this first PoC implementation greatly outperform other in-kernel solutions and is comparable with more efficient DPDK-based platforms.

2 DESIGN

Figure 1 illustrates the high-level architecture of a mobile network, with different instances of a mobile gateway that are placed on the the same servers where the others MEC services are running. The gateway handles the data traffic of several user equipments (UE), which is encapsulated into GTP-u tunnels, and acts as a point of contact with packet networks like Internet. Moreover, it includes functionalities such as access control, per-flow QoS, guaranteed and maximum bit rate and traffic classification/monitoring.

We implemented a subset of these functionalities as four separate in-kernel network services, leveraging one or more eBPF programs deployed through the Polycube [5] eBPF framework, which in addition to simplifying the development and chaining of such services, provides an automatically generated REST API to control the service behavior. These modules include:

1) GTP Handler. In the upstream direction (UE-to-MGW) this module acts as a GTP tunnel terminator; it removes the GTP headers (i.e., GTP, UDP and outer IP) and retrieves the Tunnel Endpoint Identifier (TEID), which is then passed to the next module in the chain (i.e., *traffic policer*) through a shared eBPF `PERCPU` hash map. On the downstream direction (MGW-to-UE), it matches the IP destination address of the packet (i.e., the IP address of the UE) with an eBPF `HASH` map containing the UE-BS mapping. Then, it encapsulates the packet into a new GTP tunnel, retrieving the TEID from the shared eBPF map, and sends it to the base station.

2) QoS Management. The next module of the chain is in charge of dropping, passing or shaping the packets of a specific traffic class that is equal to the TEID. For bandwidth management, we implemented and evaluated three different policing mechanisms. They have low memory overhead since they are bufferless, and they have small CPU overhead because there is no need to schedule or manage queues. More complex traffic shapers (e.g., pacing, hierarchical token bucket) are not entirely implementable in eBPF/XDP but require a cooperation with the Linux Traffic Control (TC) subsystem for buffer management and queuing.

Fixed Window Counter (FWC): A userspace thread is in charge of resetting, every W seconds an eBPF `HASH` map containing the mapping TEID - Window Counter (WC), which is defined as the product of the desired rate R and the window size W. Once a new packet is received, the MGW atomically decreases the WC size in the map based on the packet size; when the value is zero the packet is discarded.

[1]https://github.com/polycube-network/polycube/tree/mobile-gateway

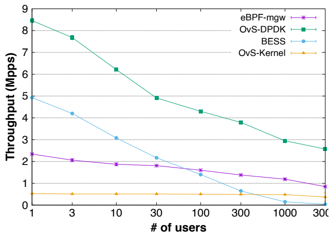

Figure 2: Multiple users scalability (downlink).

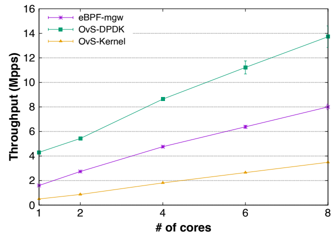

Figure 3: Multicore scalability (downlink).

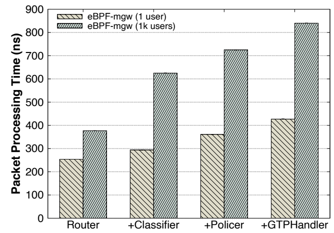

Figure 4: Packet processing time breakdown.

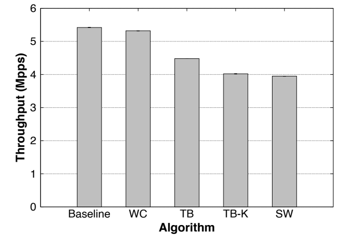

Figure 5: Throughput with different rate limiters.

Token Bucket (TB): Unlike the FWC, the TB requires to read and update the eBPF map value to refill the bucket with the correct number of tokens. Unfortunately, this operation can not be atomically performed in user space, given the impossibility to acquire a lock to an eBPF map entry in that context. Therefore we perform this task directly in the data plane, using the `bpf_spin_lock` to read and modify several variables atomically, and we store an additional timestamp value in the eBPF map, which is used to retrieve the number of tokens to be added after the last refill.

Sliding Window (SW): Given the rate limit of R and burst limit of B, a fixed window of size $W = B/r$ is defined. Every time a new packet arrives, its arrival time is checked against the information saved in the eBPF map. If the time is inside the window, the packet is allowed and the window is shifted forward by a value $t = S/r$ where S is the packet size, otherwise the packet is discarded.

3) Traffic Classifier. This module is used to map a packet in the downlink direction to its corresponding TEID, which is used to enforce the correct QoS. To support more complex classification rules we used the same algorithm defined in [4], which is compatible with the limitation present in eBPF.

4) Router. The router component can work in both "shared" mode, where the host FIB table is used to decide the next hop of the packet through the Internet, or in "private" mode where a separate BPF `LPM_TRIE` map is used and configured by the MGW control plane.

3 EVALUATION

We compared our eBPF MGW with equivalent pipelines based on different data plane technologies (BESS [2], OvS-DPDK and OvS-kernel [6])[2] available in TIPSY [1].

Scalability with multiple users. We scaled the number of configured users (each one with a single tunnel) up to 3000, setting one tunnel endpoint every 100 users and one route on the PDN every 10 users. Moongen generated an average of 10 UDP flows per user. Fig. 2 shows that the eBPF pipeline outperforms both other in-kernel alternatives and also (user space) BESS with an high number of configured users, due to the poor scalability of the latter.

Multicore scalability. We configured 100 users, 10 routes and 1 base station, scaling the pipeline with an increasing number of cores, with an average of 10 flows per user. Fig. 3 shows that the scalability of the eBPF implementation is in line with the one of its in-kernel and user space counterparts.

Modules overhead. We analyzed the impact of the different modules on the performance of the eBPF gateway with both low (1) and high (1000) number of configured users. Fig. 4 shows the average time needed to process each packet, starting only with the *Router* and then gradually adding other modules. Results show that the most resource-hungry service is the *Classifier*, whose algorithm scales linearly with the number of rules we use in this scenario, but that can be reduced with a more careful implementation.

Rate limit algorithms overhead. Tests in fig. 5 evaluate the overhead introduced by different rate limiters. The *Policer* has been attached to a simple eBPF program forwarding packets between the interfaces of the DUT and configured with a high rate limit in order not to influence the number of forwarded packets. The *WC* shows the best performance, thanks to its simplest data plane that reaches almost the baseline speed. The *TB* (used in all the previous tests) has an additional cost due to the spinlocks, while the TB-K bar show the overhead introduced by the timestamping alone (`ktime` helper). The *SL* shows the poorest performance, relying both on spinlocks and `ktime` timestamping.

REFERENCES

[1] TIPSY Authors. 2018. TIPSY: Telco pIPeline benchmarking SYstem. (2018). https://github.com/hsnlab/tipsy

[2] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. SoftNIC: A software NIC to augment hardware. (2015).

[3] Høiland-Jørgensen et al. 2018. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. 54–66.

[4] Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Mauricio Bernal, Yunsong Lu, and Jianwen Pi. 2019. Securing Linux with a Faster and Scalable Iptables. (2019).

[5] Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Mauricio Vásquez Bernal, Yunsong Lu, Jianwen Pi, and Aasif Shaikh. 2019. A Service-Agnostic Software Framework for Fast and Efficient in-Kernel Network Services. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE.

[6] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*.

[2] Tester and DUT connected with a dual-port Intel XL710 40Gbps NIC. DUT with Intel Xeon Gold 5120 14-cores CPU @2.60GHz (hyper-threading disabled) and Ubuntu 18.04.1 LTS. Moongen packet generator. Kernel 5.6 for eBPF, kernel 5.0 with DPDK 19.11 for other technologies.