Monitoring 6G UPFs: A Software-based Network Tomography Framework

Florian Raskob[1], Wilton Arthur Poth[2], Tobias Meuser[1], Björn Scheuermann[1]
[1]Communication Networks Lab, Technical University of Darmstadt
[2]Department of Computer Science, ETH Zürich
E-Mail: {florian.raskob, tobias.meuser, scheuermann}@kom.tu-darmstadt.de, uni@wilton-poth.de

***Abstract*—The architecture of 6G networks comprises many software-based Network Functions (NFs) that can run on and migrate between different devices. To make future 6G networks resilient, a flexible framework is needed to detect failures in individual NFs. In this work, we present a software-based Network Tomography (NT) monitoring framework for 6G NFs, enabling flexible real-time monitoring. As a case study, we measured the performance of the User Plane Function (UPF), which is responsible for forwarding user traffic in the core network. To determine packet loss and latency, the framework matches ingress and egress packets of UPFs while it monitors packet and data rates by analyzing only the egress packets. In our evaluation, we demonstrated the framework's accuracy of a Mean Percentage Error (MPE) of 0.2% for packet rates, 0.13% for packet loss, and 0.05% for data rates. Latencies were measured with 1.86 µs precision. Our findings demonstrate that software-based monitoring can achieve high-precision performance measurements, which is fundamental for enabling resilience in future 6G networks.**

***Index Terms*—6G, mobile networks, resilience, monitoring, network tomography**

I. Introduction

As future 6G networks will play a critical role in our society, they must be resilient, i.e., autonomously deal with challenges like failures and attacks while enhancing themselves by learning from preceding challenges [1]. The 6G core network adopts a Service-based Architecture (SBA) consisting of multiple software-based NFs and can be distributed by placing different NFs on different devices [2]. NFs should be monitored to detect failures and ensure their correct operation. For this purpose, the 6G standard defines mechanisms enabling NFs to provide status information. However, during a failure, the functionality of an NF might be impaired, making this information unreliable. This raises the need for a flexible software-based monitoring framework that can be deployed in parallel to individual NFs. While integrating this monitoring in the NF themselves is possible, in this work, we propose a NF monitoring framework based on the concept of NT [3], i.e., without any modification to the original NF. Based on these considerations, we aim to answer the following questions.

1) How can we measure the performance of 6G NFs in real-time using a software-based approach that considers only ingress and egress data?
2) What is the impact of such an approach on resource utilization?
3) What accuracy can be achieved when measuring the performance of NFs?

To answer these questions, we developed a software-based framework to measure the performance of 6G UPFs in real-time. The UPF is central for forwarding packets and, therefore, highly related to the overall 6G network's functionality and performance. The framework measures the packet loss rate and latency of UPFs by matching ingress and egress packets and data rate and packet rate by only considering the egress packets. The framework is lightweight, fully software-based, and flexible, as it can be easily hot-plugged into any UPF's execution environment. Although we focus only on UPFs, the concepts can be easily adapted to monitor any NF. Our framework enables the network to detect potential failures and attacks by continuously monitoring the functionality of NFs. This is a fundamental enabler for overall resilience [4].

We structure the rest of this paper as follows. We present related work in Section II. We introduce our framework's design and implementation in Section III. We describe how we evaluated the framework's accuracy and present and discuss the results in Section IV. Finally, we conclude this work and present future directions in Section V.

II. Related Work

In the following, we present work that is related to ours. Khaloopour et al. introduced the Resilience-by-Design framework, emphasizing the importance of embedding resilience into the design phase of future 6G networks [1]. This includes all layers provided by the network, from electronics to services. In their work, they embrace the conceptual framework of Sterbenz et al. [4] to make communication networks resilient. Sterbenz et al. define a resilient system as one that can autonomously *detect* threats, *defend* itself against them, *remediate* itself to get on an acceptable level of functionality, and finally *refine* itself to increase its capability to cope with future challenges like failures and attacks. Regarding this definition, our work contributes to the *detect* phase by providing a framework to monitor the performance of UPFs in real time.

In the context of monitoring 5G networks, Mamushiane et al. performed stress testing on the open5gs's UPF and analyzed its CPU utilization [5]. They used UERANSIM for Radio Access Network (RAN) and User Equipments (UEs) simulation and generated traffic using the ping utility. However, they could not generate enough traffic to push the UPF's CPU utilization to maximum due to limitations by UERANSIM.

ISBN 978-3-903176-72-0 © 2025 IFIP

To circumvent this limitation, we used PacketRusher in our experiments. Soos et al. measured the end-to-end throughput, latency, jitter, and packet error rate of a full non-standalone 5G testbed using iPerf3 and Coscos's TRex [6]. While their experiments assess the overall network performance, they do not offer insights into the performance of individual network components.

In previous works, latency and packet loss of data plane traffic were often measured using timestamping [7], [8]. Therefore, two timestamps are added to a packet: one at the ingress and one at the egress point of the device under test (DUT). The forwarding time of the DUT can then be evaluated by calculating the difference between the two timestamps. Rischke et al. measured latency, inter-packet delays, packet rate and data loss in a full 5G testbed [9]. Therefore, they added timestamps and sequence numbers into the packets' payload utilizing a Network Interface Card (NIC) with Data Plane Development Kit (DPDK) support. The timestamps had a precision of less than 1 µs by using the `clock_gettime()` method. When using timestamps, the main challenge is to create the timestamps with high precision, which can be effectively achieved through hardware. To achieve this in a software-based setup, Orosz et al. [7] altered libpcap, which is used by, e.g., Wireshark or tcpdump. They point out that for this high precision, the clock source, the NIC driver architecture, and the OS kernel play a central role. They also showed that the overhead of creating the timestamps increases for lower-frequency CPUs.

Although hardware-based timestamping provides high-accuracy measurements, deploying specialized hardware to measure the performance of 6G NFs is not always feasible due to its distributed and flexible architecture. Furthermore, it is not possible to write timestamps into the payload of packets without corrupting them. In 2002, Coates et al. introduced the concept of NT to cope with the heterogeneous and unregulated structure of the internet [3]. The main idea is to infer internal network characteristics, such as delay and packet loss, using end-to-end measurements without direct access to intermediate nodes. More recently, Kakkavas et al. emphasize that NT can effectively adapt to the dynamic nature of 5G networks, addressing the limitations of hardware-based monitoring techniques [10]. They examine existing NT techniques and highlight their ability to provide efficient, low-overhead monitoring. In this work, we provide a monitoring framework to measure the performance of 6G UPFs utilizing the concepts of passive NT, i.e., without using any test packets or probes for measurements.

Wernet et al. present a mechanism to mirror one 5G UPF's session state to a second failover instance [11]. During their evaluation, they trigger failovers based on heartbeat messages sent between the UPF and SMF. However, the RTT and variance of these heartbeats are system-dependent, and relying on them comes with a risk of false positive failovers. Therefore, more reliable frameworks to monitor the state of UPFs are needed.

III. System Design & Implementation

In the following, we present our design of a software-based framework to monitor UPF performance, which is optimized for deployment in distributed 6G networks. Running in parallel to the UPF, as shown in Fig. 1, the framework can be hot-plugged into the UPF's execution environment. It runs independently from the Control Plane (CP) and can be executed with the UPF on the same machine or on a separate machine. The framework utilizes an NT approach, treating the UPF as a black box and operating independently of any information it provides. By matching incoming with outgoing packets, the framework transparently measures key performance metrics such as *packet rate*, *packet loss*, *data rate*, and *latency*. In this work, we focused on measuring the performance of UPFs, but all concepts can be directly applied to any other Network Function utilizing appropriate matching functions.

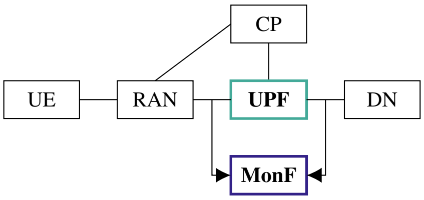

Fig. 1. Our monitoring framework (MonF) operates in parallel to the 6G user plane, sniffing the UPF's ingress and egress packets.

A. Sniffing Packets

Our framework sniffs all packets entering and leaving the UPF using two POSIX Raw-Sockets combined with the *poll()* interface to simultaneously receive packets from both UPF interfaces. Packets can enter and leave the UPF on both UPF interfaces, as they can be sent to the Data Network (DN) (uplink) or the RAN (downlink). Packets captured between the RAN and the UPF are sent uplink if their destination port matches the UPF's GTP-U server port and downlink if their destination port matches the data interface port of the Next Generation Node B (gNB). Packets captured between the UPF and the DN are sent uplink if their source IP address is in the UE's subnet and downlink if their destination address is in the UE's subnet.

B. Mapping Incoming and Outgoing Packets

Incoming and outgoing packets must be mapped to determine if the UPF forwarded or dropped a packet and to calculate the time a packet needed to pass the UPF. Our framework stores incoming packets and some metadata in a hash table, allowing outgoing packets to be mapped to them later. Therefore, we must consider that some headers can change while traversing the UPF. For example, the TTL is decremented by one, and the IPv4 source address might be changed for packets sent to the DN due to NAT. We overcome this problem by decrementing the TTL before storing the packet and zero headers for which it is unknown if and how the UPF changes them. Thus, outgoing packets that are looked up have the same contents as their equivalent previously stored in the hash table. GTP-U headers are always ignored when mapping packets.

We calculate the 64-bit hashes for the hash table based on the sum of the first and last eight bytes of the IP packets' payload. The hash is calculated similarly to map an outgoing packet to its incoming counterpart. Then, the framework performs a lookup in the hash table and compares the entry with the outgoing packet. If the framework finds a matching entry, the UPF has forwarded the packet; however, the UPF can also drop packets. To detect this, we select a threshold, and if a packet resides in the hash table longer than that threshold, we delete it from the hash table and assume it was dropped by the UPF either intentionally or not. The threshold must be selected based on the highest expected latency of the UPF, as the framework considers packets erroneously lost if forwarding them takes longer than the threshold. If the threshold is too big, packets reside longer in the hash table, increasing the probability of hash collisions and delaying the detection of lost packets. This also reduces the time granularity with which lost packets are detected. A packet might be considered lost if the UPF did not process it correctly and unintentionally changed it while forwarding it. In this case, our framework cannot match the egress to the ingress packet. To determine how long a packet resides in the hash table, we store a timestamp with each entry. We continuously sweep the hash table's buckets and check if timestamps have exceeded the given threshold. In addition, we sweep all buckets every threshold's duration unless all of them were swept in the last second. When an entry is matched or has timed out, we delete it from the hash table.

C. Measuring Performance

Our implementation considers four events. (1) *inpkt:* an incoming packet was received, (2) *outpkt:* an outgoing packet was received, (3) *matched:* an outgoing packet was successfully mapped to an incoming packet, and (4) *lost:* a packet is considered lost. The following will discuss each performance metric our framework measures and how we assess it using these events. Each of the metrics is measured for uplink and downlink separately.

1) Packet Rate / Loss: The packet rate describes how many packets are forwarded by the UPF. We determine it by incrementing a counter on each occurrence of the *outpkt* event. Packet loss is determined similarly considering the occurrence of *lost* events. The loss ratio is calculated by dividing the packet loss by the sum of both values.

2) Data Rate: For each occurrence of an *outpkt* event, we increase a value by the length of the outgoing packet.

3) Latency: All incoming packets are stored along with a timestamp in the hash table. We determine the latency on a *matched* event by calculating the delta between this timestamp and when the corresponding outgoing packet was received.

D. Performance Considerations

Taking all packets entering and leaving the UPF into account can result in high resource utilization, especially at high packet rates. This is a problem as our framework should not interfere with the UPF's operation even when running on the same machine. To address this issue, we use subsampling and sample every n^{th} ingress packet, reducing the number of packets that have to be handled by our framework. We then approximate the actual number of lost packets by multiplying the measured number of packets lost by n.

Sampling egress packets would complicate determining packet loss and latency. If we do not find a matching egress packet for an ingress packet, we cannot tell if the packet was lost or not sampled. Determining the latency for a packet is only possible on a *match* event; however, if we independently sample every n^{th} ingress and egress packet, the probability of finding two matching packets is quite low. Applying subsampling to the ingress packets does not affect the framework's measurement of packet and data rates, as we determine these metrics based only on egress packets. We decided to sample every n^{th} packet instead of taking time-based periodic samples as this was shown to create more accurate measurements by Drobisz et al. [12]. Higher sampling rates (smaller n) lead to greater accuracy at the expense of increased resource utilization. Therefore, n should be chosen based on the expected packet rates, such that the framework provides sufficient accuracy while keeping its resource utilization as low as possible.

IV. Evaluation

The evaluation aims to determine the accuracy of our proposed framework. Therefore, we measured the UPF's performance using our monitoring framework in parallel to iPerf3 and a hardware-based packet stamper for reference. In the following, we present our experimental setups and results for each metric.

Regarding packet rate, packet loss, and data rate, we evaluated our framework using two machines connected by a 100 Gbit/s optical fiber cable, as shown in Fig. 2. Both machines have an AMD Ryzen Threadripper PRO 5955WX, 128 GB of memory, and run Ubuntu 22.04. Regarding the 5G network, we run PacketRusher [1] simulating a UE and gNB on Machine 1 and an open5gs [2] core network on Machine 2. We use iPerf3 for traffic generation and run its client and server on Machine 1 to avoid time synchronization problems. Our monitoring framework is deployed on Machine 2 along with the core network. It sniffs packets on the Ethernet interface connected to Machine 1 via the optical fiber cable and the UPF's tun interface.

To evaluate the accuracy of our framework, we generated UDP traffic of different data rates ranging from 1 Mbit/s to 2 Gbit/s using iPerf3 and sent it through our test setup. We increased the data rates in steps of 25 Mbit/s and ran the

[1] https://github.com/HewlettPackard/PacketRusher

[2] https://open5gs.org/

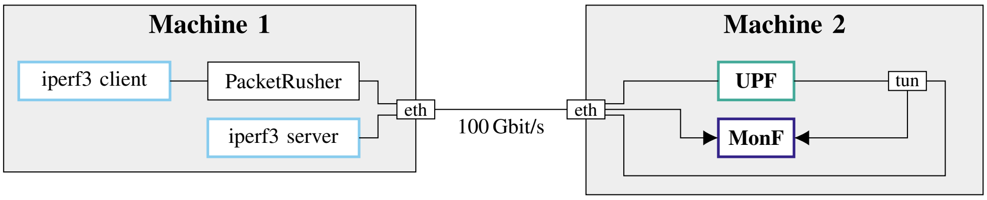

Fig. 2. We used two Linux machines in our experimental setup to evaluate our monitoring framework's accuracy in measuring packet rates, packet loss rates, and data rates. We ran iPerf3's client and server alongside PacketRusher on Machine 1 and the core network with our monitoring function on Machine 2. Both machines were connected via a 100 Gbit/s fiber link.

experiment for 5 min for each data rate. We conducted the same experiments for uplink and downlink; however, in the following, we will mainly focus on uplink, as the results did not differ significantly. We set the sampling rate such that the framework samples every 1000th packet during all experiments. As iPerf3 provides one measurement point per second, we repeatedly calculate the average of all data points within the last second for each metric, leading to a 1 s resolution.

A. Packet Rate / Loss

Fig. 3 shows the packet and packet loss rates measured by iPerf3 and our monitoring function. It also shows the CPU utilization of our framework and the UPF under test during each experiment. We calculated the averages of the 5 min experiments for each target data rate. One can see that up to 1 Gbit/s, the measured packet rate increases linearly with the target data rate. This is expected, as all UDP packets generated by iPerf3 are the same size. For target data rates higher than 1 Gbit/s, the packet rate converges against 120,000 pps, and the packet loss increases. The reason for this can be seen in the CPU utilization of the UPF. As the single-threaded UPF implementation reaches 80% CPU utilization, it can no longer process all incoming packets and begins to drop them. When the UPF's CPU utilization reaches 100% in our experiment, the packet loss is about 30%. Our monitoring framework's CPU utilization remains lower than that of the UPF. Across all experiments, it reaches a maximum of 80%. When we compared the measurements provided by iPerf3 with the ones provided by our framework, we observed an average mean percentage error (MPE) of 0.2% for the packet rate and an MPE of 0.13% for packet loss. Even for the higher target data rates, the errors did not increase.

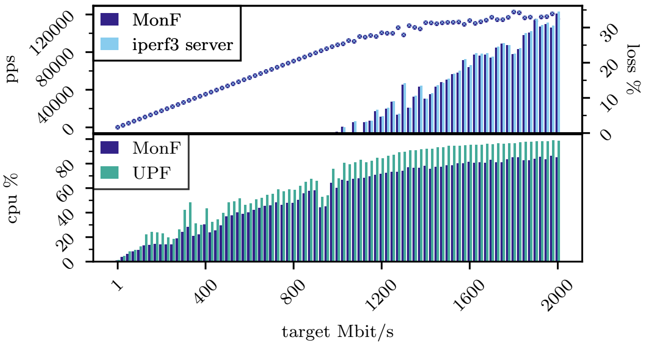

Fig. 3. For each target data rate generated with iPerf3, the upper plot shows the packet rates measured by our framework as dots and those measured by iPerf3's server as crosses, while the bars show the respective packet loss. The lower subplot shows the average CPU utilization of both our framework and the UPF.

The results show that our monitoring framework can accurately measure UPFs' packet rates and packet loss and that our framework can monitor the UPF even on high network loads. When the UPF's CPU utilization reaches 100% during our experiments the framework was at 80% and therefore not impaired by CPU limitations. However, this might not be the case for more efficient UPF implementations or under different circumstances, e.g, in virtual environments or when using different hardware. However, we leave a detailed analysis of resource utilization to future work, as this work focuses on assessing the accuracy of our framework.

B. Data Rate

Fig. 5 shows the data rates measured by our framework and by iPerf3. Each point's x-value represents the data rate measured by the iPerf3 server. The y-value represents the data rate measured by our framework. If a point lies on the diagonal, the iPerf3 server measured the same data rate as our framework. Up to 1 Gbit/s, the data rates measured by our framework align with the ones measured by iPerf3 and the MPE is 0.05%. For target data rates higher than 1 Gbit/s, the measured data rates start spreading, and we observed an MPE of 0.49%. We also did not measure data rates much higher than 1.5 Gbit/s even if we generated data rates of up to 2 Gbit/s. The reason for the spread and the limited data rate is that the UPF reaches its CPU utilization limit, as discussed in the previous section. The MPE over the whole data rate measurements is 0.27%.

One can see that the single data points per second align with the target diagonal up to 1 Gbit/s. This indicates that the deviation of our results from the ones provided by iPerf3 is negligibly small, even for the single data points. Regarding data rates higher than 1 Gbit/s, most of the points still lie on the diagonal. However, some outliers also indicate a deviation of up to 500 Mbit/s between the data rate measured by our framework and the one measured by iPerf3. This is because each point represents the average over all samples within one second, but packet loss does not occur equally distributed.

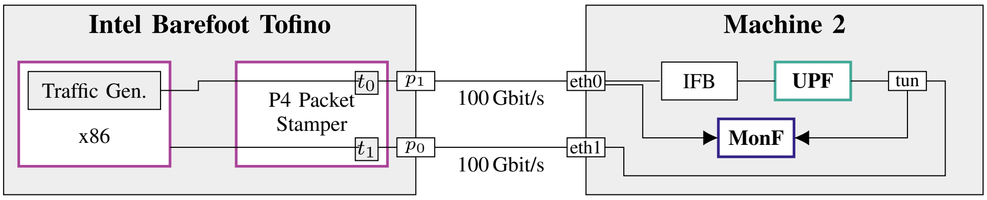

Fig. 4. To evaluate how accurately our framework measures latency, we used an Intel Barefoot Tofino and a Linux Machine (Machine 2). On Machine 2, we ran open5gs along with our framework and added an Intermediate Functional Block (IFB) between $eth0$ and the UPF to emulate latencies. We deployed P4STA on the Tofino and used it to generate GTP-U traffic, which was sent to Machine 2 via a 100 Gbit/s fiber, first passing the IFB and then the UPF. Then, the traffic was sent back to the Tofino via a second 100 Gbit/s fiber.

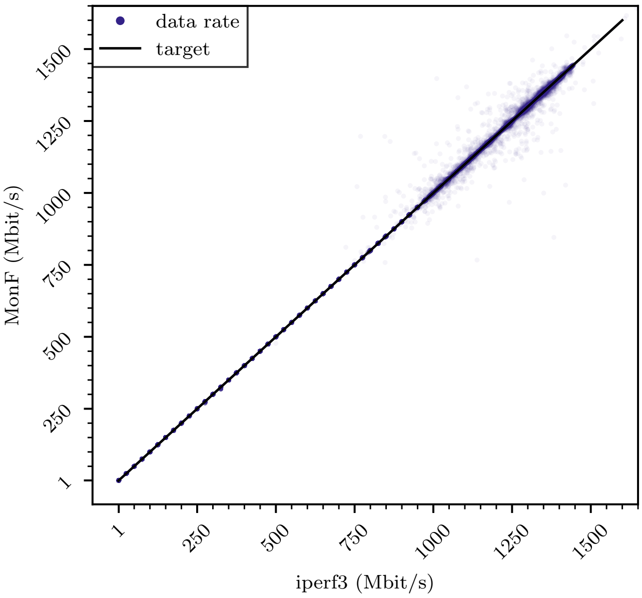

Fig. 5. Each point's x-value represents the data rate measured by our framework, while its y-value represents the one measured by iPerf3. If we measured the same data rate as iPerf3, the respective point lays on the diagonal.

Therefore, within some of the one-second intervalls, there are not enough samples to accurately determine the data rate.

C. Latency

To evaluate how accurately our framework measures the UPF's latency, we compared its results with those provided by P4STA [8], which is a framework leveraging hardware-based packet stamping to measure the latency of network devices. To use P4STA, we had to rebuild the test setup as outlined in Fig. 4. P4STA is executed on an Intel Barefoot Tofino, comprising an x86 unit and a P4 programmable switch ASIC, which runs the packet stamper. The packet stamper writes a timestamp t_0 into each leaving packet and a timestamp t_1 into each entering packet. The Tofino then calculates the delta between both timestamps in each packet to determine the latency. The Tofino has two external ports p_0 and p_1, both connected via an 100 Gbit/s fiber to an Ethernet interface $eth0$ and $eth1$ of Machine 2. We used the x86 unit on the Tofino to generate GTP-U encapsulated UDP traffic. All traffic is then sent through the packet stamper and via p_0 to $eth0$ of Machine 2. Machine 2 routes incoming GTP-U traffic through an intermediate functional block device (IFB) before it is decapsulated by the UPF. Our monitoring framework runs in parallel to the IFB and the UPF. Thus, we can artificially set the IFB's latency using traffic control emulation (tc-em) to simulate the UPF's latency. The encapsulated IP packets leaving the UPF are returned to the Tofino's p_1 via $eth1$.

For each experiment, we generated GTP-U encapsulated UDP traffic with 1 Gbit/s for 5 min. We repeated this and increased the delay of the IFB in steps of 5 µs starting from 0 µs up to 100 µs. Fig. 6 shows the averages for each of the experiments. The latencies measured by our framework and P4STA increase depending on the delay added by the IFB. As the UPF also adds some latency by itself, the latency measured by our framework is always higher than the delay added by the IFB by an average offset of 80.1 µs. The minimal offset is 76.72 µs and the maximal offset is 88.06 µs. The latencies measured by P4STA are again higher by an average offset of 41.47 µs than the ones measured by our framework with a minimum offset of 36.89 µs and a maximum offset of 44.47 µs. This is because the packets need some additional time to travel between the Tofino and Machine 2. When subtracting the average offset between the measurements between P4STA and our framework, the Mean Average Error (MAE) is 1.86 µs. The average standard deviation of the framework's measurements is 11.52 µs, indicating a high variation among the measured latencies per second. We assume that varying latencies caused by the UPF mainly cause this. In future work, we plan to obtain additional insights into that behaviour using P4STA's per-packet latency measurements.

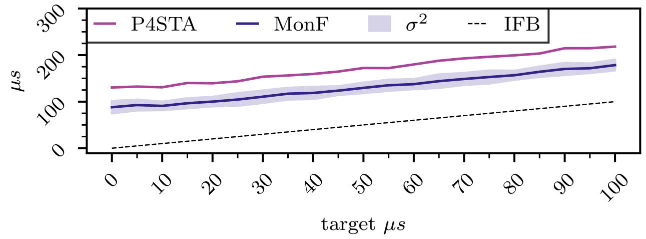

Fig. 6. We emulate UPF latencies of 0 µs to 100 µs and measure them simultaneously with our framework and P4STA. The plot shows the average measured latency for each emulated latency.

D. Limitations

Even if our approach yields all the concepts necessary to measure the performance of UPFs, it comes with some limitations that we will discuss in the following. We consciously did not consider most of these limitations in our design as they did not affect our evaluation. Currently, our framework ignores fragmented IP packets, as it would have to reassemble them to match them. This would add additional complexity and potential resource overhead, but has to be addressed by future work. Another problem arises when the same packet passes the UPF multiple times as with TCP retransmits. In this case, our framework cannot distinguish between these packets, potentially resulting in faulty results. We could overcome this by storing a counter with each ingress packet and increasing it every time we receive the same packet again. If we receive a matching egress packet and the counter is greater than one, we decrement it. Otherwise, the packet is deleted from the hash table. However, this approach must be further refined, as a timestamp for each packet is needed to determine packet loss. As the UPF does traffic policing, it might drop packets intentionally. Our framework does not distinguish between intentional drops and drops caused by overload. Therefore, the operator using our framework must further interpret the packet loss measurements it provides.

V. Conclusion

To enable resilience in future 6G networks, operators must be able to monitor the functionality of individual NFs. In this work, we presented a framework based on the NT concept to measure the performance of 6G UPFs in real time. The fully software-based framework can be easily hot-plugged into the UPF's execution environment. This enables a flexible deployment suiting the distributed SBA of 6G networks. We evaluated the accuracy of our framework by comparing its measurements to those of iPerf3 and P4STA. We demonstrated that our monitoring framework measures packet rates with an MPE of 0.2%, packet loss with an MPE of 0.13%, data rates with an MPE of 0.05%, and latency with 1.86 µs precision. These results show that monitoring 6G NFs with high accuracy is possible using a flexible and fully software-based framework. Our framework assumes that the UPF decreases the TTL of packets and that it adds or removes a GTP-U header. This concept of predicting egress packets based on ingress packets can be applied to other NFs in the CP to detect potential anomalies.

In future work, two approaches can be leveraged to improve the performance of our framework further. First, the framework can be implemented using the DPDK, which is optimized for building network applications by bypassing the Linux kernel. In addition to that, we see enhancing the way our framework samples packets as the most promising direction. Future work should elaborate on setting the sampling rate dynamically based on the packet rate; the higher the packet rate, the lower the sampling rate. This is important to prevent weak accuracy at lower packet rates and also to prevent high resource utilization for high packet rates. In this work, we measured the performance of UPFs, but our framework can be extended to monitor any NF. Therefore, the concept of matching egress with ingress packets to monitor the functionality of NFs should be pursued further.

Acknowledgment

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the project "Open6GHub" (grant number: 16KISK014).

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the *emergenCITY* center [LOEWE/1/12/519/03/05.001(0016)/72] and by the Federal Ministry of Education and Research of Germany (BMBF) through the project "Open6GHub" (grant number: 16KISK014).

References

[1] L. Khaloopour, Y. Su, F. Raskob, T. Meuser, R. Bless, L. Janzen, K. Abedi, M. Andjelkovic, H. Chaari, P. Chakraborty, M. Kreutzer, M. Hollick, T. Strufe, N. Franchi, and V. Jamali, "Resilience-by-design in 6g networks: Literature review and novel enabling concepts," *IEEE Access*, vol. 12, pp. 155666–155695, 2024.

[2] "5G; System Architecture for the 5G System (3GPP TS 23.501 version 15.3.0 Release 15)," 2018.

[3] A. Coates, A. Hero III, R. Nowak, and B. Yu, "Internet tomography," *IEEE Signal Process. Mag.*, vol. 19, no. 3, pp. 47–65, 2002.

[4] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, 2010.

[5] L. Mamushiane, A. A. Lysko, T. Makhosa, J. Mwangama, H. Kobo, A. Mbanga, and R. Tshimange, "Towards stress testing open5gs core (upf node) on a 5g standalone testbed," in *IEEE AFRICON*, pp. 1–6, 2023.

[6] G. Soós, D. Ficzere, P. Varga, and Z. Szalay, "Practical 5g kpi measurement results on a non-standalone architecture," in *IEEE/IFIP Netw. Operations and Manage. Symp.*, pp. 1–5, 2020.

[7] P. Orosz and T. Skopko, "Performance evaluation of a high precision software-based timestamping solution for network monitoring," *Int. J. on Advances in Softw.*, vol. 4, no. 1, 2011.

[8] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, "P4sta: High performance packet timestamping with programmable packet processors," in *IEEE/IFIP Netw. Operations and Manage. Symp.*, pp. 1–9, 2020.

[9] J. Rischke, P. Sossalla, S. Itting, F. H. P. Fitzek, and M. Reisslein, "5g campus networks: A first measurement study," *IEEE Access*, vol. 9, pp. 121786–121803, 2021.

[10] G. Kakkavas, A. Stamou, V. Karyotis, and S. Papavassiliou, "Network tomography for efficient monitoring in sdn-enabled 5g networks and beyond: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 59, no. 3, pp. 70–76, 2021.

[11] L. Wernet, L.-M. Spang, F. Siegmund, and T. Meuser, "Resilient user plane traffic redirection in cellular networks," in *IEEE Conference on Netw. Function Virtualization and Softw. Defined Netw.*, pp. 1–6, 2024.

[12] J. Drobisz and K. Christensen, "Adaptive sampling methods to determine network traffic statistics including the hurst parameter," in *IEEE Annu. Conf. on Local Comput. Netw.*, pp. 238–247, 1998.