Exploratory TEID-Based Defence against Flooding Attacks in 5G Slicing

2026-02-04, Author's copy: this paper is accepted for publication in IEEE ICC 2026.

Adam Ali Husseinat
Dept. of Systems and Computer Engineering
Carleton University
adamhusseinat@cmail.carleton.ca

Ashraf Matrawy
School of Information Technology
Carleton University
ashraf.matrawy@carleton.ca

***Abstract*—Network slicing, one of the main enablers of 5G networks, is susceptible to flooding attacks. As the case with other types of networks, spoofing the IP addresses of the attackers complicates the tasks of detection and mitigation. Traditional mitigation by blocking the detected source IPs might lead to impacting and blocking legitimate IP addresses.**

In 5G networks, each traffic stream between the User Equipment (UE) and the Network Core (NC) gets segregated and encapsulated into communication flows utilizing tunnels with distinct Tunnel Endpoint Identifiers (TEIDs) regardless of the inner IPs used in the traffic packets. In this paper, we leverage the TEID of each UE, in addition to traditinal IP and port filtering, to mitigate flood attacks when source IP addresses are spoofed in 5G slicing scenarios. This paper explores a mitigation solution for the 5G network architecture and combines protection layers: 1) IP and port-level filetring layer deployed at the User Plane Function (UPF), and 2) MAC and TEID (Tunnel Endpoint Identifier)-level layer deployed at the gNBs.

Our results show the effectiveness of the proposed solution in the hope of expanding research that utilizes new 5G architectural properties to provide defences that may not be possible in earlier architectures.

***Index Terms*—5G Network, Network Slicing, XR Traffic, iPerf Traffic, Slicing Security, Free5GC.**

I. Introduction

5G Network slicing partitions the physical infrastructure into logical networks, known as network slices, to allow resource sharing using different techniques of network virtualization. However, infrastructure and functional sharing present challenges, particularly security and privacy challenges.

The GPRS Tunneling Protocol (GTP) is a network protocol utilized in mobility networks such as 4G LTE and 5G. It segregates and encapsulates traffic into communication flows, utilizing tunnels with distinct Tunnel Endpoint Identifiers (TEIDs). It resides above the transport layer of the Internet Protocol suite, encapsulated by the User Datagram Protocol (UDP) within IPv4 or IPv6 headers. GTP facilitates uninterrupted sessions with the UPF and other services, permitting seamless transitions and routing modifications [1]. In this paper, we are leveraging the TEID to mitigate flooding attacks where the source IP addresses may be spoofed. allowing the mitigation of these flooding attacks without blocking the spoofed IPs.

In our earlier work [2], we studied the impact on flood attacks in 5G slicing on XR traffic and traffic generated by iperf [3]. We found that the flooding attacks have severe effects on this traffic in 5G slices; they change the traffic's characteristics (burstiness) and throughput, as illustrated in [2]. In the experiments of this paper, we utilize XR and iPerf traffic for testing and examine their characteristics (throughput and burstiness) through different 5G slicing configurations in the presence of different attacks (ping flood, UDP flood, and registration flood attacks).

Our **contributions** in this research are:

1) Detection and mitigation of flooding attacks (UDP flood, Ping flood, and Registration flood) at the UPF leveraging the IP address and the source port of the attack.
2) Leveraging the Tunnel Endpoints Identifier (TEID) to detect and mitigate the flooding attack at the next-generation node B (gNB), by mitigating the attack at the edge of the network, we aim to minimize the effect of these attacks on the other parts of the network.
3) Examining the effectiveness of our solution in maintaining the nature and characteristics of the XR traffic (throughput and burstiness) to mitigate the severe effects of flooding attacks on XR traffic (illustrated in our previous work [2]). We also show the results of the flooding attacks on iPerf traffic when our solution is deployed.

The rest of this paper is organized as follows: Section II summarizes the related works. Section III describes the system model and the experimental setup we consider in this work. Our proposed solution in section IV. Results are explained in Section V. Finally, Section VI concludes this work.

II. Related Work

The authors in [4] initially examine DoS/DDoS attacks on network slices and their effects on the performance metrics of slice users, including bandwidth and latency. They introduce a fresh dataset of DoS/DDoS attacks, gathered from a simulated 5G network slicing ttestbed and utilize it to construct a deep-learning-based bidirectional LSTM (Long Short Term Memory) model, referred to as SliceSecure, for the detection of DoS/DDoS attacks. Their model demonstrated a detection accuracy of 99.99% on the dataset they developed. The authors in [5] introduced a framework that utilizes open-source tools: Open5GS and UERANSIM for authentic 5G network emu-

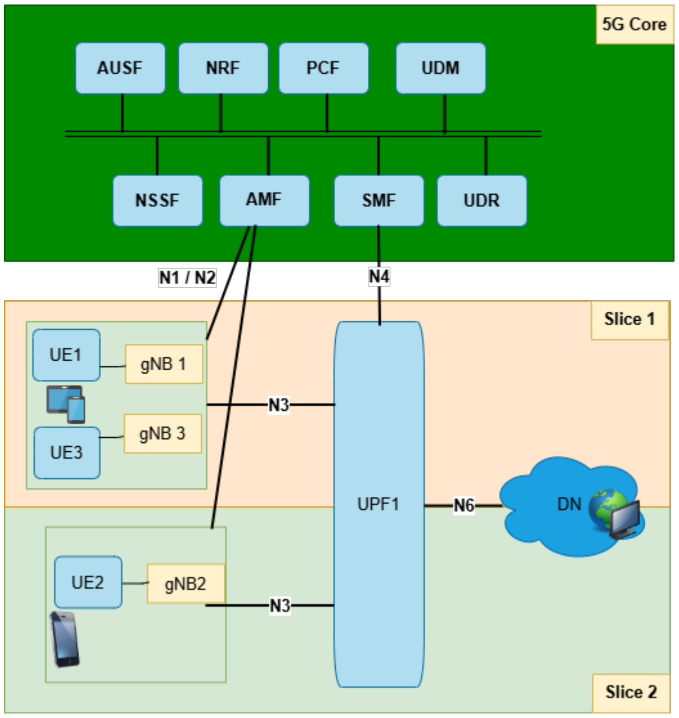

Fig. 1. Shared UPF and Shared SMF [2]

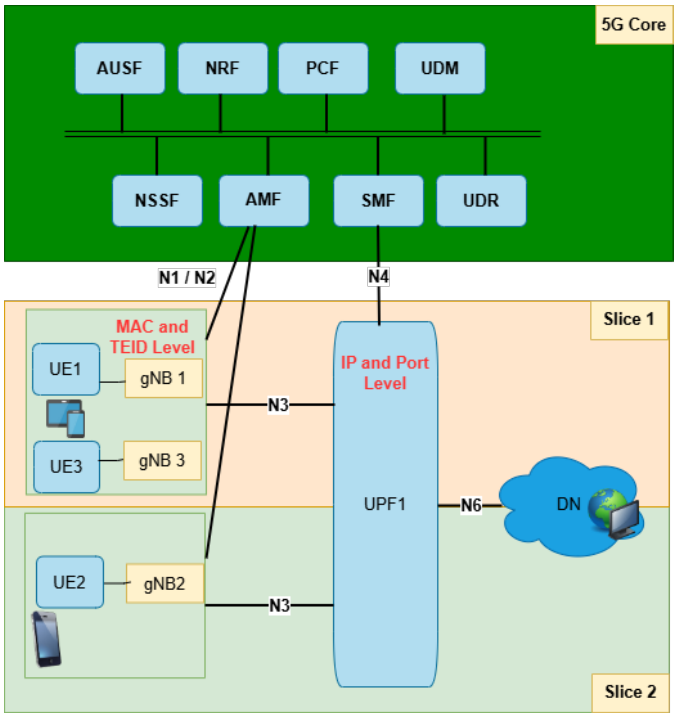

Fig. 2. Two-layer Mitigation Solution

lation, Docker for effective virtualization of the training infrastructure, and 5Greply for simulating attack situations. This creates a dynamic learning environment in which cybersecurity professionals may participate in real-time attack and countermeasure exercises, thereby enhancing their preparedness for 5G-specific cyber threats. They simulated three unique attack scenarios (Security Mode Command (SMC)-Reply, DoS, and DDoS attacks) and implemented countermeasures to illustrate the cybersecurity training. The authors in [6] propose a zero-touch security management solution using machine learning to detect and mitigate Distributed Denial-of-Service (DDoS) attacks on 5G CN components. It includes a novel closed-control loop, an ML algorithm for predicting MTC device attachment requests, a detection algorithm, and a mitigation algorithm that detaches the suspected MTC devices involved in the attack. The Quarantine Slice Manager architecture, a hierarchical and distributed two-level network design, is introduced in [7] to protect 5G networks from security holes by managing network slices well and responding quickly to possible threats.

The authors in [8] present an automated defense mechanism for software-defined networking (SDN) that integrates traffic analysis, external blacklist data, and dynamic capacity utilization to enhance load management, filtering, and service adaptability during attacks. The results demonstrate that the mechanism effectively sustains around 50% to 60% service levels, even when subjected to a significant onslaught. A novel actor-critic Reinforcement Learning model, Slice Isolation-based Reinforcement Learning (SIRL), is introduced in [9] utilizing five ideal graph features to construct a problem environment. SIRL exhibits a 54% superior covered requests ratio when compared to four non-RL models and nine state-of-the-art models.

All the above-mentioned related work did not investigate the effects of the flooding attacks on the XR traffic in the context of 5G slices, as our work does.

III. System Model

In this paper, we employed the same system model and setup we used in [2], which included the use of the emulators Free5GC and UERANSIMM to perform our experiments [10], [11]. We are only presenting the configuration that is associated with the results in Section V which has shared UPF and shared Session Management Function (SMF). As illustrated in Fig. 1, the network consists of the 5G core VM, including all functions except the UPF that has been deployed on another VM, the gNB and UE VM, and the DN VM. In Fig. 1, (AMF) is the Access and Mobility Management Function, (AUSF) stands for Authentication Server Function, (NRF) is the Network Repository Function, (PCF) is the Policy Control Function, (UDM) is the Unified Data Management, (NSSF) is the Network Slice Selection Function, and (UDR) is the Unified Data Repository.

In our experiments, we employed a configuration with two slices that share both the UPF and the SMF where we expect that flood attack impact to be severe as we learned in our earlier work [2]. See Fig 1 for the configuration. This configurations were established by allocating IP addresses to the various entities comprising the slice, ensuring exclusive usage by the users of that particular slice. The experiments consist of the 5G core (which includes the UPFs and the SMFs), two gNBs (gNB1 and gNB3) and (UE1, UE3) in slice 1, one gNB (gNB2) and UE2 in slice 2, and the DN.

IV. Proposed solution

Our mitigation solution, as shown in Fig. 2, is divided into two protection layers:

1) IP and port-level layer deployed at the UPF.
2) MAC and TEID (Tunnel Endpoint Identifier)-level layer deployed at the gNBs.

Both protection layers work in parallel to provide the complementary protection intended in our research.

A. IP and Port filtering at the UPF

In this layer of protection, we leverage the port traffic volume and the IP addresses associated with that traffic. This layer monitors the different ports' traffic and compares it with a predefined data rate threshold to identify any prospective flooding attack; the threshold can be defined based on different applications and slice use cases. When an attack is identified, the defense solution identifies the IP addresses responsible for the flooding attack and applies new IP-table rules to block that traffic at the specified port; hence, all the traffic from the identified IP addresses will be blocked. The tests we performed are based on the traffic volume and uses a threshold to trigger the solution rather than studying the performance in terms of the number of UEs.

We developed two algorithms;

- The first algorithm detects and mitigates the UDP flood attack: Algorithm 1.
- The second algorithm detects and mitigates the Ping flood (ICMP) attack: Algorithm 2.

Algorithm 1 UDP Flood Detection and Mitigation Algorithm

```
Require: Network interface (interface), Packet threshold (threshold)
Ensure: Detected and blocked attack traffic
1:  Function TrafficMonitoring(interface, threshold):
2:  sessions ← empty dictionary;
3:  while true do
4:      packet ← CaptureUDPPacket(interface);
5:      if packet is not null then
6:          source_port ← ExtractSourcePort(packet);
7:          sessions[source_port].packets ← +1;
8:          elapsed_time ← CurrentTime() - sessions[source_port].start_time;
9:          if elapsed_time ≥ 1 second then
10:             rate ← sessions[source_port].packets / elapsed_time;
11:             if rate ≥ threshold AND NOT sessions[source_port].is_blocked then
12:                 BlockTraffic(source_port);
13:             end if
14:             sessions[source_port].packets ← 0;
15:             sessions[source_port].start_time ← CurrentTime();
16:         end if
17:     end if
18: end while
19: Function BlockTraffic(src_port):
20: if NOT IsPortBlocked(src_port) then
21:     AddIPTableRule(”-t raw -I UDP_FLOOD -p udp –sport ” + src_port + ” -j DROP”);
22:     AddIPTableRule(”-I UDP_FLOOD -p udp –sport ” + src_port + ” -j DROP”);
23:     AddIPTableRule(”-t raw -I UDP_FLOOD -p udp –dport ” + GTP_PORT + ” -m u32 –u32 ’0 22&0x3C@8 16=” + src_port + ” -j DROP’”);
24:     MarkPortAsBlocked(src_port);
25:     LogBlockingAction(src_port);
26: end if=0
```

Algorithm 2 ICMP Flood Detection and Mitigation Algorithm

```
Require: Interface, PACKET_THRESHOLD
Ensure: Detected and blocked attack traffic
1:  Function MonitorTraffic(interface, threshold):
2:  sessions ← empty dictionary;
3:  cmd ← ”tcpdump -i ” + interface + ” -n ’icmp[icmptype]=8’”;
4:  while true do
5:      packet ← CapturePacket(cmd);
6:      if packet is not null then
7:          src_ip ← ExtractSourceIP(packet);
8:          sessions[src_ip].packets ← sessions[src_ip].packets + 1;
9:          elapsed_time ← CurrentTime() - sessions[src_ip].start_time;
10:         if elapsed_time ≥ 1 then
11:             rate ← sessions[src_ip].packets / elapsed_time;
12:             if rate ≥ 100 then
13:                 LogHighTraffic(src_ip, rate);
14:             end if
15:             if rate ≥ threshold AND NOT sessions[src_ip].is_blocked then
16:                 BlockICMPTraffic(src_ip);
17:             end if
18:             sessions[src_ip].packets ← 0;
19:             sessions[src_ip].start_time ← CurrentTime();
20:         end if
21:     end if
22: end while
23: Function BlockICMPTraffic(src_ip):
24: if NOT IsIPBlocked(src_ip) then
25:     AddIPTableRule(”raw”, ”ICMP_FLOOD”, ”icmp”, src_ip, ”DROP”);
26:     AddIPTableRule(”raw”, ”ICMP_FLOOD”, ”icmp –icmp-type echo-request”, src_ip, ”DROP”);
27:     MarkIPAsBlocked(src_ip);
28:     LogBlockingAction(src_ip);
29:     DisplayCurrentRules(”ICMP_FLOOD”);
30: end if=0
```

We deployed our solution in the UPF as a center point with respect to both attack kinds, as shown in Fig. 2; the XR (or iperf) traffic started from UE2 with four different traffic generation parameters and in four different slice configurations. The attacks were launched from UE1 and UE2, and the traffic was captured and analyzed. Our proposed solution mechanism is as follows:

1) The UPF monitors the traffic coming from all ports in all served slices.
2) The attack detection relies on examining the incoming traffic rate from each slice and each port against a certain traffic rate threshold that can be adjustable based on the Service Level Agreement or the heuristics of the traffic.
3) When an attack is detected, the algorithm applies new rules to the IP tables in order to mitigate the flood attack by dropping the attack traffic coming from the attacking source port or IP address.

B. TEID and MAC filtering at the gNodeB

We studied the case where an attacker would spoof the IP addresses of the attack traffic. This would result in the IP filtering defence blocking all the spoofed IP traffic by the algorithm deployed at the UPF. Leveraging the MAC address of the attacker and its TEID to mitigate the spoofed traffic scenario, we deployed the second layer of our solution at the gNB as a center point with respect to the attacking UEs connected to the same gNB. We did not implement or apply an actual spoofing attack. Rather, we assume that all traffic that comes from the same physical device is the spoofing attack traffic, which will have the same TEID. Algorithm 3 shows the TEID-based flood mitigation solution and its traffic-blocking function. The layer two mechanism is as follows:

1) The gNB monitors the traffic coming from all ports and IP addresses for all served slices.
2) The attack detection relies on challenging the incoming traffic rate from each slice and each port against a certain traffic rate threshold (adjustable) based on the Service Level Agreement or the heuristics of the traffic.
3) When an attack is detected, the algorithm deep-inspects the incoming attack traffic packets in all GTP tunnels and extracts the TEID of the attacking tunnels, then blocks these tunnels regardless of the packet's IP address that has been used.

The algorithm applies early packet dropping (mangle) rules to block the attacking UE if it is spoofing other IPs or has powerful capabilities to send that amount of traffic from its real IP. GTP blocking results in the attacker's disconnection. The way to deal with the attacker after being blocked depends on the actions available to the defenders and is outside the scope of this paper.

V. Results

In this section, we present our experiment results for three scenarios: (1) UDP attack mitigation, (2) Ping attack mitigation, and (3) TEID-based mitigation.

A. UDP Attack Mitigation

Algorithm 1 shows the UDP flood attack mitigation solution and its main traffic blocking function. It consists of the detection mechanism based on the traffic threshold that can be configured based on the slice application or use case and the blocking function that takes the input from the detection mechanism and applies new IP-table rules to block the specified attacking traffic source.

Our algorithm uses the IP tables and port blocking to apply new rules to drop all the UDP flood attack packets after detecting the attack. We deployed the solution and ran the same previous experiments in our previous work in [2] by sending the XR and iPerf traffic through the network, launching the UDP attack, and monitoring the traffic throughput and burstiness. The results show a very fast response from

Algorithm 3 TEID-Based Flood Mitigation Algorithm

```
Require: Network interface (interface), GTP port (port),
    Packet threshold (pkt_threshold)
Ensure: Blocked malicious UE traffic
 1: Function MonitorAndMitigate(interface, port,
    pkt_threshold):
 2: packets ← CaptureGTPTraffic(interface, port);
 3: for each packet in packets do
 4:     teid ← ExtractTEID(packet);
 5:     src_ip ← ExtractInnerIP(packet);
 6:     src_mac ← ExtractMAC(packet);
 7:     UpdatePacketCounter(teid);
 8:     rate ← CalculateRate(teid, window_size);
 9:     if rate > pkt_threshold then
10:         BlockMaliciousUE(teid, src_ip, src_mac);
11:     end if
12: end for
13: Function BlockMaliciousUE(teid, src_ip, src_mac):
14: // Block at MAC layer (ebtables)
15: AddEbtablesRule("-A GTP_FLOOD -s " + src_mac + "
    -j DROP");
16: // Block at IP layer (iptables)
17: AddIPTablesRule("-A INPUT -s " + src_ip + " -j DROP");
18: AddIPTablesRule("-A FORWARD -s " + src_ip + " -j
    DROP");
19: // Block GTP tunnel
20: AddIPTablesRule("-t mangle -A PREROUTING -p udp –
    dport 2152 " + "-m u32 –u32 '28=" + teid + " -j DROP"');
21: // Apply rate limiting
22: ConfigureTC("tc qdisc add dev " + interface + " root
    handle 1: htb");
23: ConfigureTC("tc class add dev " + interface + " parent 1:
    classid 1:1 htb rate 100kbit");
24: // Disrupt interface
25: ExecuteCommand("ip link set " + interface + " down");
26: Sleep(1);
27: ExecuteCommand("ip link set " + interface + " up");
28: LogBlockingAction(teid, src_ip, src_mac); =0
```

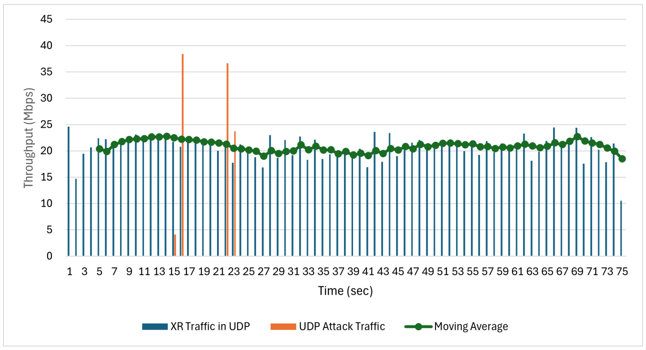

Fig. 3. XR (30 Mbps/30 fps) Shared UPF and SMF with UDP Attack

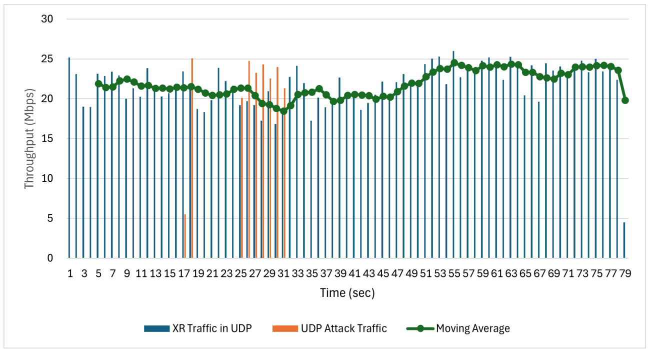

Fig. 4. XR (60 Mbps/60 Fps) Shared UPF and SMF with UDP Attack

our solution in detecting the attack from different ports and blocking it, resulting in undisrupted XR traffic and eliminating the attack effects found earlier in our experiments. Figures 3 and 4 show the results of our experiments applying our mitigation solution, and they show an instantaneous response from the presented solution in detecting the UDP attack (orange bars) and blocking its traffic at different XR generation parameters (data rate/frame size). Fig. 5 shows the results of the UDP attack on iPerf traffic with our solution deployed.

B. *Ping Attack Mitigation*

The Ping flood attack mitigation solution algorithm is shown in Algorithm 2, and its ICMP traffic blocking function. Our algorithm uses the IP tables and port blocking to apply new rules to drop all the Ping flood attack packets after detecting the attack. We deployed the solution and ran the same previous experiments by sending the XR traffic through the network, launching the Ping attack, and monitoring the traffic throughput and burstiness. Figures 6 and 7 show the results of our experiments applying our mitigation solution, and they show an instantaneous response from the presented solution in detecting the Ping attack (orange bars) and blocking its traffic at different XR generation parameters (data rate/frame size). Fig. 8 shows the ping attack's results on iPerf traffic with our solution deployed.

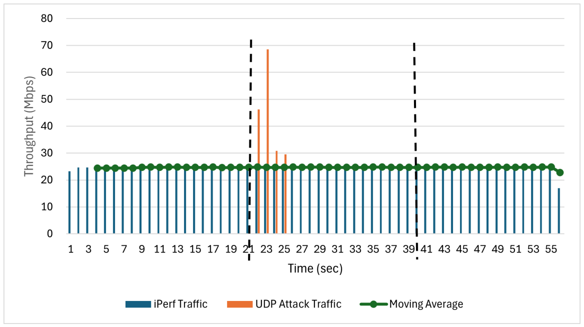

Fig. 5. iPerf Traffic with Solution Deployed under UDP Attack

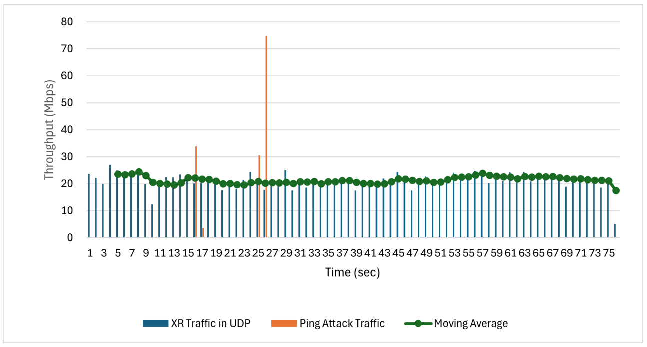

Fig. 6. XR (30 Mbps/30 Fps) Shared UPF and SMF with Ping Attack

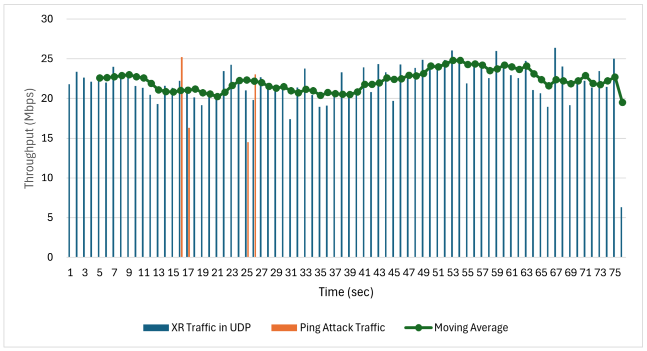

Fig. 7. XR (60 Mbps/60 Fps) Shared UPF and SMF with Ping Attack

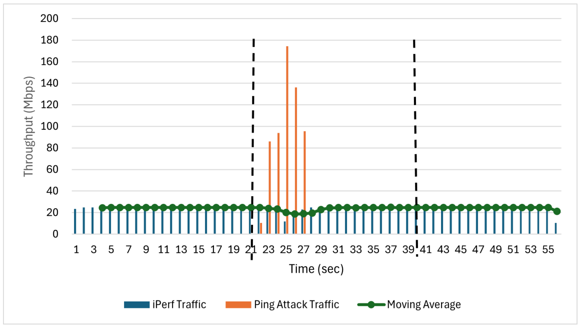

Fig. 8. iPerf Traffic with Solution Deployed under Ping Attack

C. *TEID-Based Mitigation*

Algorithm 3 defines our TEID-based flood mitigation mechanism and its ICMP traffic-blocking function. In the experiment, we applied a UDP attack, captured the traffic, and presented

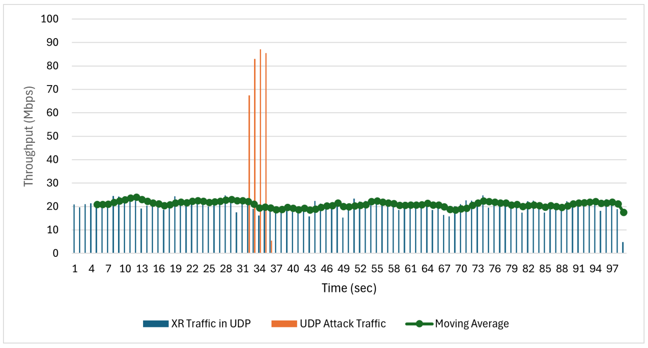

Fig. 9. TEID-based Solution Results

the results in Fig. 9.

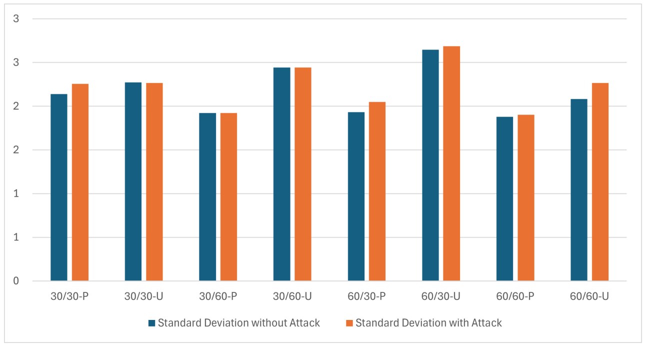

Fig. 10. Traffic Standard Deviation with the Solution Deployed

The results show a very fast response from our solution in detecting the flood attack (where we assume the IPs may be spoofed) and blocking it, resulting in undisrupted XR traffic and eliminating the attack effects found earlier in our experiments as shown in Fig. 9.

VI. Conclusion

By using the traffic rate threshold and leveraging the TEID of the UEs traffic, we explored our presented solution that could mitigate the flood attack in a two-layer scheme; the first layer is at the gNB, which identifies and blocks the attacking UE based on MAC and TEID, and the second layer is at the UPF, which blocks the attack traffic at the port and IP level.

The solution detects and blocks attacks from different ports, resulting in undisrupted XR traffic and eliminating previous attack effects. Fig. 10 shows the variance of the XR traffic with (blue bars) and without (orange bars) applying the attack, and it shows that our solution works against the Ping flood and UDP flood attacks, with a small difference in traffic throughput and burstiness when deploying the mitigation solution compared to the results without the solution found in our previous work [2].

Another note to add about XR traffic is that it is bursty and sensitive to delay and loss, hence we assume that if our mitigation solution works for this type of traffic, it might work with other types of traffic that less sensitive and less bursty, however, this remains to be tested in future work.

The solution detects UPD and Ping attacks from different UEs and reduces their effects by blocking flood attack packets upon reaching the UPF. It also considers the case if there is IP spoofing by identifying the attacking UE and stopping its traffic from all IPs it could use to spoof at the network's edge (gNB). This prevents extra resources in other network parts while allowing other users' traffic to use their TEID. The two-layer design stops the flood attack at the network's edge if it is detectable by the gNB or launched from different devices connected to the same gNB.

As mentioned earlier, this work is exploratory as we hope it encourages discussions and research that utilizes the new 5G architectural properties to provide defences that may not be possible in earlier architectures.

Acknowledgement

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] J. Collins, "Gprs tunneling protocol (gtp)," in *Encyclopedia of cryptography, security and privacy*. Springer, 2022, pp. 1–3.

[2] A. A. Husseinat, A. AbdulGhaffar, and A. Matrawy, "A study of xr traffic characteristics under flooding attacks on 5g slicing," *Authorea Preprints*, 2024, (updated version to appear in IEEE ICC 2025).

[3] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu. (2023) iPerf - The Ultimate Speed Test Tool for TCP, UDP and SCTP. ESnet/Lawrence Berkeley National Laboratory. A tool for active measurements of network bandwidth. [Online]. Available: https://iperf.fr/

[4] M. S. Khan, B. Farzaneh, N. Shahriar, N. Saha, and R. Boutaba, "Slicesecure: Impact and detection of dos/ddos attacks on 5g network slices," in *2022 IEEE Future Networks World Forum (FNWF)*, 2022, pp. 639–642.

[5] M. A. Hamza, U. Ejaz, and H.-c. Kim, "Cyber5gym: An integrated framework for 5g cybersecurity training," *Electronics*, vol. 13, no. 5, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/5/888

[6] R. Niboucha, S. B. Saad, A. Ksentini, and Y. Challal, "Zero-touch security management for mmtc network slices: Ddos attack detection and mitigation," *IEEE Internet of Things Journal*, vol. 10, no. 9, pp. 7800–7812, 2023.

[7] A. PC, A. Vashistha, Y. S. Rao, P. Yiyang, and S. Sumei, "Defending 5g networks against ddos attacks using quarantine slice manager architecture," in *2024 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, 2024, pp. 1–5.

[8] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding ddos mitigation and traffic management with software defined networking," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015, pp. 248–254.

[9] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Reinforcement learning-based slice isolation against ddos attacks in beyond 5g networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3930–3946, 2023.

[10] "Free5gc." [Online]. Available: https://free5gc.org/

[11] "GitHub - aligungr/UERANSIM: Open source 5G UE and RAN (gNodeB) implementation. — github.com," https://github.com/aligungr/UERANSIM, [Accessed 19-11-2024].