Offloading Media Traffic to Programmable Data Plane Switches

Elie F. Kfoury*, Jorge Crichigno*, Elias Bou-Harb†
*Integrated Information Technology, University of South Carolina, USA
†The Cyber Center For Security and Analytics, University of Texas at San Antonio, USA

***Abstract*—According to estimations, approximately 80% of Internet traffic represents media traffic. Much of it is generated by end users communicating with each other (e.g., voice, video sessions). A key element that permits the communication of users that may be behind Network Address Translation (NAT) is the relay server.**

This paper presents a scheme for offloading media traffic from relay servers to programmable switches. The proposed scheme relies on the capability of a P4 switch with a customized parser to de-encapsulate and process packets carrying media traffic. The switch then applies multiple switch actions over the packets. As these actions are simple and collectively emulate a relay server, the scheme is capable of moving relay functionality to the data plane operating at terabits per second. Performance evaluations show that the proposed scheme not only produces optimal results regarding Quality of Service (QoS) parameters (no packet loss, minimum delay, negligible delay variation, high Mean Opinion Score) but also scales much better than current solutions. Evaluations conducted with up to 35Gbps of media traffic or its equivalent of 400,000 simultaneous G.711 media sessions (limited only by the traffic generator rather than by the switch) show an ideal operation of the switch-based solution (using ∼1% of the switching capacity). In contrast, a relay server with a modern CPU model used for evaluations can process up to 900 simultaneous G.711 media sessions per core.

***Keywords*—Programmable switches, P4 language, Real Time Protocol, Network Address Translation, offloading.**

I. Introduction

The exponential increase in media traffic, fueled by the affordability of consumer electronics (e.g., smartphones, tablets), is rapidly changing the networking landscape. The growing number of applications generating media traffic is constantly requiring service providers to upgrade their infrastructure. Applications include WhatsApp [1] and Skype [2], which have more than 1.5 billion and 300 million monthly active users respectively [3]. Other examples are operators' platforms using standard protocols such as Session Initiation Protocol (SIP) [4] and Real-time Transport Protocol (RTP) [5]. According to estimations, media traffic represents approximately 80% of the total traffic over the Internet [6].

The infrastructure requirements are exacerbated by the network address translation (NAT), which is used by virtually all home and most enterprise and campus networks. Furthermore, recent studies show that the number of network operators deploying Carrier-grade NAT (CGN) is increasing [7]. CGN is a scheme that extends the traditional NAT to a large-scale deployment inside the service provider's network. Survey results [8] reveal that CGN has a widespread adoption and that over half of operators have deployed or will deploy CGN. NAT introduces issues such as violation of the end-to-end principle, scalability and reliability concerns, and traversal of end-to-end sessions. The latter is a problem that severely affects media traffic. For example, for an end user to be reachable for an end-to-end media session (voice, video), the user must wait and accept incoming connections at a well-known port. With NAT, the user is not reachable because it is assigned a private IP address. Furthermore, port numbers are also allocated dynamically. Moreover, these dynamic allocations interfere with the operation of signaling protocols, as end devices rely on opening ephemeral ports during the session establishment to send and receive media traffic [6].

Technical solutions to these problems include Session Traversal Utilities for NAT (STUN) [9], Traversal Using Relays around NAT (TURN) [10], Interactive Connectivity Establishment (ICE) [11], Port Mapping Protocol (PMP) [12], and Port Control Protocol (PCP) [13]. Essentially, the general solution of the NAT traversal problem requires the use of a publicly available server that acts as a relay between two end devices that may be behind NAT. Variants of this approach are widely implemented by media applications listed above. Using relay servers has associated costs to service providers, as these devices must be provisioned according to the number of users. Thus, an important open research problem is the design of a scalable network architecture to support media traffic.

A. Contribution

This paper presents a scheme for offloading media traffic from a relay server to a programmable switch. The scheme is motivated by the advent of new-generation P4 switches [14], which permit the programmer to describe the behavior of the data plane. The contribution of the paper is summarized as follows:

1) A working scheme is presented to offload media traffic from relay servers to programmable switches.
2) The number of media sessions supported by the proposed scheme is orders of magnitude higher than that supported by relay servers.
3) The scheme produces optimal results regarding the following Quality of Service (QoS) parameters: packet loss, delay, delay variation, Mean Opinion Score (MOS).
4) The prototype employs standard signaling and media protocols, namely SIP and RTP. Similarly, the switch's forwarding behavior is described using P4.

5) After the session establishment, the proposed approach fully offloads traffic to the switch and thus the CPU usage of the relay server becomes negligible.

The rest of the paper is organized as follows. Section II describes related work. Section III provides background information. Section IV describes the proposed system. Section V analyzes and compares the performance of the proposed system against a traditional system. Section VI describes the resource consumption and lessons learned. Section VII concludes the paper and describes future work.

II. Related Work

Many proposal has been presented to solve the NAT traversal problem using relay servers. STUN relies on a publicly available server that provides a method for a device behind NAT to determine its public IP address. While it solves the NAT traversal problem for some scenarios, it does not work when symmetric NAT is applied. The STUN architecture can be extended with TURN servers to support symmetric NAT [10]. In essence, the TURN server is a device that relays TCP or UDP packets carrying media traffic.

ICE [11] is a framework that was recently proposed as a general solution for the NAT traversal problem. ICE integrates both STUN and TURN servers and is frequently used in conjunction with other signaling protocols, such as Session Description Protocol (SDP) [15]. ICE only works if both end devices support it, and this requirement is one of the reasons why ICE has not gained much traction. Other efforts to solve the NAT traversal problem include the PMP [12] and PCP [13] protocols. PMP permits a device to request the NAT for its private-public IP addresses and port mappings. With this information, the device can communicate its parameters to another peer. PCP operates with IPv4 and IPv6 addresses and has an improved keep-alive procedure.

The above solutions are open standards and depend on the use of publicly available relay servers. While proprietary solutions such as Skype are not disclosed, the basic operations are inferred from measurements studies [2], [16]. For example, Skype nodes are organized into a hierarchical overlay network, with each node classified as a supernode or an ordinary node (users). When two parties behind NAT want to establish a session, a supernode is used to relay traffic between the two end devices. Note that the NAT traversal standards and proprietary solutions discussed above rely on relay servers. Independently of the signaling and media protocols, the relay server is a key component of the system.

Recent work on programmable switches includes NetCache [17], NetPaxos [18], and NetChain [19]. NetCache [17] uses programmable switches to detect, index, cache and serve hot key-value items in the switch data plane rather than in servers. NetPaxos [18] uses programmable switches to accelerate consensus protocols, but it does not offer a replicated key-value service, and the performance is bounded by the overhead of application-level replication on servers. NetChain [19] extends the ideas of NetCache and NetPaxos by offering a replicated key-value service, guaranteeing consistency, and handling switch failures. Key observations of these works [17]-[19] are the use of the data plane to store key-values into a table, from which they are then retrieved to serve queries for those keys at line rate, and the use of the control plane to populate the table. Kfoury et al. [20] also propose using programmable switches to measure and compute the TCP sending rate in high-speed networks (e.g., Science Demilitarized Zones [21]).

III. Background

A. Signaling and Media Protocols

The proposed scheme uses SIP [4] as the signaling protocol and RTP [5] as the media protocol. A SIP server/registrar maintains an index that maps a SIP identifier to current IP and port used for signaling. SIP is responsible for initiating, maintaining, and terminating multimedia sessions between communicating endpoints. SIP also encapsulates SDP [15] messages. SDP conveys media details, such as IP address, port, codec, and others. RTP runs over UDP and encapsulates the samples of the media signals (voice, video). Note that, in general, the IP address and port used for signaling (SIP) are different from those used for media (RTP).

B. Relay Server

The relay server is an intermediary device that allows RTP traffic to flow between end devices. Consider Fig. 1, where users A and B are behind NAT and the relay server has the IP address IP_R. The IP addresses and ports to be used by users A and B for RTP traffic are IP_A-P_A and IP_B-P_B respectively. This information is encapsulated in SIP/SDP. At the time of establishing the session, see Fig. 1(a), since RTP traffic is not generated yet, the NAT-translated IP addresses and ports are unknown. Assume that (1) user A initiates the session by sending an `INVITE` message including the SDP session information. The message is received by the SIP server and (2) forwarded to the relay server. The relay server (3) allocates a port P_{RA}, which will be used to receive RTP traffic on behalf of user A. The information IP_R-P_{RA} (4) is provided to the SIP server, which replaces the original information IP_A-P_A within the `INVITE`/SDP message, and (5) forwards it to user B. Once user B responds, a similar procedure occurs; the information IP_R-P_{RB} replaces IP_B-P_B within the SIP/SDP message forwarded to user A. The allocated ports are stored in the forwarding table at the relay server. As a result, when RTP traffic is generated, see Fig. 1(b), the relay server receives packets from both users. The relay server then learns the NAT-translated IP addresses and ports, $\text{IP}_{A'}$-$\text{P}_{A'}$ and $\text{IP}_{B'}$-$\text{P}_{B'}$, which enables it to establish the RTP channel.

C. Programmable Data Plane and P4

Several programmable switches implement the Protocol Independent Switch Architecture (PISA). PISA is an abstract processing model that consists of a programmable parser, programmable match-action pipeline, programmable deparser and programmable header/metadata bus that carry the headers and metadata throughout the pipeline. PISA provides protocol independence by allowing programmers to specify how a

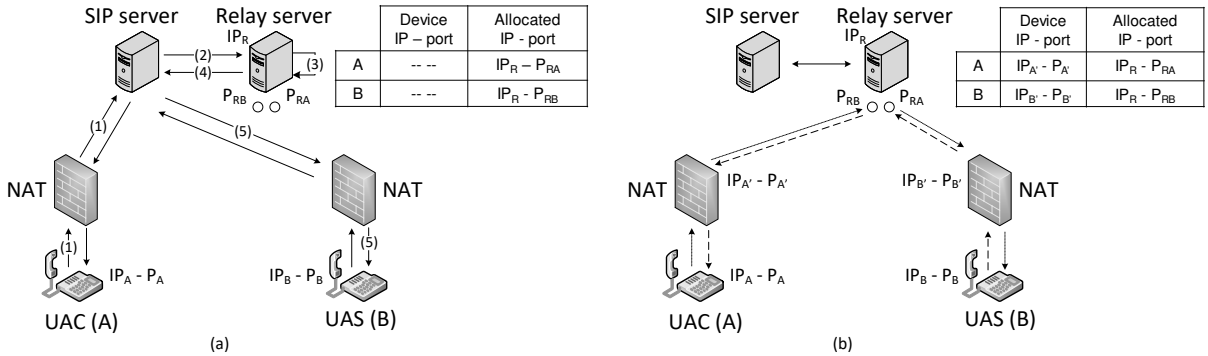

Fig. 1: Relay server mechanism. (a) Session establishment. IP_A-P_A and IP_B-P_B refer to the IP addresses and ports to be used for RTP traffic by users A and B, and IP_R is the IP address of the relay server. (1) User A sends an `INVITE`/SDP message; (2) the message is forwarded to the relay server, which (3) allocates the port P_{RA}, to be used to receive RTP traffic on behalf of user A; (4) the port allocation is provided to the SIP server, which incorporates the information into the `INVITE`/SDP message; (5) the SIP server forwards the `INVITE`/SDP message to user B. A response from user B initiates a similar process, with the relay server allocating port P_{RB}. (b) RTP traffic. When the session has been established, the relay server receives RTP packets. At this point, it learns the NAT-translated IP addresses and ports, $\text{IP}_{A'}$-$\text{P}_{A'}$ and $\text{IP}_{B'}$-$\text{P}_{B'}$, and stores them in the relay table. This information enables the relay server to establish the RTP channel.

packet should be parsed and processed by defining tables that match on specified fields in the packet or intermediate results (metadata), the actions that operate on the packet fields and metadata (including adding/removing headers) as well as as the processing algorithm itself [22]. As long as the P4 program compiles, it runs on the chip at line rate.

IV. Proposed System

A. Overview

The proposed architecture uses a programmable switch to emulate the behavior of the relay server, which must: 1) parse the incoming packet carrying media traffic from the first party, say user A; 2) identify the session this packet belongs to by using the 5-tuple {source IP, source port, destination IP, destination port, protocol}. The destination IP and the destination port in the incoming packet refer to a local socket in the relay server; 3) replace the source IP with that of the relay server and the source port with that used by the relay server to receive traffic from user B; 4) replace the destination IP and the destination port with the NAT-translated IP and port corresponding to user B; 5) recalculate both IPv4 and UDP checksums, and 6) forward the packet to user B. These steps can be implemented with match-action pipelines available at programmable switches. For each media session, the proposed scheme computes a unique session identifier given by the hash of the 5-tuple. The identifier is stored in a table by the control plane. When a packet arrives, the data planes parses the packet (step 1), identifies the session by matching the hash of the 5-tuple of the incoming packet to an entry (step 2), replaces the header fields (steps 3 and 4), recalculates checksums (step 5), and forwards the packet (step 6).

B. Session Establishment

Fig. 2 illustrates the proposed architecture. When the user A initiates a session, the following sequential events take place:

1) The user A sends a SIP `INVITE` message directed to the SIP server, addressed to user B.
2) The SIP server extracts and sends the SIP call-id to the relay server. The relay server allocates a new port for receiving media traffic on behalf of user A.
3) A monitoring agent learns the allocated port.
4) The user B replies back with a SIP `RESPONSE` message.
5) The SIP server forwards the message to the relay server, which in turns extracts the SIP call-id and performs a lookup on the existing sessions. The relay server allocates a port for receiving media traffic on behalf of user B.
6) The agent learns the allocated port.
7) The agent stores into a lookup table the ports allocated by the relay server, the NAT-translated IP addresses and ports of users A and B, and the 5-tuple hashes (one for each traffic direction). This table is referred to as `relay` and is similar to the relay table shown in Fig. 1(b).

When a packet carrying RTP traffic is received, the switch matches the 5-tuple hash of the packet with the hash entries in the `relay` table. If there is a match, it modifies the header fields and forwards the packet.

C. Monitoring Agent

Fig. 3 illustrates the main components of the agent. It learns the ports allocated to a media session by the relay server. The Rule Generator uses the source and destination IP addresses, protocol, and source and destination ports allocated to the

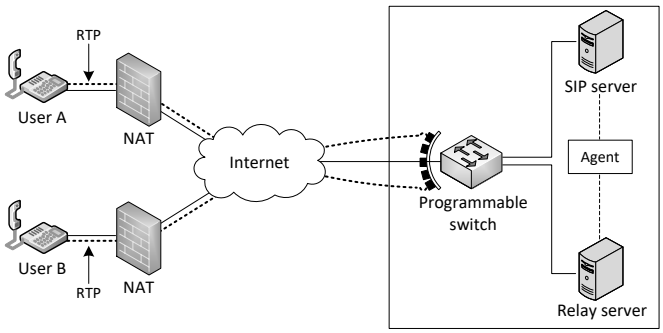

Fig. 2: System architecture.

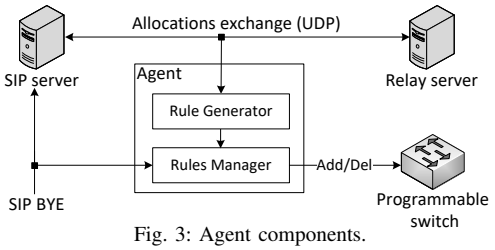

Fig. 3: Agent components.

media session to construct a unique session identifier. The Rules Manager then connects to the switch's control plane and adds a new entry into the `relay` table, which stores the identifiers of the media sessions currently traversing the switch and the new headers' values (i.e., source port, destination port, and destination IP address). The `relay` table is used by the data plane to match incoming UDP packets with a media session. The Rule Manager is also responsible for clearing media sessions allocated in the switch when a call is teared down. The Rule Manager intercepts SIP BYE messages received by the SIP server, determines the media session corresponding to the call identifier, and deletes allocated media sessions from the switch.

D. Switch Processing

The programmable switch implements regular functions of a layer-3 switch. Additionally, it matches UDP packets against the `relay` table. Pseudocode 1 describes the packet processing logic on the switch (excluding regular functionalities of a layer-3 switch). If the IPv4 and UDP headers are valid, then the session identifier (*index*) is calculated by hashing the 5-tuple. Subsequently, the packet is matched against the `relay` table, and upon having a *hit*, the switch modifies the packet headers to redirect the packet directly to the UAS/UAC without going through the relay server.

Pseudocode 1: PACKET PROCESSING(pkt)

```
1  eth ← parse(pkt.eth)
2  ipv4 ← parse(pkt.ipv4)
3  udp ← parse(pkt.udp)
4  if valid(ipv4) = true then
5      if valid(udp) = true then
6          index ← hash(5-tuple)
7          if relay.hit(index) = true then
8              eth.srcAddr ← eth.dstAddr
9              ipv4.srcAddr ← ipv4.dstAddr
10             ipv4.dstAddr ← relay[index].dstAddr
11             udp.srcPort← relay[index].srcPort
12             udp.dstPort ← relay[index].dstPort
```

V. Implementation and Evaluation

The topology used to conduct experiments is shown in Fig. 4. The following performance measures were used:

1) Delay: the time interval starting when a packet i is received from the UAC by the switch's ingress port, at time T_1^i, and ending when the packet is forwarded by the switch's egress port to the UAS, at time T_2^i. The delay of the packet i is computed as $D^i = T_2^i - T_1^i$. This metric measures the delay contributions of the switch and the relay server. To compute delay, the switch adds the timestamps T_1^i and T_2^i to each packet at the ports facing the UAC and UAS respectively, see Fig. 4. Timestamps are captured at the start of packet.
2) Delay variation: the absolute value of the difference between the delay of two consecutive received packets i and $i-1$. The delay variation is analogous to jitter, as defined by RFC 4689 [23], and is expressed as $\left|D^i - D^{i-1}\right|$.
3) CPU usage: the percentage of the CPU's capacity used by the relay server.
4) Packet loss: the number of packets that fail to reach the destination. The calculation is based on the sequence number field of the RTP header.
5) MOS: estimation of the quality of the media session based on the propagation delay, packetization delay, and jitter buffer. It is a reference quality indicator standardized by ITU-T [24].

The following components are used for the experiment: 1) OpenSIPS [26], an open source implementation of a SIP server. 2) RTPProxy [30], a high-performance relay server for RTP streams. 3) SIPp [25]: an open source SIP traffic generator that can establish multiple concurrent sessions and generate media (RTP) traffic. 4) Iperf [28]: traffic generator used to generate background UDP traffic.

Table I lists the hardware specifications. It also describes the software components along with their versions and additional remarks. The SIPp instances (UAC and UAS) were installed on two devices with Intel Xeon Silver 4114 CPU (four cores were allocated in each, 2.20GHz), running Debian GNU/Linux 9. The SIP and relay servers were both installed on two servers with the same hardware specifications. The relay server was allocated one core for relay purposes. For their communication, the SIP and relay servers use UDP. Edgecore Wedge100BF-32X [29] is the programmable switch used in the experiment. It is designed for high-performance data centers with programmable Tofino switch silicon. Wedge100BF-32X has 32 100Gbps ports, and thus can support up to 3.2Tbps. The experiment is limited to a single port of the switch, and 40Gbps link is used instead of 100Gbps.

Two scenarios are considered. The first scenario uses the relay server to relay media between end devices, without the intervention of the programmable switch. We refer to this scenario as "Server-based relay". The second scenario uses the programmable switch to relay media. We refer to this scenario as "Switch-based relay".

Scenario 1 – Server-based relay: In this scenario, the UAC

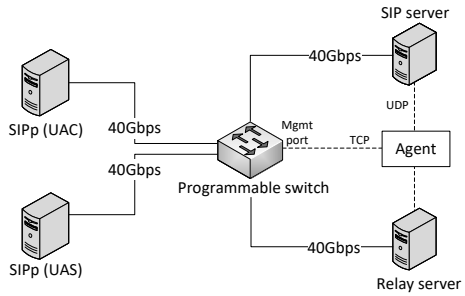

Fig. 4: Evaluation topology.

TABLE I: System components (software and hardware).

	Name	Version	Open Source	Remarks
Traffic generator	SIPp [25]	3.2	Yes	Generates concurrent sessions with RTP support.
Codec	G.711	N/A	Yes	Packetizes media traffic at a rate of 87.2Kbps.
SIP server	OpenSIPS [26]	2.4.6	Yes	Industry-grade, RFC 3261 [27] compliant.
Relay Server	RTPProxy [25]	2.1.0	Yes	High-performance software proxy.
Background traffic generator	iPerf [28]	2.0.9	Yes	Measurement tool, also used to generate traffic load.
Parser and Rule Generator	Custom Python Script	N/A	Yes	Intercepts allocations and installs/removes rules.
Stream analyzer	Wireshark/Tshark	3.0.6	Yes	RTP streams analysis and statistics.
P4 programmable switch	Edgecore Wedge100BF-32X [29]	N/A	No	Tofino switch silicon from Barefoot Networks.
CPU (UAC, UAS, servers)	Intel Xeon Silver 4114	N/A	N/A	2.20 GHz, 64 GB RAM.

(SIPp) generates 900 media sessions that include RTP streams to the UAS (SIPp). The number of simultaneous sessions was chosen empirically, and is considered a safe target. The authors noted that when the number of sessions is above 900, the relay server drops some packets as a consequence of the CPU usage. The rate at which sessions arrive is 30 per second, until 900 sessions are active. The test lasts for 300 seconds, and the codec used for media encoding is G.711.

Scenario 2 – Switch-based relay: the same traffic distribution as in scenario 1 was applied to scenario 2. The only difference is that the programmable switch now acts a relay server once the session is established.

A. Relay Server CPU Usage

Fig. 5 shows the percentage of CPU used by the relay server during the experiment with both scenarios. In the server-based relay scenario, the relay server consumes a significant portion of the CPU while new sessions are being added. The graph indicates that the CPU usage during this period increases up to 80%. Sessions stop arriving at t = 30, when a total of 900 sessions are active. The consumption drops to approximately 50% at t = 50, 20 seconds after sessions stop arriving. On the other hand, in the switch-based relay, the relay server peaked at 15% for a short period of time as sessions arrive at 30 per second. At t = 40, the CPU usage starts decreasing. The CPU consumption is approximately zero once sessions stop arriving. At this point, although 900 sessions are simultaneously active, they are fully processed by the switch.

B. Delay

Fig. 6 (a) shows the Cumulative Distribution Function (CDF) of the delay observed during the experiments in both scenarios. The CDF was formed with the data set composed of the delay measurements for all packets in both scenarios.

The delay of the server-based relay ranges from 0.250 milliseconds (ms) to 17ms, with a mean (μ) of 6.88ms and a standard deviation (σ) of 3.74ms. Approximately 50% of the packets experienced a delay above 7.5ms, and 20% of the packets have delays above 10ms. Note that the observed delay is mostly a consequence of the software component in the relay server (e.g., Linux's process scheduling, varying interrupt processing delay). The delay of the switch-based relay is almost-constant, 440 nanoseconds.

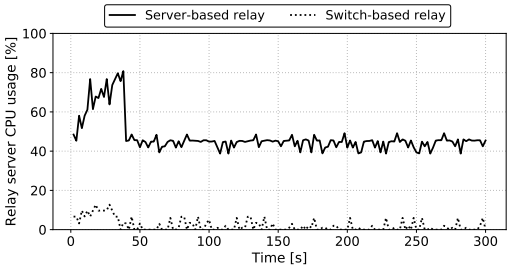

Fig. 5: Relay server CPU usage.

C. Delay Variation

Fig. 6 (b) shows the CDF of the delay variation observed during the experiments in both scenarios. The CDF was formed with the data set composed of the calculations of delay variation in both scenarios. The delay variation of the server-based relay ranges from 100 microseconds (µs) to 3ms, with a mean of 0.26ms and a standard deviation of 0.57ms. On the other hand, the delay variation of the switch-based relay is negligible. The proposed solution relies on the precise timing characteristics of the switch, removing the delay variation introduced by the relay server.

D. Packet Loss Rate

Fig. 6 (c) shows the CDF of the packet loss rate observed during the experiment in the server-based relay scenario. The loss rate varies from 0.15% to 0.5%, and all sessions experience some packet losses over time. Note that the performance tests deliberately generated a maximum of 900 active media sessions simultaneously. The authors observed that this was the safe limit supported by the relay server. Beyond 900 sessions, the relay server started to experience packet losses. In contrast, the switch-based relay did not experience packet losses.

E. Maximum Number of Sessions

In the switch-based relay, the current testbed cannot stress test the capacity of the switch (this has a switching capacity of 3.2Tbps). According to measurements conducted in the lab, the bottleneck of the testbed is the device hosting the UAC (specifically, the CPU usage of this device reached 100%). As the number of active sessions exceeds 1800, the number of packets dropped in the system significantly increases.

To further test the proposed scheme, a third scenario was created. The third scenario used iPerf instead of SIPp to emulate media traffic. The UAC generates dummy media session in main memory, which is then moved down through the protocol stack and over the network media. The UAS receives the data and moves it up through the protocol stack. No input/output operations occur with iPerf, which reduces

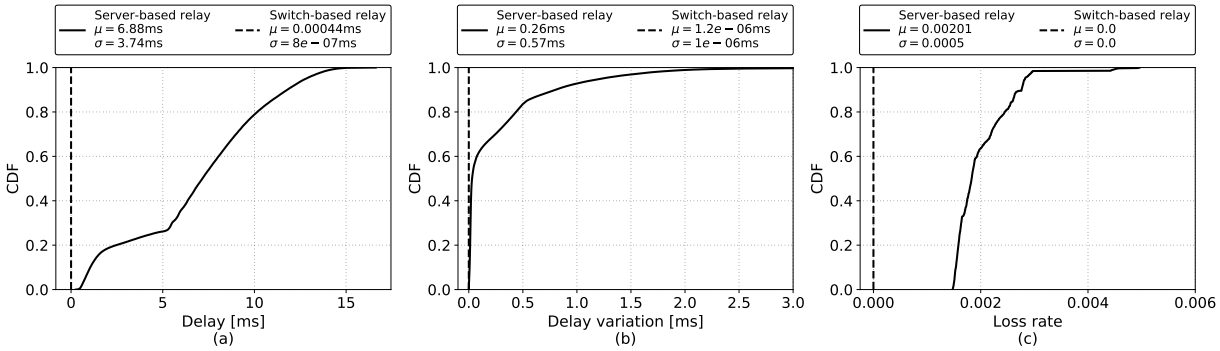

Fig. 6: CDF of delay (a), delay variation (b), and loss rate (c).

the CPU usage. The UAC generated approximately 35Gbps of dummy media traffic. Additional forwarding rules were inserted in the lookup table of the programmable switch, so that to relay the dummy media traffic from UAC to UAS.

Measures obtained in the scenario showed that the programmable switch relays 35Gbps of media traffic without experiencing packet losses. Other measures (delay, delay variation) are consistent with those obtained in scenarios 1 and 2. These results confirm the capability of the switch to relay 35Gbps of media traffic, or its equivalent of 400,000 media sessions operating with G.711 codec (each session generates 87.2Kbps of media traffic). In contrast, the relay server can serve up to 900 sessions per core before the QoS deteriorates.

F. Mean Opinion Score

Fig. 7 reports the MOS CDF considering three different scenarios: the first with 750 simultaneous sessions, the second with 1500 sessions, and the third with 1800 sessions. When the number of sessions is 750, the network resources are underutilized and the two schemes attain the maximum score for G.711 ($\sim$4.4). As the number of simultaneous sessions increases to 1500 and 1800, the MOS values of the server-based relay scheme decrease; for example, for 1800 simultaneous sessions, approximately 60% of sessions have a MOS score below $\sim$3.7. With this score, only some users are satisfied [24].

VI. Resource Consumption and Lessons Learned

A. Resource Consumption

For each media session, the proposed system stores in RAM a session identifier, given by the hash of the 5-tuple (match), and the header fields to be modified (action data). The hash is 32-bit long and the action data is 64-bit long, for a total of 96 bits per media session. To evaluate the overhead in terms of hardware resources, the prototype is implemented in two different scenarios: 1) on top of the baseline switch program (switch.p4), and 2) standalone implementation. The baseline switch.p4 implements various networking features needed for typical cloud data centers, including Layer 2/3 functionalities, Access Control List (ACL), QoS, etc. With the switch.p4 program, the relay table accommodates 64,000 media sessions by increasing the SRAM usage by 16.2% beyond the usage of switch.p4. As a standalone program, the relay table was able to accommodate up to 1,050,000 entries for media relay, with some additional resources to spare. Table II shows the resources used in both cases. Note the linear-like increase in SRAM utilization when increasing the table size.

B. Lessons Learned

Advantages of offloading relay functionality to the switch include: a) performance: the number of media sessions supported by the switch-based solution is orders of magnitude larger than that of the server-based solution; b) QoS: the switch-based solution exhibits optimal results regarding QoS parameters; c) flexibility: using a P4 switch permits to forward packets using non-standard fields. In the proposed approach, a packet is forwarded based on its session identifier; d) timing information: the authors noted that measuring delay and delay variation on the P4 switch results in precise high-resolution timing information.

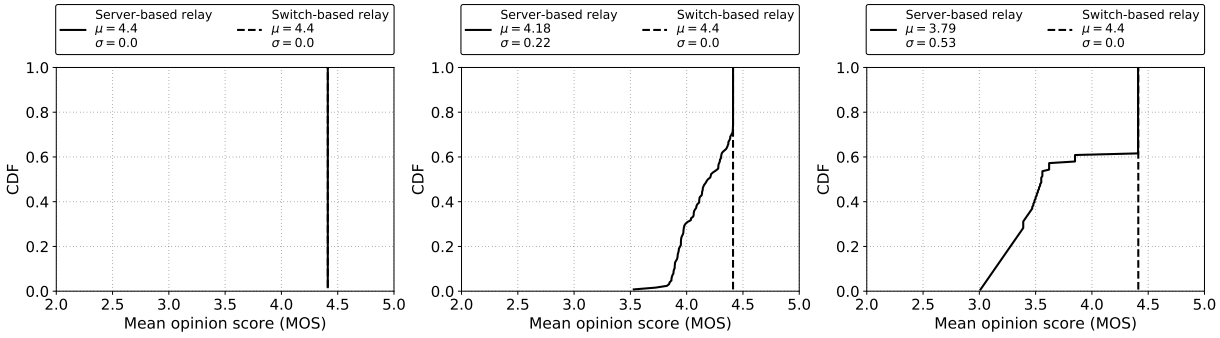

Fig. 7: CDF of Mean Opinion Score (750, 1500, and 1800 simultaneous sessions).

TABLE II: Additional hardware resources used by the offloading feature when deployed on top of switch.p4 and as a standalone program.

On top of switch.p4			
Table size	**SRAM**	**Hash Bits**	**TCAM**
32,000	+8.45%	+2.7%	+0%
64,000	+16.2%	+4.6%	+0%
Standalone program			
Table size	**SRAM**	**Hash Bits**	**TCAM**
500,000	———	———	——-
1,000,000	+97.84%	+86.4%	+0%
1,050,000	+107.5%	+89.8%	+0%

From Table II, note that the P4 switch enables the programmer to free unused resources (e.g., SRAM consumed by features on switch.p4) and customize the program, in order to accommodate additional sessions. Also, avoiding complex application logic and using simple operations (implemented with few match-action tables) facilitate the implementation of the switch-based relay.

VII. Conclusion and Future Work

This paper presents a scheme for offloading media traffic from relay servers to P4 programmable switches. The scheme permits to move media relay functionality from a general-purpose server to the data plane of the switch operating at terabits per second. Performance evaluations show that the proposed scheme not only provides optimal results regarding QoS parameters but also scales much better than current solutions based on general-purpose CPUs. For example, evaluations conducted with up to 35Gbps of media traffic or its equivalent of 400,000 simultaneous G.711 media sessions (limited only by the traffic generator rather than by the switch) show an ideal operation of the proposed solution (using $\sim$1% of the switching capacity). As a reference, the relay server with a modern CPU model used for evaluations can process up to 900 simultaneous G.711 media sessions per core before QoS deteriorates. Future work intends to offload the signaling traffic required to establish and tear down media sessions.

Acknowledgement

This work was supported by the U.S. National Science Foundation, awards 1829698 and 1907821. The authors would like to acknowledge Vladimir Gurevich from Barefoot Networks, an Intel Company. He provided insightful feedback on measuring delay and delay variation. He also provided helpful advice on various technical issues examined in the paper.

References

[1] K. P. O'Hara, M. Massimi, R. Harper, S. Rubens, and J. Morris, "Everyday dwelling with WhatsApp," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pp. 1131–1143, ACM, 2014.

[2] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang, "Profiling Skype video calls: rate control and video quality," in *2012 Proceedings IEEE INFOCOM*, pp. 621–629, IEEE, 2012.

[3] CNBC Tech News, "Microsoft's Skype gets a redesign, ditching Snapchat-like feature' highlights." https://tinyurl.com/ycsmk55f, 2018.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, *et al.*, "RFC 3261 - SIP: Session initiation protocol," 2002.

[5] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne, "RFC 3550 - RTP: A transport protocol for real-time applications," 2003.

[6] H. W. Barz and G. A. Bassett, *Multimedia networks: protocols, design and applications*. John Wiley & Sons, 2016.

[7] I. Livadariu, K. Benson, A. Elmokashfi, A. Dhamdhere, and A. Dainotti, "Inferring carrier-grade NAT deployment in the wild," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2249–2257, IEEE, 2018.

[8] P. Richter, F. Wohlfart, N. Vallina-Rodriguez, M. Allman, R. Bush, A. Feldmann, C. Kreibich, N. Weaver, and V. Paxson, "A multi-perspective analysis of carrier-grade NAT deployment," in *Proceedings of the 2016 Internet Measurement Conference*, pp. 215–229, ACM, 2016.

[9] D. Wing, P. Matthews, R. Mahy, and J. Rosenberg, "RFC 5389 - STUN: Session traversal utilities for NAT," 2008.

[10] M. Petit-Huguenin, S. Nandakumar, G. Salgueiro, and P. Jones, "RFC 7566 - TURN: Traversal using relays around NAT (TURN) uniform resource identifiers," 2013.

[11] J. Rosenberg and C. Holmberg, "RFC 8445 - ICE: Interactive connectivity establishment: a protocol for Network Address Translator (NAT) traversal," 2018.

[12] M. Krochmal and S. Cheshire, "RFC 6886 - NAT-PMP: NAT port mapping protocol," 2013.

[13] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk, "RFC 6887 - PCP: Port control protocol," 2013.

[14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[15] M. Handley, C. Perkins, and V. Jacobson, "RFC 4566 - SDP: Session description protocol," 2006.

[16] J. F. Kurose, *Computer Networking: A top-down approach featuring the Internet, 7/E*. Pearson, 2017.

[17] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: balancing key-Value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 121–136, ACM, 2017.

[18] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, "Netpaxos: consensus at network speed," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 5, ACM, 2015.

[19] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: scale-free sub-RTT coordination," in *15th USENIX Symposium on Networked Systems Design and Implementation NSDI-18*, pp. 35–49, 2018.

[20] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, and G. Srivastava, "Enabling TCP pacing using programmable data plane switches," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 273–277, 2019.

[21] J. Crichigno, E. Bou-Harb, and N. Ghani, "A comprehensive tutorial on science dmz," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, Secondquarter, 2019.

[22] Barefoot Networks, "The world's fastest & most programmable networks." https://tinyurl.com/r9kk2hf, 2019.

[23] S. Poretsky, J. Perser, S. Erramilli, and S. Khurana, "RFC 4689: Terminology for benchmarking network-layer traffic control mechanisms," 2006.

[24] J. A. Bergstra and C. Middelburg, "ITU-T recommendation G. 107: the E-Model, a computational model for use in transmission planning," 2003.

[25] R. Gayraud and O. Jacques, "Sipp 2.0 reference documentation," 2004.

[26] F. E. Goncalves and B.-A. Iancu, *Building Telephony Systems with OpenSIPS*. Packt Publishing Ltd, 2016.

[27] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "Session initiation protocol," 2002.

[28] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool.," p. 38, 2005.

[29] "Wedge 100BF-32X, 100GbE data center switch." https://tinyurl.com/sy2jkqe.

[30] Sippy, "RTPproxy." https://github.com/sippy/rtpproxy, 2009.