Performance Evaluation of GTP-U and SRv6 Stateless Translation

Chunghan Lee* Kentaro Ebisawa* Hitoshi Kuwata† Miya Kohno‡ Satoru Matsushima§

* Toyota Motor Corporation † APRESIA Systems,Ltd. ‡ Cisco Systems § SoftBank Corp.
* {chunghan_lee, kentaro_ebisawa}@mail.toyota.co.jp † hitoshi.kuwata.gt@apresiasystems.co.jp
‡ mkohno@cisco.com § satoru.matsushima@g.softbank.co.jp

***Abstract*—The GPRS Tunneling Protocol User Plane (GTP-U) has long been deployed for GSM, UMTS and 4G LTE. Now for 5G, IPv6 Segment Routing (SRv6) has been proposed as an alternative user plane protocol to GTP-U in both 3GPP and IETF. SRv6 based on source routing has many advantages: stateless traffic steering, network programming and so on. Despite the advantages, it is hard to expect to replace GTP-U by SRv6 all at once, even in a 5G deployment because of a lot of dependencies between 3GPP nodes. Therefore, stateless translation and co-existence with GTP-U have been proposed in IETF. However there are no suitable measurement platform and performance evaluation results between GTP-U and SRv6. In particular, it is hard to measure latency on commercial traffic generators when a receiving packet type is different from a sending packet type. In this paper, we focus on the performance evaluation between GTP-U and SRv6 stateless translation. We designed an SRv6 measurement platform using a programmable switch, and measured GTP-U and SRv6 functions with pre-defined scenarios on a local environment. Well-known performance metrics, such as throughput and packets per second (PPS), are measured by the traffic generator while the latency at the functions was measured using telemetry on our SRv6 platform. In our evaluation, we cannot find the abrupt performance drop of well-known metrics at SRv6 stateless translation. Moreover, the latency of SRv6 stateless translation is similar to GTP-U and their performance degradation is negligible. Through the evaluation results, it is obvious that the SRv6 stateless translation is acceptable to the 5G user plane.**

***Index Terms*—SRv6, GTP-U, 5G, P4, Mobile user plane, Measurement**

I. Introduction

With the explosive growth of smartphones and the rapid developments of cloud computing and mobile technology, mobile networks have been drastically evolving as 5G networks that have low latency and high throughput. For the mobile user plane, the GPRS Tunneling Protocol User Plane (GTP-U) has been deployed for GSM, UMTS, and 4G LTE networks.

IP is connectionless, and has a potential to mitigate session load, so IPv6 Segment Routing (SRv6) [1][2][3] has been proposed as an alternative user plane protocol in both 3GPP and IETF. SRv6 based on source routing has quite a few advantages: stateless traffic steering, common and simplified dataplane across domains, state reduction and service programming [4]. However, it is hard to expect to replace GTP- U by SRv6 all at once, even in a 5G deployment because of a lot of dependencies between 3GPP nodes. Currently, a method [5] for stateless translation between GTP-U and SRv6 has been proposed, and co-existence with GTP-U has been also discussed for the 5G user plane within IETF.

Unfortunately, there are no quantitative performance evaluation results between GTP-U and SRv6. Thus, it is hard to clarify the validity of SRv6 stateless translation method. Moreover, there is no suitable measurement platform although some GTP-U and SRv6 functions are supported by CPU-based and ASIC-based platforms. In CPU-based platform, it would be hard to achieve the pure performance evaluation for the 5G networks. Alternatively, ASIC-based platform as a commodity programmable switch, such as Barefoot Tofino [6], is reasonable for the evaluation.

Although several SRv6 functions are implemented on the programmable switch [7], there are problems with existing p4 code. The first problem is that there are no implementations of translation functions. The second one is that unnecessary switching and routing functions are involved to the SRv6 functions and they are widely spread out hardware resource. The last one is that we cannot measure the latency on a commercial traffic generator, such as IXIA, when packet types are changed by the translation functions on the programmable switch. In order to solve the problems, we should design and implement GTP-U and SRv6 translation functions with the latency measurement.

Our research goal of this paper is to evaluate the quantitative performance of stateless translation between GTP-U and SRv6, and to clarify the possibility of co-existence of GTP-U and SRv6. To achieve our goal, we firstly designed and implemented GTP-U and SRv6 stateless translation functions with the minimum resource on the programmable switch. Next, we injected nanosecond timestamp to a packet header as telemetry to measure the latency at the functions. Lastly, we measured well-known performance metrics, such as throughput, packet loss and packets per second (PPS), at the functions under light (100Mbps) and heavy (100Gbps) conditions on a local environment.

This paper shows the quantitative performance evaluation of GTP-U and SRv6 stateless translation using the programmable switch. We clarified that there are no significant different of well-known metrics between GTP-U and SRv6 on the local environment. Although the latency at the SRv6 stateless translation functions is slightly high in comparison with GTP-

978-3-903176-24-9

U and their impact is small under the heavy condition. Finally, a discussion of significant implications for future 5G user plane derives from the basis of our performance evaluation.

The rest of this paper is organized as follows. We first introduce related work for SRv6 and discuss what are different points in Section II. Next, we describe the overview of SRv6 stateless translation and our measurement method for the stateless translation in Sections III and IV. Then, we present evaluation results on the local environment and discuss implications from our evaluation in Section V. Finally, we conclude with a summary of the main points in Section VI.

II. Related work

Many SRv6 applications [8][9] which could be applicable to Datacenter Network (DCN)s were widely proposed. In the applications, the Segment Routing Headers (SRH) [3] was efficiently used to apply both traffic steering and service policies. Lebrun et al. [10] designed and implemented Software Resolved Networks (SRNs) based on Software Defined Networking (SDN) to apply SRv6 to enterprise networks. In SRNs, a DNS server was extended as an SDN controller and it provided dedicated network paths to users for steering their traffic. Y.Desmouceaux et al. [11] applied SRv6 for network load balancer (LB). SRv6 forwarded packets from a new connection through a set of candidate servers until the connection is accepted to a dedicated server. Ventre et al. [12] designed and implemented Southbound API between an SDN controller and the SRv6 device. gRPC, REST, NETCONF, and remote command line interface (CLI) were implemented to analyze performance differences when each protocol is used as Southbound API. Finally, there were no previous works using the programmable switch as the SRv6 platform supporting the mobile user plane.

Some SRv6 applications have been relied on SRv6 extension in the Linux kernel. Duchene et al. [13] proposed SRv6Pipes to enable service function chain (SFC) based on mapping the SRH to bytestream, such as TCP flow, by using some kernel system calls. In SRv6Pipes, SR-Aware TCP proxy provided transparency at network stack in the kernel. To perform several network functions on the proxy, a large enough IPv6 address space is allocated for the proxy, the function, and the parameters of the function. SRv6 test framework (SRPerf) was also proposed by Abdelsalam et al. [14]. Their evaluation focused on the performance of SRv6 functions, such as *T.Encaps*, *T.Insert*, *End.X*, *End.DX6*, and so on, in the kernel. Poor performance was observed by both *End.X* and *End.DX6*. The major cause of poor performance was that caches on routing tables were not activated although a next-hop was already known. Y.Desmouceaux [15] applied eBPF to SRv6 to process SRHs in the kernel. They showed use cases of SRv6: LB, ECMP nexthop, and latency measurement. Again, it would be hard to expect to measure the pure performance on the kernel. In summary, our performance evaluation results were fundamentally different from those used in previous works.

III. Overview of the stateless translation

A. GTP-U

GTP-U, specified in 3GPP, is currently a mainstream mobile user plane protocol and it is a connection-oriented protocol. A GTP-U tunnel is used to carry encapsulated Transport Protocol Data Unit (T-PDU)s. The Tunnel Endpoint ID (TEID), which is present in the GTP header shall indicate which tunnel a particular T-PDU belongs to. A value of TEID is allocated on each endpoint and indicates which tunnel a particular T-PDU belongs to [16]. In this manner, packets are multiplexed and de-multiplexed by GTP-U between a given pair of tunnel endpoints [17]. Thus, the same number of TEIDs is required to process the same number of sessions.

B. SRv6

Segment Routing (SR) [2] leverages the source routing paradigm with abstraction of network resource as segment, called as "Segment ID" (SID). In SR, an ingress node steers a packet through an ordered list of the SIDs as instructions, called as "SID list".

SRv6, specified in IETF, is the IPv6 dataplane instantiation of SR so that a SID in SRv6 is a 128 bits length IPv6 address. SRv6 introduces the concept of network programming [1]. In SRv6, a 128 bits SID is divided into arbitrary length of locator, function and argument parts. A SID can be bound to any function/service in a router or compute instance of the locator so that SRv6 achieves a networking objective that goes beyond mere packet routing.

By that, SRv6 has many advantages: common dataplane, tunnel elimination, state reduction, and traffic steering. We here introduce how to reduce the states using SRv6 briefly. Only the ingress nodes have to indicate SID list that a packet must traverse. Any other nodes in the network do not need to maintain the states.

C. Stateless GTP-U translation with SRv6

Co-existence with GTP-U is vital to deploy SRv6 user plane in a step-by-step basis. However if the co-existence introduces additional states in the user plane, it could be an obstacle to transition from GTP-U to SRv6. Thus, the solution for GTP-U translation with SRv6 is required to eliminate the obstacle so that some stateless translation solution is anticipated.

The idea of the stateless translation is that all identifiers of a GTP-U tunnel can be embedded as an argument of a SRv6 SID. It enables that just one configuration of a prefix allocated to the translation aggregates all tunnels states into one configured SID. This method was proposed to IETF in "SRv6 for Mobile User Plane" [5] which defined following functions:

- *T.M.Tmap*
  Translate a GTP-U over IPv4 packet to a SRv6 packet.
- *End.M.GTP4.E*
  Translate an SRv6 packet to a GTP-U over IPv4 packet.
- *End.M.GTP6.D*
  Translate a GTP-U over IPv6 packet to a SRv6 packet.

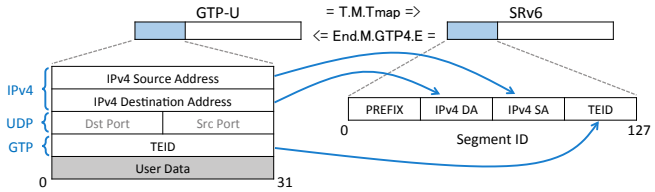


Fig. 1. Mapping between GTP-U and SRv6.

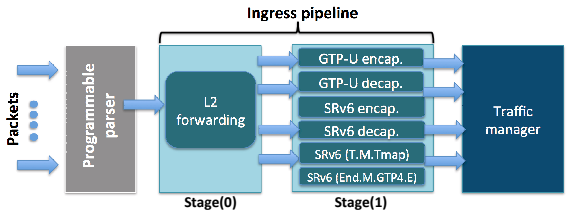


Fig. 2. Programmable switch for GTP-U and SRv6.

- *End.M.GTP6.E*
  Translate an SRv6 packet to a GTP-U over IPv6 packet.

For *T.M.Tmap* and *End.M.GTP4.E* cases, the IPv4 destination address and TEID of the GTP-U packet are embedded as an argument of the SID in the sending, or receiving SRv6 packet respectively, as depicted in Fig. 1. Although this method provided the GTP-U and SRv6 stateless translation, there are no performance evaluation results. In this paper, we propose a measurement methodology and analyze our evaluation results for GTP-U and SRv6 stateless translation.

IV. Measurement Methodology

A. SRv6 platforms

Multiple hardware and software platforms have demonstrated support for SRv6. We classify two SRv6 platforms: CPU-based and ASIC-based SRv6 platforms. In the CPU-based platform, there are two types of approach: Linux kernel and DPDK [18]. Several SRv6 functions [19] are implemented in the kernel. However the functions are processed by network stacks on the kernel and the extra overhead of packet processing will be induced.

FD.io VPP [20] using DPDK has been supported as the platform. We can expect the high throughput and low latency with enough CPU cores. When the small number of CPU cores is used, we cannot expect to evaluate the pure translation performance. Moreover, the performance of packet processing can be decreased when the number of VPP graph nodes is increased. In VPP, GTP-U and SRv6 functions were implemented after L2 and L3 VPP nodes and the multiple nodes were one of performance bottlenecks [21]. For above the reasons, the CPU-based platform will be adopted as our measurement platform in the near future.

A programmable switch like Barefoot Networks Tofino [6] using the P4 programming language [22] can be used as a SRv6 platform. Tofino is ASIC-based so that we can expect to evaluate the pure translation performance for the 5G networks. Here, our objective is to evaluate the pure translation performance between GTP-U and SRv6. Previously, S. Lange et al. [23] evaluated performance of LTE Serving Gateway (SGW) which processes GTP packets, but it was on the CPU-based platform. To achieve our objective, the programmable switch is desirable for stable performance. Thus, we adopted Tofino as the measurement platform.

TABLE I
Function list for GTP-U and SRv6.

Functions	Description	Sending type	Receiving type
GTP-U encap.	Encapsulated as GTP-U	IPv4	GTP-U
GTP-U decap.	Decapsulated as IPv4	GTP-U	IPv4
SRv6 encap.	Encapsulated as SRv6	IPv4	SRv6
SRv6 decap.	Decapsulated as IPv4	SRv6	IPv4
SRv6 (T.M.Tmap)	Translated to SRv6	GTP-U	SRv6
SRv6 (End.M.GTP4.E)	Translated to GTP-U	SRv6	GTP-U

B. Programmable switch for the stateless translation measurement

To fulfill the measurement on the adopted platform, we had to solve several problems with our P4 code in addition to the existing P4 code provided by Barefoot. The first problem is that there was no implementation of the stateless translation functions while some other SRv6 functions were available on Tofino. The second problem is that SRv6 functions were widely spread out to multiple stages in the pipeline which were also involving some other functions, such as VRF and L3 routing, which were out of scope of the measurement. We had to get rid out of those stages and functions since it could cause extra latency which makes us hard to evaluate the functions. The last problem is that we cannot measure the latency of functions on a commercial traffic generator, such as IXIA, when a receiving packet type is different from a sending packet type. In order to overcome the problems, we designed and implemented the GTP-U and SRv6 functions with the minimum stages which was one of the challenging works in our measurement.

Our programmable switch for GTP-U and SRv6 is adopted on Wedge100BF-32 [24] (Fig.2). The switch has a programmable parser, an ingress pipeline which consists of two stages, and a traffic manager. To decide the baseline of latency, we added the L2 forwarding function at the 1st stage (*Stage(0)*). Next, we prepared six primitive functions and fixed the location of functions at the 2nd stage (*Stage(1)*) to reduce extra latency on the pipeline. Each function is described in Table I. The function applied to a packet is determined by IPv4, UDP, and IPv6 header fields matched to an entry in Match/Action table which is equipped to the functions to determine the processing and forwarding of the packets. The table is simultaneously looked-up in parallel using SRAM or TCAM. Thereby it is expected that we can get stable

translation and forwarding performance unless the number of the entries reaches the upper-limit of the table. However as the impact was not yet investigated, the switch was configured with only one match/action entry for the measurement to exclude potential performance impact. In particular, we used 48 bits timestamp as nano scale provided from Tofino to measure the latency at the functions. It is similar to Inband Network Telemetry (INT) [25] but without adding additional headers and avoid impacting packet length by using source MAC address to store the timestamp.

In the switch, incoming packets are processed as follows. Firstly, the timestamp is recorded when the packets are received by the switch. Next, MAC, IPv4, UDP, and IPv6 header fields are parsed by the programmable parser and the fields are extracted as metadata. The metadata is used to classify the packet type and to match the primitive functions. Next, the packets are unconditionally matched to the L2 forwarding function at *Stage(0)*. If the metadata is related to SRv6 or GTP-U packets, the packets are matched to the primitive functions at *Stage(1)*. After the matching, the packets are forwarded to the Traffic manager and the timestamp recorded when receiving the packet is written into the source MAC address field. Then the packet is looped back to the programmable switch and the 2nd timestamp is recorded when receiving the packet. This time, no primitive function processing is done on the packet except for calculating the latency, by subtracting the 1st timestamp stored in the source MAC address field from the 2nd timestamp, and set to the source MAC address field.

C. Measurement Scenarios

In this section, we introduce our measurement scenarios. In our experiment (Fig. 3), our programmable switch on Wedge100BF-32 is used as a DUT, and our traffic generator (IXIA) is used as a tester. Our measurement method is based on RFC 2544 [26], but slightly different from the original RFC method. Well-known performance metrics, such as PPS, throughput, and packet loss, are measured on the tester while the latency at the function is measured on the DUT. It would be possible to use an open source packet generator, such as TRex [27], to measure the latency. But, the current version of TRex cannot measure the latency with nanoscale timestamp.

The experiment is occurred as follow. Firstly, packets are sent from the tester to the DUT to evaluate the functions. Next, the packets are encapsulated, decapsulated, or translated by the functions on the DUT. In this phase, the latency at the functions is measured using the telemetry. Then, the tester receives the changed packets. Finally, the well-known performance metrics are measured by the tester and the latency is extracted from the source MAC address field.

We prepared two types of measurement conditions: light (100Mbps) and heavy (100Gbps). Under the conditions, the PPS at both the sending and the receiving is equally achieved to without packet loss. We also prepared two packet sizes: short and long to clarify the performance impacts caused by packet size differences. Particularly, the short packet size was mainly observed at IoT application [28] and multimedia

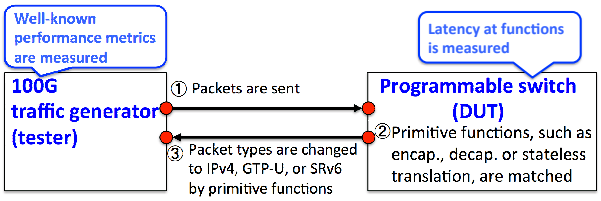


Fig. 3. Measurement method on local environment.

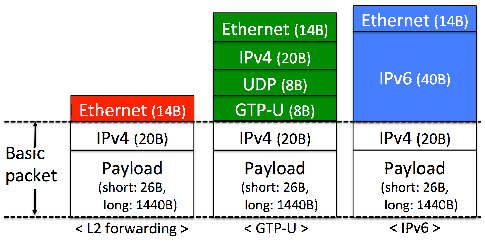


Fig. 4. Packet structure for measurements.

traffic in mobile network [29]. It is reasonable to evaluate the functions with the short packet size.

In our experiment, a basic packet consists on an IPv4 header (20 bytes) and a payload which we managed the sizes at 26 bytes and 1440 bytes, for short and long packet sizes respectively. L2-forwarding and simple encapsulation/decapsulation using GTP-U and SRv6 were measured to evaluate performance impacts on the stateless translation functions. The packet structure with the size of the headers and the payload is depicted in Fig. 4. The total sizes of short/long packets were 64/1478, 100/1514 and 104/1518 bytes for L2-forwarding, GTP-U and SRv6 respectively. The experiments at each condition run five times. To summarize, the target functions are measured with the four scenarios of the combination of light/heavy conditions and short/long packet sizes.

V. Evaluation Results

A. Well-known performance metrics at primitive functions

Here, we present the performance of well-known metrics, such as PPS, packet loss, and throughput. To achieve the well-known metrics, we used the statistics from the traffic generator. In our experiment, there is no packet loss at both light and heavy conditions. The average of well-known metrics is summarized in TABLE II. We focused on the generation of the same number of PPS under the measurement scenarios. We clearly achieved the same number of PPS at the functions.

Regardless of GTP-U and SRv6 under the heavy condition, the throughput is close to approximately 100 Gbps. Although the throughput at the GTP-U is slightly different from that at the SRv6, the throughput gap between the SRv6 and GTP-U is negligible. Because the packet size at the SRv6 is slightly larger than that at the GTP-U (Fig. 4), the throughput gap is observed. From the results, we confirmed that there were no abrupt performance changes in the well-known metrics under the scenarios.

TABLE II
AVERAGE OF WELL-KNOWN PERFORMANCE METRICS (PPS AND THROUGHPUT).

	Light (short)		Light (long)		Heavy (short)		Heavy (long)	
	PPS	Throughput	PPS	Throughput	PPS	Throughput	PPS	Throughput
GTP-U functions (*GTP-U encap. and decap.*)	100,805	96.7 Mbps	8,127	99.7 Mbps	100,805,260	96.7 Gbps	8,127,358	99.7 Gbps
SRv6 functions (*SRv6 encap. and decap.*)	100,805	99.9 Mbps	8,127	99.9 Mbps	100,805,260	99.9 Gbps	8,127,358	99.9 Gbps
SRv6 translation functions (*T.M.Tmap* and *End.M.GTP4.E*)	100,805	99.9 Mbps	8,127	99.9 Mbps	100,805,260	99.9 Gbps	8,127,358	99.9 Gbps

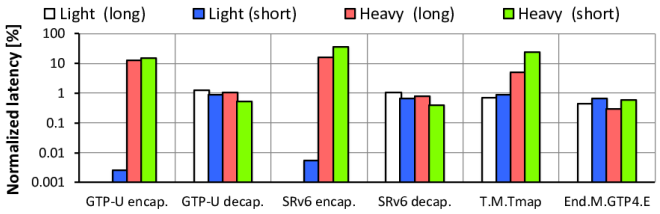


Fig. 5. Normalized latency on local environment.

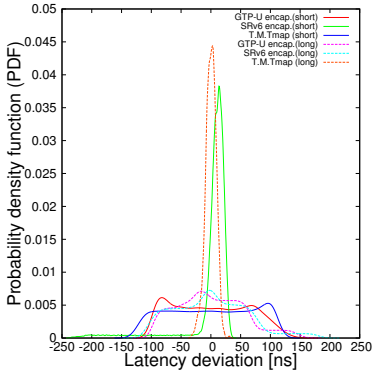
B. Latency at primitive functions

Since packets are forwarded to the additional stage, the latency at the functions will be higher than the L2 forwarding. Again, there is no way to measure the latency on the traffic generator when the packet type was different between the sending and the receiving. Alternatively, we measured the latency using the telemetry function at the programmable switch. The maximum latency was under one microsecond. Unfortunately, the absolute values are omitted for Barefoot network's NDA. We present the normalized latency based on the baseline (Avg. latency of L2 forwarding) in Fig. 5.

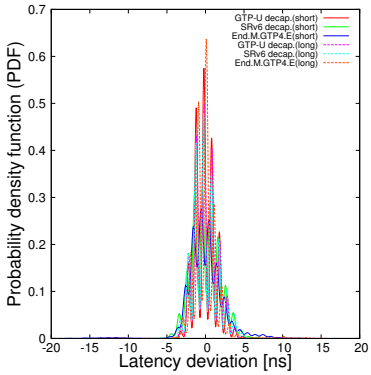
Under the light condition, there are no abrupt changes in the latency (the white is the long and the blue is the short) regardless of the packet sizes. With the long case, the normalized latency on both *GTP-U encap.* and *SRv6 encap.* is zero. Next, the normalized latency (the red is the long and the green is the short) under the heavy condition is also shown in the same Fig. 5. Interestingly, it was observed that the latency was noticeably impacted most at *SRv6 encap.*, followed by *T.M.Tmap* and *GTP-U encap.* for both of the short and the long cases while much low latency impact was observed on the rest of functions, such as *GTP-U decap.*, *SRv6 decap.* and *End.M.GTP4.E*. Due to the high PPS rate, it was also observed that the latency impact on the short was relatively high compared to the long.

When it comes to the group of latency-impacted functions, we found one common characteristic among them, i.e., the header size of forwarding packets was increased after the function processing. On the contrary, another group of functions processes the forwarding packets to decrease header size of the packets. We classified the patterns of the results into increased and decreased patterns.

As the variation of latency impacted-patterns under the heavy condition, we analyzed the PDF of latency deviation at all functions (Fig. 6). In the figure, the zero value on x-axis is the avg. latency of the functions. The evaluated function is unstable when the deviation is far to the zero. In contrast, the function is stable when the deviation is close to the zero. In the increased pattern, it was expected that the PDF values (Fig. 6(a)) of the short cases could be fluctuated than the long due to the high PPS rate. However in the long case, *T.M.Tmap* only followed that expectation. *T.M.Tmap* has the smallest gap in terms of header size difference between before/after the processing, and the lowest latency is observed in the long within the increased pattern. Interestingly, we found that *SRv6 encap.* was stable from the PDF value even in the short. It is one of our future work for the investigation. On the other hand, the PDF values (Fig. 6(b)) at all functions in the decreased



(a) Increased pattern



(b) Decreased pattern

Fig. 6. PDF of latency deviation under heavy condition.

pattern are concentrated in the avg. latency. Thus, we cannot find any particular differences between the functions.

C. Discussion

The implication derived from the measurement results on the local environment is summarized as follows. In the heavy condition, the extra latency and unstable performance were observed. However due to that it is commonly observed throughout the increased pattern, it would be a genuine characteristic of the switch. Thus it is elaborated that the pure performance of the translation functions, such as *T.M.Tmap* and *End.M.GTP4.E*, is low and stable in terms of the latency. Thus, our evaluation results are enough to show the possibility of co-existence of GTP-U and SRv6.

The latency observed in the increased pattern could be caused by some functions related to the packet buffer management of the switch. It means that some space still exists to be improved on that functions in the future.

VI. Conclusion

In this paper, we focused on the performance evaluation by measuring the primitive functions for GTP-U and SRv6 stateless translation using an industry-grade programmable switch for co-existing with GTP-U as 5G user plane. In order to evaluate the pure translation performance, the well-known performance metrics are measured by the traffic generator while the latency at the functions was measured by the telemetry on the programmable switch. In our local experiments, the latency at the increased pattern was noticeably increased when the throughput was close to 100 Gbps. It was interesting that the stable and lowest latency was achieved by one of the stateless translation function (*T.M.Tmap*). Through the quantitative results, the performance gap among the stateless translation, GTP-U and SRv6, is small and it is negligible. We hereby believe that our performance evaluation results will be helpful to consider the co-existence of GTP-U with SRv6 and the transition methods to SRv6 in 5G deployment.

In future work, we intend to evaluate various SRv6 functions on other platforms, such as VPP and the kernel. We will also propose and deploy a full 5G network with suitable scenarios for co-existing with the GTP-U.

References

[1] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "SRv6 Network Programming," Internet Engineering Task Force, Internet-Draft draft-filsfils-spring-srv6-network-programming-07, Feb. 2019.

[2] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018.

[3] C. Filsfils, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 Segment Routing Header (SRH)," Internet Engineering Task Force, Internet-Draft draft-ietf-6man-segment-routing-header-18, Apr. 2019.

[4] F. Clad, X. Xu, C. Filsfils, D. Bernier, C. Li, B. Decraene, S. Ma, C. Yadlapalli, W. Henderickx, and S. Salsano, "Service Programming with Segment Routing," Internet Engineering Task Force, Internet-Draft draft-xuclad-spring-sr-service-programming-02, Apr. 2019.

[5] S. Matsushima, C. Filsfils, M. Kohno, P. Camarillo, D. Voyer, and C. Perkins, "Segment Routing IPv6 for Mobile User Plane," Internet Engineering Task Force, Internet-Draft draft-ietf-dmm-srv6-mobile-uplane-04, Mar. 2019.

[6] "Barefoot tofino," https://barefootnetworks.com/products/brief-tofino/ [Online].

[7] "Segment routing ipv6 – interoperability demo is already there!" https://blogs.cisco.com/sp/segment-routing-ipv6-interoperability-demo-is-already-there [Online].

[8] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: a comprehensive survey of research activities, standardization efforts and implementation results," *CoRR*, vol. abs/1904.03471, 2019.

[9] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 464–486, Firstquarter 2019.

[10] D. Lebrun, M. Jadin, F. Clad, C. Filsfils, and O. Bonaventure, "Software resolved networks: Rethinking enterprise networks with ipv6 segment routing," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '18. New York, NY, USA: ACM, 2018, pp. 6:1–6:14.

[11] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "6lb: Scalable and application-aware load balancing with segment routing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 819–834, April 2018.

[12] P. L. Ventre, M. M. Tajiki, S. Salsano, and C. Filsfils, "Sdn architecture and southbound apis for ipv6 segment routing enabled wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1378–1392, Dec 2018.

[13] F. Duchêne, D. Lebrun, and O. Bonaventure, "Srv6pipes: enabling in-network bytestream functions," in *IFIP Networking 2018*, 2018.

[14] A. Abdelsalam, P. L. Ventre, A. Mayer, S. Salsano, P. Camarillo, F. Clad, and C. Filsfils, "Performance of ipv6 segment routing in linux kernel," in *2018 14th International Conference on Network and Service Management (CNSM)*. Washington, DC, USA: IEEE Computer Society, Nov 2018, pp. 414–419.

[15] M. Xhonneux, F. Duchene, and O. Bonaventure, "Leveraging ebpf for programmable network functions with ipv6 segment routing," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 67–72.

[16] 3GPP, "System architecture for the 5g system (5gs)," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 23.501, 2019.

[17] 3GPP, "General packet radio system (gprs) tunnelling protocol user plane (gtpv1-u)," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 29.281, 2019.

[18] "Intel dpdk," https://01.org/packet-processing.

[19] "Srv6 linux kernel implementation," https://segment-routing.org/ [Online].

[20] "Fd.io," https://fd.io/ [Online].

[21] "Improve the performance of gtp-u and kube-proxy using vpp," https://ossna2017.sched.com/event/BEN4/improve-the-performance-of-gtp-u-and-kube-proxy-using-vpp-hongjun-ni-intel [Online].

[22] "p4," https://p4.org/ [Online].

[23] S. Lange, A. Nguyen-Ngoc, S. Gebert, T. Zinner, M. Jarschel, A. Köpsel, M. Sune, D. Raumer, S. Gallenmüller, G. Carle, and P. Tran-Gia, "Performance benchmarking of a software-based lte sgw," in *2015 11th International Conference on Network and Service Management (CNSM)*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 378–383.

[24] "Wedge 100bf-32x," https://www.edge-core.com/productsInfo.php [Online].

[25] "In-band network telemetry (int)," https://p4.org/assets/INT-current-spec.pdf [Online].

[26] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544, Tech. Rep. 2544, Mar. 1999.

[27] "Trex," https://trex-tgn.cisco.com/ [Online].

[28] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying iot traffic in smart cities and campuses," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Washington, DC, USA: IEEE Computer Society, May 2017, pp. 559–564.

[29] S. Fowler, J. Sarfraz, M. M. Abbas, E. Bergfeldt, and V. Angelakis, "Evaluation and prospects from a measurement campaign on real multimedia traffic in lte vs. umts," in *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE)*. Washington, DC, USA: IEEE Computer Society, May 2014, pp. 1–5.