WHITE PAPER

NFVi Acceleration
SRv6

Segment Routing Over IPv6 Acceleration Using Intel® FPGA Programmable Acceleration Card N3000

Authors

SRv6 Solution Team
HCL Technologies

Chuck Tato
Director,
Communications Business Division
Intel® Corporation

Table of Contents

Introduction

Segment Routing (SR) is an emerging technology that helps in addressing the requirements of the new Network Functions Virtualization (NFV) and Software Defined Network (SDN) architecture. It provides a unified solution for networking programmability, Service Function Chaining (SFC), protocols simplification, traffic engineering, and mobile versus fixed network convergence.

SR leverages the source routing paradigm. A node steers a packet through a SR Policy instantiated as an ordered list of instructions called "segments" (often referred by their Segment Identifiers, or SIDs). A segment can represent any instruction, either topological (to forward packets through an identified path) or service-based (within an NFV/SDN architecture a service is represented by a Virtual Network Function (VNF) implemented by a container or a Virtual Machine (VM)). A segment can have a semantic local to an SR node or global within an SR domain. By leveraging this technology, the network no longer needs to maintain a per-flow or per-application state.

The SR architecture can be instantiated on various data planes, such as Multiprotocol Label Switching (MPLS) or IPv6. In this paper, we will refer to SR implemented over IPv6 (SRv6). There are a variety of efficient SRv6 implementation options available, ranging from pure software implementation to offloading certain functionality of SRv6 to hardware, to pure hardware-based products.

This white paper gives an overview of the available SRv6 software implementations; however, the main objective is to present a solution built for the Intel® FPGA Programmable Acceleration Card (Intel FPGA PAC) N3000 for the Vector Packet Processing (VPP) data plane. For this solution, the performance results in saving valuable CPU cycles and resources are given.

Outline

- Section 3 provides a brief background on accelerator solutions and SRv6 protocol
- Section 4 describes the main SRv6 software implementations available
- Section 5 describes the SRv6 acceleration approach based on VPP data plane and related processing details
- Section 6 presents performance data for Service Function Chaining through SR dynamic proxy

Background

NFVi Acceleration Solutions

Network Functions Virtualization infrastructure (NFVi)-proposed approach utilizes IPv6 routing-based architecture in data centers. Moreover, the usage of SRv6-based fabric in data centers can help reduce underlying protocol stacks needed (for instance Border Gateway Protocol (BGP) and Internal Gateway Protocol (IGP) in data centers, simplifies the interconnection with core network). It also allows implicit handling of Equal Cost Multi Path (ECMP) for traffic engineering and simple implementation for Ethernet VPN (EVPN) overlays. This leads to SRv6 domain extending to the Tunnel Endpoint Termination (TEP) located in data plane servers.

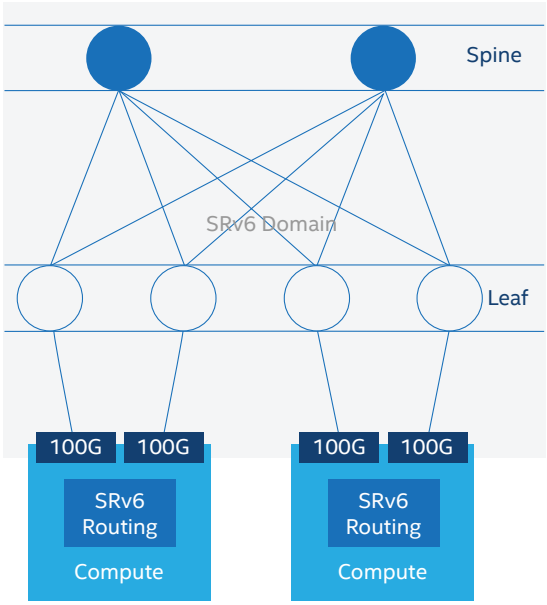

Figure 1. Data Center Architecture

The NFV industry is increasingly realizing that standard network interface card (NIC) products may have performance limitations when supporting emerging NFV/SDN workloads. This performance limitation in the network will require a higher level of dedicated compute resources for networking in the servers, which will result in less resources being made available for actual VNFs. Even by accepting this additional overhead for networking, performance becomes less predictable than solutions where the same networking functions are realized in hardware.

While keeping the NFVi as an open platform, Intel FPGA acceleration solutions allows for offloading of low-level specific network functions to the hardware, increasing data path performances and predictability, saving CPU cycles and cores, thus leaving additional CPU server resources available for VNFs and workloads.

With the Intel FPGA PAC N3000, the FPGA-based solution can be re-programmed using a standard Intel FPGA development environment such as the Intel Quartus® Prime software, allowing for almost any type of functionality being offloaded to provide a high level of performance.

The HCL solution presented here is based on an implementation of SRv6 protocol processing offloaded to the Intel FPGA PAC N3000.

This solution will provide two main advantages to the telecom operator's virtual installation environments:

- Greatly improve the data plane throughput while still using open hardware platforms
- Allow for smooth network upgrade of new SRv6 features without changing the installed hardware base

SRv6 Overview

SRv6-based architecture relies on the source routing paradigm. An ordered list of instructions, called segments can be used to steer packets through a set of intermediate nodes towards the destination. In an SR domain, the IPv6 segments are transported through a specific IPv6 Routing header called the SRv6 header which is an extension header within the IPv6 header space. There are two types of network segments:

- Adjacency segments: A local (to the node) segment that allows forwarding on a specific link
- Prefix segments: A global (to the SR domain) segment that is attached to a network prefix, a node segment being a special case of prefix segment

Moreover, in the context of Service Function Chaining, the service segment (local to the node) represents a specific service to apply to a packet.

To be able to enforce specific paths, edge nodes must insert Segment Routing Header (SRH) containing the ordered list of Segments (see Figure 2). This information can be obtained by a routing protocol or through an SDN controller configuration.

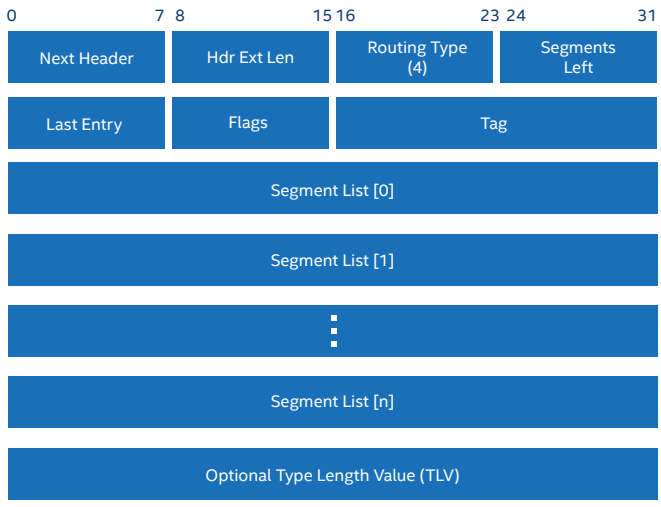

Figure 2. SRv6 Header Format

In SRv6, segments are encoded as regular 128-bit IPv6 addresses. The segment list is contained in the SRH, and up to 128 segments can be inserted. Segments are stored in reverse order in the segment list. For example, the segment at index 0 corresponds to the last instruction to execute.

Here is a brief SRH field explanation:

- Next Header: Indicates which type of header is following the SRH
- Hdr Ext Len: Length of the SRH header in eight-bytes units, not including the first eight bytes
- Routing Type: Always set to 4
- Segments Left (SL): Indicates the number of remaining segments
- Last Entry: Specifies the index of the last segment in the segment list
- Flags: Cleanup flag is of interest for the penultimate segment endpoint SRH stripping
- Tag: Can be used to mark the packet as belonging to a class of packets sharing the same properties
- Type-Length-Value (TLV): TLVs are 3-tuples that can be used to store additional data in the SRH; the structure of this 3-tuple allows TLVs of variable length to be implemented, the Length field specifies the number of bytes contained in Value, and finally Type stores a unique identifier for each TLV type

With the introduction of the SRH, new packet processing rules for SRv6 packets must be defined.

The three types of nodes that can be involved in SRv6 packet processing are:

- Source SR Node: A node inserting an SRH into an IPv6 packet. This can either be an end host originating an IPv6 packet, or an SRv6 ingress router encapsulating a received packet in an outer IPv6 header with an SRH
- Transit Node: A node forwarding an IPv6 packet whose destination address (DA) is not an SID belonging to the node; they operate as pure IPv6 routers, forwarding packets based on IPv6 DA
- SR Segment Endpoint Node: A node receiving an IPv6 packet whose DA is an SID belonging to the node

The Internet Engineering Task Force (IETF) Draft on SRv6 Network Programming defines the SRv6 architecture and elaborates the concept of SRv6 SID and possible functions associated with it. The SRv6 SID can be encoded as a LOC:FUNCT where the locator part is the most significant L bits of the SID (it represents a network location or an SRv6 Node) and the function part is the less significant 128-L bits (which represents a function implemented on the Node). A function may require additional arguments (ARGS) that would be placed immediately after the FUNCT in the form of LOC:FUNCT:ARGS::. Within the same IETF Draft there is also a set of "well-known" function definitions that can be associated with an SID.

SR Proxy for Service Function Chaining

SRv6 supports NFV SFC by simply mapping VNFs to be traversed in ordered list of SIDs. For SR-unaware VNFs, compute nodes can run an SR proxy function which can alter the packet to allow processing by Service Function. The generic behavior of an SR proxy is to intercept traffic destined to the VNF via a locally instantiated service segment, modify it by removing SR related information and send it out to IFACE-OUT VNF interface. Post that, when traffic is sent back from VNF to SR proxy via IFACE-IN, the SR information is restored and forwarding takes place.

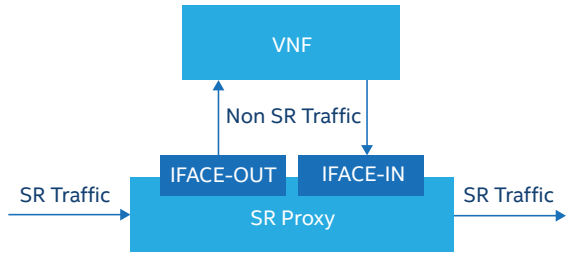

Figure 3. Generic SR Proxy

Four different SR proxy mechanisms have been defined within the IETF:

- Static proxy, available in VPP
- Dynamic proxy, available in VPP
- Shared proxy
- Masquerading proxy, available in VPP

SRv6 performance data presented later is related to the SR dynamic proxy scenario.

SRv6 Software Implementations

Data plane implementations of SRv6 have been made available both from open-source software organizations (Linux and fd.io VPP) and hardware/ASIC telco vendor providers. Interoperability tests have also been reported under IETF. While this paper presents performance data for VPP-accelerated data plane only, a brief overview of Linux* kernel-based implementation can be found in the following sections. Some data about Linux kernel performance is also available in literature.

Linux SRv6 Implementations Overview

SRv6 implementation has been merged into Linux Kernel 4.10, based on Light Weight Tunnel (LWTunnel). The SRv6 SIDs are configured as FIB entries via IPv6 route. iproute2 user-space utility was also extended to support addition of SID and its associated behavior.

Linux 4.18 custom SRv6 functions can be implemented in Linux kernel using the extended Berkeley Packet Filter (eBPF) through End.BPF hook. Currently supported behaviors include T.insert, T.encaps for Transit and End, End.T, End.X, End.DX2, End.DX6, End.DT6 as Endpoints.

SREXT is a kernel module providing both basic and advanced segment routing functions. It can be used as a standalone SRv6 implementation or as a complement to existing SRv6 kernel implementation. SREXT supports a completely independent "my local SID table" and adds support for End.AD and End.AM to the already available Linux Kernel Endpoint set.

VPP Implementation

Initial support for SRv6 in VPP was made available in the version 17.04 release from April 2017. Currently, the following SRv6 Endpoints are supported: End, End.X, End.DX6, End.DT6, End.DX4, End.DT4, End.DX2, End.B6, End.B6.Encaps. Moreover, T.insert, T.encaps policy behaviors are also supported. To fully support the programmability option (which is crucial for SR paradigm), VPP allows the developer to easily create custom SRv6 functions which, through ad-hoc application programming interfaces (APIs), can be connected to VPP as plugins. Plugin require code for the custom behavior through a new graph node, while basic SRv6 features are already implemented in VPP. The SRv6 enpdoint implementation of SR unaware service function chain proxy functions are available as a VPP plugin.

SRv6 Acceleration Approach

Compute nodes within SRv6-enabled NFVi installations can benefit from additional acceleration provided by the Intel FPGA PAC N3000 and is seamlessly integrated with the infrastructure, simplifying the deployment and networking (see Figure 4).

This means that developing orchestration support is crucial for operators to be able to introduce and manage such kind of solutions inside their existing environments.

Open Programmable Accelerator Engine (OPAE) developed by Intel is used as the basic framework to develop solution management application. SRv6 acceleration approach proposed here deals with improving the performances of VPP-based SRv6 implementation, where the entire SRv6 processing is done through VPP/DPDK software. The basic idea is to offload CPU intensive operation to the Intel FPGA PAC N3000, defining a new functional splitting of SRv6 behaviors between hardware and VPP software. Interworking between the Intel FPGA PAC N3000 and VPP is supported by introducing new graph nodes in VPP, which oversees encoding/decoding/process hardware-related information.

As a result, throughput performance gain is basically given by the difference between the CPU core cycles saved by partially offloading SRv6 processing in hardware and the additional processing needed by the newly added hardware interworking functions.

SR Dynamic Proxy Offload

The basic idea to support this use case is to offload SRH encapsulation or decapsulation operations and consequently SR Cache handling to the Intel FPGA PAC N3000. In ingress direction, the Intel FPGA PAC N3000 performs an SID lookup, and based on lookup result can strip the outer IPv6/SRH headers and add the proper metadata header instead, for the VPP to be able to process inner frames/packets. In egress direction inner frames/packets are received on SmartNIC together with metadata, which instruct the Intel FPGA PAC N3000 on SRH cache entry to be used for that specific flow. Route lookup in this approach is still done in VPP.

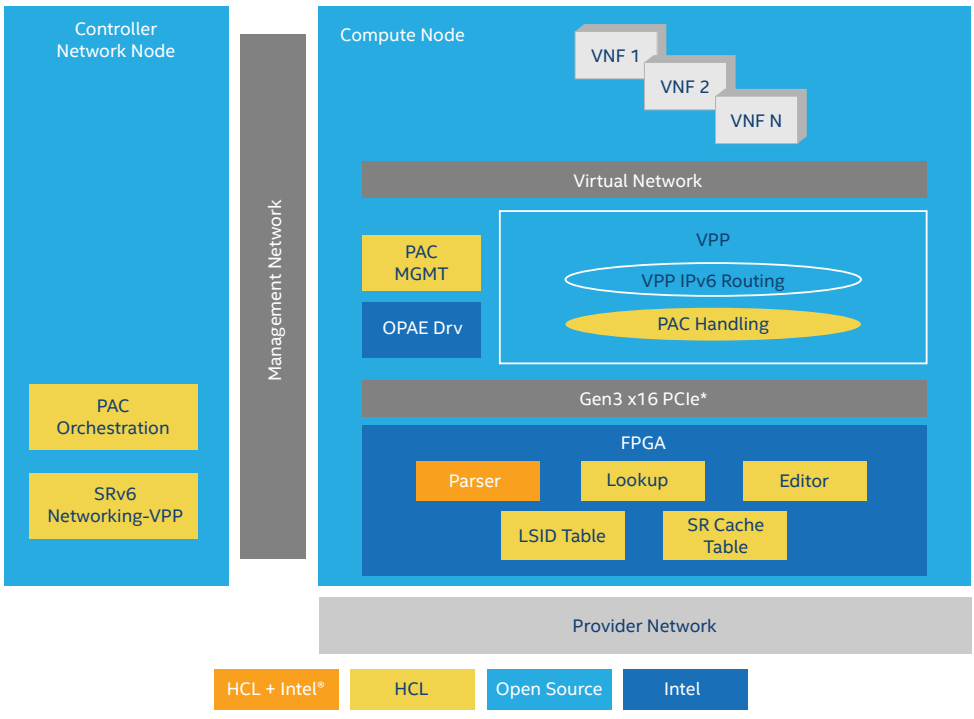

Figure 4. Accelerator in OpenStack Reference Architecture

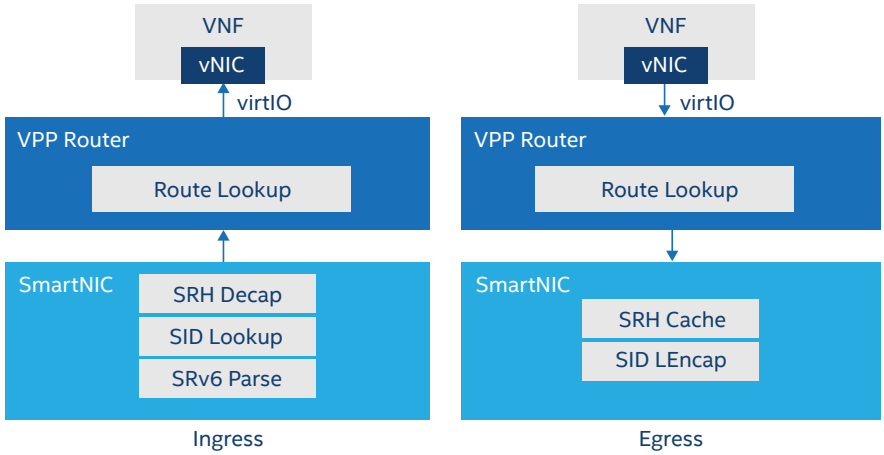

Figure 5. Basic Processing

SRv6 Throughput Performance

In our experiments, we tested throughput performances of End.AD2 behavior in both accelerated and non-accelerated scenarios.

Throughput measurement definition follows IETF RFC 1242 and it is reported in packet per second (pps). Either Non Drop Rate (NDR) or Partial Drop Rate (PDR) are defined, depending on acceptable tolerance. In this paper, we provide PDR-based performance measurements with a threshold of 0.5%.

Test Setup

The test setup we used is shown in Figure 6 with configuration settings as reported in Table 1.

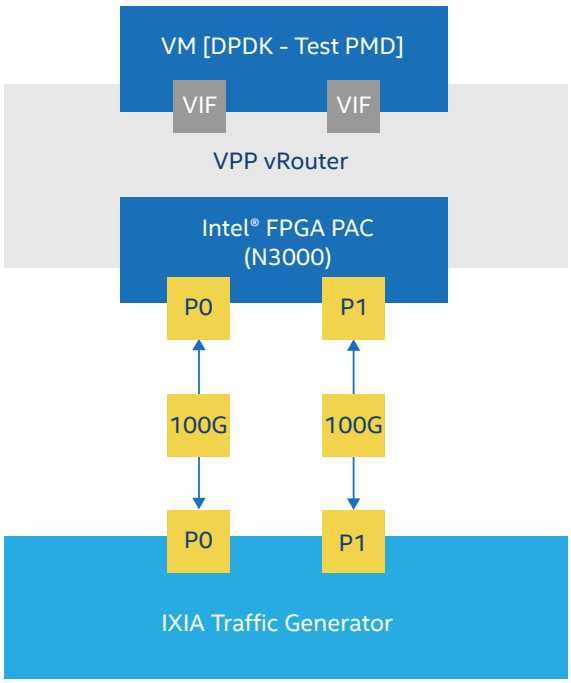

Figure 6. Test Setup

DEVICE UNDER TEST (DUT) CONFIGURATION	
CPU	Family 6, Model 85, Model Name Intel® Xeon® Platinum 8180M CPU at 2.50 GHz
Socket	CPU Socket 1 Used
Core/socket	56 (28 Physical + 28 Logical)
Intel HyperT	Disabled
Intel VT	Enabled
Intel Turbo Boost	Enabled, 3 GHz.
RAM	192 GB
SmartNIC	PAC N3000 1x4x25G
Transceivers	QSFP28 –SR4 with 12-core Fiber MTO cable
Host operating system (OS)	Centos 7.4 kernel 3.10
Huge pages	2 MB
Data Plane Development Kit (DPDK) version	19.05
VPP version	19.08
QEMU version	2.6

Table 1. DUT Configuration

SRv6 traffic matching End.AD2 case was sent in both directions, starting from full line rate and decreasing the rate of both Cisco TRex traffic generator ports until losses fell below the given threshold. The same test was repeated for both legacy VPP with the Intel FPGA PAC N3000 in "passthrough" mode.

Results

Performance measurements were carried out for three different packet sizes: 192B, 512B, and Internet Mix (IMIX) where the traffic profile is according to Table 2.

PACKET SIZE (BYTES)	DISTRIBUTION IN PACKETS	DISTRIBUTION IN BYTES
192	58.33%	22%
576	33.33%	47%
1,500	8.33%	31%

Table 2. IMIX Profile

Results are reported in Figure 7 for 192B packet size. Core Cycles Per Packet (CPP) saving ratios are reported in Table 3. Throughput performance is inversely proportional to the average CPP.

PACKET SIZE (BYTES)	CPP SAVING
192	84%
512	72%
IMIX	67.5%

Table 3. Cycles Per Packet Saving

Results show higher performances for shorter packet sizes, since with short packets CPP savings are more impactful compared to CPU cache processing. With reference to Figure 7, maximum throughput achievable for a pure VPP-based solution using 12 cores is obtained by the accelerated solution with six cores only. This performance gain is visible with 512 bytes packet size as well. By looking at the IMIX profile, presence of a relevant percentage of 1,500 bytes packets further decreases achievable gain.

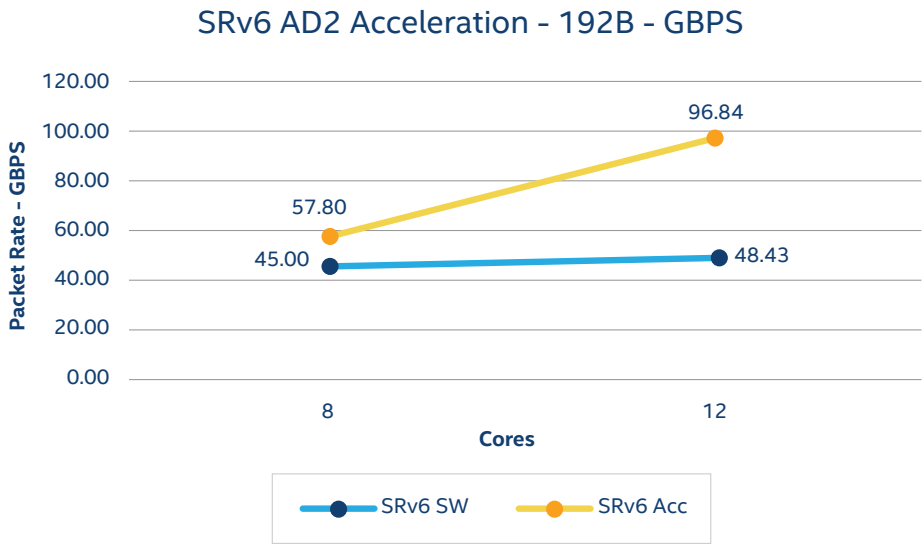

Figure 7. Performance Data with 152 Bytes Packet Size

Conclusion

VPP or DPDK implementation of SRv6 already provides an optimized solution with good throughput performances. The proposed acceleration solution with the Intel FPGA PAC N3000 increases the overall throughput, allowing for more predictable performance, and freeing up CPU Cores/Cycles for other VNFs and Workloads. Furthermore, additional performance improvement is possible, though it would require the placing of routing functions and interfaces between the network and VMs within the hardware, and maximizing the Intel FPGA capabilities to the fullest possible extent.

About HCL Engineering and R&D Services

HCL ERS, a division of HCL Technologies, enables technology-led organizations to go to market with innovative products and solutions. HCL partners with its customers through flexible engagement models, combining accelerated product development, adoption of new technologies, and agile and industrialized service delivery to deliver a world class customer experience, and creates associated solution delivery ecosystems to help bring market leadership. HCL develops engineering products, solutions and platforms across semiconductor, telecom and networking, consumer electronics, software, online, servers and storage, medical devices, aerospace and defense, automotive, and industrial manufacturing for its customers.

Where to Find More Information

For more details contact: erx@hcl.com.

Follow us on Twitter: http://twitter.com/hclers and our blog http://ers.hclblogs.com/

Visit our website: www.hcltech.com/engineering-services/

References

[1] C. Filsfils et al., "Segment Routing Architecture", Internet Requests for Comments RFC Editor RFC 8402, available: https://tools.ietf.org/html/rfc8402

[2] C. Filsfils et al., "SRv6 Network Programming", IETF Internet-Draft, available: https://tools.ietf.org/html/draft-ietf-spring-srv6-network-programming-00

[3] C. Filsfils et al., "IPv6 Segment Routing Header (SRH)", IETF InternetDraft, available: https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-18

[4] F. Clad et al., Service Programming with Segment Routing, available: https://tools.ietf.org/html/draft-xuclad-spring-sr-service-programming-02

[5] C. Filsfils et al., "SRv6 interoperability report", IETF Internet-Draft, available: https://tools.ietf.org/html/draft-filsfils-spring-srv6-interop-02

[6] "VPP", available: https://wiki.fd.io/view/VPP

[7] “SRv6: Segment Routing for IPv6”, available: https://docs.fd.io/vpp/17.07/srv6_doc.html

[8] S. Bradner, "Benchmarking Terminology for Network Interconnection Devices", Internet Requests for Comments RFC Editor RFC 1242, July 1991, available: https://tools.ietf.org/html/rfc1242

[9] TRex realistic traffic generator, available: https://trextgn.cisco.com/

[10] “srext – a Linux kernel module implementing SRv6 Network Programming model”, available: https://github.com/netgroup/SRv6-net-prog

[11] A. Abdelsalam et al., "Performance of IPv6 Segment Routing in Linux Kernel," 2018 14th International Conference on Network and Service Management (CNSM), Rome, 2018, pp. 414-419

[12] P. Lapukhov et al., " Use of BGP for Routing in Large-Scale Data Centers", Internet Requests for Comments RFC Editor RFC 7938, August 2016, available: https://tools.ietf.org/html/rfc7938

WP-01295-1.0