Green Network Traffic Engineering Using Segment Routing: an Experiment Report

Jacob Van Groningen
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
jacobvangroningen@cmail.carleton.ca

Chung-Horng Lung
Department of Systems and Computer Engineering
Carleton University, Ottawa Canada
chlung@sce.carleton.ca

Abstract— With the ever-expanding network-based services, environmental impact has become a concern, as the surge in network traffic between devices has only intensified in increased energy consumption. This paper aims to exploit Segment Routing over IPv6 (SRv6) for energy efficiency purposes for data forwarding. SRv6 is a traffic engineering mechanism that enables data packet steering using segments in IPv6 headers. The main idea of the paper is to use SRv6 with automatic rerouting of network traffic based on the resource usage of network devices for higher energy efficiency compared to the traditional IP forwarding based on the shortest path first (SPF) algorithm. The method and system outlined in this paper dynamically created network topologies within `Mininet` and performed SRv6 using the `ROSE` platform to route packets through the most energy-efficient paths, all while actively collecting device usages, calculating dynamic weights, computing energy-efficient paths, and rerouting the traffic using SRv6. This paper successfully achieved the goal of energy-aware traffic rerouting. The results showed that the resource usage for SRv6 could be more than 70% lower than that of the SPF-based forwarding, depending on the network topology.

Keywords—Green networking, segment routing, SRv6, software-defined networking, Mininet, ROSE platform

I. INTRODUCTION

The number of network devices has increased dramatically due to the huge volume of Internet traffic. Hence, the total energy demand for network devices continues to grow significantly. Energy-aware or green networking approaches have been proposed. Green networking can be supported at different layers, such as hardware, device, network, and protocol. Further, green networking techniques have been investigated for different types of networks, such as data center networks, Ethernet, and backbone networks. Green networking has gained more popularity in the literature [1].

Traditional network traffic engineering (TE) using the Interior Gateway Protocol (IGP) centers on performance and reliability by overprovisioning and exploiting redundancy. For instance, in [6], the average link utilization for a specific backbone network was around 10% and the link utilization was over 50% only when there were failures in the network. Further, a common objective for the traditional TE approaches is to minimize the maximum link utilization (MLU) [7] for higher performance and reliability. As a result, those TE approaches consume more energy or generate a higher carbon footprint in order to reduce MLU for performance and reliability purposes.

Various green TE approaches have been proposed [2-5,14-17]. They share a common feature, i.e., reducing the overall carbon footprint by turning off some ports, links, or even nodes, or putting some of them to sleep mode, when the utilization is low for the corresponding device. For the backbone networks, that means efficient path computations and traffic rerouting are moving away from the distributed Multiprotocol Label Switching (MPLS) technique to software-defined networking (SDN) that has a central controller to effectively monitor devices and network telemetry and recompute data forwarding paths. Segment Routing IPv6 (SRv6) [2-5] is based on the SDN concept for routing and offers flexibility and programmability to support TE. SRv6 has gained a great deal of attraction from network operators recently.

SR-based green TE techniques have been proposed in the literature, such as [8-10]. Those methods all share similarities in putting underutilized resources into idle mode. However, they were proposed for SR over MPLS (SR-MPLS), not SRv6. This paper focuses on traffic path computations, reroutes, and data forwarding for backbone networks using SDN and SRv6. Further, there is a lack of experimental resource usage measurements for SRv6-based green TE compared to regular IGP-based data forwarding. Hence, the objective of the paper aims to reduce the carbon footprint of a network while maintaining the quality of service provided by the traditional TE. The contribution of our paper is to conduct an experimental evaluation of energy-aware traffic rerouting by monitoring resource usage and subsequent dynamic weight calculation using a testbed implementation with the SRv6 platform *ROSE* [11]. We tested various network topologies and compared the resource usage of methods with shortest path first (SPF) and energy-aware rerouting using SRv6. The results showed that the SRv6-based data forwarding results in lower energy consumption than SPF-based data forwarding.

The rest of the paper is organized as follows: Section II highlights related works. Section III describes the proposed approach. Section IV presents the experiments and results. Finally, Section V is the conclusions and future directions.

II. RELATED WORKS

A. SDN, Mininet, and Ryu Controller

SDN has attracted a great deal of attention since its innovation. It is a new paradigm that separates the control logic from the data plane to reduce the complexity of network

978-3-903176-66-9 ©2024 IFIP

operations. As a result, network devices have been simplified and most control functionalities have been shifted to the central controller. The central controller monitors the status of network devices of the entire network, makes decisions for each device, and sends instructions to target network devices for execution.

One prominent example of an SDN controller is the `Ryu` controller [12]. `Ryu` provides a flexible framework for building SDN applications and network management solutions. However, other controllers can also be used. `Mininet` [13] is a popular network emulator that facilitates the creation of customized networks for experiments and exploratory research.

B. SRv6

SR simplifies the control plane by using source routing to replace complex Resource Reservation Protocol (RSVP) while keeping efficient MPLS data plane forwarding. SR can be realized using MPLS (SR-MPLS) or IPv6 (SRv6) [2-5]. Similar to MPLS data forwarding, SR uses predefined labels or segments for a traffic flow to follow the shortest path or an explicit path. A segment can be either a 32-bit label for SR-MPLS or a 128-bit IPv6 address for SRv6. This paper focuses on SRv6, as it offers more flexibility and advantages. Further, a stack of multiple SRv6 segments conceptually has higher processing overhead, because each segment has 128 bits if compression is not used. Hence, it is important to investigate SRv6's impact on the energy consumption for reroutes, as rerouted paths usually are longer than the shortest paths.

The *`ROSE`* platform [11] for SRv6 is built upon the open Linux ecosystem with the support of an industrial partner, Cisco. It is easy to access and useful for exploratory research.

C. Green TE

Green TE has been investigated for different routing protocols, i.e., traditional IGP [14], MPLS [15][16], and SR [2–5][17]. SDN has also been used [9] and is assumed to be the default architecture in many existing green TE techniques or SR in general [2–5]. The authors in [10] presented important related techniques that have been adopted for green TE approaches.

Our paper focuses on green TE using SRv6 and SDN for two main reasons: (i) SRv6 supports explicit routing for traffic steering with higher header overhead as explained earlier, but without the complex MPLS operations at the control plane; and (ii) SDN controller provides a global view of the network, which reduces the complexity of distributed routing information dissemination and simplifies path computations.

III. Design of Green TE Using SRv6

This section presents the design of green TE using SRv6. To support green TE for rerouting traffic, resource usages need to be continuously monitored, and dynamic link weights be updated based on usages. The updated weights will then be used for path computations and rerouting. There are different methods for weight and path computations, but they are not the focus of our paper. The main objective of the paper is to conduct an experimental evaluation and comparison of resource usage for IGP-based data forwarding using SPF and energy-aware SRv6-based reroutes for data forwarding. Different methods for path computations can be used for similar purposes.

$$Weight(edge) = CPU_{tx_{score}} + Memory_{tx_{score}} + CPU_{rx_{score}} + Memory_{rx_{score}} \quad (1)$$

$$X_{score} = \begin{cases} usage, & usage \leq LimitTh \\ 0, & usage > LimitTh \end{cases} \quad (2)$$

Equations (1) and (2) are used to calculate the *link* or edge weights based on measured CPU and memory usage on each side of a link for dynamic SR path computations for each (source, destination) pair using a modified weighted Dijkstra SPF algorithm. $CPU_{tx_{score}}$ and $CPU_{rx_{score}}$ represents the CPU score for transmitting (`tx`) and receiving (`rx`) data packets measured for a node on each side of a link, respectively. Similarly, $Memory_{tx_{score}}$ and $Memory_{rx_{score}}$ denotes the memory score for transmitting and receiving data packets for a node on each side of an edge, respectively. The usage data collected are sent to the controller for path computations. *`LimitTh`* is a configurable threshold allowing for potential traffic bursts. Cisco's recommendation of 75% for reservable resources was used as a reference. Hence, the value of *`LimitTh`* is 70, i.e., a buffer of 30% CPU and memory capacities are reserved to accommodate potential usage spikes and ensure sufficient resource remains available during traffic spikes caused high packet volumes. Maintaining *`LimitTh`* below a certain level contributes to router stability in such scenarios.

Each link CPU and memory score of a *node* is calculated by taking the modulo with *`LimitTh`* of 70. For instance, if the CPU usage is at 65%, it would be 65. However, if the CPU or memory usage on either endpoint of a link is over *`LimitTh`*, the score for that link is set to 0, and the link will be considered overloaded. If all links are overloaded, the one with the least usage will be used for path computation. The purpose is to reduce the likelihood of nodes having usages over *`LimitTh`* to be used in the SPF calculations if an alternate path is available. A simple method for the final weight calculations is used by subtracting each metric score from 100. For instance, 100 – 65 = 35 is the final score for a metric. The total weight for the link is then summed by taking these final four CPU and memory scores for a node on each side of the link, they are then summed to determine the smallest score for each link, which is then used in SPF. The higher sum is the score for the link, except in the instance of all links being over *`LimitTh`*. The links with higher sum (lower utilization after subtraction from 100) are less favored using SPF, as SPF chooses an edge with the lowest cost first. These metrics are collected with the `psutil` (process and systems utilities) system call at nodes on each side of the link.

The algorithm `calculate_weight` to compute link or edge weights is depicted as follows. Currently, four metrics are used for each endpoint of an edge, as shown in Equ. (1), hence the combined edge score is 4 × (100 – each resource score).

Algorithm: `calculate_weight`
Purpose: Calculate weights for links/edges based on host measurements. Higher weights indicate better feasibility for SR.
Inputs:
- `self`: Reference to the network object containing the function
- `get_edges()`: Function retrieves the network's edges

Outputs:

- Updates the `weighted_edges` Python dictionary with edge weights for Segment Routing

Steps:

1. Initialize an empty dictionary `weighted_edges` for all edges.
2. Call `get_edges()` to get all edges and store them in `edges`.
3. Iterate through each `edge` in `edges`:
   - Create a dictionary `overloaded_endpoint_scores` to track overloaded endpoints (initially 0 for all endpoints).
   - Create a dictionary `edge_scores` to store scores for each endpoint in the edge (initially empty).
   - Iterate through each endpoint `endpoint` connected to `edge`:
     - Calculate scores for endpoint resource usage (memory, CPU) using modulo with *LimitTh*.
     - If endpoint memory or CPU usage exceeds *LimitTh*:
       - Add the usage amount to `overloaded_endpoint_scores[endpoint]`.
       - Set the corresponding score in `edge_scores[endpoint]` to 0 (infeasible).
   - Calculate a combined edge score for each endpoint: `edge_scores[endpoint]` = 4 × (100 – each resource score).
   - Set a flag `endpoint_feasible` to `True` if all resource scores for the endpoint are positive (feasible).
4. Based on `endpoint_feasible`:
   - If feasible:
     - Create temporary weights for each endpoint on the edge: `weight` = `edge_score` + 1.
     - Update `weighted_edges` with these temporary weights.
   - If not feasible:
     - Find the endpoint with the lowest `score` (least overloaded).
     - Create temporary weights for all endpoints on the edge with a weight of 4 × 100 (highest score or infeasible).
     - Set the weight for the least overloaded endpoint to its actual edge score.
     - Update `weighted_edges` with these temporary weights.
5. Use the `nx.set_edge_attributes` function (assumed to be from the `NetworkX library`) to set the calculated edge weights on the network graph `self.G`.

Note that CPU and memory usage data are used for link weight calculations here, as both measurements can be obtained quickly from the system calls. The proposed algorithm was called while a packet was being routed. Since the metrics are actively being evaluated, there may be potential changes from the previous path calculations. An alternative approach would be to call this algorithm after the metrics are evaluated and then save that result for all packets being routed until the next evaluation. The weight calculation can also be replaced with another formula, for example, link bandwidth utilization if both sides can be effectively measured. The purpose of the paper is to show dynamic link weight calculations and subsequent SR path computations and reroutes for proof-of-concept.

IV. Experimental Setup and Results

A. Experiment Setup

To validate the design, several experiments were conducted using different topologies and various tools. Specifically, four topology types were evaluated: Linear, Tree, Mesh, and Web. The tools used included the *ROSE* platform [11] for SRv6 path creation and data forwarding, `Mininet` for network emulation, `Ryu` for the SDN controller, `NetworkX` for network topologies generation and a modified weighted SPF algorithm implementation, `InfluxDB` for querying resource usage, and `Grafana` for visualizing the results. TABLE I shows the parameters for four different types of topologies

TABLE I. Topology Types and Parameters for Experiments

Topology	Minimum Size	Maximum Size
Linear	2	20
Tree	2	4
Mesh	2	20
Web	2	4

The size of each linear or mesh topology is directly proportional to the size of the nodes, with the number of nodes being $2\times SIZE$, while the size of the Web and Tree topologies have a structure of $SIZE$ nodes stemming off each node $SIZE$ layers deep from the central router, thus the number of nodes is:

$$n = \sum_{i=0}^{SIZE} SIZE^i \quad (3)$$

To generate the traffic for the network, the hosts of the network were iterated through each sending a single ICMP packet at a time from that host to every other host continuously during a 5-minute timespan. Each packet had a one-second timeout, leading to a minimum packet density of 300. However, with the ICMP request duration closer to one millisecond, the actual packet density was about 300,000. The traditional SPF-based routing was realized using the `SimpleSwitch` method provided by `Ryu`, while the SR implementation was based on the aforementioned weighted Dijksktra algorithm to compute paths based on collected data and the calculated weights.

Various tools were used to automate topology generation and evaluation using `CustomTopo` which is an interface for the `NetworkX` Python library. Further, IPv6 addresses must be assigned to hosts and the network must follow a specific format. These IPv6 addresses are then added to the `/etc/hosts` file using the `Python-Hosts` library. Following the addressing of the hosts, they are configured for operation with the `Free Range Routing` protocols `Zebra` and `IS-IS` using the `configure_hosts` method to create configuration files for each protocol and will start the protocols at each node. The dynamic topology creation suite also can plot the network and its shortest paths. The topology can be displayed circularly by taking the network with `NetworkX` along with the Python library `PyGraphviz`. Additionally, a shortest path can be passed to the `plot_topologies` method to visualize the shortest path, as shown in Fig. 1.

One strength of SR is its flexible forwarding compared to the SPF-based method. As stated, the paper adopted a simple dynamic weight calculation method based on CPU and memory usage. The forwarding paths may be rerouted for energy-saving purposes using SRv6. The dynamic topology creation suite not only can plot the network topology but also show the paths.

Based on the results, there was little difference between different sizes for the same type of topology. Hence, this paper only shows the average values for each topology type in TABLE II for SRv6 and traditional SPF routing.

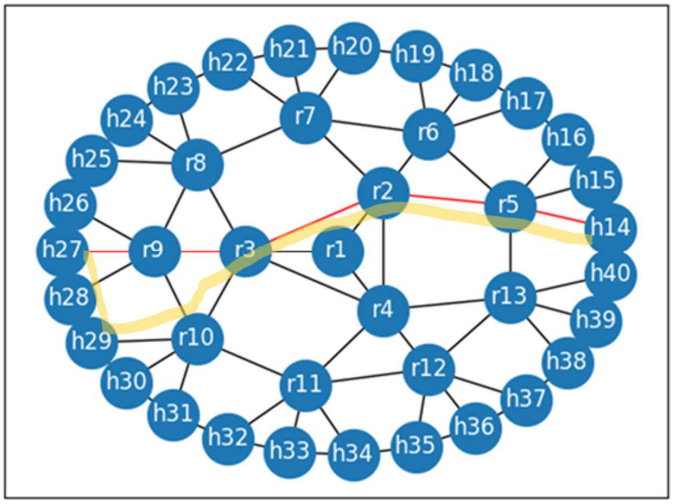

Fig. 1. Web topology of size 3 (r1: root, child node degree: 3, mesh for the 2nd and 3rd levels, the number of hosts for each leaf router: 3, each host is also an edge node with two neighbors, and SPF path in red and SRv6 path in yellow).

The green SRv6-based solution monitored the resource usage of each node every minute (for experiment purposes) and then calculated the shortest path based on updated usage and weights. The ingress node of a path performed rerouting using SR, if needed. For instance, based on collected resource usage, weight calculation, path computation at the controller, and rerouting using SRv6, Fig. 1 shows the new computed path in yellow **h14**, **r5**, **r2**, **r3**, **r10**, **h29**, **h28**, and **h27** in contrast to the static shortest path in red **h14**, **r5**, **r2**, **r3**, **r9**, and **h27**.

TABLE II. EXPERIMENT RESULTS

Topology	Operation State	Average CPU Usage (%)	Average Memory Usage (%)
Linear	**SRv6**	80.13	55.57
	SPF	74.39	50.61
Tree	**SRv6**	17.25	59.02
	SPF	76.82	71.69
Mesh	**SRv6**	0.33	55.77
	SPF	86.07	55.27
Web	**SRv6**	10.47	56.72
	SPF	84.23	55.83
Average	**SRv6**	27.05	56.77
	TR	80.38	58.35

As shown in TABLE II, the results for the Linear topology are lower for the traditional SPF than that of SRv6. These averages were calculated from data points collected during testing, with the overall average being a weighted average based on the different network sizes used. The memory usage results for mesh and Web topologies do not reveal much difference. However, the CPU usage outcomes for SRv6 are significantly lower than those of SPF in Tree, Mesh, and Web topologies. Energy consumption decreases as CPU utilization decreases. A CPU consists of millions of transistors. Modern CPUs are designed to be energy efficient, as most parts of the CPU are switched off when the CPU is idle. For advanced CPUs, lower CPU utilization will save even more energy [17]. Based on the measured resource usage results, SRv6-based rerouting already saves energy as the CPU utilization is mostly lower compared to SPF routing. Further, it has the potential to save even more energy if some links could be put into the energy-saving mode.

V. CONCLUSIONS AND FUTURE DIRECTIONS

Green TE is an important issue not only for network operators but also for environmental concerns. This paper presented a preliminary evaluation using SRv6 and SDN to reduce energy consumption for underutilized devices. The main contribution of the paper was an experimental validation and feasibility investigation by monitoring resource usage, invoking dynamic weight calculation, and performing energy-aware rerouting with SRv6/SDN. Experiments were conducted using various tools. The results showed that the SRv6-based green TE method has the potential to save energy consumption.

Further research is needed to investigate detailed resource usage on `ROSE`, `Mininet`, and real routers for SPF- and SRv6-based forwarding and link utilization using practical network traffic data, alternative weight calculation, and effective traffic reroute policies. SRv6 features, such as SID compression [4] and ECMP, warrant more research and evaluation.

REFERENCES

[1] GreenNet conference https://sites.google.com/view/greennet2024/home, (accessed May 2024).

[2] C. Filsfils, et al., *Segment Routing Part I*, 2017.

[3] C. Filsfils, et al., *Segment Routing Part II*, 2019.

[4] Z. Li, Z. Hu, and C. Li, *SRv6 Network Programming Ushering in a New Era of IP Networks*, CRC Press, 2021.

[5] P. L. Ventre *et al.*, "Segment routing: A comprehensive survey of research activities, standardization efforts, and implementation results," *IEEE Commu. Surveys and Tutorials*, pp. 182–221, Jan. 2021.

[6] A. Hassidim, D. Raz, M. Segalov, and A. Shaqed, "Network utilization: the flow view", *Proc. of IEEE INFOCOM*, 2013, pp. 1429–1437.

[7] B. Fortz, J. Rexford, M. Thorup, "Traffic engineering with traditional IP routing protocol", *IEEE Commu. Magazine*, 40(10), 2002, pp. 118–124.

[8] R. Carpa "Segment routing based traffic engineering for energy efficient backbone networks," *Proc. of IEEE Int'l Conf. on Advanced Networks and Telecommunications Systems*, Dec 2014, pp. 1–6.

[9] C. Thaenchaikun *et al.*, "Mitigate the load sharing of segment routing for SDN green traffic engineering," *Proc. of Int'l Symp. on ISPACS,* 2016.

[10] C.-H. Lung and H. Elbakoury, "Exploiting segment routing and SDN features for green TE", *Proc. of IEEE 8th Conf. on Network Softwarization*, 2022, pp. 49–54.

[11] S. Salsano *et al.*, "Netgroup/rose-SRV6-tutorial: SRV6 on a mininet topology with IS-is routing and a Controller," https://netgroup.github.io/rose/ (accessed May 2024).

[12] "Ryu SDN Framework." https://ryu-sdn.org/ (accessed March 29, 2024)

[13] "MiniNet: An instant virtual network on your laptop (or other PC)." http://mininet.org/ (accessed March 29, 2024).

[14] N. Okonor *et al*, "Dynamic link sleeping reconfigurations for green traffic engineering," *Int'l J. of Communication Systems*, 30(9), 2017.

[15] G. Yan, J. Yang, Z. Li, "OSPF extensions for MPLS green traffic engineering," *IETF draft-li-ospf-ext-green-te-01*, 2014.

[16] M. Zhang, C. Yi, B. Liu, B. Zhang, "GreenTE: power-aware traffic engineering," *Proc. of IEEE Int'l Conf. on Network Protocols*, 2010.

[17] F. Lamharras *et al*., "Green traffic engineering: solution for reducing energy consumption in SDN network with segment routing", *Journal of Theoretical and Applied Information Technology*, 2023, pp. 7266–7274.

[18] B. Z. M. Feng and C.-H. Lung, "A green computing based architecture comparison and analysis", *Proc. of IEEE/ACM Int'l Conf. on Green Computing and Communications*, 2010, pp. 386–391.