Best nexthop Load Balancing Algorithm with Inband network telemetry

Jiyoon Lim*, Sukhyun Nam*, Jae-Hyoung Yoo*, and James Won-Ki Hong*

*Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{limjiyoon, obiwan96, jhyoo78, jwkhong}@postech.ac.kr

***Abstract*—We proposed a load-balancing algorithm that helps avoid the congestion path by using global network information, then compared the proposed algorithm with the ECMP algorithm. The proposed algorithm collects the network information in real time by using In-band network telemetry. The collected network information is hop latency, queue depth, and link utilization. Each switch stores the network information and calculates the degree of congestion as the sum of metric values for each path. When the traffic arrives, each switch divides the traffic into flowlet units and forwards each flowlet to the least-congested path. We compared the throughput of the proposed algorithm to that of the ECMP algorithm in three scenarios. In the first scenario, we send 10 flows sequentially and measured the throughput per the number of flows. The proposed algorithm shows stable throughput, but the ECMP algorithm shows significant descent throughput after the number of flows exists three. In the second scenario, we send traffic from two different sources to two destinations. The proposed algorithm showed 27% higher throughput than the ECMP algorithm in this scenario. In the third scenario, we send a large burst of traffic from a single source to a single destination. The proposed algorithm showed 81% higher throughput than the ECMP algorithm in this scenario. These results show that the proposed algorithm performs better than the ECMP algorithm in congested status.**

***Index Terms*—data center, load balancing, p4**

I. Introduction

The topologies and traffic patterns of the data center network differ from the public Internet. There are two main reasons: first, since lots of Internet services are provided in cloud environments and each service requires high reliability and high performance, data centers use topologies designed to guarantee high path diversity that provides high bisection bandwidth. Secondly, data center network traffic fluctuates fastly and the topologies are less volatile than that of the Internet. In such situations, many studies are in progress to provide load balancing and to satisfy short flow completion time and high throughput.

ECMP [1] is the widely used load balancing algorithm used in the data center network. It distributes the flow evenly over each shortest path through static hashing. ECMP uses multi-path and it is simple to deploy it in the data center network. However, there is a limitation that when several large flows use the same path, ECMP distributes flows consistently regardless of congestion. To solve the problem, various load balancing algorithms that identify and utilize the network status information have been presented. Hedera [2] measures the amount of flow at each switch periodically by a single centralized controller. Besides, when a large flow is measured, a path is allocated by the minimax determination method from among the paths that can accommodate that large flow. The advantage of routing control with a centralized controller is that it can easily assign optimal paths to distribute the traffic load. However, the centralized mechanism shows limited scalability problem because all the computational burden is concentrated on the controller. Also, it is hard to process short flows because of the increased overhead on the controller. As a result, the controller may also process packets late. Delayed packet processing makes a significant impact on performance [3]. Therefore, many studies are being conducted in a distributed way to ensure that packets can be processed quickly and sent in the proper path even when such heavy traffic occurs.

From the perspective of operation granularity, we can classify load balancing algorithms in three categories: Flow-level, Packet-level, Sub-flow level. The flow-level load balancing algorithm suffers from performance because it is hard to achieve optimal path selection [4]. Packet-level load balancing algorithms can select optimal path however packet reordering problems can occur. The sub-flow level load-balancing algorithm is the compromise of the flow-level algorithm and packet-level algorithm. A sub-flow level load balancing algorithm can decide the optimal path and doesn't suffer from packet reordering problems. A flowlet [5] is one of the sub-flow concepts that divide a flow into bursts of traffic by time gap between inter-packet, as shown in Figure 1. The threshold value of the time interval that distinguishes flowlet is called flowlet timeout. If the time interval between two packets is shorter than the flowlet timeout, these two packets are in

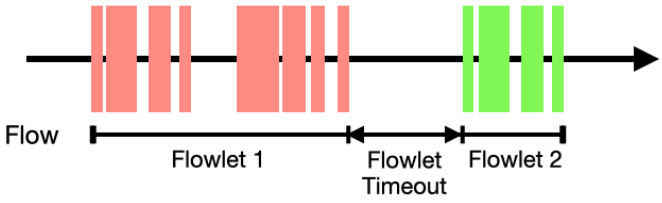

Fig. 1. Concept of flowlet.

978-3-903176-31-7 ©2020 IFIP

the same flowlet. Otherwise, two packets belong to different flowlet and independently routed. Flowlet has a characteristic that adjusts the number of packets in the flowlet according to the degree of congest. The more congested the network, the lower the number of packets in the flowlet. Due to that characteristic, it shows resilience to network asymmetry. Flowlet also can prevent packet reordering problem by making flowlet timeout exceed the highest value of latency among the pathways of the flow. E. Vanini *et al.* [5] shows a much better performance than Weighted ECMP (WCMP)using these kinds of features of flowlet. They simply allow each switch to select a random path for each flowlet without any complex mechanism. Since flowlet-level granularity show better performance, lots of load balancing algorithms use the flowlet.

Recently, P4 [6] and programmable switches are also used to collect precise network status information. P4 is the Domain-Specific language (DSL) for programmable data planes. The P4 program can program the protocol's header format definition, header parser, match-action table structure, and control flow. With P4, it is possible to parse the headers of packets coming into the switch according to defined rules, change the header through the match-action table, reassemble the packet, and export it to the output port. This allows information about networks to be collected in real-time and can be used for load balancing, even in distributed environments.

In this study, we propose an algorithm that collects three metrics in the data center network with Inband network telemetry and utilizes them for selecting the least congested path. Also, we validate the performance of our proposed algorithm with the virtual environment with two physical switches. We use the ONOS as a controller. The data plane is implemented with P4 and Control plane is implemented with Java ONOS application. We generate traffics that follows datacenter traffic pattern with the empirical traffic generator [7]. It shows 81% higher throughput than that of ECMP.

The remainder of this paper is organized as follows. Chapter 2 introduces the related work. Chapter 3 introduces the algorithms proposed in this study. Chapter 4 explains the experimental environment and results. Finally, Chapter 5 describes conclusions and future research and concludes this paper.

II. Related Works

A. P4 and In-band Network Telemetry

P4 is a programming language for expressing how packets are processed by the data plane to enable a programmable forwarding element. The name P4 comes from the original paper that introduced the language, "Programming Protocol-independent Packet Processors". P4 language and programmable switch concepts provide several advantages. First, it is protocol independent. The programmer can define header format and processing algorithms. Second, P4 is target independent language, thus the programmer doesn't have to consider the low-level hardware. P4 has many other advantages, so it is preferred over other systems.

Traditional network monitoring methods can be divided into two categories, active monitoring, and passive monitoring. The criteria for dividing two things are whether additional packets are generated or not. The active monitoring method generates additional packets and the passive monitoring doesn't. Traditional passive monitoring method such as SNMP has some limitations. The network monitoring tool installed on network equipment cannot be changed dynamically. Also, problems such as inefficient packet forwarding due to routing path setting error, intermittent packet loss due to equipment error, and asymmetric packet distribution of ECMP are hard to solve. To solve these limitations, a packet-level network monitoring system, so-called, Inband network telemetry (INT) [10] is proposed. INT collects packet-level network information in real-time. In particular, INT does not generate a separate probe packet by using the In-band method. Network traffic packets gather the network information from the network device, inserting the network information into the packet header, and carrying the network information. At specific switches, network information is extracted from the packet and store or send to another module (i.e. controller).

Information that can be collected through INT includes switch ID, information related to incoming and outgoing ports, flow delay information, and switch queue usage. Because of these characteristics, INT is used for several network monitoring studies. J. Hyun *et al.* [11] presents the design of the overall INT management architecture.

As shown in Figure 2, Each switch in INT detected target network is divided into three types: Source switch, Transit Switch, Edge switch. Packets collect network information by passing these types of switch sequentially. Source switch is the switch that inserts the telemetry instruction header into the packet header. Telemetry instruction defines what information to collects. Transit switch is the switch that inserts the network information called congestion data into the packet header from the network device according to telemetry instructions. The sink switch is the switch that extracts the congestion data and sends it to the Inband network telemetry collector module. Then it removes the telemetry instructions and congestion data from the packet and sends the packet to the network.

It has good features for monitoring such as visibility and collects data in real-time. However, it is hard to collect network information on every switch in the network. It is because the INT method only collects the network information of the switch on the path. To collects the network information of every switch, it should send at least one packet to every path. In the same way, it is hard to distribute the network information to every switch. To gather overall network information, we use the weighted-ECMP method that each switch distributes the packets across all the paths. It allows us to collects information throughout the network without additional traffic.

B. Load Balancing

In this section, we briefly introduce some load balancing algorithms.

Currently, Equal-cost MultiPath (ECMP) [1] is a load-balancing algorithm mainly used in data centers. ECMP distributes the flow evenly among the paths when there are

multiple shortest paths from the source to the destination. However, it has a problem that allocates to the same path when multiple large volumes of traffic are generated in the data center. So, recent studies try to grasp the degree of network congestion in real-time and use it for load balancing. These studies generally compare FCT (Flow Completion Time) with ECMP when measuring their performance.

Flare [5] is the first load balancing algorithm that proposes flowlet aware routing. It splits flows into flowlets at each switch and each flowlet is scheduled by weighted ECMP algorithm. The weight of each path is determined by time. Flare uses empirically obtained values of flowlet timeout. It shows lower error and better performance than the flow-based algorithm. Also, FLARE has a little overhead to split traffic into flowlets.

Let It Flow [8] is the simple load-balancing algorithm that uses flowlet. It only calculates the flowlet timeout. Using the characteristic that the flowlet size depends on the traffic state, this algorithm just distinguishes flowlet with pre-defined flowlet timeout, then each switch randomly selects a path for each flowlets. It is simple and has similar performance to other algorithms that use global network information. Also, it shows resilience to asymmetry.

Clove [13] is the centralized load balancing algorithm that uses flowlet. It uses the flowlet and schedules short flows with ECMP, and elephant flows with weighted round robins algorithm. To implement a weighted round-robin algorithm, it collects the global network information using probe packets. Clove can reduce flow completion time. It is deployed in soft edge switches to avoid hardware modification. But because of that, it is hard to deploy in the non-virtualized datacenter. Also, its flowlet timeout doesn't adjust for dynamic traffic.

CONGA [12] uses the decentralized structure to respond quickly to changes in network status by executing the load balancing algorithm on each switch. They identify network status information by adding a header to a packet that can store traffic congestion information. All leaf switches store traffic congestion information to other leaf switches. Each switch specifies the next path with the least congestion for each flowlet. They show the performance of 1.2 times that of ECMP. These studies have the disadvantage of storing the traffic congestion information of the entire network on all switches, which can not be used on general switches. Also, it can only be applied to a two-tier topology.

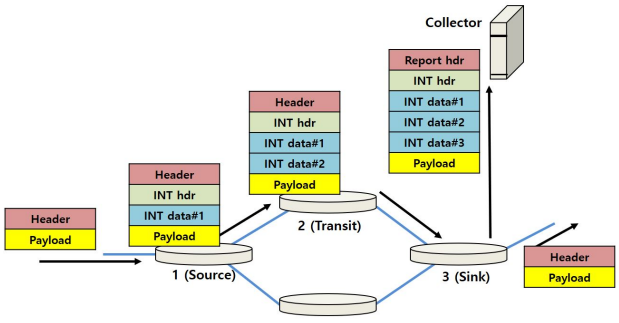

Fig. 2. Inband network telemetry and INT Collector

III. Design

In this section, we describe the process of the proposed load balancing algorithm method. It consists of three processes: (1) Network monitoring phase (2) Routing decision phase (3) Flowlet assignment phase. To implement our algorithm, we need three bottleneck register, a best-port register, a last-time register, a flowlet-id register, and a flowlet-port for each switch.

During the network monitoring phase: To collect INT data, the control plane assigns to each switch the function of either source switch or transit switch. The function is assigned depending on whether or not the switch is connected to the host. The source switch is connected to the host, and the transit switch is not. When a data packet arrives, the source switch inserts three INT headers into the packet. The INT header refers to the type of data is to be collected. Each INT header refers to three metrics; hop latency header, queue occupancy header, and link utilization header. A Transit switch measure values that correspond to each INT header and inserts the values into corresponding INT headers. If the INT header is empty, the transit switch inserts the measured value into the INT header. Otherwise, the transit switch compares the measured value and value in the INT header and inserts the bigger value into the INT header. The transit switch has three bottleneck registers that correspond to the three INT headers. Each register stores the key, bottleneck measurement. The key refers destination IP Address, egress port. The bottleneck measurement is the biggest measured value of the link in the path.

For example, In switch 1, the bottleneck register value of the port 3 is 1 that is the link utilization of link 1 (Fig 3. Since the link utilization value of link 2 is 10 which is greater than the link utilization value of link 4, the congestion data value of the path corresponding to port 1 in Switch 4 is 10.

During the routing decision phase, to avoid the congested paths, we calculate the least congested path by using the information from the Network monitoring phase. Computation of the least congested path entails three assumptions: 1) each switch knows the path to each host in advance; 2) each link has at least one packet to forward; 3) the possible range of each metric is known. The second assumption guarantees that each switch stores the overall network information. The third assumption enables the computation of the degree of congestion. The degree of congestion is defined by the sum of each metric with min-max normalization. Min-max normalization is the method that subtracts the current value by the lower bound of the metric and divides it by the size of the range. After calculating the degree of congestion, the transit switch calculates the least congested path. A transit switch has one best-port register that stores key, least-congested-value, best-port. The least-congested-value is the least degree of congestion among the possible ports. The best-port is the port that corresponds to the least-congested-value. We put the

source IP address as the key and store the least-congested-value and the best-port into the best-port register.

During the Flowlet assignment phase, before a packet is forwarded, we assign a flowlet id to it. We use a last-time register, a flowlet-id register, and a flowlet-port register in this phase. The last-time register stores the departure time of the previous packet with flow hash value as a key. The flowlet-id register stores the current flowlet id with flow hash value as a key. The flowlet-port register stores the port with flowlet id as a key. To assign the flowlet id of the current packet, we calculate the hash value of Source IP address, Destination IP address, Source port, Destination port, Protocol of the current packet. Then get the current flowlet id from the flowlet-id register, and get the departure time of the previous packet from the last-time register with the hash value. If the time gap between the current time and the departure time of the previous packet is bigger than the flowlet timeout, then we assign flowlet id + 1 and store the flowlet id in the flowlet-id register. Otherwise, we assign the same flowlet id as the previous packet. We define the flowlet timeout as a 50ms which is the round trip time of the current network. Then we store the best-port from the Routing decision phase with a flowlet id into the flowlet-port register.

IV. Experiment and Evaluation

In this experiment, we compared proposed algorithms with ECMP by implementing and comparing the throughputs.

First of all, we implement link utilization calculation methods. BMv2 switch and programmable switch support the direct extraction of hop latency and queue occupancy among congestion information. However, link utilization is not yet supported in the BMv2 switch and programmable switch. Therefore, we implement the additional function to get link utilization using DRE (Discounting Rate Estimator) algorithm which proposed in [12]. DRE could measure link utilization in a heuristic way. When DRE sends a packet from the switch to a specific port, it increases the link utilization value of that port by 1 and decreases the link utilization value by a constant percentage over a certain period. The DRE algorithm is simple to implement and respond fast to traffic bursts compared to the EWMA (Exponential Weighted Moving Average), which is used widely. In this experiment, the link utilization value is set to be halved every 50 ms.

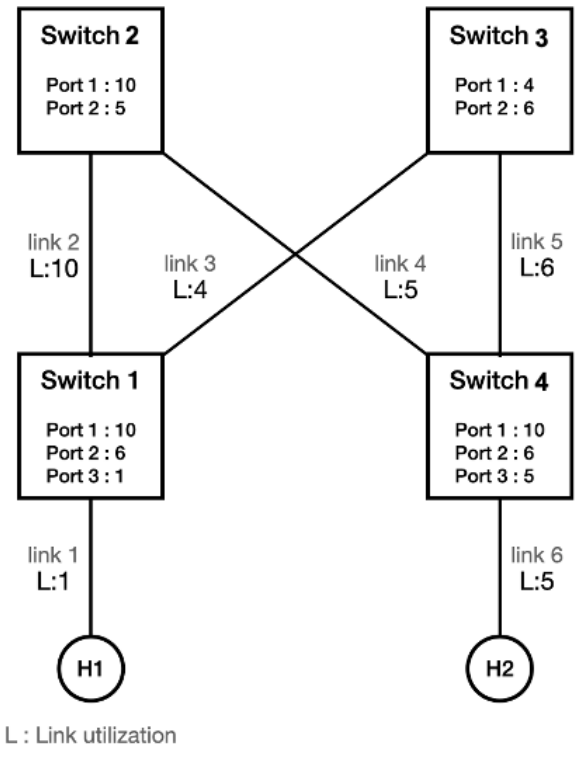

Fig. 3. Collected network information after network monitoring phase

We also implement a P4 program for load balancing is installed into each switch. The P4 program consists of header parser, match-action tables. We implement the flowlet-control match-action table, the basic routing match-action table, and the best next-hop-routing match-action table. The flowlet-control match-action table distinguishes different flowlet by the inter-packet gap, and assign different flowlet ID. The basic routing match-action table is for basic routing without the best next hop. In our topology, aggregation switches and core switches are forwarding the packets using this match-action table. The best-next-hop match-action table is for out load balancing algorithm. The matching key of the best-next-hop table is the source IP address, destination IP address, protocol number, source port, destination port, and flowlet ID. The action of the best-next-hop table is to do ECMP.

The load balancing program operates as follows: First, it recognizes and parses Ethernet, TCP/IP protocols, and probe packet protocol. Subsequently, the protocol number is checked to distinguish whether the packet is a UDP or TCP. If the switch is a source switch, it inserts the INT header into a packet. If the switch is a transit switch, it inserts network information corresponding to the INT header into a packet header field. If the switch is a sink switch, it extracts network information and INT header and sends network information to INTCollector and forward packet in a normal way. The flow is divided into flowlets based on the predefined flowlet timeout, and the flowlet unit is sent by the smallest congestion. We use 50ms as flowlet timeout because [8] suggests 50ms to 100ms as proper flowlet time out value.

Also, we implement the ONOS java application to easily deploy corresponding routing rules for each table. We test our load balancing programs with two testbeds, that is leaf-spine tree and fat-tree with different traffic generator and different test scenario.

The leaf-spine tree is constructed with Mininet and BMv2. We construct leaf-spine tree topology consisting of two leaf switches and four spine switches as shown in Figure 4. Each leaf switch connects with two hosts. The capacity of the link from the host to the leaf switch is 25Mbit/s, and the capacity of the link from the leaf switch to the spine switch is set to 8Mbit/s.

We use Distributed Internet Traffic Generator (D-ITG)

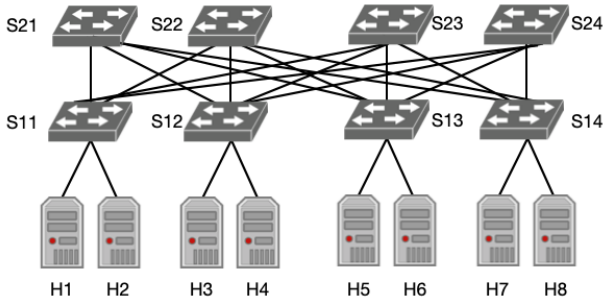

Fig. 4. Leaf-Spine tree Testbed

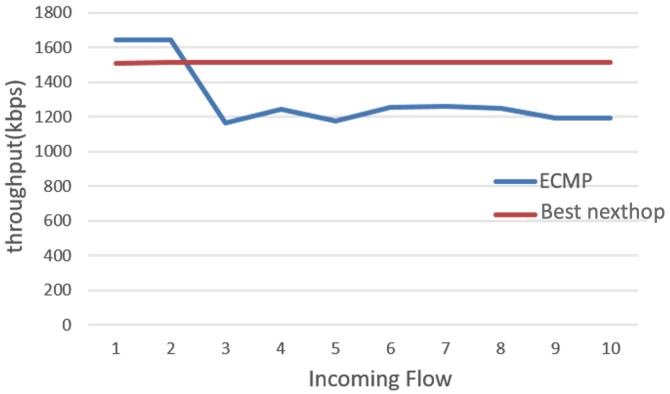

Fig. 6. throughput per the number of flows

and Iperf tools to generate traffic in the leaf-spine tree and empirical traffic generator in fat-tree topology. D-ITG is a program that can generate traffic at the packet level and can generate a flow by setting the time interval between packets and the packet size to follow a specific probability distribution. In this experiment, D-ITG is executed by setting the probability distribution of packet size and time interval to follow the distribution of data center network traffic. D-ITG can also measure flow completion time, delay, jitter, average throughput, etc.

We measured the throughput, and flow completion time of the best nexthop algorithms and the ECMP algorithm. In the simple forward experiments, we send 10 flows sequentially from host h1 to host h8 by using D-ITG and make one specific path (H3 - S11 - S23 - S12 – H7) especially more congested by sending additional traffic using D-ITG. Each flow generated by D-ITG is a total of 12800 kbytes and the size of the packet in the flow is 1500bytes. Inter departure time follows Poisson distribution on average 4.2ms. Then we measure the throughput by the number of flows (Fig. 6). Even the number of flows is increased, the best-nexthop algorithm shows the almost constant throughput; It shows 1507 kbps throughput regardless of the number of flows. On the other hand, the ECMP algorithm shows a significant variation depending on the number of flows; the throughput when the number of flows less than three is 1612 kbps, and the throughput when the number of flows equals and more than three is 1207 kbps. This result means that the best-nexthop algorithm is more resilient than the ECMP algorithm in the congestion state.

Fat-tree is constructed with Mininet, BMv2, and two Tofino switches, as shown in Figure 7. Switches with prefix C are core switches and they are physical programmable switches. Switches with prefix A are aggregation switches and switches with prefix E are Edge switches. They are BMv2 switches in Mininet. Each edge switch has one host. The capacity of the link from the host to the leaf switch is 10Mbit/s, and the capacity of the link from the leaf switch to the spine switch is set to 10Mbit/s.

Also, we tested in fat-tree with an empirical traffic generator. Empirical traffic generator generates traffic based on the user-defined probability distribution of traffic size. Table I shows a CDF of the request packet size distribution and it follows the traffic pattern from [9].

Using fat-tree we measure the flow completion time of the best nexthop algorithms and the ECMP algorithm with two scenarios; the asymmetric traffic scenario and the burst traffic scenario.

In the asymmetry traffic scenario, we send two kinds of

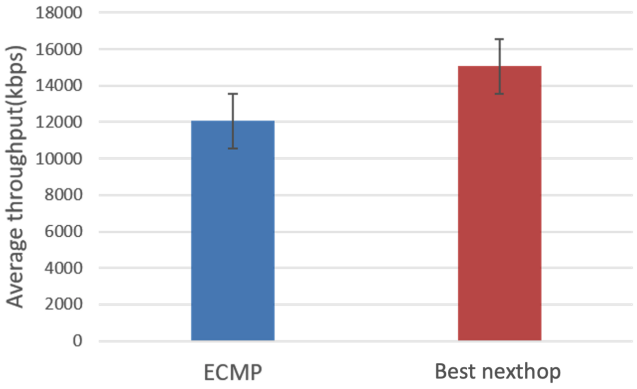

Fig. 5. Average throughput.

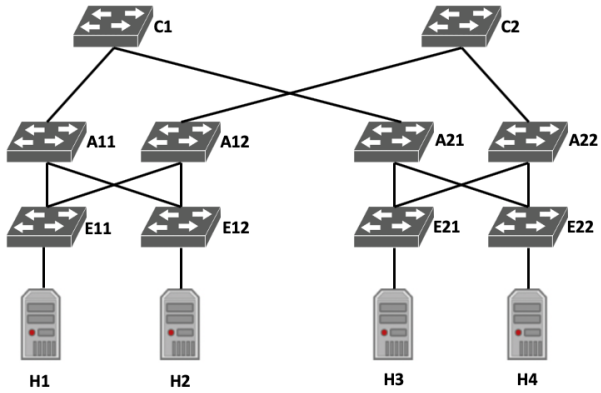

Fig. 7. Fat-tree Testbed

TABLE I
CDF OF GENERATED TRAFFIC PATTERN

Packet Size (bytes)	CDF
0	0
100	0.15
200	0.2
300	0.3
500	0.4
800	0.53
2000	0.6
10000	0.7
20000	0.8
50000	0.9
100000	0.97
300000	1

traffic and measure the flow completion time; First traffic is 5000 flows from H1 to H3 and H4, and the second traffic is 5000 flows from H3 to H1 and H2. Compared to the ECMP algorithm, the average throughput of the best-nexthop algorithm shows 8.13 Mbps which is 31% higher (Table II). Also, it shows the average flow completion time as 198.9 ms which is 4.4% lower than the ECMP algorithm. However, it shows a minimum flow completion time as 13.3 ms which is 12% higher minimum flow completion than the ECMP algorithm.

The second scenario is the burst traffic scenario which sends a total of 10000 flows from H1 to H3 and H4. More precisely, we send 3000 flows to H3 and sends 7000 flows to H4. Compared to the ECMP algorithm, the average throughput of the best-nexthop algorithm shows 9.2 Mbps which is 81% higher (Table III). Also, it shows a higher maximum flow completion time and average flow completion time. However, it shows a lower minimum flow completion time than the ECMP algorithm.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed an algorithm to collect information for each link in the network in real-time using a probe packet and use it to identify the congestion for each link and use it for load balancing. INT is network telemetry written with P4 language and is suitable for reading metrics with packet unit details in real-time. We also conducted a verification experiment of the proposed algorithm. Compared to the ECMP algorithm, the best-nexthop algorithm has a bigger overhead to compute the path but finds a less congested path. So, the best nexthop algorithm shows lower throughput or higher flow completion time in a non-congested state than the ECMP algorithm. Also, the best-nexthop algorithm shows higher throughput and lower flow completion time in a congested state. For future work, we will develop an algorithm using the INT method and compared it with the proposed algorithm. Also, we calculated the degree of congestion by simply adding the collected metrics. We will use a variety of techniques such as machine learning in calculating the degree of congestion.

Algorithm	Flow completion time			Throughput
	Max	Average	Min	Average
Best-nexthop	67870.6	393.2	6.9	9.2
ECMP	129520.9	610.6	6.3	5.1

TABLE II
RESULT OF FAT TREE SCENARIO 1 : ASYMMETRIC TRAFFIC

Algorithm	Flow completion time			Throughput
	Max	Average	Min	Average
Best-nexthop	78657.9	198.9	13.3	8.1
ECMP	104275.9	208.1	11.9	6.2

TABLE III
RESULT OF SCENARIO 2 : BURST TRAFFIC

ACKNOWLEDGMENT

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Government of Korea (MSIT) (No. 2017-0-00195, Development of Core Technologies for Programmable Switch in Multi-Service Networks).

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (2018-0-00749, Development of Virtual Network Management Technology based on Artificial Intelligence).

REFERENCES

[1] M. Chiesa, G. Kindler, and M. Schapira, "Traffic Engineering with Equal-Cost-MultiPath: An Algorithmic Perspective," IEEE Conference on Computer Communications, 2014.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," NSDI, 2010.

[3] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, p. 266, 2011.

[4] S. Rost and H. Balakrishanan. Rate-aware splitting of aggregate traffic. Technical report, MIT, 2003.

[5] S. Sinha, S. Kandula, D. Katabi, "Harnessing tcp's burstiness with flowlet switching," In Proceedings of the Third Workshop on Hot Topics in Networks HotNets-III, 15–16 November 2004.

[6] P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors," ACM SIGCOMM Coumputer Communication Review, vol. 44, no.3, pp. 87-95, 2014.

[7] Cisco. [n. d.]. Simple client-server application for generat- ing user-defined traffic patterns. https://github.com/datacenter/ empirical-traffic-gen. ([n. d.]). (Accessed on 07/24/2020).

[8] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," NSDI, 2017.

[9] M. Alizadeh et al., "Data center TCP (DCTCP)," ACM SIGCOMM Comput. Commun. Rev., vol. 40, no. 4, pp. 63–74, 2010.

[10] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band Network Telemetry via Programmable Dataplanes," ACM SOSR, 2015, pp. 2-3.
[11] J. Hyun, N. V. Tu, J. Yoo and J. W. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," International Journal of Network Management (IJNM) (SCIE), Volume 29, Issue 6, 2019, https://doi.org/10.1002/nem.2080.
[12] M. Alizadeh et al., "CONGA: distributed congestion-aware load balancing for datacenters," Proceedings of the 2014 ACM conference on SIGCOMM, 2014.
[13] N. Katta, M. Hira, A. Ghag, C. Kim, I. Keslassy, and J. Rexford, "Clove: Congestion-Aware Load Balancing at the Virtual Edge," in Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies -CoNEXT, 2017.