Real-time Virtual-network-traffic-monitoring System with FPGA Accelerator

Yuta Ukon, Shuhei Yoshida, Shoko Ohteru, and Namiko Ikeda

Abstract

There is a growing demand for a virtual-network-traffic-monitoring system for managing and controlling network services. Such a system must be able to handle high-load processing such as analyzing encapsulated packets and network-traffic classification using many header fields. To visualize network traffic for a virtual machine in real time, we propose a real-time virtual-network-traffic-monitoring system with a field-programmable gate array (FPGA) accelerator. Our system consists of a resource-saving hash-based network-traffic classifier (NTC) that classifies virtual network traffic at high speed using many search conditions. The hash-based NTC reduces memory resources by using a two-step hash search. Our system with this hash-based NTC provides a real-time visualization of multiple statistics such as the number of packets, bytes, microbursts, and histograms of jitter and latency for each virtual machine. To verify the performance of the hash-based NTC, we evaluated the number of searches per input packet. As a result of classifying virtual extensible local area network (VXLAN) packets into 10,000 categories using 17 header fields, the average number of searches executed with the hash-based NTC was about one-fourth that of a search-tree-based NTC. In addition, memory and logic-resource usage of the hash-based NTC were on average about 40 and 80%, respectively, which were less than those of several FPGA-based ternary content addressable memories with the same rules. Finally, we demonstrated that our system with the hash-based NTC visualizes VXLAN traffic for each virtual machine in real time.

Keywords: traffic monitoring, virtual network, FPGA, hash-based search

1. Introduction

Real-time network-traffic monitoring is important for managing network services in a datacenter. Datacenter traffic has been diversifying as a result of advances in network virtualization technology such as software-defined networking [1] and network function virtualization [2]. Therefore, network operators require a real-time network-traffic-monitoring system to detect problems in the virtual network.

A virtual-network-traffic-monitoring system constructed within a server enables the monitoring of communication with external machines and between virtual machines (VMs) in the server. However, in-server monitoring degrades the performance of VMs because the virtual-network-traffic-monitoring system uses the server's computing resources. To avoid this problem, such a system should be constructed on the virtual network. In this case, it must monitor virtual-network traffic to multiple servers. Traffic-monitoring software is not suitable for monitoring a high volume of virtual-network traffic because it cannot handle high-load processing such as analyzing encapsulated packets and classifying network traffic using many header fields at high speed. We propose a real-time virtual-network-traffic-monitoring system with

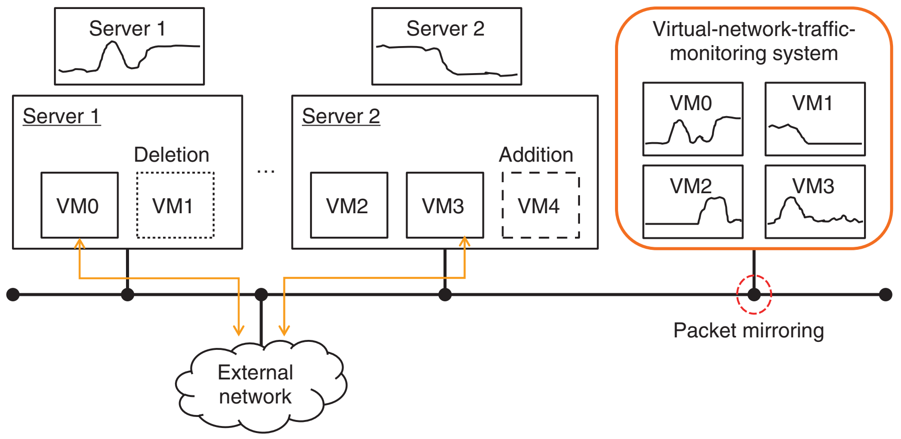


Fig. 1. Network-traffic monitoring using virtual-network-traffic-monitoring system.

a field-programmable gate array (FPGA) accelerator that processes encapsulated packets at high speed.

Our system visualizes virtual-network traffic at the VM, server, and network level to identify the cause of failure in network service. To provide this function, the system must have a network-traffic classifier (NTC) to classify virtual-network traffic. NTCs using a search tree [3, 4], which are often used in traffic-monitoring software, do not operate at high speed on the FPGA because it takes many processing cycles to search for a rule consisting of multiple conditions. A ternary content addressable memory (TCAM) [5–8], on the other hand, enables high-speed processing by searching for rules in parallel. However, virtual-network-traffic classification requires a large amount of memory resources when using many search conditions. Therefore, it may not be possible to implement a TCAM with sufficient rules in the FPGA. To classify virtual-network traffic using more rules, our system consists of a resource-saving NTC based on a hash method. This hash-based NTC uses a two-step hash search to reduce the memory resources needed.

The rest of the article is organized as follows. In Section 2, we introduce our real-time virtual-network-traffic-monitoring system with an FPGA accelerator. Section 3 discusses the shortcomings of conventional NTCs and describes our hash-based NTC. Section 4 discusses the experimental results from evaluating our hash-based NTC's processing performance and circuit area. Finally, Section 5 concludes the article.

2. Our virtual-network-traffic-monitoring system with FPGA accelerator

The systems described in this article are for monitoring virtual-network traffic on a network. **Figure 1** shows an example of monitoring for four VMs implemented on two servers. A network-traffic-monitoring system visualizes network traffic for each VM by analyzing packets copied with a router and network test access point. Such a system must process many encapsulated packets at high speed. High-speed packet processing can be achieved using an FPGA accelerator. An FPGA accelerator can change its configuration when the target network changes. Therefore, an FPGA-based network-traffic-monitoring system is suitable for monitoring a virtual network that continues to evolve rapidly.

2.1 System architecture

Our virtual-network-traffic-monitoring system uses an FPGA accelerator to analyze and classify encapsulated packets at high speed and calculate multiple statistics such as the number of packets, bytes, microbursts, and histograms of jitter and latency. These statistics are visualized using open-source software such as Kibana [9] and Zabbix [10]. The system also captures packets received before and after detecting microburst traffic. This function helps reduce the cost analyzing network failures.

Figure 2 shows a block diagram of our system. The FPGA accelerator contains a packet receiver, packet-header analyzer, NTC, statistics aggregator, and microburst detector [11, 12]. The packet receiver supports 10-Gigabit Ethernet. The packet-header analyzer

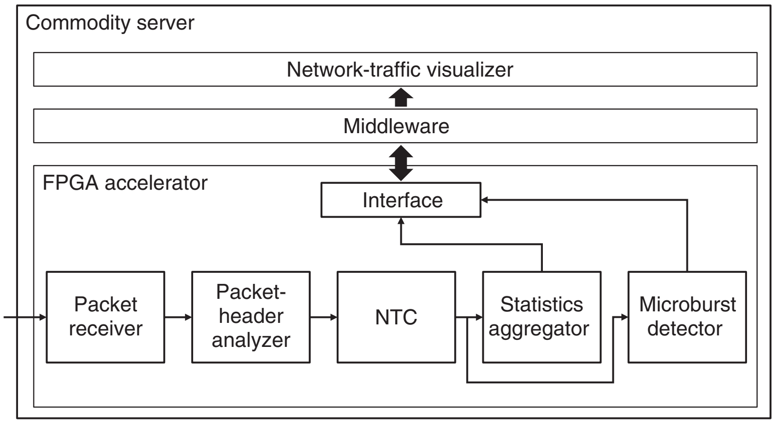


Fig. 2. Block diagram of our virtual-network-traffic-monitoring system with FPGA accelerator.

Table 1. Header fields used for virtual-network-traffic classification.

No.	Header	Field	# of bits
1	Outer Ethernet	Source MAC address	48
2		Destination MAC address	48
3		VLAN ID	12
4	Outer IP	Source IP address (v4 and v6)	32 or 128
5		Destination IP address (v4 and v6)	32 or 128
6		IP protocol number	8
7	Outer UDP	Source port	16
8		Destination port	16
9	VXLAN	VXLAN network ID (VNI)	24
10	Inner Ethernet	Source MAC address	48
11		Destination MAC address	48
12		VLAN ID	12
13	Inner IP	Source IP address (v4 and v6)	32 or 128
14		Destination IP address (v4 and v6)	32 or 128
15		IP protocol number	8
16	Inner TCP/UDP	Source port	16
17		Destination port	16

ID: identifier
IP: Internet Protocol
MAC: media access control
TCP: Transmission Control Protocol
UDP: User Datagram Protocol

analyzes a virtual extensible local area network (VXLAN) and virtual LAN (VLAN) packet and extracts several header fields from the packets. This analyzer also supports packets encapsulated in both VXLAN and VLAN. Although analyzing an encapsulated packet requires more calculation than analyzing an un-encapsulated packet, the packet-header analyzer achieves high throughput by pipeline processing. The NTC uses inner header fields as well as outer header fields to classify the packets in accordance with the VM, server, and network. **Table 1** shows the 17 header fields used in the NTC. Since classification using multiple header fields increases the amount of calculation, the system requires a high-performing NTC to process many packets at high speed. The statistics aggregator aggregates the number of packets, bytes, and histograms of jitter and latency for each VM, server, and network. These

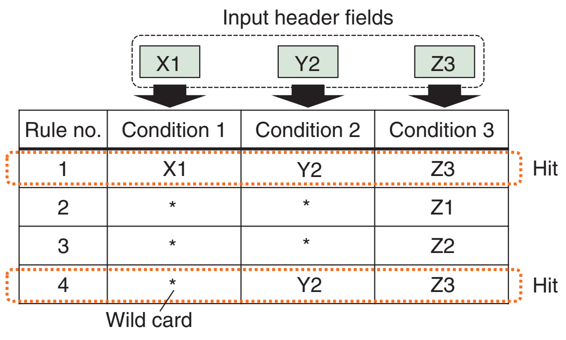


Fig. 3. Concept of NTC using partial match search.

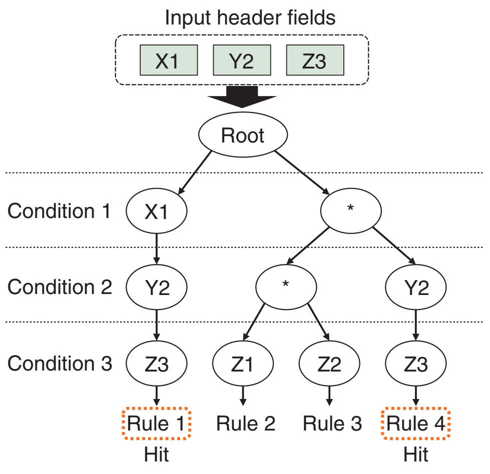


Fig. 4. Network-traffic classification using search tree.

statistics are then sent to the middleware via a PCI (Peripheral Component Interconnect) Express bus, and the microburst detector detects network traffic that increases rapidly within 100 microseconds and counts the number of microbursts. The microburst detector captures packets in the ring buffer until a microburst is detected. This method enables only packets received before and after the microburst to be captured. As mentioned above, our system enables real-time virtual-network-traffic monitoring by offloading high-load processes to the FPGA accelerator.

3. NTC

To classify network traffic by using a VM, server, and network, the NTC searches for rules that have different conditions. For example, outer and inner Internet Protocol (IP) addresses and VXLAN network interfaces (VNIs) are necessary for VM classification, whereas only a VNI is used for network classification. To enable flexible traffic classification, our system uses an NTC with a partial match search algorithm. The algorithm can potentially reduce FPGA memory resources because it requires minimal rules for flexible classification. **Figure 3** shows the concept of this NTC using the partial match search. Three input header fields (X1, Y2, and Z3) are compared with four rules consisting of three conditions. Comparing the input header fields with Conditions 1, 2, and 3, X1 matches the first rule, Y2 matches the first and fourth rules, and Z3 matches the first and fourth rules. These input header fields also match wild cards. Hence, X1 and Y2 also match the second, third, and fourth rules and the second and third rules, respectively. From the above, it is determined that the first and fourth rules match for all the input header fields. Therefore, the partial match search algorithm enables flexible searching with a small rule table since unnecessary conditions can be compressed using wild cards. In implementing this algorithm, it is important to suppress the amount of computation for achieving high-speed operation. It is also important to suppress the volume of memories and logic resources because our system has to implement it on an FPGA.

3.1 Conventional implementation approach

A search tree with wild card nodes can potentially conserve memory resources by reducing the number of nodes. However, it may be computationally expensive because all follower nodes of the wild card node must be checked. **Figure 4** shows an example of network-traffic classification using the search tree under the same conditions when using the NTC illustrated in Fig. 3. In this case, Rule 1 is found as a matching rule by searching the X1 node of Condition 1, Y2 node of Condition 2, and Z3 node of Condition 3 in this order. Rule 4 is also found by searching all nodes on the right side of the root node. As this example shows, almost all nodes must be checked to find multiple matching rules. The search tree compares input header fields with nodes sequentially; hence, the processing speed is slow on the FPGA when using a low clock frequency.

A TCAM enables high-speed classification by searching rules in parallel. It also classifies network traffic flexibly using rules consisting of 0s, 1s, and wild cards. **Figure 5** illustrates the TCAM architecture consisting of AND circuits and one-bit comparators

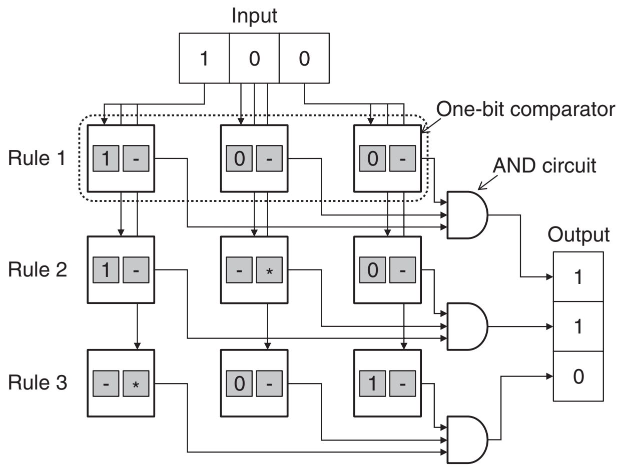


Fig. 5. TCAM architecture.

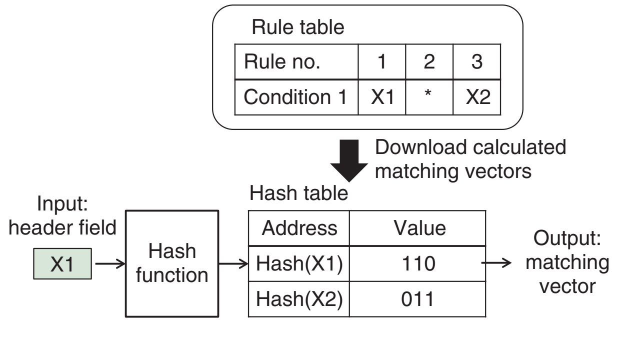


Fig. 6. Operation of hash-based NTC, which obtains a matching vector from the hash table. The matching vector is calculated in advance and downloaded to the hash table.

containing two memories: condition and wild card. The condition memory holds a one-bit rule, and the wild card memory holds a valid flag. The one-bit comparator outputs a high-level signal if the condition memory matches the input or the wild card memory is valid; otherwise, it outputs a low-level signal. The AND circuit collects outputs of one-bit comparators and determines whether the rule matches the input data. For example, if the TCAM shown in Fig. 5 receives input data “100,” all one-bit comparators in Rules 1 and 2 output the high-level signal. The AND gates collect these signals and output the matching vector “110,” which means the input data match Rules 1 and 2. These one-bit comparators work in parallel, so the TCAM operates at high speed. However, the TCAM uses a large volume of memory and logic resources since it requires two memories and a comparator for each one-bit comparator. Therefore, this approach is unsuitable for our system implementation.

3.2 Implementation approach for our system (hash-based NTC)

For the above-mentioned reason, we devised a hash-based NTC for high-speed virtual-network-traffic classification. **Figure 6** shows the operation of the hash-based NTC. When receiving the input header field, the hash-based NTC converts it into a hash value using the hash function and obtains the matching vector from the hash table by using that value as the search key. In this example, Rules 1 and 2 are matched to the input header field X1, and the matching vector “110” is output with a single access to the hash table. Therefore, the hash-based NTC operates at high speed since it can obtain the matching vector with a small number of memory accesses. The hash-based NTC can also reduce memory and logic resources because it does not compare the input and conditions bit-by-bit. Although this NTC does not allow bitwise wild cards, flexible virtual-network-traffic classification is still feasible by using wild cards per condition.

Figure 7(a) shows the overall architecture of the hash-based NTC. The architecture consists of hash functions, condition comparators, and an aggregator. The condition comparator evaluates an input header field using a rule table for one condition. The aggregator collects the evaluation results of the condition comparators and calculates the matching vector. This hash-based NTC works at high speed even if the number of conditions increases because the condition comparators operate in parallel, making it suitable for classifying network traffic using many header fields.

A typical hash table uses a large memory space to avoid memory-address conflict; however, this results in inefficient memory usage. Our condition comparator solves this problem by searching tables in two steps. **Figure 7(b)** shows a detailed diagram of the condition comparator. It includes the address table (hash table), multiple candidate memories (CMs), and a selector. The search mechanism is as follows. When receiving the hash value, the condition comparator divides the hash value into two values: a search key and reference value. The search key is used in the address table to extract an address for the CMs. When a small search key is used, the address table may output the same CM address for different inputs due to a memory-address conflict. This problem

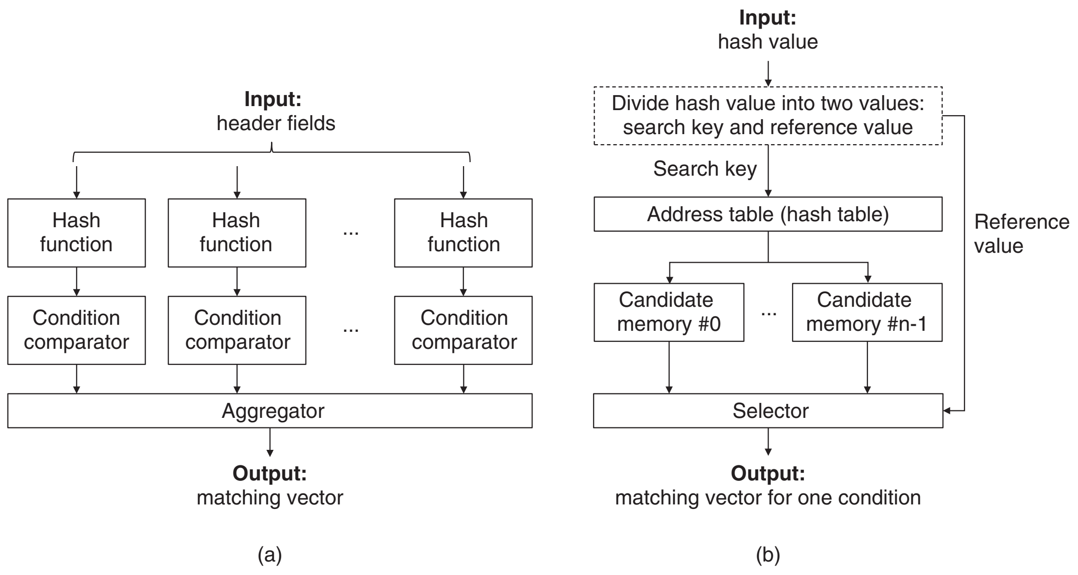


Fig. 7. Block diagrams of hash-based NTC: (a) overall architecture and (b) detailed condition comparator.

can be avoided using multiple CMs and the selector. The CMs receive the CM address from the address table and output candidate values and comparative values stored at the address. The selector selects one of the candidate values by comparing the comparative values with the reference value and outputs it as a matching vector for one condition. If the bit widths of the reference value and comparative value are sufficient, the selector can select the appropriate candidate value. Although this architecture has overhead, which holds the comparative values in the CMs, the memory-resource usage becomes highly efficient as the total usage is reduced by storing candidate values in the CMs.

The memory-resource usage of the address table and the CMs can be estimated using the following equations. Note that MRU_{AT} and MRU_{CM} are memory-resource usages of the address table and CMs, respectively.

$$MRU_{\mathrm{AT}} = 2^N \times \lceil \log_2 R \rceil \qquad (1)$$
$$MRU_{\mathrm{CM}} = C \times R \times M + C \times R \times R, \qquad (2)$$

where N is the bit width of the search key, R is the number of rules, C is the number of CMs, and M is the bit width of the reference value. The first and second terms on the right side in Eq. 2 are the memory-resource usage required to store the comparative values and candidate values, respectively. The MRU_{AT} increases exponentially for N because the search key is used as addresses in the table. In contrast, as N increases, MRU_{CM} decreases because fewer CMs are used to store candidate values. Due to the trade-off between Eqs. 1 and 2, the condition comparator should be designed with the appropriate parameters.

4. Experiment

To verify the advantages of the hash-based NTC, we evaluated its processing performance and circuit area.

4.1 Implementation

We designed our hash-based NTC that classifies VXLAN traffic using rules consisting of 17 conditions. MurmurHash3 [13] was used in the hash function because it is relatively unlikely to have output collisions between different inputs. To determine the optimal parameters for our implementation, we evaluated the maximum number of output collisions of the hash function. **Figure 8** shows the maximum number of output collisions when 500 random hash keys were input and the memory-resource usage of the hash-based NTC was estimated from Eqs. 1 and 2. Note that the maximum numbers of output collisions were the highest values in several simulations. The bit width of the hash function output was set to 32-bit. This figure shows that the maximum number of collisions decreases as the bit width of the search key increases. However, the memory-resource usage

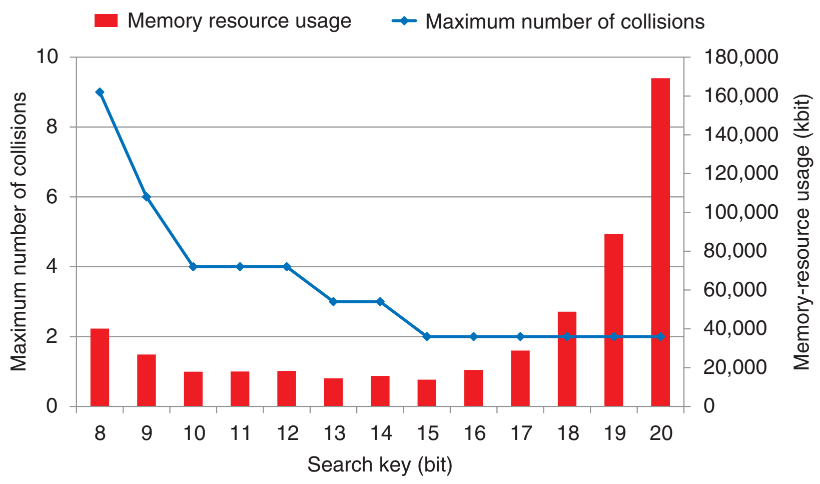


Fig. 8. Maximum number of output collisions when 500 random hash keys are input to the hash function using MurmurHash3 and memory-resource usage of the hash-based NTC.

Table 2. Implementation parameters.

Parameter	Variable	Value
Search key (bit)	N	16
Maximum # of address collisions	R	4
# of rules	C	500
Reference value (bit)	M	16

of the hash-based NTC increases when using 16-bit or higher search keys because the memory-resource usage of the address table increases rapidly. The figure also shows that the memory-resource usage of the hash-based NTC is minimized using the 15-bit search key and two CMs for each condition comparator. Given these results, we used the parameters shown in **Table 2** for the hash-based NTC and implemented it with an Arria 10 GX1150 FPGA [14].

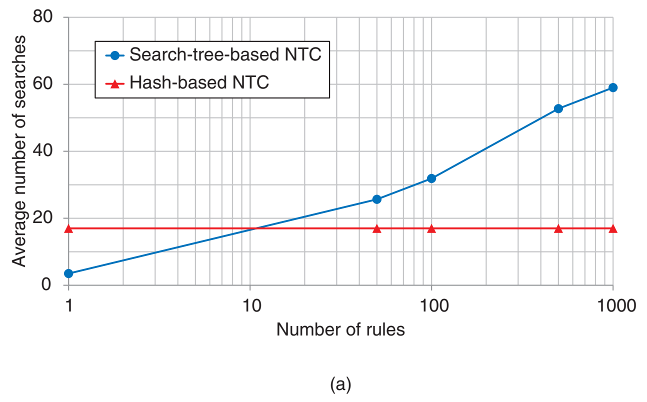
4.2 Evaluation

Figures 9(a) and **(b)** show the average number of searches for 100,000 inputs for a search-tree-based NTC and the hash-based NTC. Note that these results were evaluated through simulation. We used the search-tree-based NTC with a Patricia tree [3], which can efficiently store long strings, for comparison. Figure 9(a) shows that on average the search-tree-based NTC executed more searches than the hash-based NTC when there were more than ten rules due to the increase in nodes to be searched. The average number of searches of the hash-based NTC was also constant regardless of the number of rules. These results indicate that each condition comparator accessed the internal memories only once for each input. Figure 9(b) shows that the average number of searches using 17 conditions in the hash-based NTC is about one-fourth that of the search-tree-based NTC. The difference in the throughput of these NTCs is even greater since the hash-based NTC executes the search for each condition in parallel.

We implemented these NTCs, which classify virtual-network traffic using 17 conditions, on an FPGA operating at 100 MHz. The search-tree-based NTC processed 1.3 million packets per second, meaning that it is capable of wire-speed processing for an input rate of about 0.9 Gbit/s. The hash-based NTC processed one hundred million packets per second and executed wire-speed processing for a higher input rate of about 67 Gbit/s.

Table 3 shows the logic- and memory-resource usages of the hash-based NTC and several FPGA-based TCAMs. The resource usage of the hash-based NTC was calculated using the Intel Quartus Prime version 17.1.1. The amount of memory and logic resources per unit was calculated by dividing the logic- and memory-resource usage by the product of the number of rules and total bits of the conditions. In other words, these indicators are the amount of resources required to store a one-bit rule.

Comparing the indicators of each circuit, the memory- and logic-resource usage of the hash-based NTC was on average about 40 and 80% less than that of several FPGA-based TCAMs, respectively. This



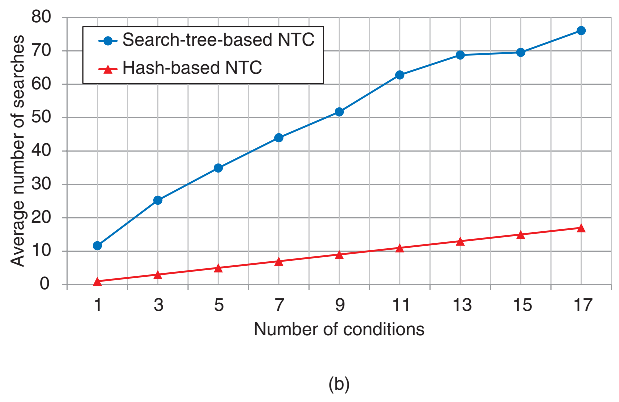


Fig. 9. Average number of searches per input for (a) number of rules and (b) number of conditions. The number of rules in the experiment was 500.

Table 3. FPGA-resource usage of the hash-based NTC and FPGA-based TCAMs.

Design	Size	Speed (MHz)	LUTs	Memory (kbit)	LUTs/Size	Memory/Size
Xilinx Locke [6]	256 × 32	130	4576	1152	0.56	140.63
UE-CAM [7]	512 × 36	202	3652	1152	0.2	62.5
RAM-based TCAM [8]	1024 × 150	150	48,552	9792	0.32	63.75
Hash-based NTC	500 × 832	100	26,480	19,040	0.06	44.7

LUT: look-up table
RAM: random access memory

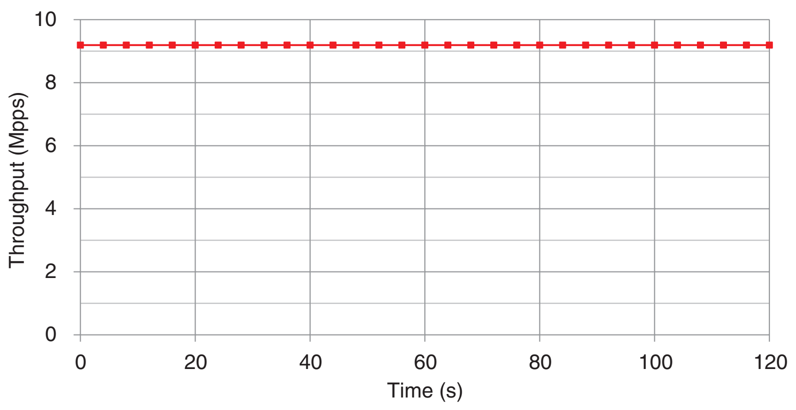


Fig. 10. Throughput for VXLAN packets with the shortest length.

indicates that the hash-based NTC can have more rules than FPGA-based TCAMs while using the same amount of FPGA resources.

Finally, we evaluated the performance of our system with the hash-based NTC. **Figure 10** shows the throughput for VXLAN short packets with a length of 116 bytes. The results indicate that the system achieved a theoretical performance of 9.19 mega-packets per second (Mpps) throughput at an input rate of 10 Gbit/s. The figure also shows that the system processed the packets without packet loss under a high workload. **Figure 11** shows the results of

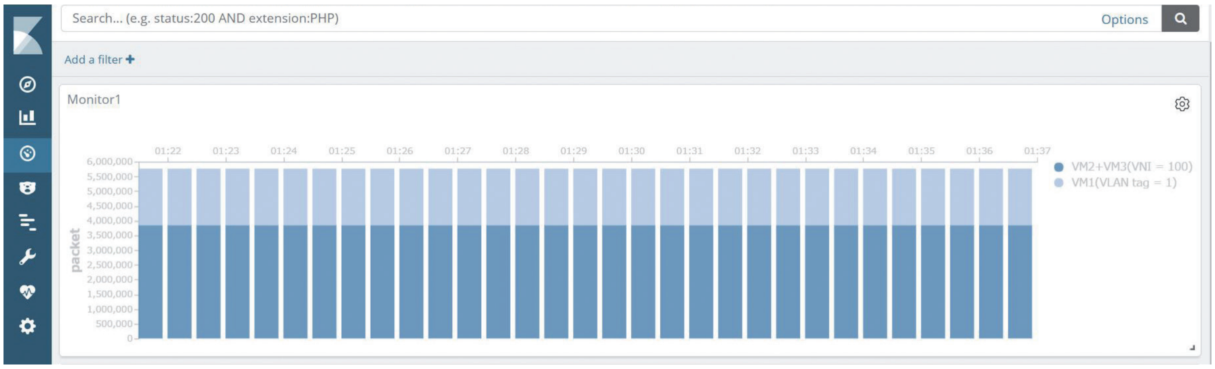


Fig. 11. Network-traffic visualization by VM and VXLAN network.

visualizing the number of packets in units of VM and VNI using Kibana. The system classified VXLAN traffic flexibly with the hash-based NTC and visualized communication volumes of different groups at short intervals.

5. Conclusion

We proposed a real-time virtual-network-traffic-monitoring system with an FPGA accelerator to monitor VXLAN and VLAN traffic. The key module in the system, the hash-based NTC, attained high-speed, flexible virtual-network-traffic classification with fewer FPGA resources by using a two-step hash search. The experimental results indicate that the hash-based NTC used fewer logic and memory resources compared with several FPGA-based TCAMs. Finally, we determined that our system with the hash-based NTC was able to visualize the amount of VXLAN traffic in real time for a 10-Gbit/s input rate.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined Networking: A Comprehensive Survey," Proc. IEEE, Vol. 103, No. 1, pp. 14–76, Jan. 2015.

[2] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," IEEE Commun. Surv. Tutor., Vol. 18, No. 1, pp. 236–262, 1st quarter 2016.

[3] K. Sklower, "A Tree-based Packet Routing Table for Berkeley UNIX," USENIX Winter, pp. 93–104, Dallas, TX, USA, Jan. 1991.

[4] P. Gupta and N. McKeown, "Algorithms for Packet Classification," IEEE Netw., Vol. 15, No. 2, pp. 24–32, Mar. 2001.

[5] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," IEEE J. Solid-State Circuits, Vol. 41, No. 3, pp. 712–727, Feb. 2016.

[6] K. Locke, "Parameterizable Content-addressable Memory," Application Note: Xilinx FPGAs, XAPP1151 (v1.0), Mar. 2011.

[7] Z. Ullah, M. K. Jaiswal, R. C. C. Cheung, and H. K. H. So, "UE-TCAM: An Ultra Efficient SRAM-based TCAM," Proc. of TENCON 2015 – 2015 IEEE Region 10 Conference, Macao, China, Nov. 2015.

[8] W. Jiang, "Scalable Ternary Content Addressable Memory Implementation Using FPGAs," Proc. of the ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 71–82, San Jose, CA, USA, Oct. 2013.

[9] Y. Gupta, "Kibana Essential," Packt Publishing Ltd, 2015.

[10] R. Olups, "Zabbix 1.8 Network Monitoring," Packt Publishing Ltd, 2010.

[11] S. Yoshida, Y. Ukon, S. Ohteru, H. Uzawa, N. Ikeda, and K. Nitta, "FPGA-based Network Microburst Analysis System with Flow Specification and Efficient Packet Capturing," Proc. of 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 29–32, July 2020.

[12] S. Yoshida, Y. Ukon, S. Ohteru, H. Uzawa, N. Ikeda, and K. Nitta, "FPGA-based Network Microburst Analysis System with Efficient Packet Capturing," J. Opt. Commun. Netw., Vol. 13, No. 10, pp. E72–E80, Oct. 2021.

[13] A. Appleby, "MurmurHash," 2008, https://sites.google.com/site/murmurhash/

[14] "Intel®Arria®10 Device Overview," A10-OVERVIEW, Nov. 2018.

Yuta Ukon

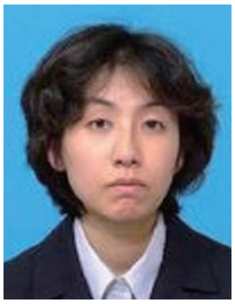
Deputy Senior Engineer, NTT Advanced Technology Corporation.

He received a B.E. and M.E. in engineering from Osaka University in 2010 and 2012. In 2012 he joined NTT Microsystem Integration Laboratories and has been engaged in research and development on a packet processing accelerator and packet processing circuit. He moved from NTT Device Innovation Center to NTT Advanced Technology in July 2021. His research interests include hardware accelerators for software-defined networking and network function virtualization, hardware and software co-design technologies, and packet processing circuit. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE).

Shuhei Yoshida

Engineer, NTT Device Innovation Center.

He received a B.E. and M.E. in computer science and systems engineering from Kobe University, Hyogo, in 2014 and 2016. In 2016, he joined NTT Device Innovation Center and has been engaged in research and development on hardware design methodology and FPGA-accelerated systems. He is a member of IEICE.



Shoko Ohteru

Research Engineer, NTT Device Innovation Center.

She received a B.S. in physics from Ochanomizu University, Tokyo, M.S. in physics from the University of Tokyo, and Ph.D. in engineering from Nihon University, Tokyo, in 1992, 1994, and 2011. She joined NTT Telecommunication Networks Laboratories in 1994. She is currently with the NTT Device Innovation Center. She is a member of IEICE.

Namiko Ikeda

Senior Research Engineer, NTT Device Innovation Center.

She received a B.E. and M.E. in inorganic materials from Tokyo Institute of Technology in 1996 and 1998. In 1998, she joined NTT, where she engaged in research and development of an image-enhancement algorithm for single-chip fingerprint sensor/identifier large-scale integrated circuits. She is currently with the NTT Device Innovation Center and engaged in research and development of a virtual-network-traffic-monitoring system with an FPGA accelerator. She is a member of IEICE.