LightFlow: Speeding Up GPU-based Flow Switching and Facilitating Maintenance of Flow Table

Nobutaka Matsumoto and Michiaki Hayashi
Integrated Core Network Control and Management Laboratory
KDDI R&D Laboratories
Fujimino, Saitama, Japan

***Abstract*—Flow-based switching is increasingly important in accordance with the growing demand for in-network processing for cloud applications. Flow switching performance tends to be degraded in proportion to the number of flow entries. To reduce the number of flow entries, they can be aggregated by applying wildcard fields. Meanwhile, the existence of the wildcard entry adversely affects the use of a hash-based lookup on a flow table, and thus a linear search is inherent in flow switching. However, the linear search is currently the primary cause of performance limitation. To date, two flow tables, one for hash-based lookup and the other for a wildcard-enabled linear search, have been used for flow switching. While hash-based table lookup is much faster than linear search, it needs to be manually updated for every exact match entry. Maintaining a hash-based table of all the flow switches is not feasible from a network operator viewpoint. In this paper, LightFlow, a mechanism to accelerate software flow switching processing and relieve the burden of maintaining the flow table is proposed. In LightFlow, two-dimensional parallelization of a linear search is introduced to accelerate lookup of the wildcard-enabled flow entries. It also introduces a mechanism that allows updating of the hash table to be performed automatically based on the result of wildcard-aware table lookup. LightFlow satisfies both the need for fast table lookup and feasibility of flow table management which needs to allow a large number of wildcard entries. Experimental results show that LightFlow can increase the speed of lookup of a wildcard-aware flow table three-fold or more compared to the current GPU-based wildcard search mechanisms.**

Keywords-wildcard; table lookup; flow switch; GPU; flow-based networking

I. INTRODUCTION

Recently, rapid growth of cloud-based services has led to the need for fine-grained traffic management in transport networks. Consequently, one objective is to carefully control the quality required for each application. The other objective is to change the functional behavior (e.g., address translation, cache, tunneling, etc.) of the network. To achieve such fine-grained manageability, flow-based networking is expected to play a key role in transport networks. OpenFlow [1] is a typical technology to support flexible flow switching in the network.

From the viewpoint of creating a switching substrate, recent progress in software-based nodes is significant [2][3][4][5][6]. The software-based nodes originally have the high flexibility that allows their functionalities to be modified. To improve packet processing performance, graphic processing unit (GPU)-based flow switching has been proposed [7]. However, even in GPU-based flow switching, forwarding performance is highly dependent on the size of the flow table, which is continuously increased as the number of accommodated users increases. This remaining issue is certainly blocking flow switches from being rolled out to large-scale networks.

The remaining issue is derived from the table lookup operation of the packet forwarding process. For flow switches, two kinds of lookup mechanisms have been introduced so far. One is the exact match table and the other is the wildcard-aware table [8]. In the exact match table, any flow entry can be deterministically identified by using a hash function, while the hash calculation requires all the flow-identifying fields (e.g., MAC addresses, VLAN, MPLS label, IP addresses, TCP/UDP port, etc.) to be explicitly determined for all the flow entries. In the wildcard-aware table, the flow entry needs to be searched linearly from the top of the table, while the wildcard aggregates the entries. Thus, the wildcard-aware table has a longer lookup time compared to the hash-based exact match table. In conventional implementations, flow tables including the aforementioned two kinds require manual or semi-automatic update. From the viewpoint of the flow table maintenance, wildcard-aware table is desirable, but the performance is too low. While exact match table provides better performance, manual update of its huge entries are not feasible for operators. In addition, solutions for coordinating them to automatically update the exact match table have not been provided to date.

In this paper, LightFlow, a mechanism to accelerate software flow switching processing and relieve the burden of maintaining the flow table is proposed. LightFlow introduces two techniques: an acceleration mechanism for looking up the wildcard-aware table and automatic updating of the hash-based exact match table. The wildcard-aware flow table lookup is designed on a GPU-accelerated platform incorporating two-dimensional parallelized lookup of the entries. Automatic update of the hash-based exact match table is carried out in coordination with the result of the wildcard-aware flow table lookup. The proposed mechanism addresses the aforementioned requirements (i.e., lookup time on wildcard-aware table and maintenance feasibility of hash-based exact match table) by advancing the parallelism of linear search for wildcard-aware flow table lookup and relieving operators of the need to manually maintain all the exact match entries.

978-1-61284-0833-6/12/$26.00 ©2012 IEEE

The rest of this paper is organized as follows: Section II describes the existing issues affecting flow table handling in flow-based networking. Section III provides the architecture and mechanisms of LightFlow. Section IV presents a demonstration of the prototype flow switching node and analyses the results of performance measurement. Section V discusses the feasibility of the proposed mechanism and compares it with other mechanisms. Section VI describes related work. Section VII concludes this paper.

II. THE EXISTING ISSUES IN HANDLING FLOW TABLES

In flow-based networking, each node recognizes "flow" by matching multi-layer information such as physical port, MAC address, VLAN ID, IP address, QoS values and port numbers. According to the policy rules, each node controls the forwarding destination, queue assignment, and updating of the packets. To make the flow-based networking practical, there are two existing issues that need to be resolved.

A. Lookup Time of Flow Entry

Fig. 1 (a) and (b) shows the conventional flow table lookup mechanisms: hash-based exact match table and wildcard-aware linear search table, respectively. The hash-based exact match table has columns for hash value, flow identifying fields, and destination. The hash value is calculated from the values of flow identifying fields. Searching an entry in the table is carried out by calculating the hash value for a received packet, and directly jumping to the corresponding entry. Since both the hash calculation and accessing the entry are a fixed-time simple process and independent of the number of the entries, this mechanism enables fast table lookup regardless of the table size. However, all the flow identifying fields are required to be explicitly determined. Since hash functions basically output different hash values from different arguments, for the entry to be matched to the hash value calculated from packets, all the flow identifying fields of the entry need to be identical to those of the corresponding packet. This limitation results in the growth of the table size.

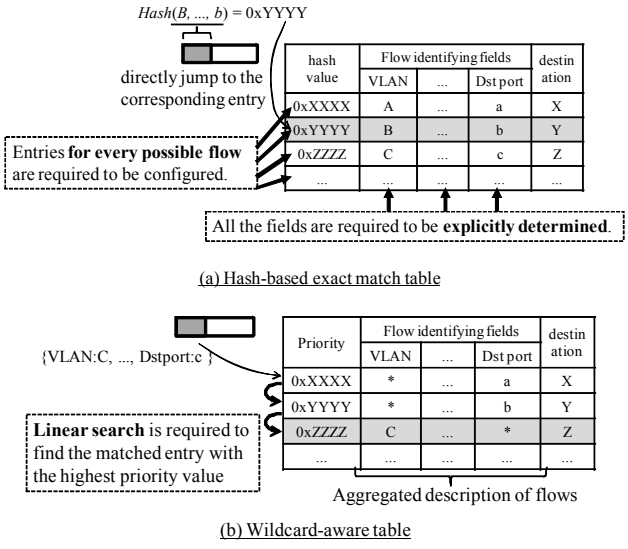

Figure 1. Conventional flow table lookup mechanisms.

The wildcard-aware linear search table has columns for the priority field, flow identifying fields, and destination. The flow identifying fields can include wildcard, which enables the aggregation of flow entries. Since each entry can include wildcard, a packet can be matched to multiple entries in this table. To distinguish which entry has priority over another, the priority field is added to each entry. Searching an entry in the table is carried out sequentially in the order of priority value. Thus, if the targeted entry fortunately exists in a highly prioritized entry, the lookup process can be completed at an early stage. However, the lookup performance of the wildcard-aware table tends to slow down in accordance with table size.

B. Maintenance of Flow Table

In conventional flow switches, flow tables require manual or semi-automatic update to reflect operators' control policy. Although there are signaling protocols [9][10][11] which can automatically specify end-to-end routes for flows in combination with underlying routing mechanisms (e.g., source routing [12] and other explicit routing mechanisms), they are unlikely to be used for carrier services since they affect the development of applications to implement such the signaling protocols. Therefore, reducing the number of the flow entries to be maintained is important from an operator viewpoint. While some flow switches have both the exact match table and wildcard-aware table, solutions for coordinating them to automatically update the exact match table have not been provided to date. Therefore, the huge entries in exact match table still require manual update.

III. PROPOSED ARCHITECTURE AND MECHANISM OF LIGHTFLOW

A. Basic Idea of LightFlow

There are two requirements for the functionality of the flow table:

- From the viewpoint of operation, aggregation of flow information with wildcard is required to avoid configuring flow table entries for all flows manually.
- From the viewpoint of packet forwarding processing, high-speed matching is required to maintain performance.

For fast table lookup, hash or other deterministic algorithms are suitable, however, such algorithms have difficulty handling wildcard, longest prefix matching, and range matching. On the other hand, once you have obtained the exact header information of the flow, it is better to use deterministic algorithms. Considering that wildcard-based matching is required only for the first packet of the flow, the rest of the packets do not require such a linear search, due to the fact that the full set of flow matching information has already been obtained while the first packet was being processed.

The basic idea of LightFlow is as follows:

- To avoid manual configuration for all the exact entries, automatic update of the hash-based exact match table is introduced. The addition of entries is based on the

matching result of the wildcard-aware table, and the deletion of entries is based on periodic monitoring.

- To accelerate lookup on the wildcard-aware flow table, a GPU-accelerated two-dimensional lookup mechanism for linear search is introduced.

Although utilizing both the tables has been considered to date, the effect of coordinating the two tables has not been investigated. LightFlow tries to relieve operators of the burden of managing flow entry directly for each exact flow by coordinating the two tables.

B. Functional Architecture

Fig. 2 shows the LightFlow architecture. The node has both a wildcard-aware flow table and a hash-based exact match table. The wildcard-aware flow table manager and the hash-based exact match table manager update entries on each corresponding table. The wildcard-aware flow table manager updates the entries of the table according to requests from the operator (including manual operation, network management systems, routing daemons). The hash-based exact match table manager monitors and deletes unused entries from the table. The forwarding controller drives the table lookup process for both tables in the appropriate sequence. The forwarding engine performs packet reception, forwarding, and provides packet information to the forwarding controller.

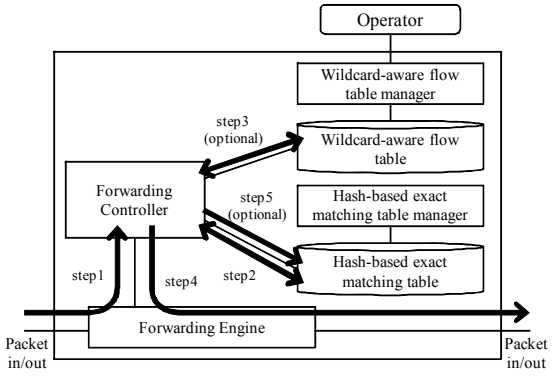

Figure 2. The LightFlow architecture.

The lookup of the destination of a packet occurs in five steps. In step 1, the forwarding engine passes the information of the received packet (e.g., incoming physical interface, MAC addresses, VLAN ID, IP addresses, port numbers) on to the forwarding controller. In step 2, the forwarding controller searches the hash-based exact match table to check whether a matched entry exists or not. If a matched entry is not found, then the forwarding controller searches the wildcard-aware flow table to find the entry that has the highest priority among all the entries that match the packet in step 3. This step is skipped in the case where matched entry is found in step 2. In step 4, the forwarding controller requests the forwarding engine to forward the packet. In the case where the destination is resolved in step 3, the forwarding controller adds the new entry for the flow to the hash-based exact match table in step 5.

To update the control policy for flows, the operator requests the wildcard-aware flow table manager to update the flow information. Then the manager updates the related flow information on the table.

C. Two-Dimensional Parallelization Method for Acceleration of Wildcard-Aware Flow Table Lookup

Fig. 3 shows the proposed lookup mechanism for the wildcard-aware flow table. The entry of the wildcard-aware flow table is composed of columns for the priority field, flow identifying fields, wildcard flags, destination, and option. The priority field value is unique for determining which entry should be matched. The flow identifying fields include multiple columns which are for layer 1 to layer 4 information, and wildcard can be used for the arbitral column. Each of the columns has its matching criteria such as exact matching, longest prefix matching and range matching. The wildcard flags field is for fast recognition of columns with wildcard. The destination field is the forwarding correspondent and is the same as that in conventional IP routing. The option field may be included to provide the additional control information (e.g., QoS control, packet rewrite).

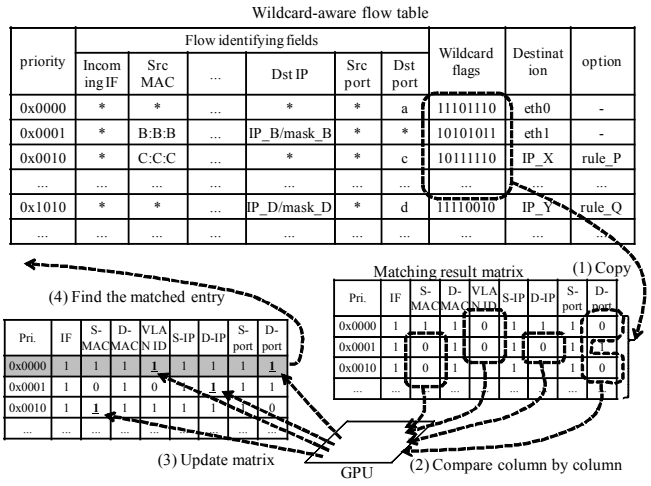

Figure 3. Table lookup on the wildcard-aware flow table.

The difference in the matching criteria for each column made it difficult to carry out parallelization of matching for multiple columns by GPU using conventional mechanisms. The proposed mechanism applies parallelization to the same column for multiple entries which use the same matching criteria. This "two-dimensional" parallelization enables a higher degree of parallelism on GPU processing to be attained.

When the forwarding controller recognizes that the incoming packet is not matched to any of the hash-based exact matching table entries, table lookup on the wildcard-aware flow table is invoked. Lookup on the table is carried out from the entries with the highest priority and multiple entries are checked simultaneously. The lookup procedure consists of the following three steps. First, wildcard flags for the selected entries are copied to a matching result matrix. Each of the elements of the matching result matrix indicates whether the corresponding column matches the packet information or not. The element with "1" indicates that it matches the packet information while the element with "0" indicates otherwise. Second, fields whose corresponding elements are "0" are

grouped by the same column and checked in parallel by the GPU since the check for the same column uses the same instruction. And the elements for matched fields are marked as "1" in the matching result matrix. Third, the match of each entry is judged by calculating the conjunction of all the elements in the entry. This process is also accelerated by the GPU. If there are entries whose result is "1", the entry with the highest priority is the matched entry. If there are no entries that match, the same operation will be carried out for the next selected group of entries.

D. Automatic Update of Hash-based Exact Match Table

Fig. 4 shows the management process of the hash-based exact match table. The entry of the table is composed of columns for the hash value, flow identifying fields, destination, discard flag and option. The hash value is calculated using all the values in the flow identifying fields. The flow identifying fields are used to check whether the entry corresponds correctly to the searched packet, since conflicting hash values can occasionally occur. The discard flag is used to recognize whether the entry is outdated and needs to be deleted or not. Destination and option are exactly the same as those in the wildcard-aware flow table. All the entries are searched using hash in this table and the hash value calculation can be accelerated by the GPU.

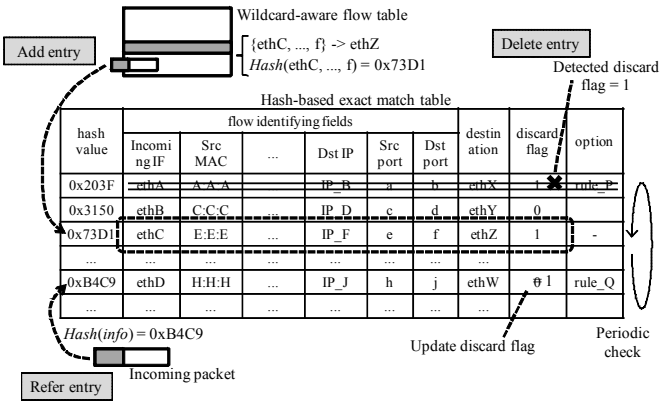

Figure 4. Table lookup and update of the hash-based exact match table.

Addition of entries to the hash-based exact match table is triggered by the lookup of the wildcard-aware flow table. When a packet matches an entry in the wildcard-aware flow table, the hash value for the packet is calculated from the header information, and the entry for the packet is newly added to the hash-based exact match table.

Since the addition of entries occurs for every new flow, it is necessary to have a deletion mechanism for outdated-entries to avoid the situation where the number of entries reaches the upper limit of the table size and any new addition of entries is rejected. Deletion of entries is carried out independently to table lookup for the wildcard-aware flow table and the hash-based exact match table. The entries are checked periodically, and the entries whose discard flag is set to "1" are deleted from the hash-based exact match table. The discard flag is set to "1" after the entry is checked, while it is set to "0" when the entry is newly added or the entry is matched for a forwarded packet.

IV. PROOF OF CONCEPT DEMONSTRATION

To prove the feasibility of LightFlow, a prototype node was implemented. With the manual flow entry configuration, the basic node operation was demonstrated. The table lookup performance was measured and compared to that of the conventional wildcard-based table lookup.

A. Environmental Setup

Fig. 5 shows the configuration of the demonstration. The LightFlow prototype is implemented on Linux PC. The specifications of the hardware consisting of commodity products are shown in TABLE I. The flow switch software is implemented on CentOS 6.2 (Linux kernel 2.6.32) and the CUDA driver version used was 270.41.19. In this prototype implementation, the forwarding controller, wildcard-aware flow table manager and hash-based exact matching table manager were implemented on the user space of the OS, while the wildcard-aware flow table and hash-based exact match table were implemented on the GPU global memory. The forwarding controller dispatches the lookup process and the other tasks such as hash calculation to the GPU through the CUDA driver. The forwarding controller also contacts the kernel TCP/IP stack to utilize some basic networking tasks (e.g. ARP resolution). The forwarding engine is implemented within the kernel and it forwards received packets to the forwarding controller bypassing the kernel. For the management of wildcard-aware flow table entries, a simple command line interface (CLI) was implemented. The traffic sender and receiver were connected to the node via a 10 gigabit Ethernet link.

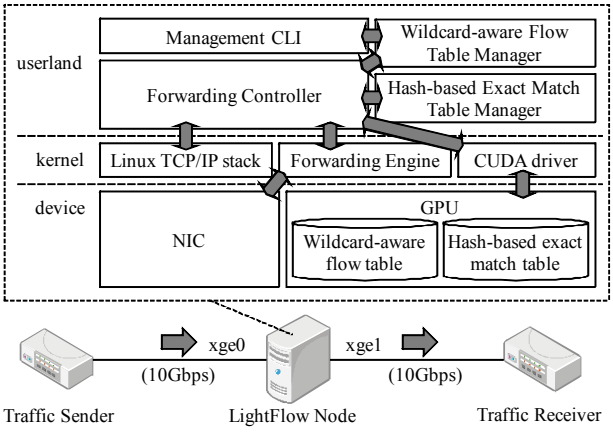

Figure 5. Setup for the demonstration.

TABLE I. SPECIFICATIONS OF THE LIGHTFLOW PROTOTYPE

Device	Hardware	Quantity
Processor	Intel Xeon 5650 2.16GHz	2
Memory	DDR3 8GB ECC-Registered	6
GPU	GeForce GTX580 (3GB VRAM)	2
Network	Intel X520-DA2 Dual port 10GbE	4
Motherboard	SuperMicro X8DAH+F	1

B. Table Lookup Performance

Using the configuration above, the lookup time was measured for both the tables by changing the number of entries.

For the hash-based exact match table, the elapsed time required for completing hash calculation and table lookup was measured. The average time ranges from 51 microseconds to 71 microseconds, independent of both the number of the entries and the values of the flow identifying fields.

For the wildcard-based flow table, the lookup time of the entry which has the lowest priority was measured for three fields with different criteria: exact match field (source MAC address), prefix match field (destination IP address), and range match field (destination port number). Fig. 6 shows the factor of speedup compared to the simulated value of conventional linear search-based mechanism [7][13]. LightFlow completed wildcard-aware flow table lookup from three- to six-fold compared to the conventional mechanism.

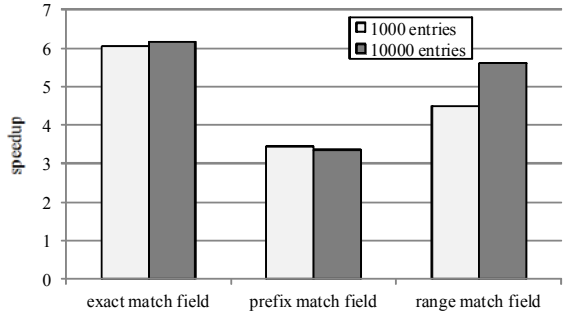

Figure 6. Speedup of wildcard-aware flow table.

Regarding the speedup effect obtained through the coordination of the wildcard-aware flow table and hash-based exact match table, the relationship between average lookup time and the number of packets per flow is shown in Fig. 7. The number of the entries is 10,000 for both tables. For each matching criteria, a highly significant speedup is observed for the proposed mechanisms especially when the number of packets per flow becomes large, since the proportion of packets which require lookup of the wildcard-aware table is decreased. The reduction in the time for looking up on the wildcard-aware flow table is most effective for prefix matching, since it requires more steps to check matching than for the other field in lookup of the wildcard-aware flow table.

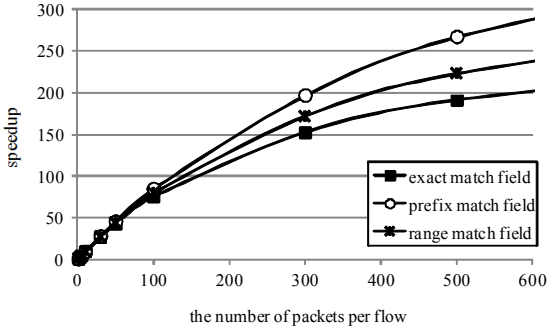

Figure 7. Speedup effect of wildcard-aware flow table coordinating with hash-based exact match table.

V. Discussion

Since the proposed mechanism uses two tables (wildcard-aware table and hash-based table), it consumes a lot of memory compared to a single wildcard-based table. Although there is a trade-off between performance and cost, the feasibility of memory usage should be considered. In the case of our prototype implementation, 128 bytes and 136 bytes are required for a wildcard-aware flow table entry and a hash-based exact matching table entry, respectively. Fig. 8 shows the relationship between the number of entries and memory usage. Both tables can hold about ten million entries within 1.4 gigabytes of memory. Considering that the current number of IPv4 routes is about 400,000 [14] and the number of flows supported by a commercial flow router is 3,000,000 [15], the memory usage of the proposed flow table is feasible. Since recent GPUs have a global memory of more than 1.5 gigabytes, it is possible for both tables to be extracted using GPU memory to hasten memory access.

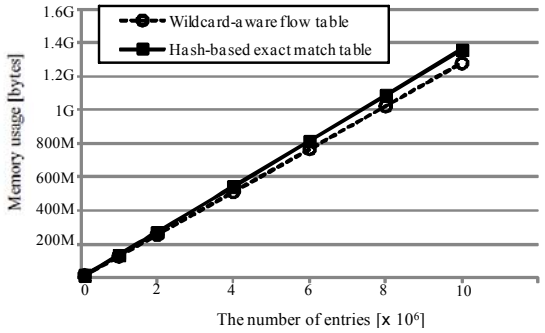

Figure 8. Relationship between the number of entries and memory usage.

There is another potential issue regarding processing jitter. For parallel processing on GPU, some packets may be subject to a waiting period until synchronization with other packets occurs. In the prototype implementation, the maximum variance for the processing time was about 600 microseconds in the case of 10,000 flow entries. Note that this value is the worst case (i.e., all packets are matched at the lowest priority in the wildcard-aware table). Compared to the telecommunication carriers' service level agreement for enterprise networks, whose typical value for jitter ranges from sub-milliseconds to tens of milliseconds, the measured jitter of the flow table lookup seems to be feasible. However, we need to conduct further investigations since other causes of jitter (e.g., communication between GPU and motherboard) should also be taken into account. In addition, there is a possibility of great throughput instability when commodity servers are used due to interruption processes and so on, compared to the router and switch appliances. Therefore, careful tuning of the OS-level configuration is also desired for practical use.

VI. Related Work

Owing to progress in software router technologies, the flexibility of packet processing and its performance have been improved. Click modular router [2] is a software router with high packet processing flexibility. Due to the enhanced performance of PCs, 20 Gbps throughput (per two ports) using

10 Gbps network interface cards has already been achieved using a commercial software router, Vyatta [3][16]. RouteBricks [5] takes a cluster-based approach to achieve 35 Gbps throughput (per four ports) and its capacity can linearly scales to the number of servers. Some recent studies have attempted to accelerate packet processing by utilizing a GPU. Since a GPU has an advantage regarding parallel processing for simple arithmetical instructions, it is suitable for application to packet pattern matching and cryptographic processing [17][18]. GPU-acceleration is also considered to be promising for IP table lookup. [19][20][21] proposed GPU acceleration for IP routing table lookup algorithms such as radix-tree [22], DIR-24-8-BASIC [23], trie on hash tables [24]. PacketShader [7] also uses a GPU for accelerating packet processing. PacketShader supports IPv4/v6 routing and flow switching based on OpenFlow reference implementation [13]. The flow table consists of hash-based exact match entries and wildcard-aware linear search entries. In PacketShader, hash value calculation and wildcard matching are offloaded to the GPU. The method proposed in [25] is another approach to accelerate flow table lookup using GPU. In the method, hash-based lookup tables are generated for each combination of wildcard fields, and lookup operation is carried out for all the tables in parallel. The method assumes that each field can be matched exactly unless the value is not wildcard. Hence, the method does not support longest prefix matching and range matching.

VII. CONCLUSIONS

This paper proposed LightFlow, a mechanism to speed up software flow switching processing and relieve the burden of maintaining the flow table. As the key technologies of LightFlow, a two-dimensional parallelization of the linear search on the wildcard-aware table and an automatic updating mechanism for the hash-based exact match table were presented. The demonstration showed that the two-dimensional parallelization increased the speed of the lookup of the wildcard-aware flow table about three- to six-fold compared to the conventional mechanism. The demonstration also showed that fast lookup of the hash-based exact match table can be utilized while operators only have to manage the wildcard-aware flow table. For future work, the detailed implementation of GPU resource allocation and utilization for each computation task should be investigated for improving the performance of this mechanism.

ACKNOWLEDGMENT

The authors wish to thank Dr. Yasuyuki Nakajima and Dr. Itsuro Morita for their continuous support of this study.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp.69-74, April 2008.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," ACM Transactions on Computer Systems vol. 18, no. 3, pp. 263-297, August 2000.

[3] Vyatta official website, http://www.vyatta.com/

[4] R. Bolla, and R. Bruschi, "The IP Lookup Mechanism in a Linux Software Router: Performance Evaluation and Optimizations," in *Proc. of HPSR 2007*, May 2007.

[5] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism To Scale Software Routers," in *Proc. of ACM SOSP 2009*, pp. 15-28, October 2009.

[6] N. Varris, and J. Manner, "Performance of a Software Switch," in *Proc. of HPSR 2011*, July 2011.

[7] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated Software Router," in *Proc. of ACM SIGCOMM 2010*, September 2010.

[8] Open Networking Foundation, "OpenFlow Switch Specification Version 1.2," December 2011.

[9] R. Braden, et al., "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification," IETF RFC 2205, September 1997.

[10] ITU-T, "Signalling protocols and procedures relating to Flow State Aware QoS control in a bounded sub-network of a NGN," Recommendation ITU-T Q.3313, October 2011.

[11] TIA, "QoS Signaling for IP QoS Support," TIA-1039, July 2005.

[12] J. Postel, "Internet Protocol," IETF RFC 791, September 1981.

[13] OpenFlow reference implementation, available at http://www.openflowswitch.org/wp/downloads/

[14] BGP Routing Table Analysis Report, http://bgp.potaroo.net/

[15] Anagran FR-1000 datasheet, http://anagran.com/assets/docs/FR-1000Datasheet031511.pdf

[16] Intel solution brief, "Integrating Services at the Edge," http://edc.intel.com/Download.aspx?id=2977

[17] G. Vasiliasdis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *Proc. of RAID 2008*, pp. 116-134, September 2008.

[18] R. Simth, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan, "Evaluating GPUs for Network Packet Signature Matching," in *Proc. of ISPASS 2009*, pp. 175-184, April 2009.

[19] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. Deng, and S. Zhang, " IP Routing Processing with Graphic Processors," In *Proc. of IEEE DATE 2010,* pp. 93-98, March 2010.

[20] Y. Lee, M. Jeong, S. Lee, and E. Im, "Fast Forwarding Table Lookup Exploiting GPU Memory Architecture," in *Proc. of IEEE ICTC 2010*, pp. 341-345, November 2010.

[21] J. Zhao, X. Zhang, X. Wang, Y. Deng, and X. Fu, "Exploiting Graphics Processors for High-performance IP Lookup in Software Routers," in *Proc. of IEEE INFOCOM 2011*, pp. 301-305, April 2011.

[22] K. Sklower, "Tree-Based Packet Routing Table for Berkeley Unix," in *Proc. of USENIX Winter Conference '91*, pp. 93-99, January 1991.

[23] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," In *Proc. of IEEE INFOCOM '98*, pp. 1240-1247, April 1998.

[24] M. Waldvogel, G. Varghese, j. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookups," In *Proc. of ACCM SIGCOMM '97*, September 1997.

[25] R. Yanggratoke, and H. Puthalath, "Method for Enhancing Table Lookups with Exact and Wildcards Matching for Parallel Environments," US Patent Application No. 2011/0292830, Decemver 2011.