Data Structure Enabling Retrieval of Time Series of Traffic with the Requested Granularity

Yoshihiro Tsuji, Yuichi Ohsita, and Masayuki Muarata
Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka, 565–0871, Japan
Email: {y-tsuji, y-ohsita, murata}@ist.osaka-u.ac.jp

***Abstract*—Some network management tasks require the fine-grained information of time series of traffic. However, it takes a large overhead to obtain the fine-grained information. One approach is to obtain the traffic information only with the required granularities because the traffic information with the fine granularity are not always necessary. In this approach, the fine-grained traffic information is stored at each monitoring point, and the network manager receive only the traffic information with the required granularity. The required granularity of information depends on time, but none of existing data structure enables immediate retrieval of time series of traffic with the required granularity that changes in time. Therefore, in this paper, we propose a data structure enabling it. Through the numerical evaluation, we demonstrate that our data structure enables immediate retrieval of time series of traffic without a large calculation time to store the traffic data.**

***Keywords*—Traffic Measurement, Flow Monitoring, Granularity, Patricia Tree, Multi-Stage Tree, Segment Tree**

I. Introduction

The information on the time series of traffic is an important information in the network management such as the traffic engineering [1]–[3], and anomaly detection [4], [5].In recent years, the demand for the fine-grained information of the time series of traffic has increased, as the technology enabling the configuration of the routes for each flow such as the OpenFlow technology [6] becomes popular, because the per-flow route configuration requires the per-flow traffic information. The traffic information with the fine time granularity has also become necessary. For example, Benson et al. demonstrated that the dynamic routing reconfiguration with seconds interval significantly improves the performance of the data center network [1]. To perform the dynamic routing reconfiguration, the traffic information with the smaller time granularity than the control interval is required.

Typically, a central server collect traffic information periodically from the routers by using sFlow [7] and NetFlow [8] to analyze the time series of traffic for the network management tasks. We call the server *manager*. However, obtaining the traffic information with the fine granularity by the NetFlow or sFlow takes a large overhead, because the manager has to collect a large amount of traffic data with a short interval. One approach to reducing the overhead is to collect only the required traffic information with the required granularity, because the traffic information with the fine granularity are not always necessary; the dynamic routing control requires only the traffic information of the aggregated flows, which passes the same routes, instead of the traffic information of each flow, though the aggregated flows change in time.

Yuan et al. [9] proposed an architecture that monitors the traffic with the required granularity. In this method, the counters are configured so as to monitor the traffic with the required granularities. However, in this method, the granularity is required to be predefined. When the traffic information of a certain IP address prefix becomes required, the past traffic information of the prefix cannot be obtained by this method even if the granularity is redefined.

Therefore, in this paper, we discuss an approach that enables retrieval of the information of the time series of traffic with the required granularity that changes in time. In this approach, computers called *traffic observer*s are deployed at each monitoring point. The traffic observer stores all traffic data from the connected or nearby routers. If traffic information is required, the manager sends a request including the required granularity to the traffic observers. Then, each traffic observer analyzes the stored data to construct the traffic information with the requested granularity, and replies the constructed information. By this approach, the overhead to collect the traffic information can be saved by collecting only the required traffic information with the required granularity.

In this approach, how to store the traffic data has a large impact on the time required to retrieve the required traffic data. If we simply store each fine-grained traffic data, it takes a long time to retrieve the coarse grained data; to obtain the traffic amount of an IP address prefix, the traffic observer is required to search all traffic data matching the IP address prefix.

Therefore, we propose a data structure to store the time series of traffic so that the traffic information with the required granularity can be retrieved immediately. Our data structure is constructed of multiple trees. Each tree indicates one field of the traffic data (e.g., time, source IP address, or destination IP address). In each tree, the leaf nodes correspond to the traffic data of the finest-grained granularity, and a parent node corresponds to the coarser granularity that includes all of its children. Each node has the pointer to the root of the tree corresponding to the next field. Therefore, the traffic data with the required granularity can be obtained by continuing to search the nodes corresponding to the required granularity of the field. Moreover, we improve the proposed tree-based data structure to reduce the data size and the calculation time to update the traffic information by using Patricia Tree and eliminating sibling nodes.

The rest of this paper is organized as follows. Section II explains the architecture of the traffic observer, and proposes the data structure for the time series of traffic. Section III evaluates the proposed data structure. Finally, the conclusion and future work are mentioned in Section IV.

II. Design of Traffic Observer

A. Overview

In this paper, we deploy computers called *traffic observers*. Each traffic observer collects fine-grained traffic information periodically from directly connected routers or nearby routers. The collection of the traffic information by the traffic observer does not takes a large overhead, because each traffic observer collects traffic information of a small number of routers. Moreover, the number of hops passed by the traffic information is small because each traffic observer collects the traffic information only from nearby routers. Any method to collect the traffic information can be applied; traffic observer can retrieve the counter values of the routers by using the NetFlow, or can count the packet through the mirrored ports.

The traffic observer stores the collected traffic information in the data structure where the information on the time series of traffic with the required granularity can be immediately retrieved. When the network manager requires the traffic information, the network manager sends a query, including the range of the required traffic information and the required granularity, to the traffic observer. Then, the traffic observers retrieve and reply the information requested by the query. In this paper, the granularities are defined by the three fields, (i.e., time, source IP address, and destination IP address). The granularity of the time is set by the length of the time slot. On the other hand, the granularities of the source and destination IP addresses are set by the length of the IP address prefix. By setting these granularities, the traffic information is aggregated into the traffic information per the defined time slot and the defined IP address prefixes.

The rest of this section explains the data structures used to store the traffic information in the traffic observers.

B. Data Structure in Traffic Observer

In this paper, we propose the data structure shown in Figure 1. As shown in Figure 1, our data structure is constructed of three kinds of trees. The first tree manages the time granularity. In this paper, we call this tree *time tree*. Our data structure includes one time tree. In the time tree, the leaf nodes corresponds to the minimum time slot. A parent node corresponds to the larger time slot, which includes all time slots of its children.

Each node in the time tree has the pointer to the corresponding tree managing the granularity of the source IP address prefix. We call this tree *source IP tree*. Our data structure includes the same number of the source IP trees as the number of nodes in the time tree, and each source IP tree corresponds to the time slot of the corresponding node in the time tree. In each source IP tree, the leaf nodes correspond to the traffic data monitored per source IP address. The sibling nodes in the source IP tree correspond to the IP addresses with the same prefix, and their parent node corresponds to the traffic data aggregated by the prefix.

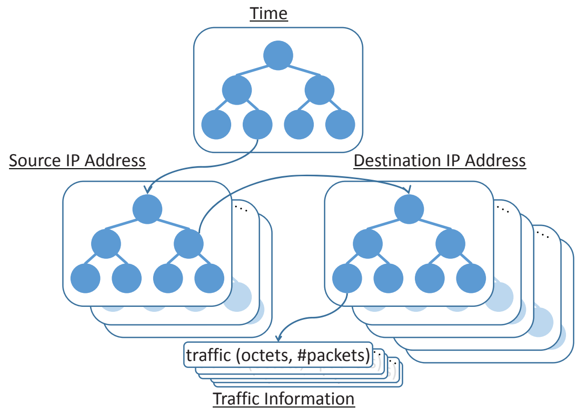


Fig. 1. Proposed data structure to store the traffic data

Similar to the time tree, each nodes in the source IP tree has the pointer to the tree managing the destination IP addresses. We call this tree *destination IP tree*. Our data structure includes the same number of destination IP trees as the total number of nodes in all source IP trees. Each destination IP tree corresponds to the time slot and the source IP address prefix corresponding to the node in the source IP tree. The structure of the destination IP tree is similar to the source IP tree. Each node in the destination IP tree has the pointer to the counter.

By searching these trees, traffic information with any granularity can be retrieved. First, the nodes corresponding to the requested time slots are found by searching the time tree, and the pointer to the corresponding source IP tree is obtained. Then, the nodes corresponding to the requested source IP prefixes are found by searching the source IP tree, and the pointer to the destination IP tree is obtained. Finally, the counter values are retrieved by searching the destination IP tree.

The rest of this subsection explains the detail of the structures of the time tree, source IP tree and destination IP tree.

1) Time tree: The time tree is constructed so as to include the nodes corresponding to the time slots that are possible to be requested. For example, the traffic data with 1-minute interval or 1-hour interval may be often requested while the traffic data with 12-minute interval is seldom requested. Thus, we predefine the possible time intervals.

Based on the defined possible time intervals, the structure of the time tree is determined. For example, we consider the case that the traffic information for one day is stored, and the possible time intervals are set to 1 day, 12 hours, 1 hour, 10 minutes, 1 minute, and 10 seconds. In this case, the root node corresponds to the traffic for one day, and has two children. The node corresponding to the traffic for 12 hours, 1 hour, 10 minutes, and 1 minutes have 12, 6, 10, and 6 children, respectively.

2) Source IP tree and destination IP tree : The source IP tree and the destination IP tree are constructed based on the binary tree; each parent node corresponds to the IP address prefix whose length is one bit shorter than its children. However, the binary tree requires a large amount of nodes in the trees.

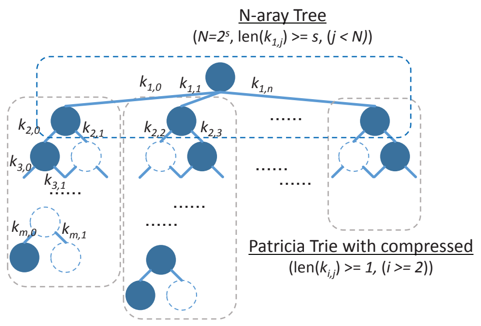

N-aray Tree
(N=2^s, len(k_{1,j}) >= s, (j < N))
k_{1,0}
k_{1,1}
k_{1,n}
k_{2,0}
k_{2,1}
k_{2,2}
k_{2,3}
k_{3,0}
k_{3,1}
k_{m,0}
k_{m,1}
Patricia Trie with compressed
(len(k_{i,j}) >= 1, (i >= 2))

Fig. 2. Overview of the source IP tree and the destination IP tree

To solve this problem, we first introduce the threshold s indicating the length of the prefix whose traffic information is not stored; the traffic information does not aggregated into the IP address prefix whose length is less than s. The root node has the same number of children as the number of s-bit IP address prefix included in the monitored traffic. This is motivated by the fact that too coarse grained information is seldom used; the information on the traffic passing the same route is useful for many applications such as network management, while the information on the aggregated traffic which includes the traffic passing the different routes is not required.

Moreover, we reduce the nodes in the source IP tree and the destination IP tree by using the sparsity of the IP addresses monitored at each traffic observer; some IP address prefixes are frequently monitored by a traffic observer, while the other IP address prefix has never monitored by it. In this case, the tree does not require the nodes corresponding to the IP addresses that are not monitored. To utilize the sparsity, we use the Patricia tree to store the traffic information. Patricia tree [10], [11] is a tree-based data structure where nodes with only one child are eliminated from the tree.

We also eliminate the information of one of the sibling nodes, because the information of the node can be recovered from the information (i.e. the pointer to the next tree) of the parent and the other sibling node; the information is recovered by calculating the difference of the parent node and the other sibling node. Even if the parent node also does not have the information, we can recover the information by continuing to recover the information of the parent.

C. Operations on Data Structure

1) Update: The traffic observer collects the traffic information periodically. When the new traffic information is received, the traffic observer updates its data structure to store it.

a) Update of the time tree: To update the traffic information, we first search the time tree to find the nodes corresponding to the current time slot. If no nodes including the current time slot are found at each layer, we add the new node to the time tree, and create the new source IP tree corresponding to the current time slot.

b) Update of the source IP trees: Then, we update all source IP trees that have pointers from the nodes of the current time slot in the time tree. We search each of the source IP trees for the nodes with the source IP address prefix corresponding to the source IP addresses included in the newly received traffic information.

If the source IP address in the newly received traffic information has not been included in the source IP tree, we update the source IP tree by adding a node corresponding to the source IP address. To add the node corresponding to the source IP address A^{new}, we first find the leaf node corresponding to the source IP address A^{leaf} where $P(A^{\mathrm{new}},A^{\mathrm{leaf}})$ is the largest, denoting $P(A,B)$ as the number of bits of the common prefix of A and B. The found leaf node is denoted by N^{leaf}. Then, we add the node corresponding to A^{new}, which is denoted as N^{new}, and the parent node of N^{leaf} and N^{new}, which is denoted as N^{parent}. The pointer whose value was the address of N^{leaf} is changed into the address of N^{parent}, and the pointers of the children of N^{parent} are set to the addresses of N^{leaf} and N^{new}.

After adding new node, we create the destination IP trees corresponding to the newly added nodes. In our data structure, only one of the sibling nodes has to have the corresponding destination IP tree. The process to create the destination IP trees depends on whether N^{leaf} has the corresponding destination IP tree. If N^{leaf} has the corresponding destination IP tree, N^{new} does not require the corresponding destination IP tree. Instead, N^{parent} requires the corresponding destination IP tree, because the node that becomes the sibling node of N^{parent} after the addition does not has the corresponding destination IP tree. The destination IP tree of N^{parent} is created by copying the destination IP tree of the N^{leaf}. On the other hand, if N^{leaf} does not have the corresponding destination IP tree, N^{parent} does not require the corresponding destination IP tree. Instead, either N^{leaf} or N^{new} requires the corresponding destination IP tree. In this case, we create the blank tree, that includes only the root node, for the N^{new}, because creating the blank tree takes little calculation time.

c) Update of the destination IP trees: Finally we update the destination IP trees. The destination IP trees required to be updated are found by searching the source IP trees for the nodes with the source IP address prefix corresponding to the source address of the flows included in the newly collected traffic information.

If a destination IP tree is found by searching the source IP tree for the node corresponding to the set of flows F, the destination IP tree is updated by the following steps. For each flow f in F, we search the destination IP tree for the nodes with the destination IP address prefix corresponding to the destination IP address of f. Then, we obtain the pointers to the corresponding counter, and update the values of the counters. If the destination IP address of f has not been included in the destination IP tree, we add the nodes by the similar way to the source IP tree. Then, the value of the corresponding counter is updated.

2) Retrieval of the required information: When the network manager requires the traffic information, it sends the request of the traffic information to the traffic observers. The request includes the range and the granularity of the required traffic information. The range and the granularity are defined by the three fields, the time, the source IP address, and the destination IP address. That is, the request includes the following information, the start time T^{start}, the ending time T^{end}, the length of time slots of the required traffic information T^{size}, the range of the source IP address (S^{start}, S^{end}), the number of bits of the prefix of the source IP address of the required traffic information S^{size}, the range of the destination IP address (D^{start}, D^{end}) and the number of bits of the prefix of the destination IP address of the required traffic information D^{size}. When receiving the request, the traffic observer retrieves and replies the requested traffic information. The requested traffic information is retrieved by the following steps.

a) Find the nodes corresponding to the requested information : We construct the subtree that has all of the required information from each tree as follows. We first find the node that includes both of the traffic information at T^{start} and T^{end}. The found node becomes the root of the subtree. Then, we eliminate the nodes whose parents have the sufficiently small time slots whose lengths are shorter than T^{size}. Then, the subtrees of the source IP trees corresponding to the nodes in the subtree of the time tree is constructed by the similar way. Finally, the subtrees of the destination IP trees are also constructed. After the construction of the subtrees, the leaf nodes correspond to the requested traffic information.

b) Restoration: In our data structure, only one of the sibling nodes has the pointer to the tree of the next field. Thus, we require to restore the information to retrieve the requested traffic information. The information of the node N which is denoted as I_N is restored by

$$I_N = I_{N^{\text{parent}}} - I_{N^{\text{sibling}}}.$$

where N^{parent} is the parent of N, N^{sibling} is the sibling node, and the operator $-$ indicates the difference of the first term and the second term. Because the information corresponding to the node in the time tree or the source IP tree is the source IP tree or the destination IP tree, we define the operator $-$ for the trees in the rest of this section.

To calculate the tree $B - A$ for trees A and B, all nodes in B are first copied to $B - A$, and then the information corresponding to the nodes in $B - A$ is calculated. The information of the node n in tree $B - A$ is calculated by

$$I_n = \begin{cases} I_{n_B} & n_A^{\text{found}} \text{ is not found} \\ I_{n_B} - I_{n_A^{\text{found}}} & \text{otherwise} \end{cases},$$

where n_B is the nodes with the same key as n in B, and n_A^{found} is the node in A whose key length is the longest among the nodes whose keys include the key of n as the prefix.

III. Evaluation

A. Traffic Data

In this paper, we use the traffic data captured from P.M. 0:00 to P.M. 0:30 on 7th Feb. 2014 at the port of the gateway of Osaka University. The maximum rate of the port is 1 Gbps. In this data, the IP addresses are anonymized by applying the hash function to the lowest 16-bit of the IP addresses. This anonymization does not have any impact on our evaluation, because (1) the anonymization process does not change the upper 16-bit of the IP address, and (2) the flows belonging to the same flow have the same IP addresses even after the anonymization process.

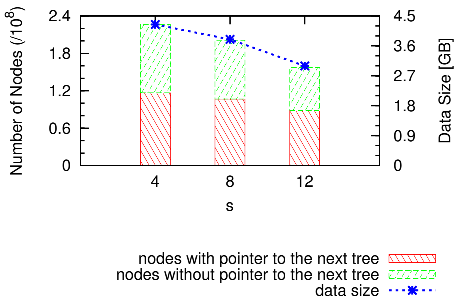


Fig. 3. Total number of nodes in our data structure when 30-minute traffic information is stored

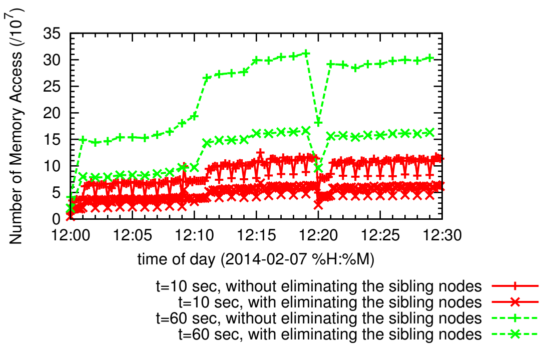


Fig. 4. The number of memory access required to update the traffic information at each time slot

B. Data Size of Data Structure

We first investigate the data size of our structure required to store the 30-minute data. We set the possible time interval to 30 minutes, 10 minutes, 1 minute, and 10 seconds. We set the minimum time interval to 10 seconds. We set s to 4, 8, and 12.

Figure 3 shows the number of nodes in trees. The data size is calculated by setting the data size of each node to 24 bytes and 16 bytes for the node with pointer to the next tree and the node without the pointer to the next tree respectively. Figure 3 indicates that setting s to a large value decreases the number of nodes in trees and the data size. Even if we set s to 4, the data size does not become large and is less than 4.5 GB.

C. Updating Complexity

We also investigated the calculation time to update the traffic information. In this investigation, we count the number of memory access required to search trees, copy trees and update counters. Figure 4 shows the number of memory access required to update the traffic information at each time slot. In this investigation, we set the same parameter of our data structure as Section III.B. We compare two cases of the time interval to collect the traffic information t. Moreover we compare our data structure with the data structure that does not eliminate the sibling nodes. This figure indicates that eliminating the sibling nodes decreases the time required to update the traffic information by half. This is because eliminating the sibling nodes reduces the number of nodes in trees required to be updated. Supposing that the time required to access the memory is 50 ns based on [12], the time required to update the traffic information is 15 sec or 5 sec in the case of $t = 60$ and $t = 10$ respectively. That is, the time required to update is much smaller than the interval to update the traffic information.

D. Extraction Complexity

We investigate the calculation time to extract the required traffic information based on the number of memory access. In this investigation, we set t to 60, s to 8.

In our investigation, we use the parameters in the request of the traffic information generated by the combination of the following fields. We use three T^{start}, 12:00, 12:10, and 12:20. Then, T^{end} is set to 10 minutes after T^{start}. T^{size} to 1 minute unless otherwise stated. For S^{start}, S^{end}, D^{start}, and D^{end}, we first select ten source and destination IP address pairs randomly from the monitored traffic data. Then, S^{start} is set to the address whose first 8 bits are set to the same as the selected IP address and the other bits are set to 0. Similarly, S^{end} is set to the address whose first 8 bits are set to the same as the selected IP address and the other bits are set to 1. D^{start} and D^{end} are set similarly. We set S^{size} and D^{size} to 12 unless otherwise stated.

We investigate the number of memory access required to retrieve the requested information by changing the granularity defined by T^{size}, S^{size} and D^{size} in Figure 5. This figure indicates the average of the number of memory access. Figure 5 indicates that setting T^{size}, S^{size} or D^{size} to a small value increases the number of memory access, because more traffic information is required to be retrieved. This figure also indicates that the total number of memory access required to obtain the required information is less than 140×10^3, which takes only 7.0 ms by supposing that each memory access takes 50 ns, even when sibling nodes are eliminated, though the restoration is required. That is, eliminating sibling nodes does not cause the large overhead to restore the eliminated information.

IV. Conclusion

In this paper, we proposed a data structure enabling the retrieval of the time series of the traffic with the requested granularity. Through the numerical evaluation, we demonstrated that our data structure enables the immediate retrieval of the time series of the traffic without a large calculation time to store the traffic data.

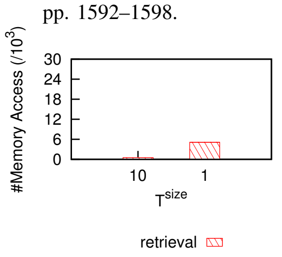
Our future work includes the discussion on more reduction of the nodes in the trees so as to save the memory size and the calculation time more.

References

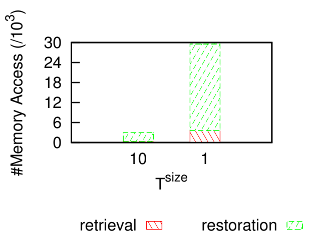
[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proc. ACM CoNEXT*, 2011, pp. 8:1–8:12.

[2] Tatsuya Otoshi, "Prediction-based control theoretic approach for robust traffic engineering," Master's thesis, Graduate School of Information Science and Technology, Osaka University, February 2014.

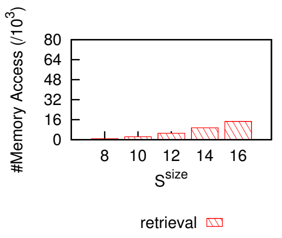
[3] T. Otoshi, Y. Ohsita, M. Murata, Y. Takahashi, K. Ishibashi, and K. Shiomoto, "Traffic Prediction for Dynamic Traffic Engineering Considering Traffic Variation," in *Proc. IEEE Globecom*, Dec. 2013, pp. 1592–1598.



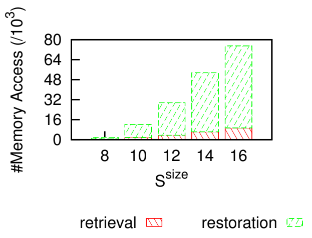
(a) Various T^{size} without eliminating sibling nodes



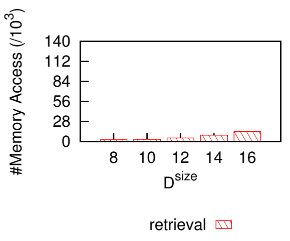
(b) Various T^{size} with eliminating sibling nodes



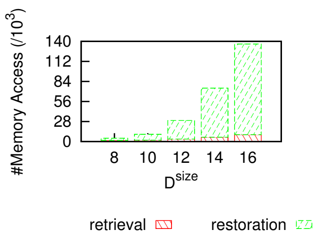
(c) Various S^{size} without eliminating sibling nodes



(d) Various S^{size} with eliminating sibling nodes



(e) Various D^{size} without eliminating sibling nodes



(f) Various D^{size} with eliminating sibling nodes

Fig. 5. The number of memory access to extract the required traffic information

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[5] A. Soule, K. Salamatian, and N. Taft, "Combining Filtering and Statistical Methods for Anomaly Detection," in *Proc. ACM SIGCOMM IMC*, Oct. 2005, pp. 31–31.

[6] 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 CCR.*, vol. 38, no. 2, pp. 69–74, 2008.

[7] P. Phaal, S. Panchen, and N. McKee, "InMon Corporations sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," RFC 3176, 2001.

[8] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Oct. 2004.

[9] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: Towards Programmable Network Measurement," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 115–128, Feb. 2011.

[10] D. E. Knuth, "The Art of Computer Programming. vol. 3: Sorting and Searching," *Atmospheric Chemistry & Physics*, vol. 1, 1973.

[11] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest-Matching Prefixes," *IEEE/ACM Trans. Netw.*, vol. 4, no. 1, pp. 86–97, 1996.

[12] Micron, "Micron Technical Note Using DDR4 in Networking Subsystems," http://www.micron.com/-/media/documents/products/technical%20note/dram/tn_4003_ddr4_network_design_guide.pdf.