Monitoring the Dynamics of Network Traffic by Recursive Multi-dimensional Aggregation

Midori Kato
Keio University
katoon@sfc.wide.ad.jp

Kenjiro Cho
IIJ/Keio University
kjc@iijlab.net

Michio Honda
NEC Europe Ltd.
michio.honda@neclab.eu

Hideyuki Tokuda
Keio University
hxt@sfc.keio.ac.jp

Abstract

A promising way to capture the characteristics of changing traffic is to extract significant flow clusters in traffic. However, clustering flows by 5-tuple requires flow matching in huge flow attribute spaces, and thus, is difficult to perform on the fly. We propose an efficient yet flexible flow aggregation technique for monitoring the dynamics of network traffic. Our scheme employs two-stage flow-aggregation. The primary aggregation stage is for efficiently processing a huge volume of raw traffic records. It first aggregates each attribute of 5-tuple separately, and then, produces multi-dimensional flows by matching each attribute of a flow to the resulted aggregated attributes. The secondary aggregation stage is for providing flexible views to operators. It performs multi-dimensional aggregation with the R-tree algorithm to produce concise summaries for operators. We report our prototype implementation and preliminary results using traffic traces from backbone networks.

1 Introduction

Traffic monitoring is crucial to network operation to detect changes in traffic patterns often indicating signs of flash crowds, mis-configurations, DDoS attacks, scanning or other unusual conditions of a network. It is also important to understand the usage of a network and its long-term trend for capacity planning and other purposes.

Flow-based traffic monitoring has been most widely used for traffic monitoring. A "flow" is defined by a unique 5-tuple (source and destination IP addresses, source and destination ports and protocol), and used to identify a conversation between nodes.

A challenge for flow-based traffic monitoring is how to extract significant flows in constantly changing traffic and produce a concise summary. A summary should be detailed enough to identify possible anomalies in the traffic, but if it is too detailed, an operator would overlook them.

A simple and common way to extract significant flows is by observing traffic volume and/or packet counts, and report the top ranking flows. However, a significant event often consists of multiple flows; for example, scanning can be identified by concentration of flows originating from one node, and flash crowds and DDoS attacks can be identified by concentration of flows destined to one node.

To this end, individual flows with common attributes in 5-tuple can be classified into an aggregated flow. It is often realized using predefined rules similar to packet classifiers for firewalls. However, the limitation is that it can identify only predefined flows and cannot detect unexpected or unknown flows, which considerably weakens its usefulness for traffic monitoring.

To overcome the limitation of predefined rules, automatic flow aggregation has been proposed [5, 7, 14]. The basic idea is to perform flow clustering on the fly in order to adapt aggregated flows to traffic changes. Although the idea is simple and promising, it is not easy in practice to cluster flows efficiently. The space required for bookkeeping flows is combinations of 5-tuple spaces, and searching the best matching aggregated flow in 5-dimensional space is nontrivial. In addition, because traffic information is continuously generated so that the clustering process needs to keep up with incoming traffic information in near real-time.

In this paper, we propose a practical methodology for multi-dimensional flow aggregation. The key insight is that, once aggregated flow records are created, they can be efficiently re-aggregated with coarser temporal and spacial granularity. The advantage is that an operator can monitor only coarse-grained aggregated flows to identify anomalies and, if necessary, he can look into more fine-grained aggregated flows by changing aggregation granularity. Thus, we employ a two-stage flow aggregation; the primary aggregation stage focuses on efficiency

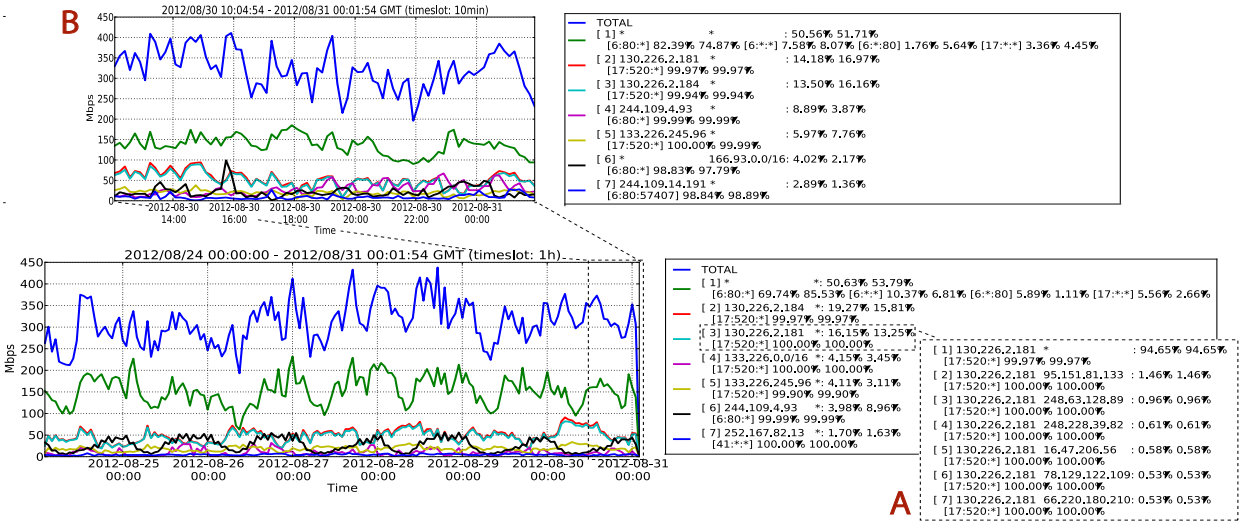

Figure 1: Proposed traffic monitoring system image: an operator should be able to identify suspicious flows at a glance, (A) click and request details, and (B) zoom into part of the view if necessary. IP addresses are anonymized with preserving prefix relationship.

while the secondary aggregation stage focuses on flexibility. The primary aggregation stage reads raw traffic records and produces rudimentary aggregated flow records on the fly. The secondary aggregation stage reads the output of the primary aggregation stage, and produces concise summaries for operators.

Our main contribution is the proposed practical technique for multi-dimensional flow aggregation. Although there exit previous proposals on multi-dimensional flow aggregation techniques, none of them is widely used, probably because the algorithms are too expensive for operational purposes. We believe that our aggregation technique is efficient enough for daily operational use, while providing flexible views with varying spatial and temporal granularities.

2 Motivation

It is important for network operators to be able to capture the network condition at a glance, without taking further actions such as scrolling or clicking. Thus, we would like to have an aggregated-flow monitoring system which provides a compact summary view to draw operators' attention to suspicious flows, with a function to zoom into the suspicious flows by changing the time granularity and/or flow granularity.

The requirements for such monitoring systems are (1) efficient CPU usage to produce traffic summary reports, (2) limited storage usage to store flow data, (3) flexibility to change spatial and temporal granularity in summarizing traffic, and (4) ability to monitor high-speed links.

To illustrate the proposed system, an example summary view of our prototype is shown in Fig. 1. The bottom view shows one-week traffic with 60-minute resolution, and the upper view shows zoomed 15-hour traffic with 10-minute resolution. All the flows are re-aggregated into 7 of the most significant flows in a single view for readability.

The flow-list on the right in a view shows each aggregated flow. The first line of an entry shows the information of the source-destination pair: the rank, source address, destination address, percentage in volume, and percentage in packet counts. The second line shows the protocol information within the source-destination pair: protocol, source port, destination port, percentage in volume, and percentage in packet counts. A wild-card, "*", is used to match any.

In the one-week view, the top ranking flow is a wild card containing flows which do not match the rest of the aggregated flows, and accounts for 50.63% in bytes and 53.79% in packets of the total traffic. The second through fifth flows are UDP source port 520 traffic from a few hosts to a wide range of destinations, which turned out to be a DDoS attack by source-spoofed RIPv1 response flooding. This is a typical anomaly that is hard to identify with simple flow-ranking, but easy to catch with flow-aggregation.

An operator can see the detailed list of a flow by clicking an entry in the flow-list (A in the figure). This functionality informs them of specific destination hosts that are hidden in the overview. Also, they can capture a new summary view with different time resolution by specify-

ing a desired time range (B in the figure). The zooming graph shows short and significant aggregated flows like sixth-flow.

3 Related work

Aguri [5], Autofocus [7] and ProgME [14] aim at traffic monitoring by aggregating flows. Aguri [5] and AutoFocus [7] use a tree structure for each attributes in 5-tuple to hold the accounting information of flows and dynamically create an aggregated attribute by aggregating small tree nodes. Although Aguri supports "recursive" flow aggregation by re-aggregating its own aggregated outputs, it does not support multi-dimensional flow aggregation. Autofocus realizes multi-dimensional flow aggregation with a flow matrix generated by the cross-product of 5-tuple fields, but it does not consider recursive flow aggregation. ProgME [14] uses dynamic packet classifiers where it keeps splitting and merging existing classifier rules to adapt classifiers to changing traffic. However, it is costly to keep updating multi-dimensional classifier rules according to changes in traffic. Our primary aggregation scheme is based on Aguri but it is extended to support multi-dimensional aggregation.

HHH [15] extracts heavy-hitter flows by clustering source and destination addresses. However, HHH is designed to detect security related incidents so that it simply ignores small clusters without incidents, which is not suitable for traffic monitoring. Our scheme aggregates small clusters into larger ones, and thus, does not lose the information of small clusters.

Multi-dimensional HHH [6] proposes heavy hitter splitting and overlapping techniques that aggregate child nodes by propagating them in the multi-dimensional hierarchy to the parent elements. Our algorithm is categorized into their splitting algorithm. Their online algorithm reduces the required number of items in the data structure by dynamically aggregating small items, which is conceptually similar to our primary stage algorithm. The main difference in our algorithm is that our primary aggregation is more efficient by aggregating each attribute separately, and our secondary aggregation is more flexible to be able to produce outputs with varying granularity from the primary aggregation results.

Multi-dimensional aggregation has been considered in the networking research for packet classification [11]. Tuple space [10] and rectangle search [12] are examples to partition classifier rules defined in multiple fields in order to realize faster classifier matching. Their focus is on fast look-up for the best matching filter set for a given packet. We use rectangle trees [3, 8, 9] for the secondary aggregation of our scheme but our focus is to identify best candidates for aggregation by finding the superset and subset relationships of aggregated flows.

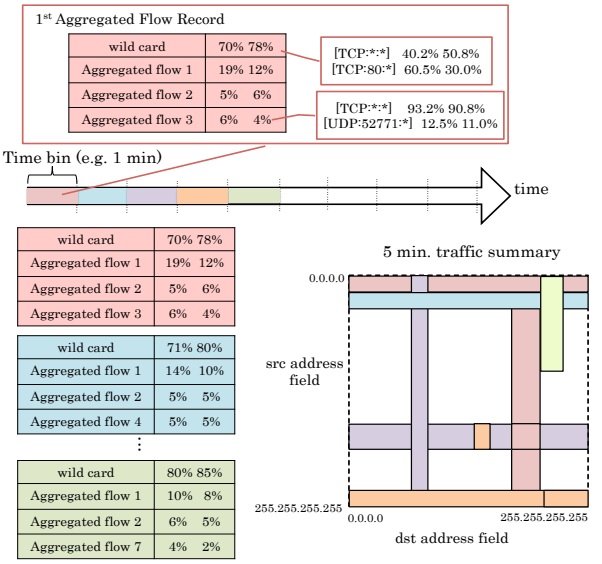

Figure 2: Agurim: In the primary aggregation, Agurim produces an aggregated flow record for each time slot. In the secondary aggregation, these records are mapped into two-dimensional address space based on the specified duration.

4 System Design

4.1 Agurim Overview

We propose a two-staged flow aggregation scheme, named "Agurim". The primary aggregation stage reads raw traffic records and produces rudimentary aggregated flow records. The secondary aggregation stage reads the output of the primary aggregation stage, and produces concise summaries for operators.

In both aggregation stages, we use source and destination IP addresses as main attributes, and protocol and ports as sub-attributes, following the operational practices. Although it is possible to treat each attribute equally, it is more expensive and the use case is fairly limited. To identify significant flows, we use both traffic volume and packet counts, so as to detect both volume-based and packet count-based anomalous flows.

Fig. 2 illustrates how the primary and secondary aggregation stages work. The time slot for the primary aggregation determines the highest time resolution for the secondary aggregation. In the example, the time slot is set to 1 minute.

The primary aggregation exports aggregated flow records at the end of each time slot. Fig. 3 shows a part of aggregated flow records. Each aggregated flow is represented in 2 lines: the first line shows the aggregated flow by the main attributes (source-destination address pair), and the second line shows the decomposition of protocol

```
%!AGURIM-1.0
%%StartTime: Sat Mar 31 11:50:00 2012 (2012/03/31 11:50:00)
%%EndTime:   Sat Mar 31 11:55:00 2012 (2012/03/31 11:55:00)
%AvgRate: 218.93Mbps 34299.30pps
# aggregated flow record
#[<rank>] <src addr> <dst addr>: <bytes> (%)  <packets> (%)
#     [<proto>:<sport>:<dport>] <bytes>(%) <packets>(%) ...
[ 0] * *: 2145550340(26.13%)   3914103(38.04%)
    [6:80:*]68.9% 31.2% [6:*:*]20.5% 26.4% [6:*:80]3.9% 26.3%
[ 1] 203.0.133.98 203.0.133.98: 1581920126(19.27%) 1108116(10.77%)
    [6:80:*]100.0% 100.0%
[ 2] 192.168.0.211 *: 996242054(12.13%) 695625(6.76%)
    [6:80:*]100.0% 100.0%
[ 3] 203.0.133.85 203.0.133.85: 819137103 (9.98%) 567032 (5.51%)
    [6:80:24626]86.1% 85.6% [6:80:*]13.9% 14.4%
  ...
```

Figure 3: A part of contents of an aggregated flow record

and port numbers within the main attributes. Because the secondary aggregation can produce the summary in the same format, it is possible to recursively apply the secondary aggregation to an output of the secondary aggregation (e.g., to create a daily summary from 1-minute summaries). The table at the top of Fig. 2 illustrates aggregated flow records. Each row of the list contains percentages of packet counter and traffic volume, and a pointer to another list for protocol and source and destination ports within the main attributes.

The secondary aggregation stage performs multi-dimensional aggregation by mapping all flow records for a specified duration into a two-dimensional address space. In the bottom of the figure, the secondary aggregation reads aggregated flow records for 5 time slots and maps each record into a two-dimensional address space.

4.2 Primary Aggregation

The primary aggregation stage is designed to efficiently process a huge volume of raw traffic records. We first aggregate each attribute of 5-tuple separately, as the original Aguri algorithm. Then, each packet is matched against the resulted aggregated attributes, and classified into an aggregated flow as a combination of the aggregated attributes of 5-tuple. In the primary stage, the flow aggregation is performed only for each attribute separately, and we do not use any multi-dimensional aggregation algorithm.

We have extended the Aguri algorithm [5] for the primary aggregation. Fig. 4 illustrates the aggregation algorithm of Aguri using a prefix-based binary tree for a single attribute. Each node represents an address with prefix length, and has 2 counters for packet and byte accounting. When a new address is observed, it is added to the tree as a leaf node. At the end of a time slot, small nodes are aggregated into its ancestor node until either byte or packet size of the node becomes larger than the predefined threshold. We empirically use 1% of the total traffic for the threshold. With the threshold of 1%, the number of aggregate attributes is usually less than 30 for

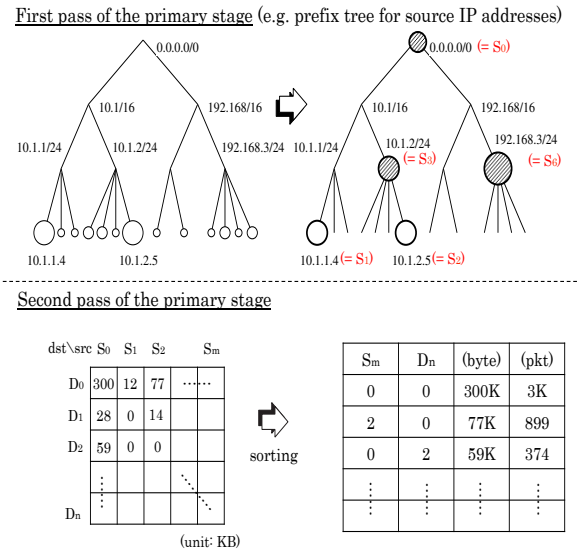

Figure 4: Primary aggregation process: The first-pass builds prefix-based trees for observed addresses. Small nodes are aggregated to ancestors at the end of a time slot. The second-pass produces an aggregated flow record that contains the combination of aggregated attributes and counts up them. Port/protocol space are processed in a similar way.

each attribute.

The same algorithm is used for port numbers to keep track of ranges of port numbers in a tree. Aguri embeds a protocol number into a port tree by concatenating a 8-bit protocol number and a 16-bit port number. Thus, Aguri maintains 4 trees for 5 tuple: for source address, for destination address, for source port (and protocol) and for destination port (and protocol).

To keep the memory usage and the performance for the tree search, Aguri uses a fixed number of tree nodes, and replaces nodes dynamically by the LRU algorithm. The reclaimed nodes are aggregated in the middle of a time slot.

In order to produce aggregated flows, each attribute of 5-tuple of a packet is matched against the aggregated attributes separately, which creates a combination of 5 aggregated attributes as an aggregated flow. The number of resulted aggregated flows are fairly small, usually less than 50 along the main attributes (source-destination address pairs), and less than 200 for 5 attributes.

The primary aggregation supports one-pass and two-pass algorithms to produce outputs. In the two-pass algorithm, at the end of a time slot, each attribute is aggregated separately in the first pass, and the entire input of the time slot is re-read in the second pass to match each input packet to the aggregated attributes to create aggregated flows. The two-pass algorithm is suitable when

saved traces are used as the input needs to be read twice.

On the other hand, the one-pass algorithm is used for online processing. In the one-pass algorithm, attribute matching is performed using the aggregated attributes produced in the previous time slot. Thus, the results are less accurate due to discrepancies in aggregated attributes between the previous and current time slots. However, a significant flow can be captured in the next time slot as long as it lasts longer than 2 time slots so that it is enough to set the time slot interval smaller than a half of the required interval. We will investigate the effect of the discrepancies introduced by the one-pass algorithm in the future work.

4.3 Secondary Aggregation

The secondary stage reads the output of the primary stage, and performs multi-dimensional aggregation to produce a traffic summary for operators. In order to generate a summary with specified temporal and spatial granularity, we use R-tree [8] algorithm.

R-trees are hierarchical data structures to hold multi-dimensional variables, and often used to support range queries over multiple dimensions. An R-tree is formed by aggregating Minimum Bounding Rectangles (MBRs) of spatial objects and storing aggregates in a tree structure. A parent node in an R-tree always covers the entire regions of its child nodes so that a range query is realized by a search for overlaps with the query rectangle from the top node to leaves.

Fig. 5 shows an example R-tree with six leaves and five MBRs. A node can be contained in multiple MBRs but has one (direct) parent MBR. For instance, o4 is contained in R3 and R4, but stored in the leaf pointed by R3.

To insert a new node, the tree is traversed from the top to find the best matching MBR. If necessary, a new MBR is created for the new node.

The secondary aggregation uses a R-tree for the two-dimensional source-destination address space. Each node entry is of the form of (source IP address with prefix length, destination IP address with prefix length). The protocol and port list is also included in each node, but they are not used as a search key in the tree. When a node is aggregated, the protocol and port list is also aggregated to the ancestor.

The secondary aggregation implements two aggregation modes: one is for intermediate summaries (e.g., daily and monthly summaries) and the other for graph plotting. The former is similar to the primary aggregation, and aggregates flows until flow's packet or byte count exceeds the threshold. The latter needs to create the specified number of aggregated flows. To create N aggregated flows, we first build an R-tree for the specified period, and repeat aggregating smaller flows until the remaining number of flows becomes N. After we have the N aggregated flows to plot, we go through each time interval and aggregate flows into these N aggregated flows.

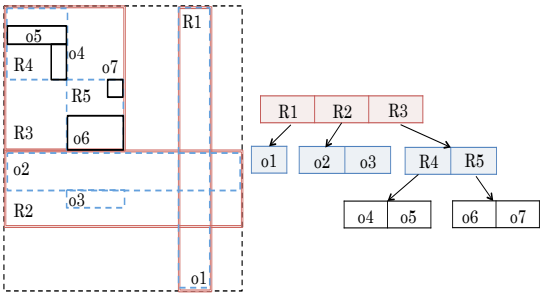

Figure 5: Spatial data access in R-tree: The R-tree data structure creates a rectangle with source addresses and destination address. The parent node holds the entire region of child nodes.

5 Preliminary Results

We have implemented a prototype of Agurim, the primary aggregation and the secondary aggregation, in C, and tested it with packet traces from backbone networks.

Note that our prototype is not tuned for performance, and the evaluation is just to show the feasibility of Agurim on a commodity PC. The prototype implementation of the primary aggregation is based on the original open source Aguri implementation, and can read pcap trace files and Netflow [16, 2] data.

We use 2 data sets: 4 x 60-minute-long packet traces collected in 2012 from a 10Gbps Tier-1 ISP link by CAIDA [4] and 7-day-long 150Mbps packet traces collected in 2012 from a transit link of the WIDE backbone [13].

To evaluate the performance of the primary aggregation and the secondary aggregation separately, we split the traces into 1-minute-long traces and used them as inputs for the primary aggregation.

For the evaluation, we used a Linux 3.0 system equipped with Intel Core(TM) i5-2520M CPU at 2.50 GHz and 8GB memory.

5.1 Primary Aggregation

We tested the primary aggregation stage using 272 x 1-minute-long `pcap` files from the CAIDA data set. The traffic volume in the traces is 2.11Gbps on average with 0.43Gbps standard deviation. The numbers of 5-tuple flows in the 1-minute-long traces are 776,792 on average,

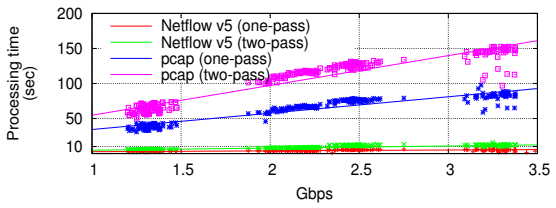

Figure 6: Processing time of primary aggregation stage: We use time command to measure performance with bit per second.

and the file size is 1.2GB on average. The primary aggregation using the 2-pass algorithm created only 22 aggregated flows for each 1-minute-long trace on average, whose size is only 8.5KB on average. When using the 1-pass algorithm, Agurim produces 60 aggregated flows on average.

We developed a pcap-netflow conversion tool based on the Netgraph tool [1] in order to compare the performance with two formats. The parameters is configured as following: We limit the number of flows in a table to 4,096 and flow count in a Netflow packet to 30. We set the inactive timeout value as 15 seconds, active timeout as 30 minutes.

Fig. 1 depicts the processing time with the bits per second. The upper plot shows that Agurim using 2-path algorithm takes around 50 seconds to process 60-second-long `pcap` data at 1Gbps, suggesting the current throughput being a little over 1Gbps. Even when we use 1-path algorithm, the decreasing of the processing time is not significant. We found that most of the CPU cycles are used for table look-up in the flow attribute trees; each packet requires 8 table looks (2 passes over 4 flow attributes). We are planning to implement a flow cache to improve the performance.

On the other hand, Agurim takes only around 10 seconds to process a 1-minute `Netflow` data at 3Gbps. The algorithm performs much better with `NetFlow` because flow records are already aggregated to some extent (no need to process every single packet). In addition, sampling can be used for `Netflow`.

5.2 Secondary Aggregation

To evaluate the secondary aggregation, we used the one-week-long packet traces from the WIDE project. (We cannot use the CAIDA data set as they are not contiguous.) We used the traffic summaries for a 1-minute created by the primary aggregation. We set the maximum number of leaves in a R-tree to four in this test.

Table 1 shows the processing time to produce a plot for users. In the table, the "duration" is the entire time period to be displayed in the plot. The "time" shows the average processing time of the secondary stage from the time to start reading data to the time to finish producing the plot data. We use the same time resolution of the plot, 60-minute, for different duration for comparison.

Duration	# of aggr flows	Time
12-hour	2,178	0.44 sec
1-day	3,796	1.35 sec
3-day	9,858	13.46 sec
1-week	23,065	75.77 sec

Table 1: Processing time of the secondary stage

The processing time in the secondary stage depends on the number of unique aggregated flows. It increases exponentially against the number of aggregated flows due to the node look-up time in a tree.

Although the visualization time for a long duration does not satisfy reasonable waiting time, we can avoid this problem by re-aggregation. It is more efficient to prepare pre-processed aggregated records (e.g., hourly records, weekly records) for plots with coarse time resolutions, because periodic re-aggregation can reduce the number of flows to be read and aggregated.

6 Conclusion

We have presented Agurim, a traffic monitoring system using multi-dimensional flow aggregation. Agurim allows operators to easily capture dynamic changes in traffic with varying resolutions.

Our system supports flow re-aggregation with a two-staged aggregation algorithm; once aggregated flow records are created, they can be efficiently re-aggregated with coarser temporal and spatial granularity. It allows operators to monitor coarse-grained aggregated flows, and then, zoom into suspicious flows.

We have tested our prototype implementation with traffic traces from CAIDA and WIDE. The performance of the prototype is promising, although there are rooms to improve.

For the future work, we will develop accuracy and performance metrics to evaluate our system to find a good balance between the accuracy and performance of flow aggregation, and compare them with the previous work. We believe that Agurim will help to manage complex and dynamic today's networks.

Acknowledgments

This research was partially supported by National Institute of Information and Communications Technology (NICT).

References

[1] Netflow implementation for netgraph. http://sourceforge.net/projects/ng-netflow/.

[2] E. B. Claise. Cisco Systems NetFlow Services Export Version 9. http://www.ietf.org/rfc/rfc3954.txt.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD, New York, NY, USA, 1990. ACM.

[4] Caida. URL http://www.caida.org/home/.

[5] K. Cho, R. Kaizaki, and A. Kato. Aguri: An aggregation-based traffic profiler. In *Quality of Future Internet Services*, 2001.

[6] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 155–166, New York, NY, USA, 2004. ACM.

[7] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *SIGCOMM*, 2003.

[8] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD, pages 47–57, New York, NY, USA, 1984. ACM.

[9] P. W. Huang, P. L. Lin, and H. Y. Lin. Optimizing storage utilization in r- tree dynamic index structure for spatial databases. In *J. Syst. Softw.*, 2001.

[10] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM, pages 135–146, 1999.

[11] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37:238–275, September 2005.

[12] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed ip routing table lookups. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM, pages 25–36, 1997.

[13] Wide project. URL http://www.wide.ad.jp/.

[14] L. Yuan, C.-N. Chuah, and P. Mohapatra. Progme: towards programmable network measurement. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM, pages 97–108, New York, NY, USA, 2007. ACM.

[15] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC, 2004.

[16] T. Zseby, S. Zander, and G. Carle. Policy-Based Accounting. http://www.ietf.org/rfc/rfc3334.txt.