Aguri: An Aggregation-based Traffic Profiler

Kenjiro Cho[1], Ryo Kaizaki[2], and Akira Kato[3]

[1] Sony Computer Science Labs, Inc., Tokyo 1410022, Japan
`kjc@csl.sony.co.jp`
[2] Keio University, Fujisawa 2528520, Japan
`kaizaki@sfc.wide.ad.jp`
[3] The University of Tokyo, Tokyo 1138658, Japan
`kato@wide.ad.jp`

Abstract. Aguri is an aggregation-based traffic profiler targeted for near real-time, long-term, and wide-area traffic monitoring. Aguri adapts itself to spatial traffic distribution by aggregating small volume flows into aggregates, and achieves temporal aggregation by creating a summary of summaries applying the same algorithm to its outputs. A set of scripts are used for archiving and visualizing summaries in different time scales. Aguri does not need a predefined rule set and is capable of detecting an unexpected increase of unknown protocols or DoS attacks, which considerably simplifies the task of network monitoring.
Once aggregates are identified and profiled, it becomes possible to make use of the profile records to control the aggregates in best-effort traffic. As a possible solution, we propose a technique to preferentially drop packets from aggregates whose volume is more than the fairshare. Our prototype implementation demonstrates its ability to protect the network from DoS attacks and to provide rough fairness among aggregates.

1 Introduction

Traffic monitoring is essential to network operation in order to understand usage of the network and identify abnormal conditions or threatening activities. Also, longer-term monitoring is needed for capacity planning or for tracking trends. Flow-based traffic profiling in which packets are categorized into traffic types and usage information is recorded for each type is commonly used for traffic monitoring [3, 9]. Flow-based traffic monitoring, combined with visualization techniques, provides a powerful tool to understand network conditions [2, 16, 20, 21].

However, a weakness common to the existing flow-based monitoring tools is that, to identify traffic types, predefined filter rules are needed. Filter rules are used to classify packets by examining fields in the packet header. Thus, without *a priori* definitions of traffic types, packets cannot be identified. Flow-based monitoring is facing a difficulty identifying new protocols and dynamically assigned ports. Even for known traffic types, it is not practical to list all possible

[0] this document is to appear in proceedings of QofIS2001 published by Springer-Verlag in the LNCS series. ©Springer-Verlag

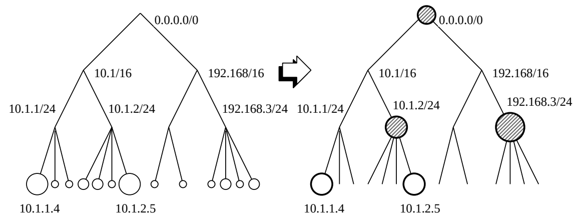


Fig. 1. aggregation profiler concept: small entries are aggregated into aggregates

combinations in the rule set so that minor traffic types are often left undefined and remain unidentified.

On the other hand, the current Internet is exposed to the menace of Denial of Service (DoS) attacks, and DoS attack detection is the highest priority for network operation. The rule-based approach lacks an ability to detect DoS attacks since forged packets can have arbitrary traffic types.

We have been monitoring the WIDE research backbone for years [8], and badly in need of an adaptive monitoring tool for trouble detection, usage reporting and long-term trend analysis. Our focus is traffic measurement to aid network operation, and thus, concise and timely summary reports are more important than precise and detailed reports.

To this end, we have developed a software package called aguri. Aguri uses a traffic profiling technique in which records are maintained in a prefix-based tree and a compact summary is produced by aggregating entries.

Powerful is the feature to produce a summary of summaries applying the same algorithm to its own outputs. Thus, derivative summaries can be produced in different time scales desirable for a specific monitoring purpose. A set of scripts have been developed to visualize summaries. It is also possible to extend the profiler as a protective measure against DoS attacks.

Aguri is targeted for near real-time, long-term, and wide-area traffic monitoring. Because automatic aggregation is used for profiling, our approach provides rough usage reports which may not be precise so that it is complementary to the existing tools.

2 Overview of Aguri

The core idea of an aggregation-based profiling is to aggregate flow entities for profiling. Small volume flows are aggregated until the volume of the aggregate becomes large enough to be identified. A summary output reports the profile of aggregates. An entry in an address profile can be a single host if it consumes a certain portion of the total traffic, or an aggregate if each host entry is small but the aggregate becomes non-negligible. Thus, a limited number of entries are produced, yet it never fails to report high volume entries.

Figure 1 illustrates the concept. A tree before aggregation is on the left and the corresponding tree after aggregation is on the right. Each node in the tree shows the address space represented by an address prefix and its prefix length. A leaf node corresponds to a single address. The size of a node shows the traffic volume of the node. The usage information recorded at leaf nodes can be aggregated to the shaded internal nodes in the right tree, and a summary reports only the remaining nodes in the right tree.

Summary Profile. It is important to produce concise summary profiles. When a traffic profile is too detailed, important symptoms are buried in excessive data, and often left unnoticed. Each summary profile produced by aguri is compact since small entries are aggregated in a profile.

Aguri produces four separate profiles for source addresses, destination addresses, source protocols and destination protocols. IP addresses are designed to be hierarchical and aggregatable so that it is natural to apply aggregation. Both IPv4 and IPv6 are supported in address profiles. Although protocol numbers are not hierarchical, the same technique can be used to identify port ranges. We concatenate the IP version, the protocol number and the TCP/UDP port number to create a 32-bit key for a protocol profile. A summary reports the total byte count used by each aggregate.

The four separated profiles are effective to capture hostile activities. A victim of a distributed DoS attack will be easily identified in the destination address profile. An originator of port scanning will be identified in the source address profile. A random attack will be identified as a range of addresses as long as some locality exists for the targets. If the locality is unusually low, it is another symptom of a random attack.

Spatial Aggregation. The basic algorithm of the spatial aggregation is quite simple. If there is no resource constraints such as memory consumption or execution time, we could profile every address and protocol occurrence in every packet and, at the end, aggregate entries whose counter value is less than an aggregation threshold. This approach would be acceptable for post-analysis of a saved packet trace. For near real-time monitoring, however, we approximate the above algorithm in exchange for the precision, by managing a fixed number of nodes in the tree using a variant of the Least-Recently-Used (LRU) replacement policy.

When a leaf node is reclaimed, the counter value of the node is aggregated to its parent node. The advantage of this approach is that counter values are never lost even though the resolution is reduced.

To produce a summary output, aguri walks through the tree in the post-order and aggregates nodes if the counter value of a node is less than the aggregation threshold, or outputs the node information if the counter value is above the threshold.

To continue profiling, it is enough to reset the counter of each node; the current tree and the LRU list are kept in tact as a cache, and used for the next profiling period.

Temporal Aggregation. The same algorithm can be used to produce a summary of summaries. Aguri can read its summary outputs, reaggregate them, and produce a new coarse-grained summary. For instance, a 1-hour-long summary can be created out of 60 1-minute-long summaries.

In this paper, an "initial summary" is used to represent a summary directly produced from non-aggregated sources such as captured packets. A "derivative summary" represents a summary produced from summaries.

The method is suitable for archiving profiles since a summary can be created in different time scales from a set of archived summaries. It is also possible to control the resolution by changing the aggregation threshold. The process to generate and archive derivative summaries can be easily automated. Network operators will usually look at only coarse grained summaries but can look into fine grained summaries if necessary.

Archiving and Visualization Utilities. A summary output is in a plain text format so that it is easily processed by various scripts. For archiving, a script is periodically invoked to generate and archive derivative summaries in different time scales such as hourly, daily, monthly, and yearly summaries. The size of a summary is about 5KB so that a small amount of disk space is required for archiving summaries.

Text-based summaries can be converted to a variety of visual images. We have developed a set of scripts for visualization to aid operators to find unusual conditions in summary outputs.

Application for Traffic Control. Once aggregates are identified and profiled, the profile records can be used for traffic control. There are many possible approaches to control aggregates. For example, a rate-limiter can be installed at a firewall to protect the network from a high-bandwidth aggregate [17].

We propose an aguri three color marker (aguriTCM) that combines an aggregation-based profiler with a preferential packet dropping mechanism. The aguriTCM identifies aggregates whose traffic volume is more than the fairshare, and probabilistically raises the drop precedence for those aggregates. The aguriTCM provides rough traffic management based on aggregates in best-effort traffic; the resolution of the control is limited by the resolution of an aggregate in the profile.

Our approach uses Diffserv components as building blocks but the primary target is a stand-alone protection mechanism to minimize the effect of DDoS or flash crowd in best-effort traffic. It also provides rough fairness among aggregates.

3 Related Work

MRTG [20] and its successor RRDtool [19] create time-series round-robin databases. They store numerical time-series data and automatically aggregate it into averages over time. Our idea of producing a summary from summaries is inspired by MRTG and RRDtool but differs in combining temporal aggregation with spatial aggregation.

Traditional flow-based monitoring tools such as NeTraMet [2] and FlowScan [21] require predefined rules to monitor a specific type of traffic. For example, in order to monitor HTTP traffic, they need to be instructed to identify TCP port 80. The approach with explicit and fixed rules has limitations on identifiable traffic types. Especially, it is a problem to cope with unknown protocols or DoS attacks.

Another approach is to report the top N flows by sorting the flow list [24, 4]. Although it does not need a rule set, there could be limitations on the maintainable number of flows or a flooding attack could easily overflow the list. Hence, it is not suitable for detecting DoS attacks. In our approach, a flooding attack may be able to reduce the resolution of the profile but the counter values are never lost. It is resilient to DoS attacks in addition to requiring no rules.

Dynamic identification of a flow is also addressed in the context of congestion control and DoS prevention. Floyd *et al.* in [11] argue on the need for end-to-end congestion control, and further, on the need for mechanisms in the network to detect and restrict unresponsive or high-bandwidth best-effort flows in times of congestion. They suggest to use the RED drop history as samples to identify misbehaving flows. The concept is known as a RED penalty-box [6].

This idea is further extended and detailed in order to cope with DDoS attacks and flash crowds [17]. It consists of a mechanism to identify aggregates, a local rate-limiter mechanism, and a pushback mechanism to propagate protective actions to neighbors. The proposed technique to identify high-bandwidth aggregates is based on the destination address in the drop history, and clusters the addresses into aggregates. The approach of identifying high-bandwidth aggregates and regulate them is similar to ours in the concept.

While their focus is to identify misbehaving flows, our focus is a traffic profiler which monitors and reports the network not only under congestion but all the time. Our observation is that a network point needing a protection mechanism is often a point to be monitored. Hence, it is practical to provide a combined solution both for performance and for simplicity. The combined method comes with visible monitoring outputs so that it could be advantageous to deployment.

4 Implementation

Aguri, as shown in Figure 2, is implemented as a user program on UNIX. The input modules on the left translate different input formats into a 4-tuple (tree, key, prefix-length, count) and pass them to the profiler engine in the center. Aguri prints summaries to the standard output or a file.

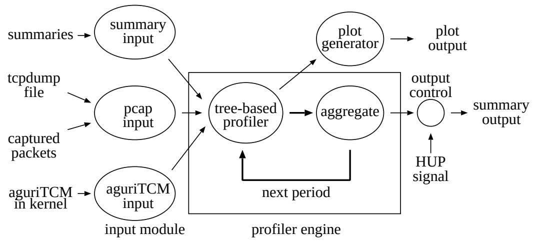


Fig. 2. aggregation profiler implementation model

The first input module reads aguri's summary outputs from files or from the standard input to produce a derivative summary. The second input module is an interface to the pcap library [15] that captures packets from a live network or reads a packet trace file saved by tcpdump [14]. The pcap interface allows us to evaluate our prototype using various tcpdump trace files. The third input module reads binary profiles produced by the aguriTCM in the kernel.

The profiler engine consists of the tree-based profiler and the aggregation module. The tree-based profiler accepts 4-tuples from one of the input modules, and maintains profile records in the trees. At the end of a profiling period, the aggregation module is called to produce a summary. While the aggregation module is walking through the tree in the post-order, each node is either aggregated or reported. To continue profiling, the profiler engine repeats this cycle.

4.1 Summary Output

Figure 3 shows an example of aguri's summary output. A summary starts with a header block, followed by a body block. Lines start with % are comment lines. The body block contains 4 profile types by default but only the destination address profile is shown in the figure. [1]

In the address profile, each row shows an address entry and is indented by the prefix length. The first column shows the address and the prefix length of the entry. When the prefix length is the full length, it is omitted in the output. The second column shows the cumulative byte count. The third column shows the percentages of the entry and its subtree.

The input for this example is a 5-second-long packet trace taken from a trans-pacific link of the WIDE backbone. The parameters of aguri is configured with 256 nodes and 1% aggregation threshold. Among 17,564 observed addresses, only 14 addresses are identified as individual addresses. 38.05% of the traffic belongs to 173.106.176/20; within this address space, 6 distinct addresses are identified. The number of individual addresses found in a typical summary is from 5 to 20. In our trans-pacific profiles, several individual addresses are still identified even in monthly summaries.

[1] IP addresses appearing in this paper are scrambled for privacy.

```
%!AGURI-1.0
%%StartTime: Sat Jan 06 14:00:00 2001 (2001/01/06 14:00:00)
%%EndTime:   Sat Jan 06 14:00:05 2001 (2001/01/06 14:00:05)
%AvgRate: 17.05Mbps

[dst address] 10658367 (100.00%)
0.0.0.0/0 105652 (0.99%/100.00%)
 0.0.0.0/2 196398 (1.84%/1.84%)
 128.0.0.0/1 141492 (1.33%/97.17%)
        133.28.0.0/16 146217 (1.37%/11.08%)
                133.28.21.21 179320 (1.68%)
         133.28.128.0/17 257220 (2.41%/8.03%)
                133.28.128.14 127541 (1.20%)
                133.28.202.127 470854 (4.42%)
   152.0.0.0/5 157159 (1.47%/25.69%)
        152.10.0.0/16 336636 (3.16%/20.28%)
         152.10.0.0/17 433037 (4.06%/15.16%)
                152.10.1.247 1182481 (11.09%)
                152.10.135.189 208992 (1.96%)
      156.96.0.0/11 253884 (2.38%/3.94%)
        156.114.0.0/16 165979 (1.56%/1.56%)
   168.0.0.0/5 315417 (2.96%/47.96%)
                168.89.12.93 275740 (2.59%)
      173.96.0.0/12 465797 (4.37%/42.42%)
          173.106.176.0/20 248236 (2.33%/38.05%)
                173.106.177.162 440466 (4.13%)
                173.106.177.163 550897 (5.17%)
                173.106.177.172 602230 (5.65%)
                173.106.177.173 1498198 (14.06%)
                173.106.187.134 559784 (5.25%)
                173.106.187.135 155322 (1.46%)
   192.0.0.0/5 111918 (1.05%/8.45%)
    194.0.0.0/7 375630 (3.52%/7.40%)
                194.105.251.45 168327 (1.58%)
                195.130.218.237 244270 (2.29%)
  208.0.0.0/4 283273 (2.66%/2.66%)
%LRU hits: 82.62% (14511/17564)
```

Fig. 3. a sample output of a destination address profile

A source address profile looks similar. A source address profile tends to identify popular www or ftp servers, whereas a destination address profile tends to identify proxy servers and mirror servers.

Figure 4 shows source and destination protocol profiles. The first column shows a 32-bit key concatenating the IP version number (8bits), the protocol number (8bits), and the TCP/UDP port number (16 bits). For example, "4:6:80" represents IPv4/TCP/HTTP.

In this summary, 96.15% of the total traffic is TCP. Only four individual ports, TCP port 20 (ftp-data), 80 (http), 6346 (gnutella), UDP port 53 (dns), are identified in the source address profile. Note that the use of gnutella is automatically detected without any knowledge about gnutella's use of port 6346.

The destination protocol profile includes a larger number of dynamically assigned ports which are usually aggregated and shown as port ranges. A source protocol profile tends to identify protocols used by servers, and a destination protocol profile tends to identify clients.

```
[ip:proto:srcport] 10570555 (100.00%)
0/0:0:0 4967 (0.05%/100.00%)
4:0/3:0 290382 (2.75%/99.95%)
4:6:0/0 164255 (1.55%/96.15%)
   4:6:0/3 540369 (5.11%/93.38%)
                4:6:20 663178 (6.27%)
                4:6:80 7329218 (69.34%)
        4:6:1024/8 106427 (1.01%/1.01%)
        4:6:1280/8 139741 (1.32%/2.75%)
         4:6:1280/9 150514 (1.42%/1.42%)
       4:6:1536/7 182444 (1.73%/1.73%)
     4:6:2048/5 564594 (5.34%/5.34%)
                4:6:6346 194004 (1.84%)
 4:6:32768/1 128925 (1.22%/1.22%)
                4:17:53 111537 (1.06%)
%LRU hits: 60.80% (10644/17506)

[ip:proto:dstport] 10570555 (100.00%)
0/0:0:0 4967 (0.05%/100.00%)
4:0/3:0 401919 (3.80%/99.95%)
4:6:0/0 579078 (5.48%/96.15%)
         4:6:0/9 327066 (3.09%/4.54%)
                4:6:80 152813 (1.45%)
       4:6:1024/7 419016 (3.96%/17.12%)
         4:6:1024/9 781275 (7.39%/7.39%)
        4:6:1280/8 609679 (5.77%/5.77%)
       4:6:1536/7 597213 (5.65%/12.77%)
        4:6:1536/8 752782 (7.12%/7.12%)
      4:6:2048/6 666539 (6.31%/21.84%)
       4:6:2048/7 155545 (1.47%/15.54%)
         4:6:2176/9 387335 (3.66%/7.96%)
          4:6:2176/10 454168 (4.30%/4.30%)
        4:6:2304/8 645406 (6.11%/6.11%)
      4:6:3072/6 893343 (8.45%/8.45%)
    4:6:4096/4 172569 (1.63%/9.51%)
       4:6:4608/7 688892 (6.52%/6.52%)
                4:6:6346 143558 (1.36%)
  4:6:49152/2 492936 (4.66%/16.44%)
                4:6:49249 1107484 (10.48%)
                4:6:49635 136972 (1.30%)
%LRU hits: 53.96% (9446/17506)
```

Fig. 4. a sample output of protocols and ports

4.2 Spatial Aggregation

The profiler engine implements the prefix-based aggregation algorithm. To produce summaries continuously in near real-time, we need an efficient algorithm in terms of CPU power and memory usage. An approximation limits the number of entries used in a tree, and thus, will make more aggregation than the ideal algorithm. As a result, it introduces two types of errors: (1) part of the counter value could be aggregated to the ancestors, and (2) the entry of a node close to the aggregation threshold could be removed and may not show up in the summary. These errors lower the precision but the impact would be limited. After all, these errors are unavoidable for derivative summaries since aggregation discards details. However, if an entry consumes a non-negligible volume of the total traffic, any approximation will be able to detect it.

To limit memory use and search time with variable length keys, we employ a Patricia tree. Patricia has been employed in the BSD kernel for the internal representation of the routing table [23], and its performance characteristics are well understood. It is suitable to handle 32-bit IPv4 addresses and 128-bit IPv6 addresses.

Patricia is a full binary radix tree. All internal nodes have exactly two children so that when the number of leaf nodes is N, the number of internal nodes is $(N-1)$. Thus, it is suitable for use with a fixed number of nodes, and nodes can be preallocated.

Each node has a prefix as a key associated with its prefix length. The key of an internal node is the common prefix of its two children.

Our use of Patricia is different from the routing table. While the routing table lookup requires best-match, we have only exact-match. In our scheme, a new node is always created when no matching node is found. If there is no available free node, an old node is reclaimed to keep the number of nodes in the tree. Thus, node insertions and deletions occur during a lookup operation.

To update an entry record, the profiler looks up the entry in the tree, and updates the counter value of the entry. A lookup starts from the root node to a leaf node, checking prefix-matching. If the prefix matches with the internal node, the bit at $(prefixlen + 1)$ of the search key indicates which branch to follow; if the bit value is 0, take the left branch, otherwise, take the right branch. If the matching leaf node is found, the search terminates and the counter of the node is updated.

If the prefix does not match, it indicates no matching node exists in the tree. A new node is created and inserted into the tree. The key is assigned to the new node, and the count is set to the counter. An insertion always creates a leaf and a branch point since single branching is not allowed. The new branch point is inserted as a parent of the unmatching node; the other child of the branch point is the newly created leaf node. The common prefix of the two child nodes is assigned to the branch point. Similarly, deleting a leaf node removes the leaf and its parent. When deleting a node, the counter value is aggregated to its parent.

A fixed number of nodes are preallocated for a tree, and a variant of the LRU replacement policy is used for managing leaf nodes. If the number of nodes is 256, the tree has 128 leaf nodes since $(N-1)$ internal nodes are needed for N leaf nodes. The LRU is selected because it is simple, cheap and well-understood. The precision could be further improved by using an elaborate algorithm such as the frequency-based replacement [22] but there is a tradeoff between the precision and the efficiency. As already mentioned, the precision is not so important in our scheme and it is evaluated in Section 5.

Since the LRU reclaims a node even when its counter value is very large, a simple heuristic is added not to reclaim a node if the sum of the counter values of the node and its parent is larger than a threshold. The current reclaim exemption threshold is set to 3.123% or 1/32 of the total count.

In the middle of a profiling period, a snapshot of the tree contains nodes with small count values. Nodes whose count value is less than the aggregation threshold are aggregated at the end of the profiling period. The aggregation threshold is set to 1% of the total count by default. The profiler walks through the tree in the post-order so that aggregation and summary output can be done in one pass.

To continue profiling, the profiler just resets the counters and keeps the tree and the LRU list in tact as a cache for the next profiling period. The profiler could reset the counters when aggregating the nodes. However, a two-pass method is used in the current implementation to show the sum of the subtree for readability. The aguriTCM, on the other hand, omits the subtree sum and employs a one-pass method.

IPv4 and IPv6 addresses have different key length. They could be managed in a single tree but separate trees are currently used for ease of debugging. The aggregation threshold is computed from the combined total count so that there is no difference in the summary. On the other hand, the key length is the same for protocol trees so that the profiler uses merged trees.

The profiler uses the same algorithm to produce derivative summaries but there are subtle differences. The size of input sets is much smaller and there are less constrains on execution time or resource usage. Another difference in the Patricia algorithm is that internal nodes are added to insert aggregates, while only leaf nodes are added for initial summaries. A single implementation is currently used for both initial and derivative summaries to reduce the maintenance cost but it could be separately optimized.

4.3 Archiving and Visualization Utilities

Archiving. Aguri prints summaries to the standard output or a file. On receiving a HUP signal, the output file is reopened so that the output file can be redirected to a new file. To archive summaries, a script is periodically invoked by *cron*. The script saves the current output file and sends a HUP signal to the running aguri program to switch the output file.

In our current setting, aguri produces a new summary every 5-seconds. A new summary file containing 24 summaries is created every 2-minutes. The script also generates hourly/daily/monthly/yearly summaries when crossing the time boundaries. It is also possible to customize the script to detect a certain condition and send an alert to the operator.

A summary output size varies depending on the traffic but is usually about 5KB. Uncompressed derivative summaries take about 150KB/hour, 3.5MB/day, 105MB/month and 1.2GB/year. If the initial summaries created every 5-seconds are saved, they consume additional 100KB for every 2 minutes. The initial summaries will take about 3MB/hour, 70MB/day, 2GB/month, and 24GB/year but these detailed summaries can be discarded after a certain period.

Plot Graph. Aguri supports a plot format output suitable to draw a plot graph. The plot format lists the counter values of the entries in a line; each line corresponds to a profiling period. It also supports conversion from byte-count

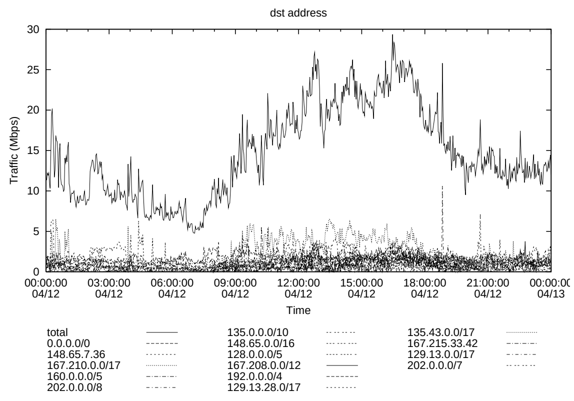


Fig. 5. a graph plotting 1-day destination addresses

to bits-per-second. A plot output is usually created from archived summaries and does not need to do in real-time. It is also needed to specify the number of entries in a plot. Thus, the plot generator uses a 2-phase algorithm which reads input files twice.

The first phase computes the cumulative byte count for each entry. At the end of the first phase, a sorted plot list is created, and the smallest entry is repeatedly aggregated until the number of nodes is reduced to the specified number. The second phase produces a plot format output for each period. For each period, if a node is not found in the plot list, it is aggregated to the nearest ancestor listed in the plot list. Hence, all counts are reflected to the plot.

Figure 5, 6 and 7 show examples of plot graphs taken from the trans-pacific link. The legend below the graph shows entries in the plot. Figure 5 plots destination addresses for 1 day on April 12, 2001, created from 2-minute summaries. Two individual addresses (148.65.7.36 and 167.215.33.42) are listed but there is no prominent address in terms of the bandwidth share.

Figure 6 plots source protocols for 10 days, from April 10 to 19, 2001, created from 1-hour summaries. The graph captures daily fluctuations of the total traffic and the high ratio of HTTP. In Figure 6, there is a change in the daily traffic pattern on the 17th. By zooming into the 17th as shown in Figure 7, we can see unusual surges of ICMP. It is a *smurf* attack and this is the cause of the distortion in the daily traffic. We can identify the target address and the address range of the originators by looking into the corresponding address profiles. This illustrates how plot graphs in different time scales can be used for trouble shooting.

Traffic Density Graph. Another graph format shows traffic density within the entire address space. From a summary, we can compute the traffic density in the address range of each aggregate, and create a time-series color graph. In

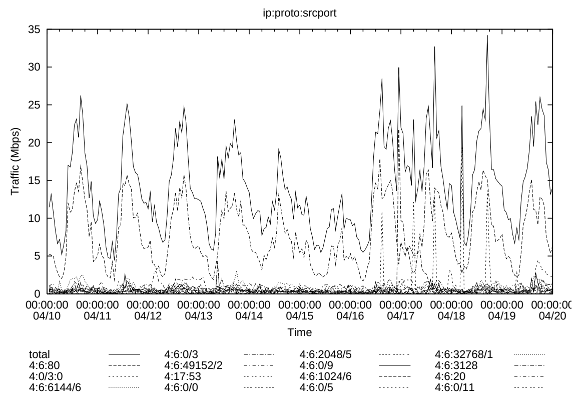


Fig. 6. a graph plotting 10-day source protocols

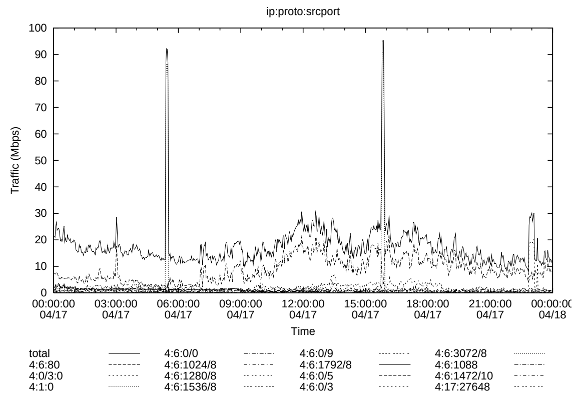


Fig. 7. a graph zooming into April 17th

a traffic density graph, the degree of traffic concentration is shown by colors and a change in traffic pattern is easily identified.

5 Evaluation

We have done a trace-driven evaluation using two 1-hour-long packet traces from the WIDE backbone [8]. Trace #1 is taken from a trans-pacific link, and trace #2 is taken from a link connected to a domestic IX. A set of shorter packet traces are extracted from the two traces. Table 1 shows the number of packets, the number of distinct addresses, and the observed rate in the traces.

The test configuration uses 256 nodes in a tree, 1% aggregation threshold, and 1/32 aggregation exemption threshold, unless otherwise specified.

Table 1. packet traces used for evaluation

trace	length	# of packets	# of addresses	rate (bps)
#1	1sec	3929	775	20.92M
	5sec	19977	1884	21.12M
	60sec	242187	7297	22.44M
	3600sec	16351933	75530	25.55M
#2	1sec	1380	295	4.27M
	5sec	6664	786	3.72M
	60sec	113680	3617	7.10M
	3600sec	5289374	25981	3.91M

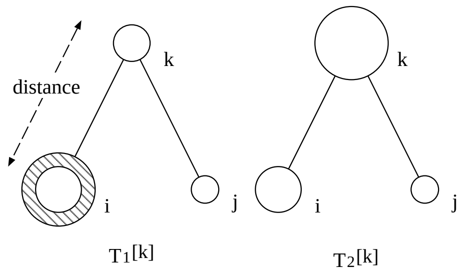


Fig. 8. distortion of two subtrees: the ideal tree on the left and the approximation on the right

5.1 Aggregation Accuracy

In our algorithm, the resolution of aggregation depends on the aggregation threshold. The number of nodes used in a tree, the replacement policy, the generation of derivative aggregation also affect the precision of a result.

Although accuracy is not the most important factor to the algorithm, it is better to understand the impact to the results. To measure the distortion in the resulting tree, we introduce the distortion index that provides a quantitative difference of two trees.

Distortion Index. The approximation in our algorithm introduces excessive aggregation in the resulting tree. We need to measure errors caused by the excessive aggregation, by comparing the resulting tree with the ideal tree. Traditional tree matching methods in graph theory (e.g., edit-distance) are not suitable for this purpose since they do not take aggregation into consideration.

Aggregation moves the counter value of a node to its ancestors but it never affects the other nodes. The aggregated value could be spread over multiple ancestors. Thus, we should do subtree-by-subtree comparison rather than node-by-node comparison.

Figure 8 illustrates the distortion. $T_1[k]$ on the left is a subtree at kth node in ideal tree T_1 and $T_2[k]$ on the left is the corresponding subtree in approximation T_2. The shaded portion of node i is aggregated to node k in the right subtree. When we compare $T_1[i]$ with $T_2[i]$, the volume of the shaded area is considered shifted by the distance from i to k that is the difference of their prefix length. $T_1[k]$ and $T_2[k]$ is considered equal since their subtrees have the same volume. If T_2 does not have a corresponding node, we assume a virtual node with size 0.

We introduce a distortion index to quantify the difference. Let $D_{12}[i]$ be the distortion index from $T_1[i]$ to $T_2[i]$. We compare the total count of the subtree at node i. $s_1[i]$ and $s_2[i]$ are the sum of the counters in $T_1[i]$ and $T_2[i]$, respectively. If $s_1[i]$ is larger than $s_2[i]$, the difference is considered to be aggregated into the ancestor nodes in T_2. Thus, we find the nearest ancestor k where

$$\frac{|s_1[k] - s_2[k]|}{s_1[k]} < \varepsilon$$

ε is an error term to allow small differences in size matching. We use 0.05 for ε. $d_{12}[i]$ represents the distance from i to k, normalized to the full prefix length.

$$d_{12}[i] = \frac{prefixlen(i) - prefixlen(k)}{prefixlen_{max}}$$

$r_{12}[i]$ is the ratio of the difference in the subtree coverage at node i, normalized to the subtree size.

$$r_{12}[i] = \begin{cases} \frac{s_1[i]-s_2[i]}{s_1[i]} & \text{if } (s_1[i] > s_2[i]) \\ 0 & \text{otherwise} \end{cases}$$

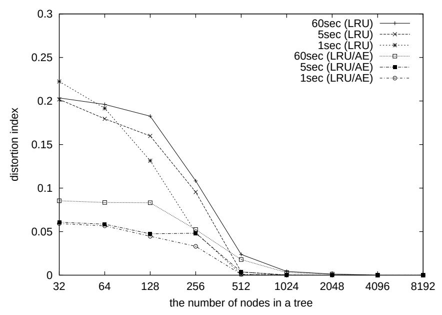
$w[i]$ is the weight of node i in the tree, and computed as the byte count of the node divided by the total byte count of the tree. Then, we get the normalized distortion at node i as

$$D_{12}[i] = w[i] \cdot r_{12}[i] \cdot d_{12}[i]$$

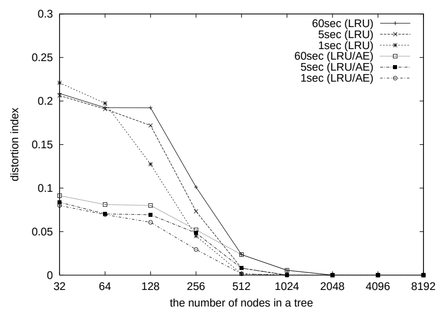
Each item ranges from 0 to 1.0. A small exponent, b, is added to each item as a bias towards small errors because small errors are expected by aggregation. We use 1.2 for b. The distortion index for the entire tree can be obtained as the sum of the indices. By making it symmetric, the distortion index becomes

$$D = \frac{\sum_{i \in T_1} w^b \cdot {r_{12}}^b \cdot {d_{12}}^b + \sum_{j \in T_2} w^b \cdot {r_{21}}^b \cdot {d_{21}}^b}{2}$$

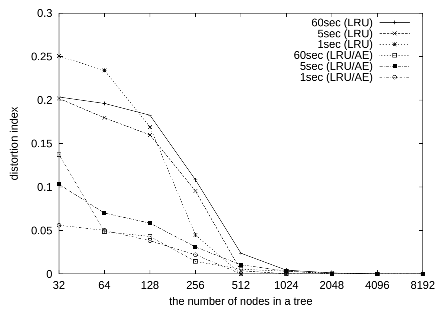
This index, albeit not perfect, at least allows us to quantify the results. When two trees are exactly the same, D becomes 0. When one tree has all the count at leaf nodes and the other tree has all the count at the root node, D becomes 0.5. When there is no overlap, D becomes 1.0. For example, one tree has all the count at leaf nodes in the left branch and the other tree has all the count at leaf nodes in the right branch.



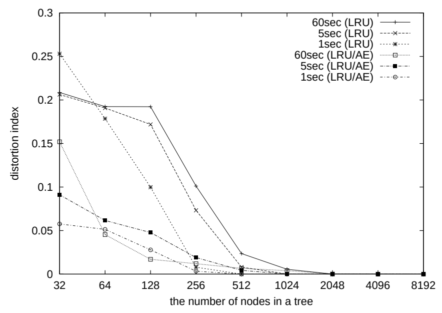
(a) trace #1 source addresses



(b) trace #1 destination addresses



(c) trace #2 source addresses



(d) trace #2 destination addresses

Fig. 9. distortion caused by LRU with varying tree size and the profiling period length

Accuracy Results. We use the distortion index to evaluate our LRU-based algorithm. Figure 9 shows the effects of the number of nodes and the profiling period length, with or without the heuristic added to the LRU algorithm, in the source and destination address trees of the two traces.

In the figure, "LRU" shows the simple LRU algorithm, and "LRU/AE" shows the LRU with the aggregation exemption threshold. The distortion index is computed with the ideal results in which there is no restriction on the number of nodes.

The effect of the different period length are tested by the traces with different length. Even though the number of the included addresses differs in orders of magnitude, the results look similar. It suggests that there is a locality in address occurrence, and thus, the results are not affected much by the trace length.

As expected, the simple LRU works well when there are enough nodes but the distortion becomes larger when nodes are insufficient. The aggregation exemption reduces distortion, especially when the profiler runs out of nodes.

Table 2 shows the differences in summary generations. "3600s" shows the initial summary directly produced from the packet trace. This is the base summary for comparison. "1sx3600" is a second-generation summary produced from 3600 1-second summaries. "60sx60" is another second-generation summary produced from 60 60-second summaries. "1sx60x60" is a third-generation summary.

Table 2. distortion in summary generations

type (gen.)	3600s (1st)	1sx3600 (2nd)	60sx60 (2nd)	1sx60x60 (3rd)	5sx24x30 (3rd)
#1 src	0.0	0.0459	0.0441	0.0488	0.0463
#1 dst	0.0	0.0425	0.0312	0.0468	0.0395
#2 src	0.0	0.0085	0.0210	0.0205	0.0213
#2 dst	0.0	0.0115	0.0140	0.0202	0.0204

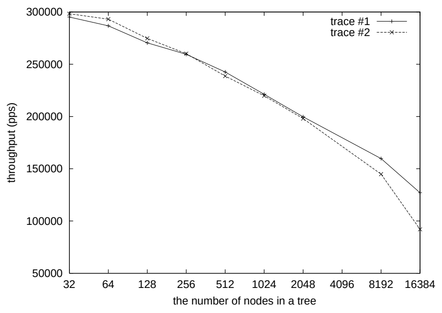


Fig. 10. performance with varying tree size

1-second summaries are first aggregated to 60 60-second summaries, and then, the final summary is created. “5sx24x30” is another third-generation summary. 5-second summaries are first aggregated to 30 120-second summaries, and then, the final summary is created. The results show that the distortion introduced by summary generations is fairly small, which justifies our approach to create derivative summaries for temporal aggregation.

5.2 Performance

For every packet, aguri looks up the matching entry in the 4 trees and manages the LRU lists. When the number of nodes in a tree is N, the lookup operation runs in $O(\lg N)$ time. On the other hand, the cost of managing the LRU list is independent from the numbre of nodes and it runs in $O(1)$ time.

The impact of the number of nodes to the performance is shown in Figure 10. As the number of nodes in a tree increases, the height of the tree becomes longer and the lookup operation becomes more costly. The execution time is measured to produce both source and destination address profiles with the 3600-second traces on a PentiumIII 700MHz/FreeBSD-4.2. The throughput is shown in packets per second (pps); we simply divide the number of packets in the trace by the user time. Thus, this is not an accurate measure but intended to provide a rough idea about the performance.

The result shows that the profiler can process about 250Kpps with 256 nodes, and about 200Kpps with 2048 nodes. The performance is good enough to monitor a 100Mbps link. In the worst case where a 100Mbps link is filled with 64-byte packets, about 190Kpps is required. As a side note, the forwarding performance of a PC router is much lower; about 80Kpps [18].

5.3 Evaluation Summary

We have evaluated the algorithm using backbone packet traces. The leaf node management using a variant of the LRU replacement policy produces decent summaries. The tree size of 128 or 256 works well even for backbone networks, and its performance is good enough. The algorithm is fairly insensitive to variations in networks, the profiling period length, and summary generations.

The packet traces used for the evaluation are backbone data, and as such, the number of included addresses are considerably larger than enterprise networks. The profiler performs much better in enterprise networks.

6 Application for Traffic Control

Once aggregates are identified and profiled, the profile records can be used for traffic control. There are many possible approaches to control aggregates.

In this paper, we propose an aguri three color marker (aguriTCM), which can be used as a component in a Diffserv traffic conditioner [1]. The aguriTCM combines a profiler with a marker. The profiler part is basically the same and the marker part is intended to be used with the Assured Forwarding (AF) Per Hop Behavior (PHB) [12]. The aguriTCM dynamically identifies and profiles aggregates as already described, and then, marks one of three colors to arriving packets. Here, the colors correspond to DS codepoints assigned for the AF drop precedence levels.

Our use of the Diffserv components is basically local to the node, which differs from the DS domain model of the Diffserv architecture. Our primary target is a protective measure against DoS attacks, and therefore, it makes sense to place a standalone traffic control node at a protection point.

Another major difference is that Diffserv markers are usually configured with traffic profile parameters (e.g., committed target rate) [13], whereas the aguriTCM does not have parameters to specify traffic profiles but automatically adapts to traffic. Again, neither class configuration nor classifier rule is needed for this mechanism.

6.1 aguriTCM

Figure 11 illustrates the traffic control model. Arriving packets are marked by the aguriTCM on the input interface, and preferentially discarded by the AF PHB on the output interface. We use the RIO dropper [10] for the AF PHB.

The aguriTCM degrades the drop precedence level of packets for aggregates whose volume is more than the fairshare. Under long-term congestion, the RIO discards packets according to the drop precedence level assigned to the packet.

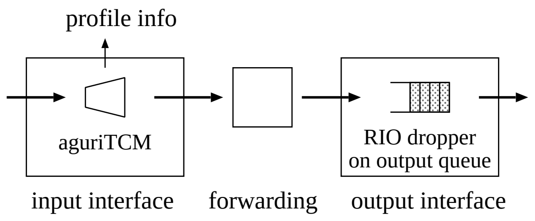


Fig. 11. traffic control model

One difference in the profiler mechanism is that, at the end of a profiling period, the counters of aggregates are not reset to zero but they are aged. The aging method avoids inaccuracy in the begining of a profiling period and smooths out marking probability.

When the counter value is c at the end of a period, the new value becomes ωc, here ω is the weight for aging. We use 0.5 for ω so that the counters are simply halved. The initial counter value for a period is saved in each node so that the profiler reports the period count by subtracting the initial value.

To find the corresponding aggregate for marking, the aguriTCM first checks whether the entry exists in the previous summary. If the saved initial counter is zero, the entry was not in the previous summary. Then, the aguriTCM goes up the tree until it encounters an ancestor with a positive initial counter, and this node is used for marking.

The aguriTCM stochastically demotes the drop precedence of packets if an aggregate uses more than the fairshare. The fairshare is derived from the number of aggregates in the previous summary. When the counter value of an entry is c and the number of aggregates in the previous summary is n, fairshare f is computed as the total count divided by n.

$$f = \frac{\sum c}{n}$$

To demote packets exceeding fairshare f, demotion probability p is computed as

$$p = \begin{cases} \frac{c-f}{c} & \text{if } (c > f) \\ 0 & \text{otherwise} \end{cases}$$

The aguriTCM computes p twice, p_{src} and p_{dst}, independently from the source address and the destination address. An arriving packet is initially considered green. The packet is demoted to red if it is marked by both criteria, and to yellow if it is marked by either criterion. In other words, the packet is marked to red with probability $min(p_{src}, p_{dst})$, and to yellow with probability $|p_{src} - p_{dst}|$.

6.2 Implementation

We have implemented the aguriTCM on the ALTQ framework [5, 7] as a Diffserv traffic conditioner component. ALTQ already implements the RIO dropper that

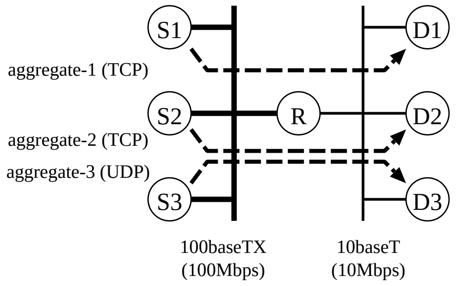


Fig. 12. test configuration with 3 aggregates

supports 3 drop precedence levels. In the current prototype, the aguriTCM always marks packets since packets are dropped only when RIO detects long-term congestion. It could be changed to turn on marking only when the RIO dropper is actively dropping packets to avoid unnecessary marking.

In the prototype, the source and destination address trees are global within the router and shared by multiple instances of aguriTCMs. The profile is produced from all the active aguriTCMs. It allows us to control outgoing packets arriving from different interfaces. Although it is effective only when aggregates share either the incoming interface or the outgoing interface, it covers the majority of the situations requiring the aguriTCM where a router has a single bottleneck or a single fat up-link. If there is less correlation among traffic from different interfaces, it would be better to assign an independent aguriTCM for each interface.

For network monitoring, the aguriTCM writes summaries to a buffer at the end of each profiling period if there is a listener for the aguriTCM device interface. The aguri program in the user space reads the binary summaries through the device interface and produces derivative summaries.

6.3 Preliminary Test Results

The aguriTCM is tested with 7 PCs in a simple configuration shown in Figure 12. 3 senders on the left are on a half-duplex 100baseTX (100Mbps), and 3 receivers on the right are on a half-duplex 10baseT (10Mbps). The aguriTCM and the RIO dropper are implemented on the router in the middle. The aguriTCM is configured with 1% aggregation threshold, 256 nodes per tree, and 5-second profiling period.

3 aggregates are generated in the tests. Both aggregate-1 and aggregate-2 consist of 4 parallel TCP sessions. Aggregate-3 is a single UDP stream sent at a constant rate of 10Mbps. Aggregate-1 starts at time 0 and aggregate-2 starts at time 10. Aggregate-3 is invoked from time 40 to time 70.

The behaviors of the aggregates are compared with and without traffic control. Figure 13 shows the original behavior and Figure 14 shows the effects of

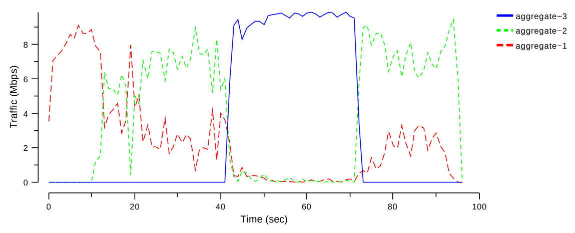


Fig. 13. throughput of aggregates without traffic control

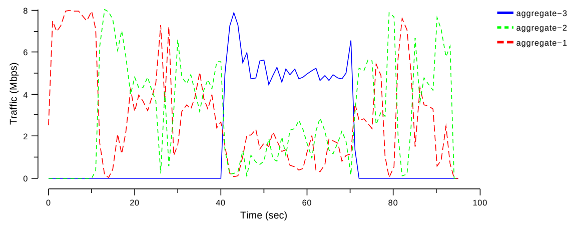


Fig. 14. throughput of aggregates with traffic control

the traffic control. The throughput is measured on the 10baseT link and plotted every second. The plots illustrates (1) resilience in the face of misbehaving flows and (2) fairness among aggregates.

In Figure 13, the UDP forces the TCPs to back off and steals the entire link capacity. On the other hand, in Figure 14, the UDP cannot fill the link after the aguriTCM starts raising the drop precedence of the UDP packets. This result demonstrates the ability to restrict the bandwidth use of misbehaving flows.

In Figure 13, the bandwidth share of the 2 TCP aggregates is not fair even when aggregate-3 is not active. It is improved in Figure 14 since packets are dropped from the aggregates using more than the fairshare. Although it is observed that the TCP throughputs go up and down in the plot, it would be improved if there are more aggregates or TCP implements better recovery mechanisms such as NewReno and SACK.

Figure 13 also shows the problem of unfairness among TCP sessions. It is common that competing TCPs have unequal bandwidth share due to the differences in various factors such as RTT, TCP implementation, and CPU power or other hardware. Among other things, unfairness by RTT is inherent in the TCP mechanism because a session with smaller RTT opens up the congestion window more quickly. The aguriTCM improves this situation since flows in a prefix-based aggregate are likely to have similar RTT.

This particular case in our test is caused by the different network cards used at $D1$ and $D2$. The network card of $D1$ seems to implement a more conservative collision recovery than $D2$. As a result, the TCPs in aggregate-1 experience ACK compression on the reverse path and frequently stall for short periods. If we use the same network cards for $D1$ and $D2$, their share becomes equal.

7 Conclusion

We were in need of an adaptive traffic profiler to track long-term trend and to discover problems in our backbone network, and have developed a tool called aguri. Aguri adapts itself to spatial traffic distribution by aggregating small volume flows into aggregates, and achieves temporal aggregation by creating a summary of summaries applying the same algorithm to its outputs. We have been monitoring our network using aguri since February 2001, and found it useful for network operation.

We have also presented a technique to combine an aggregation-based traffic profiler with a preferential packet dropping mechanism in order to protect the network from DDoS attacks and to provide rough fairness among aggregates. The preliminary test results on our prototype look promising but further investigation and parameter tuning are needed.

The implementation of aguri along with the related tools and other information is available from http://www.csl.sony.co.jp/~kjc/software.html.

References

1. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, Internet Engineering Task Force, December 1998.
2. N. Brownlee. Traffic flow measurement: Experiences with NeTraMet. Request for Comments 2123, Internet Engineering Task Force, March 1997.
3. N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. Request for Comments 2722, Internet Engineering Task Force, October 1999.
4. Kenjiro Cho. Tele Traffic Tapper. http://www.csl.sony.co.jp/~kjc/software.html, 1996.
5. Kenjiro Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In *USENIX 1998 Annual Technical Conference*, pages 247–258, June 1998.
6. Kenjiro Cho. Flow-valve: Embedding a safety-valve in red. In *Global Internet Symposium, Globecom*, pages 1753–1762, December 1999.
7. Kenjiro Cho. *The Design and Implementation of the ALTQ Traffic Management System*. PhD thesis, Keio University, January 2001.
8. Kenjiro Cho, Koshiro Mitsuya, and Akira Kato. Traffic data repository at the WIDE project. In *USENIX 2000 Annual Technical Conference: FREENIX Track*, pages 263–270, June 2000.
9. Kimberly C. Claffy, Hans-Werner Braun, and George C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.
10. D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

11. Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transaction on Networking*, 7(4):458–472, August 1999.
12. J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, Internet Engineering Task Force, June 1999.
13. J. Heinanen and R. Guerin. A two rate three color marker. RFC 2698, Internet Engineering Task Force, September 1999.
14. V. Jacobson, C. Leres, and S. McCanne. tcpdump. ftp://ftp.ee.lbl.gov/, 1989.
15. V. Jacobson, C. Leres, and S. McCanne. libpcap. ftp://ftp.ee.lbl.gov/, 1994.
16. Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and K. Claffy. The architecture of the CoralReef internet traffic monitoring software suite. In *PAM 2001*, Amsterdam, The Netherlands, April 2001.
17. Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *draft paper*, February 2001.
18. Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The Click moduler router. In *Proceedings of SOSP'99*, pages 217–231, Kiawah Island Resort, SC, December 1999.
19. Tobias Oetiker. RRDtool: Round Robin Database Tool. http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool/.
20. Tobias Oetiker. MRTG: The multi router traffic grapher. In *USENIX LISA Conference*, pages 141–147, Boston, MA, December 1998.
21. Dave Ponka. FlowScan: A network traffic flow reporting and visualization tool. In *USENIX LISA Conference*, New Orleans, LA, December 2000.
22. John T. Robinson and Murthy V. Devarakonda. Data cache management using frequency-based replacement. In *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134–142, May 1990.
23. Keith Sklower. A tree-based packet routing table for berkeley UNIX. In *USENIX Winter Conference*, Dallas, Texas, January 1991.
24. S. Waldbusser. Remote network monitoring management information base. Request for Comments 1757, Internet Engineering Task Force, February 1995.