Entropy based flow aggregation

Yan Hu, Dah-Ming Chiu, and John C. S. Lui

The Chinese University of Hong Kong
yhu4@ie.cuhk.edu.hk, dmchiu@ie.cuhk.edu.hk, cslui@cse.cuhk.edu.hk

Abstract. Flow measurement evolved into the primary method for measuring the composition of Internet traffic. Cisco's NetFlow is a widely deployed flow measurement solution that uses a configurable static sampling rate to control processor and memory usage on the router and the amount of reporting flow records generated. But during flooding attacks the memory and network bandwidth consumed by flow records can increase beyond what is available. In this paper, we propose an entropy based flow aggregation algorithm, which not only alleviates the problem in memory and export bandwidth, but also maximizes the accuracy of legitimate flows. Relying on information-theoretic techniques, the algorithm efficiently identifies the clusters of attack flows in real time and aggregates those large number of short attack flows to a few metaflows. Finally, we evaluate our system using real trace files from the Internet.

1 Introduction

Traffic measurement and monitoring are crucial to operating IP networks. Especially, flow-level measurement, such as done in Cisco's NetFlow [1], is widely used for applications such as network planing, traffic profiling, usage-based accounting and security analysis. The ever increasing speeds of transmission links and high volume of traffic present great challenges for flow measurement. For high speed interfaces, the processor and the flow memory of the router can not keep up with the high packet rate. Another problem is that the volume of complete measurements of all traffic requires too much resource, both in the bandwidth required to transmit the flow records to the collector, and the resource needed to store and process the records at the collector.

A standard solution to these problems is to perform packet sampling. Cisco's sampled NetFlow uses a static sampling rate set manually according to the normal traffic volume. But when there is an anomaly such as flooding attacks in the network, the large number of small flows generated may overwhelm the router memory and the export bandwidth to the collector. One countermeasure to this problem is performing adaptive sampling, as is done in Adaptive NetFlow [2]. This algorithm guarantees a stable flow cache and export bandwidth even under severe DoS attacks. But its sampling rate could decrease to a very low level, resulting in poor overall accuracy in per flow counting including legitimate flows. Besides sampling, another method of data reduction is to do flow aggregation. Cisco implements router-based flow aggregation, which summarizes NetFlow data on the router before the data is exported to the collector.

Adaptive flow aggregation [3] has recently been proposed to allow the flow monitoring systems to cope with sudden increases in the number of flows caused by security attacks. Flows of security attacks usually have some common patterns and form conspicuous traffic clusters. The algorithm identifies these traffic clusters in real-time and aggregates these large number of short flows into a few metaflows. Compared to adaptive sampling, this solution not only alleviates the problem in memory and export bandwidth, but also guarantees the accuracy of other legitimate flows. Without any predefined schemes or rules, identifying appropriate clusters and performing aggregation in real-time are not simple tasks. In this paper, we propose an entropy based flow aggregation algorithm. Based on the concept of entropy from information theory, we use the parameter of APP to indicate the priority of clusters to be aggregated. An efficient algorithm is used to identify those clusters as well as pick out some large normal flows belonging to the identified clusters.

2 Entropy based flow aggregation algorithm

We first provide a short description of our flow monitoring system, and more details can be found in [3]. The system collects network traffic data or just reads trace file and emits it as NetFlow flow records towards the specified collector, just as Cisco's NetFlow does. When the memory usage reaches a maximum value that the system allows, the system will perform flow aggregation. Using a new data structure called two-dimensional hash table, all flows with the same srcIP or dstIP will be put in one list. We also maintain a top list for srcIP and dstIP, which records the IP addresses with the most number of flows. The objectives of the old adaptive flow aggregation algorithm in [3] are, first, flow entries freed during aggregating these clusters can satisfy the memory's requirement, second, the level of these identified clusters should be as high as possible. After the algorithm identifies the desired clusters, the system merges all flows in one cluster to one metaflow. In the rest part of this section, we will describe the newly proposed entropy based flow aggregation algorithm.

2.1 Aggregation Priority Parameter (APP)

We define a *cluster* as a set of flows with the same values in one or several of the four keys, srcIP, dstIP, srcPort (plus protocol), dstPort (plus protocol), which are typically used to define a flow. We focus on clusters with a fixed srcIP/dstIP because almost all abnormal traffic has either a fixed source or destination IP address. For example, packets of DoS attacks often have the same dstIP, while packets of worm spreading usually have the same srcIP. In addition, some attacks have other fixed keys. For example, in Figure 1, all flows from one host form cluster A, while worm spreading flows from this host form cluster B. We define the biggest cluster which only has the fixed srcIP/destIP $L1$ (level 1) cluster such as cluster A, define the clusters which have fixed value in two (three) dimensions $L2$ ($L3$) cluster such as cluster B (D). If we choose the higher level cluster B instead of cluster A to do aggregation, we can keep more information (srcIP and dstPort).

Besides fixed values in one or several keys, other properties of the clusters containing attack traffic include: first, the number of flows in the clusters is usually large enough to become a flooding attack; second, the size of the flows (number of packets or bytes) is often much smaller than normal flows; third, some keys other than the fixed value, such as srcIP in DoS attack traffic, dstIP in worm spreading traffic and dstPort in port scan traffic, are often randomly or uniformly distributed. In addition, if there are several big flows in the identified cluster, we would pick them out from the identified cluster and do aggregation on the rest flows, because the big flows may be normal flows mixed with attack flows. Then now the concept of the cluster is extended to the remaining flows in the original cluster. For example, in Figure 1, large flow C and $L3$ cluster D are picked out from $L2$ cluster B, the remaining flows in cluster B can also be considered as a cluster $F := B - C - D$.

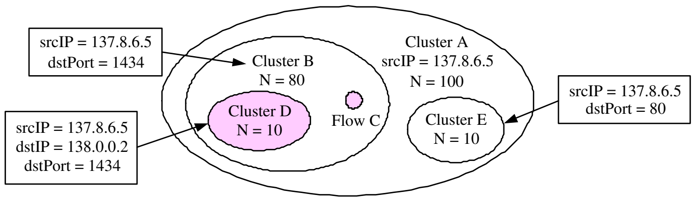

Fig. 1. examples of clusters

We call those dimensions which have more than one value (e.g. dstIP and srcPort of cluster B) as *random dimensions*, and those dimensions which have one fixed value (e.g. srcIP and dstPort of cluster B) as *fixed dimensions*. When all flows in a cluster are merged to one metaflow, the information of its fixed dimensions will be kept, while the information of its random dimensions will be lost. Intuitively, among all clusters in Figure 1, we should choose cluster B to do aggregation for the following reasons. First, cluster B contains enough flows compared with cluster D. Second, the degree of randomness of its random dimensions is large compared with cluster A. Third, cluster B contains one more dimension of information (dstPort) than cluster A. After picking out the big flow C and $L3$ cluster D from cluster B, the one we finally choose to do aggregation is cluster F. To characterize those properties of cluster F, we propose a metric named Aggregation Priority Parameter (APP) based on the concept of entropy.

Let random variable X be one of the four dimensions (srcIP, dstIP, srcPort and dstPort). The probability distribution on X is given by $p\ (x_i) = m_i/m$, where m is the total number of traffic observed, and m_i is the number of traffic that take the value x_i. We calculate number of traffic in terms of bytes instead of flows because we need to differentiate between big flows and small flows. Entropy of one dimension is a good indicator of its degree of uncertainty or randomness.

It tells us if there are some significant values that stand out from others or all values are uniformly distributed.

APP of a cluster is defined as the minimum of the entropy of its random dimensions. The larger the *APP* of a cluster, the higher priority this cluster would be aggregated because it characterizes those properties we want. Firstly, high *APP* means the number of flows in this cluster is large. Secondly, *APP* being large means none of those random dimensions has any significant value. In Figure 1, *APP* of cluster A is small than cluster B because it has a significant value 1434 in the dimension of dstPort. Third, the cluster has no flow much larger than other flows because we compute entropy in terms of number of bytes.

2.2 Algorithm description

Using the data structure and top list in our flow monitoring system, now we have some big $L1$ clusters with fixed srcIP or dstIP. What our entropy based flow aggregation algorithm should do is that, for every $L1$ cluster, find out its sub-clusters which have the largest *APP*. These identified sub-clusters could not be subordinative to or overlap with each other. Among them, the cluster whose *APP* is the largest will be chosen. However, if several sub-clusters do not contain or overlap with each other (we call them *distinct cluster*) and have similar *APP*, they would all be identified.

Algorithm 1 finding out sub-clusters

```
Input:   Cluster C; random dimensions: RD
Output:  sub-clusters of high $APP$: CList
FindSubCluster(C, RD) {
1.  for every dimension d in RD
2.    $C_m[d]$ = GetMaxEntropySubset (dimension d of cluster C);
3.  end for
4.  $C_P$ = MaxAPPCluster (C, $C_m[d]$);
5.  for every dimension d in RD
6.    for every $S_i$ whose number of flows greater than $f_r$
7.      CList[d] = CList[d] + FindSubCluster($S_i$, RD-d);
8.    end for
9.  end for
10. CList = MaxAPPDistinctCluster ($C_P$, CList[d]);
11. return CList;
}
```

We use Algorithm 1 to get the sub-clusters with the largest *APP*. The input to the function is a cluster C with random dimensions RD. The output of the function is a list of its sub-clusters with the largest *APP*. First, for each random dimension d of cluster C, the function finds out its maximum entropy subset $C_m[d]$. Maximum entropy subset is a subset of a cluster with the maximum entropy among all subsets of this cluster. For example, we assume all flows in cluster B have the same size except flow C, whose size is 10 times of that of other flows. Cluster D has 10 flows with the same dstIP. Then for the dimension of dstIP, the entropy of cluster B is 5.73, while the entropy of cluster F is 6.11, which is the maximum entropy of all subsets of cluster B. We use an efficient algorithm to find the maximum entropy subset of dimension d of a cluster C. For more details, please refer to technical report version of this paper [4].

Cluster C and $C_m[d]$ are not distinct clusters, the one with the largest APP (C_P) is chosen as a candidate for the desired sub-clusters. The fact is there may be some sub-clusters other than those maximum entropy subsets that have larger APP. They may be picked out because their sizes are large enough, or their sizes may be so small that they are subsumed in the maximum entropy subsets. So we need to recheck those sub-clusters (S_i) whose number of flows is large enough to have a large APP, as described in line 5 to 9 in the function. The last step as stated in line 10 is to choose several distinct sub-clusters from these candidates including C_P and $CList[d]$.

After the algorithm identifies the desired clusters, the system merges all flows in one cluster to one metaflow. The number of packets/bytes is the sum of packets/bytes of all aggregated flows. When new incoming packets do not belong to any active flow but belong to one metaflow, the number of packets/bytes of this metaflow will be updated. So we can get accurate packet and byte counts for the metaflow. The number of flows of the metaflow can not be counted directly. We use the multiresolution bitmap algorithm proposed in [5] to estimate it.

3 Experimental evaluation

In this section, we use experiments to evaluate our *entropy based flow aggregation* algorithm, and compare its performance with *adaptive flow aggregation* algorithm in [3] and *adaptive NetFlow* in [2]. Under normal conditions, our system works just as basic NetFlow does. When the memory usage exceeds a predefined maximum memory, our system will perform flow aggregation, while *adaptive NetFlow* will decrease the sampling rate. Without memory constraint, *basic Netflow* can get accurate result for any flow aggregate. We use *basic Netflow* as the benchmark, and compare the performance of the three solutions. The data set we use is a 5 minute trace of the traffic on an OC48 IP backbone link, provided by Caida. We artificially generate a "DDoS" data set which simulates a DDoS attack on a single victim, and mix it with the OC48 data set.

The comparison on memory usage, export bandwidth, CPU run time and more details about the experiment can be found in [4]. Here we give out some examples of the relative error of these three schemes, as shown in Table 1. These hosts are chosen from the top dstIPs, and the top 1 is the victim of the DDoS attack. For the number of bytes, both *adaptive flow aggregation* and *entropy based flow aggregation* give out accurate results for all these hosts, while *adaptive NetFlow* affects the accuracy inevitably. Some hosts also have accurate results for the number of flows, which are not affected by the flow aggregation because they do not belong to the identified clusters. The new entropy-based flow aggregation algorithm accurately identifies the cluster of DDoS attack, so only the flow counter to the victim host is affected. However, the old algorithm identifies and aggregates some other clusters (eg. web traffic to host 241.46.185.161), so flow errors of those hosts do not equal to 0.

Table 1. Relative error (%) of destination IP address breakdown

	%	adaptive NetFlow		flow aggregation		entropy-based	
dstIP	of total	byte Err.	flow Err.	byte Err.	flow Err.	byte Err.	flow Err.
162.131.189.129	30.7	0.83	81.14	0.00	12.88	0.00	12.99
162.131.199.254	12.4	0.41	41.53	0.00	0.00	0.00	0.00
162.131.175.235	9.5	0.66	56.18	0.00	0.00	0.00	0.00
241.46.185.161	3.2	0.57	37.23	0.00	0.78	0.00	0.00
241.46.188.127	2.6	0.49	46.65	0.00	0.16	0.00	0.00
0.3.117.37	2.1	2.02	48.17	0.00	0.15	0.00	0.00

4 Conclusion

To overcome NetFlow's problem of overrunning available memory for flow records during abnormal situations, this paper proposes an entropy based flow aggregation algorithm. Based on the concept of entropy from information theory, we use the parameter of APP to indicate the priority of clusters to be aggregated. The algorithm can efficiently identify the clusters containing attack flows as well as pick out some large normal flows belonging to the identified clusters. After identifying these clusters, the system merges flows in the clusters to metaflows, and updates information of the metaflows from new incoming flows belonging to these clusters. The measurements for bytes and packets for the metaflows are completely accurate, and measurements for flows are nearly accurate using the bitmap algorithm. We use experiments on real trace file to evaluate our system and compare it with *adaptive NetFlow* and *adaptive flow aggregation*. The results show that our solution provides better accuracy.

References

1. http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml.
2. Estan, C., Keys, K., Moore, D., Varghese, G.: Building a better netflow. In: Proc. SIGCOMM '04. (2004)
3. Hu, Y., Chiu, D.M., Lui, J.: Adaptive flow aggregation - a new solution for robust flow monitoring under security attacks. In: Proc. NOMS '06. (2006)
4. Hu, Y., Chiu, D.M., Lui, J.: Entropy based flow aggregation: Tech. report (2006) http://personal.ie.cuhk.edu.hk/~yhu4/paper/entropy_tech.pdf.
5. Estan, C., Varghese, G., Fisk, M.: Bitmap algorithms for counting active flows on high speed links. In: Proc. IMC '03. (2003)