Packet Filtering for Aggregate-based Congestion Control

N. Chevrollier
NIST
Gaithersburg, MD 20899

R. E. Van Dyck
NIST
Gaithersburg, MD 20899

Abstract — **We provide a short overview of the problem of congestion control in IP networks, including a discussion of some related work in countering denial-of-service attacks and packet classification. Then, we propose an adaptive packet filtering method for achieving aggregate-based congestion control. The method emphasizes approaches based on unsupervised learning, in combination with congestion detection. Initial simulation results suggest substantial improvements can sometimes be obtained.**

Keywords – **Packet Filtering, Congestion Control, Internet, Clustering.**

I. Introduction

As the Internet becomes larger and more complex, the problem of network management becomes more difficult. Recent work has suggested the creation of a knowledge plane that will provide a distributed control mechanism as an overlay to the existing data network. As a step in this direction, the research discussed here initially focuses on two important and related problems in current IP networks. These networks are vulnerable to denial-of-service (DoS) attacks and flash crowds. In both cases, the amount of traffic at particular servers can be so high as to significantly degrade their performance. Additionally, the traffic in specific links near the servers may be sufficient to adversely affect other network traffic.

One of the main problems in the above scenario is the lack of information about the global (distributed) network state available at each node, thereby limiting its decision making. Some processing can be done locally on the network routers, and it is here that we shall first concentrate. Specifically, if there is congestion, one typically needs to discard packets. Determining which ones is not an easy problem. A useful paradigm is to filter the individual packets traversing the link into some number of classes so that each class can be rate limited. Gupta and McKeown [1] provide an overview of some of the standard packet filtering approaches. Typically, the destination address is the most important parameter, and a one-dimensional classification is often done based on this field.

Mahajan *et al.* [2] point out that the problem of network congestion is neither due to a single flow refusing to use end-to-end congestion control nor is it due to a general increase in overall network load. Instead, one can characterize the increase in traffic as coming from an aggregate of flows. They further suggest that it is useful to base a congestion control algorithm on the detection of these aggregates, as opposed to concentrating on either per-flow control or active queue management. The basic reasoning is that the number of flows may be so large and transient, that it is a waste of system resources to attempt to track each flow separately. Instead, if one can cluster the flows into a smaller number of aggregates, then it is possible to process (*e.g.* queue and rate control) each aggregate separately, depending on the level of congestion of each outgoing link and local *a priori* policies.

Two questions immediately come to mind: (1) should the aggregates be related to specific categories/classes of traffic as opposed to certain destination addresses, and (2) how should the segmentation into aggregates actually be done? Typically [2], the aggregation is done based on the destination address. One potential problem with this approach is that the congestion may be caused by a single server responding to requests from multiple addresses. For example in a flash crowd caused by many requests for a video stream, the short request packets may get to the server but the congestion is actually caused by the multiple copies of the video stream headed to many different destinations. Automatic software updates from a common server also fit this scenario. To help answer these questions, we look at other work aimed at preventing denial-of-service attacks and in flow classification.

While neural networks and support vector machines can be used to build powerful classifiers [3], the standard supervised learning methods for training imply the existence of a set of labeled training data. That is, a sequence of input/output pairs where the output is the desired classification. Our concern is that the existence of such training data may be quite limited or non-existent. Consequently, we initially propose using unsupervised statistical learning based on the k-means clustering algorithm [3]. Further extensions using self-organizing maps [4] and independent component analysis [3] are also possible. Recently, there has been some interesting work on using both labeled and unlabeled data for designing classifiers [5] [6], and we also discuss the use of such techniques for this problem.

The outline of the rest of this paper is as follows: Section II describes some related work on dealing with denial-of-service attacks. Section III considers traffic flow classification, and it includes the description of our proposed system and algorithms. Section IV then provides some initial results, and the final section discusses further research directions.

II. Denial-of-Service Attacks

There has been a relatively large amount of work [7] [8] [9] [10] [11] in the related areas of intrusion detection (ID) and countering denial-of-service attacks. It is useful to make the distinction between network-based and host-based intrusion detection; the former is concerned with detecting the presence of intruding packets and packet flows into a network or subnetwork, while the latter detects intrusion into a specific end system. In this work, we are interested in network-based intrusion detection, especially its relationship to DoS attacks and congestion control.

Intrusion detection is often considered as a problem of anomaly detection [1]; this paradigm tries to find a baseline of normal behavior and then determine if the current behavior deviates sufficiently from normal. At a single router, the

[1]An alternative paradigm is one of misuse detection [8], where attacks on a system are modeled as specific patterns. The misuse detector then scans the system for occurrences of these patterns. Since the patterns must be known *a priori*, this viewpoint is more suitable for host-based intrusion detection.

goal is to determine whether or not each network link is in a normal state. If not, then it is possible that an attack is taking place, and the classification problem becomes determining the attack packets so that they can be filtered out. Both rule-based and statistical approaches have been developed in the literature. The latter approach is formulated as a statistical hypothesis testing problem, which can be addressed by a number of methods. Feinstein *et al.* [7] compute the entropy and chi-square statistics of the packets. Alternatively, given a training sequence containing normal and abnormal network behavior, neural networks can be trained to classify the traffic. For instance, Ghosh and Schwartzbard [8] use a conventional multilayer perceptron neural network with backpropagation training, while Zhang *et al.* [9] tested five different types of neural networks including radial-basis functions.

Similarly, Huang and Pullen [10] propose a congestion triggered packet sampling and filtering approach. For a given type of attack condition, they model the packets as coming from a binomial distribution, so they can precompute the confidence intervals around given thresholds. Now if the observed percentage of bad packets is above the top of the confidence interval, then there is a greater than 95% probability that the true number of malicious packet is above the threshold. While the idea of precomputed thresholds is generally useful, they do not specify, in general, how to determine the alarm conditions; they give a couple of examples, including the case of looking for spoofed source addresses that do not come from the incoming ISP.

III. Flow Classification and Packet Filtering

One definition of an IP traffic flow is a sequence of packets that share some common properties. For example, packets with the same destination address prefix constitute a flow to a particular area in a network. A flow of finer granularity would be IP packets with the same source and destination addresses and ports (IP address/port quadruple). One could consider keeping track of all of these flows going through a single router (the all service flow classification scheme mentioned below does this). Aside from the serious problem of scale, there are other problems, especially for DoS attacks. For example, the source address can often be spoofed, and the use of encryption at the IP layer or network address translation may make the port numbers unreliable also.

Ilvesmäki *et al.* [12] studied the problem of determining if flows should be routed at the IP layer (layer 3) or switched at layer 2. They used the IP address/port quadruples in conjunction with a time-out value to determine flows, and they considered four flow classification schemes: (1) all service flow classification, (2) packet-count based flow classification, (3) selected service flow classification, and (4) learning vector quantization (LVQ). They specifically proposed the last method and showed that it can provide better results than the three previous ones. While this work is of interest, there is a potential scaling problem since they are not aggregating the flows.

In terms of aggregating the flows, one goal could be to classify the network traffic into a relatively small number (5-20) of application classes. Ideally, this could be treated as a supervised learning problem, where the packet headers are the input vectors and the application class is the output. However, the implicit assumption is that there is a reasonable training sequence of input-output pairs, which may not be the case. This concern has not appeared to stop many researchers from adopting this approach. Alternatively, one can consider the

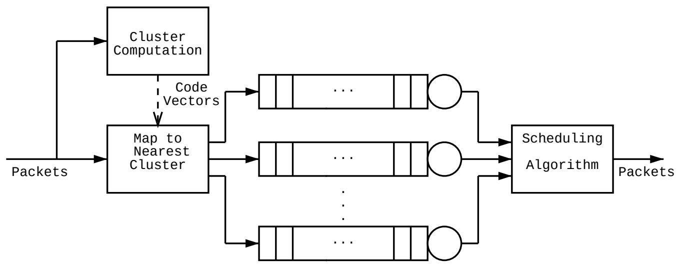

Figure 1: System block diagram: unsupervised learning and queueing.

unsupervised learning problem of segmenting the packets into a number of clusters, with each cluster processed differently.

In both the supervised and unsupervised cases, the aim is to be able to filter (that is, either classify or cluster followed by separate queueing and rate control) each packet that enters the router. Trivedi *et al.* [13] take a somewhat different approach where they build a histogram of packet lengths for 8 different application classes. A multilayer perceptron neural network was then trained on these fifty dimensional vectors. While the results appear quite promising, there is a significant issue here. When testing the classifier, it is assumed that all the packets used to make up an input histogram come from the same application class. Yet in a real network, the packets entering the router will have many different classes interspersed. Thus, while the work is interesting, it is presently not suitable for implementation.

In contrast to the work just mentioned, Singh *et al.* [14] propose a state-of-the-art packet classification algorithm. Like those discussed in [1], it uses a type of decision tree architecture to provide fast classification. Moreover, it argues that content addressable memories may be too limited for general packet filtering and that algorithmic approaches may be necessary. However, it is somewhat conventional in the sense that it takes as input a set of rules, and then it develops the classifier based on them. Again, one can consider this as a type of supervised learning in that the rules are given. In other words, this method does not try to determine rules based on the data.

Packet Filtering Algorithm

Fig. 1 shows the block diagram of our system. On each output line card of a router, instead of a single output queue there are now a number of queues. Packets entering the system are compared to a relatively small number (*e.g.* 8, 16, or 32) of code vectors, and each packet is mapped to the cluster whose code vector is the best match. The packets in each cluster are then put into a first-in-first-out queue. Presently, the scheduling algorithm uses (work conserving) round robin.

The cluster design is done using the k-means algorithm [3], with training operating in a batch processing mode. Specifically, the input packet stream is segmented into 5,000 packet epochs. During the first epoch, only a single queue is used, while the packets are also collected and used in the code vector design. The resulting code vectors are then used for clustering during the next two epochs. In general, training occurs during epochs $2n$, for integer n, and the code vectors are used for the next two epochs. To initialize the code vectors in the base algorithm, a splitting procedure is used; first, the centroid of the training samples (packets) $\mathbf{c}$ is found, and then it is perturbed by a small vector ϵ to get code vectors $\mathbf{c}+\epsilon$ and $\mathbf{c}-\epsilon$. The k-means algorithm then gives two optimized code vectors $\mathbf{c}_1$ and $\mathbf{c}_2$. The process is repeated to generate four code vectors, then eight, then sixteen *etc.* The complete base algorithm to create 8 codevectors is described below.

```
    Base Algorithm

  Compute centroid of training samples
  Split centroid into 2 code vectors
  Run k-means algorithm to optimize 2 code vectors
  Split into 4 code vectors
  Run k-means algorithm to optimize 4 code vectors
  Split into 8 code vectors
  Run k-means algorithm to optimize 8 code vectors
```

Since the algorithm uses Euclidean distance

$$\|\mathbf{x}-\mathbf{y}\|_2 = (\sum_{j=1}^{p}(x_j-y_j)^2)^{1/2}, \quad (1)$$

for p dimensional vectors $\mathbf{x}$ and $\mathbf{y}$, there is the issue of scaling the different dimensions. This is accomplished by normalizing each dimension by its maximum value in the epoch.

In the work of Mahajan *et al.* [2], as well as in our algorithm, there is an implicit assumption that roughly equal numbers of packets go to each queue. This assumption is more likely to be valid in [2], because they are using ranges of destination addresses for the aggregation; one could presumably measure the traffic over "normal" operating conditions and then build the rules that determine the address ranges. Since our algorithm is adaptively creating clusters, there is no guarantee that the clusters will have approximately equal sizes. The effect on performance is discussed in Section IV.

Thus, to ensure that no legitimate traffic is dropped under normal conditions, the two clusters with the largest numbers of training packets are split when there is no congestion. When there is congestion, this splitting procedure is not used, and the initial code vectors are those from the previous epoch. This leads to the congestion monitoring algorithm described below. The basic idea is that the bad traffic will (hopefully) tend to cluster into one or two of the queues, so that the rate limiting is more effective.

```
    Congestion Monitoring Algorithm

if ( average traffic load < threshold) ) then
  Base Algorithm
  Split 2 largest code vectors
  Run k-means to optimize 10 code vectors
else
  Use 10 code vectors from previous epoch
  Run k-means to optimize 10 code vectors
end if
```

IV. Preliminary Results

First let us consider the problem of classifying packets into the following four application classes: (1) ftp, (2) smtp, (3) http, and (4) other/bad. The data is taken from an OPNET simulation where packets traverse a particular link in a small network (11 subnets for a total of 500 end-user devices, 3 http servers, one ftp server and one mail server). The topology of the network used in the simulation is shown in Fig. 2. The legitimate traffic is a combination of http, smtp and ftp packets arranged as follows: the major contribution comes from http which represents 90% of the total traffic in bits, then smtp and ftp, which contribute 5% each. Thus, each application is characterized with a pre-defined OPNET profile and attributes. Based on the attack model developed in [15], 3 attackers from 3 different subnets are trying to overflow the bottleneck link of the network by sending UDP packets from spoofed IP addresses. Over 10 minutes of simulation, the network is under attack for about 4 minutes. Each packet header is represented by a six-dimensional vector that contains: (1) source address, (2) destination address, (3) protocol, (4) packet size, (5) source port, and (6) destination port.

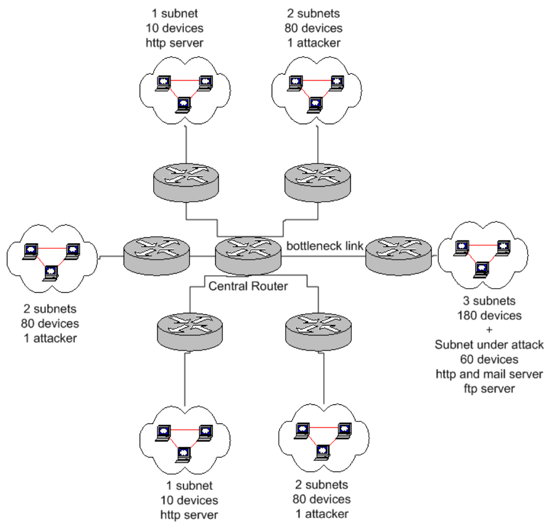

Figure 2: Simple test network topology.

Base Algorithm

In this subsection, we discuss the performance of the base algorithm. We start with a single simulation, where 120,475 packets are sent through a single queue into the output link of the router. 20,001 of these packets are "bad", in the sense that they are coming from the three attackers. During periods of no congestion, the link is about 65 percent full on average, while during the congested period, the offered load is somewhat over 100 percent of capacity. Table 1(A) shows the total number of packets (first row) and the number of packets dropped (second row) for each application type for the original system where there is a single output queue. The total number of good packets (*i.e.* ftp, smtp, http) dropped is 4,558, while 1,907 bad packets are dropped.

Next, we use the base clustering algorithm and multiple output queues, as discussed in Section III. The length of each of the eight queues is one-eighth of the length of the single queue, so that there is no net increase in memory usage. Table 1(B) shows the total packets (top 8 rows) and packets dropped (bottom 8 rows) when there are 8 clusters and queues. Now, a total of 1,831 good packets are dropped, along with 3,034 bad packets for this simulation run. Most of the good dropped packets come from the http traffic class, primarily because the clustering algorithm is sufficiently good

Cluster	ftp	smtp	http	other/bad
1	3,108	6,514	90,852	20,001
1	93	204	4,261	1,907
1	0	0	10,759	4,834
2	1,227	1,382	3,242	0
3	0	0	12,253	2,718
4	573	1,610	15,262	0
5	0	0	7,968	7,791
6	674	1,805	14,783	0
7	0	0	12,401	4,658
8	477	1,247	13,698	0
1	0	0	232	0
2	7	15	153	0
3	0	0	969	37
4	0	0	0	0
5	0	0	4	1,116
6	1	21	372	0
7	0	0	0	1,881
8	0	0	57	0

Table 1: $\boxed{\frac{(A)}{(B)}}$ Total packets and packets dropped for (A) 1 cluster/queue and (B) 8 clusters/queues with round robin scheduling.

to form large clusters (*e.g.* 15,262 packets). Still, the total number of good packets dropped is decreased.

Congestion Monitoring Algorithm

The Congestion Monitoring Algorithm, however, provides greatly improved performance without requiring additional memory. We set the congestion threshold at 80 percent of capacity, and the ten queues have lengths of one tenth the single queue. In our attack model, the bad packets come from three different sources that start transmitting at slightly different times. As the link initially becomes congested, the attack packets are clustered into the existing ten clusters. Once the next training epoch is reached, the system designs ten new code vectors, using the ones from the previous epoch as the initial locations. Table 2 shows the distribution of packets among the ten clusters/queues for a single simulation run. One can see that the attack traffic has enough self-similarity so that it ends up almost in a single cluster, whence it is easy to rate limit. As a result, no good packets are dropped, while 4,084 bad packets are dropped, compared to 4,558 good packets and 1,907 bad packets for a single queue system. The main reason for this difference is that the good traffic uses TCP, while the attacks use UDP. Examining the code vectors created before and after congestion shows that the protocol type is one of the main features used in the clustering process.

While the results in Table 2 are quite promising, they are obtained for an attack of a particular intensity. It is also interesting to evaluate the system performance as this load varies. Since most of the data is http traffic, we focus on it. Each data point in the figures below is an average of five different simulations, each using a different random seed. Fig. 3 shows the percentage of http packets dropped versus the total number of bad packets (intensity of the attack) for the one queue system, eight queue system with the base algorithm, and ten

Cluster	ftp	smtp	http	other/bad
1	0	0	12,665	985
2	612	536	3,003	0
3	510	528	3,532	0
4	138	509	9,905	0
5	195	911	10,869	0
6	0	0	4,892	19,016
7	290	711	7,088	0
8	346	1,137	11,675	0
9	16	65	14,244	0
10	512	1,089	9,383	0

Table 2: Total packets for 10 clusters/queues with round robin scheduling.

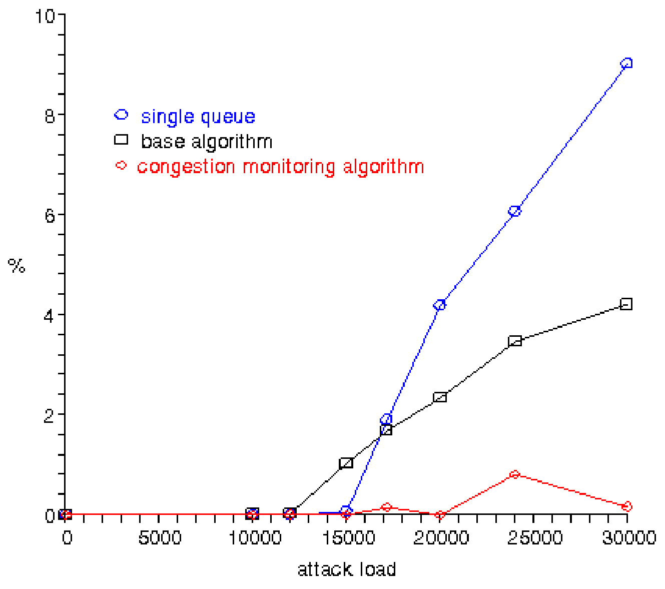

Figure 3: Average percentage of http packets dropped as a function of load.

queue system with the congestion monitoring algorithm. For low level attacks, say below 12,000 total UDP packets, the performance of the three systems is very similar. There is sufficient bandwidth available, and very few packets are dropped. As the the attack intensity increases, the base algorithm is substantially better the the one queue system, while the congestion monitoring algorithm is even more effective. Fig. 4 shows the corresponding percentage of bad packets lost also versus the load; this leads to similar conclusions.

Fig. 5 shows the TCP delay of the http traffic collected at the http server as a function of the load for the single queue system, the base algorithm used with 8 queues, and the congestion monitoring algorithm used with 10 queues. This delay is an average over the simulation time, and it is somewhat similar for the single queue system and the 8 queue base algorithm. The performance increase is much better when using the congestion monitoring algorithm along with 10 queues. The results are even more significant during the congestion period. The TCP delay for the single queue system is on the order of 2 to 5 seconds, compared to about a hundred milliseconds for the 10 queues system.

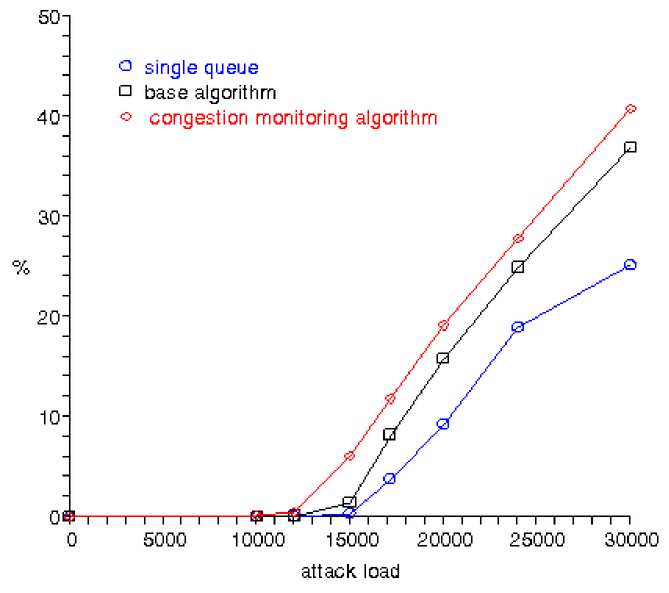

Figure 4: Average percentage of bad packets dropped as a function of load.

If one can afford a system with 18 queues (of length one-eighteen of the single queue), then it may be possible to lower the packet dropping rate even more. For the fairly simple attacks used in this work, the improvement is not very substantial. Given that the computational complexity increases with the number of code vectors, it is not clear that it is worthwhile to use this many queues. However, the situation may be different for links containing more complicated flows.

V. Conclusions and Extensions

Initial simulation results suggest that packet filtering based on unsupervised learning can be effective in combating some denial-of-service attacks. Much investigation is required to determine optimal values for the number of code vectors, the congestion threshold, the epoch length, and the packet header fields to use. It should be noted that only the comparison of the packet headers to the code vectors must be done at line speed; the cluster computation can be done in the background. Still, methods to reduce the computational complexity will be studied. The immediate extension of the above work is to examine more sophisticated traffic and attack scenarios, and then to develop attacks that will attempt to defeat the packet filtering algorithm. Additional work will consider the use of semi-supervised learning [5] [6] in conjunction with weighted queueing, based on *a priori* estimates of typical traffic.

Another direction for future work is to propagate packet classification information to immediate upstream routers so that rate limiting can be more effectively applied. The intention is that by pushing back the processing closer to the sources, less of the useful network traffic is discarded [2]. To accomplish push-back, information will need to be propagated both up and down the source tree from the congested router. Papadopoulos *et al.* [16] propose a distributed "watchdog architecture" to allow this communication, in conjunction with a relatively simple spectral analysis to detect an attack. In addition to improved detection/classification techniques, we claim that the incomplete network state information suggests use of "soft information" sent to the upstream routers.

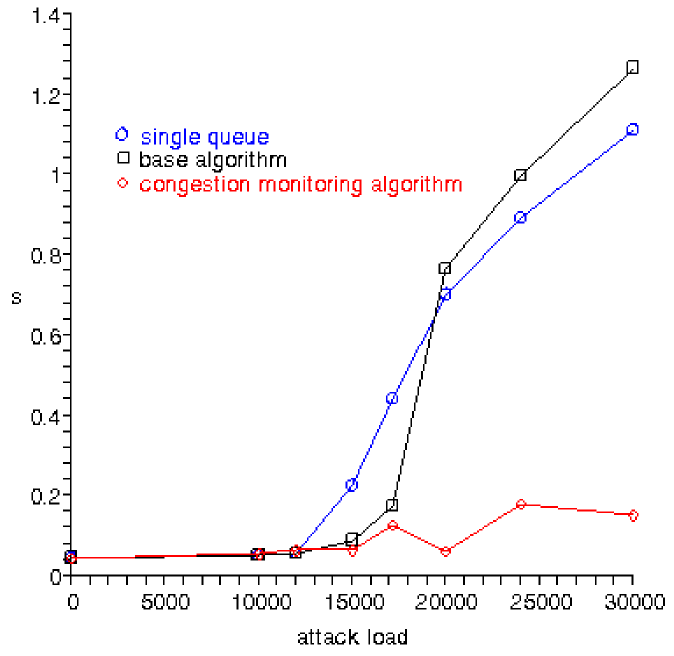

Figure 5: Average TCP delay in seconds as a function of load.

References

[1] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, pp. 24-32, Mar./Apr. 2001.

[2] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *SIGCOMM Computer Communications Review*, Vol. 32, Num. 3, July 2002.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer Verlag, 2001.

[4] T. Kohonen, *Self-Organizing Maps*, Springer Verlag, 3rd Edition, 2000.

[5] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Machine Learning*, pp. 1-34, 2000.

[6] D. J. Miller and J. Browning, "A mixture model and EM-based algorithm for class discovery, robust classification, and outlier rejection in mixed labeled/unlabeled data sets," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 25, No. 11, pp. 1468-1483, Nov. 2003.

[7] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to DDoS attack detection and response," *Proc. DARPA Information Survivability Conference and Expositions (DISCEX III)*, Washington, D.C., Apr. 22-24, 2003.

[8] A. K. Ghosh and A. Schwartzbard, "A study in using neural networks for anomaly and misuse detection," *Proc. 8th USENIX Security Symposium*, Washington, D.C., Aug. 23-26, 1999.

[9] Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, "HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification," *Proc. 2001 IEEE Workshop on Information Assurance and Security*, U.S. Military Academy, West Point, NY, June 5-6, 2001.

[10] Y. Huang and J. M. Pullen, "Countering denial-of-service attacks using congestion triggered packet sampling and filtering," *Tenth Int. Conf. on Computer Communications and Networks*, pp. 490-494, 2001.

[11] K. Park and H. Lee, "On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack," *Proc. IEEE Infocom*, pp. 338-347, vol. 1, 2001.

[12] M. Ilvesmäki, M. Luoma, and R. Kantola, "Flow classification schemes in traffic-based multilayer IP switching – comparison between conventional and neural approach," *Computer Communications*, Vol. 21, pp. 1184-1194, 1998.

[13] C. Trivedi, M.-Y. Chow, A. Nilsson, and H. J. Trussell, "Classification of Internet traffic using artificial neural networks," Technical Report, North Carolina State University, 2003.

[14] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," *Proc. SIGCOMM'03*, Karlsruhe, Germany, Aug. 25-29, 2003.

[15] W. J. Blackert, D. M. Gregg, A. K. Castner, E. M. Kyle, A. L. Hon, and A. M. Jokerst, "Analyzing interaction between distributed denial of service attacks and mitigation technologies," *Proc. DISCEX III*, Washington, D.C., April 22-24, 2003.

[16] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan, "COSSACK: coordinated suppression of simultaneous attacks," *Proc. DISCEX III*, Washington, D.C., Apr. 22-24, 2003.