Towards NIC-based Intrusion Detection

M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, D. Panda

Department of Computer and Information Science, The Ohio State University
Contact Email: {otey,srini}@cis.ohio-state.edu

ABSTRACT

We present and evaluate a NIC-based network intrusion detection system. Intrusion detection at the NIC makes the system potentially tamper-proof and is naturally extensible to work in a distributed setting. Simple anomaly detection and signature detection based models have been implemented on the NIC firmware, which has its own processor and memory. We empirically evaluate such systems from the perspective of quality and performance (bandwidth of acceptable messages) under varying conditions of host load. The preliminary results we obtain are very encouraging and lead us to believe that such NIC-based security schemes could very well be a crucial part of next generation network security systems.

1. INTRODUCTION

In today's information age, where nearly every organization is dependent on the Internet to survive, it is imperative to guarantee the privacy and security of the information being exchanged. This issue has been brought further into the foreground by the recent thrust toward cyber-space security and the almost omnipresent deployment of network intrusion detection systems. The goal of an intrusion detection system is to detect inappropriate, incorrect, and unusual activity on a network or on the hosts belonging to a local network by monitoring network activity.

How do we know if an attack has occurred or if one has been attempted? This typically requires sifting through huge volumes of data gathered from the network, host, or file system to find suspicious activity. There are two general approaches to this problem: signature detection (also known as misuse detection), where we look for patterns signaling well-known attacks, and anomaly detection, where we look for deviations from normal behavior. Most work on signature and anomaly detection has relied on detecting intrusions at the host processor level, even those that detect intrusions occurring only at the network layer and below. A problem with these approaches is that even if anomalous/intrusion activity is detected, one is often unable to prevent the anomalous packets from causing havoc in the form of disrupting the system and over utilizing the system CPU (e.g. via denial-of-service attacks). *This paper targets the emerging application of network intrusion detection using Network Interface Cards (NICs). Specifically, we study a novel architecture for network intrusion detection using NICs and empirically evaluate its feasiblity.*

The primary role of NICs in computer systems is to move data between the system's components and the network. A natural extension to this role would be to actually police the packets forwarded in each direction by examining packet headers and simply not forwarding suspicious packets. The rationale for NIC-based intrusion detection coupled with conventional host-based intrusion detection can be stated as follows: First, functions such as signature-based and anomaly-based packet classification can be performed on the NIC, which has its own processor and memory. This makes the system virtually impossible to bypass or tamper with as can be the case with software-based systems that rely on the host operating system to function. Second, if the host is loaded (with other programs running simultaneously), an intrusion detection system that relies on host processing capability may be slowed down, thereby adversely affecting the bandwidth available for acceptable network transmissions. A NIC-based strategy will not be affected by the load on the host and therefore will not suffer the same slowdown. Third, there is a potential to naturally handle the scalability problem associated with centralized intrusion detection systems, since each individual NIC can handle the in-bound and out-bound traffic of the particular processor/local area network it is concerned with, thus effectively distributing the work concerned. Fourth, NIC-based strategies provide better coverage (one-to-one mapping between hosts and intrusion detection systems) and functional separation (e.g. internal NICs can detect port-scans while NICs at the firewall can detect host-scans). Finally, the NIC-based scheme is inherently flexible, dynamically adaptive, and can work in conjunction with a host-based intrusion detection system. The host-based intrusion detection system can download new rules/signatures into the NIC on the fly, making the detection process adaptive. Figure 1 represents the overall architecture for NIC-based security. The above advantages notwithstanding, the current disadvantage to NIC-based intrusion detection is that processing capability on the NIC is much slower and the memory sub-system is much smaller when compared with host-based methods. The task of implementing algo-

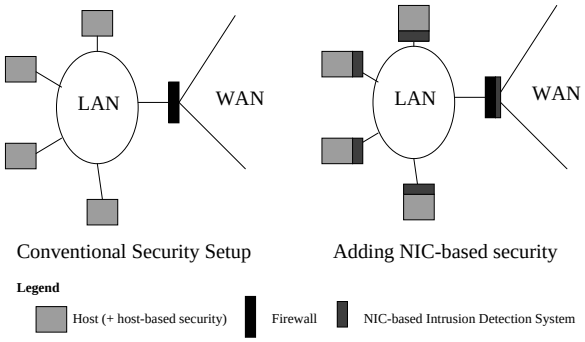


Figure 1: Motivation for adding NIC-based Security

rithms on the NIC presents several new challenges. For example, the NICs we used were not capable of performing floating point operations. As a result, we were forced to eliminate those operations or resort to estimates based on fixed-point operations. We also need to limit the impact on bandwidth and latency for non-intrusive messages. Given this fact, the question now becomes how best to use the NIC's processing capabilities for intrusion detection? This is the key question that this paper seeks to answer.

We describe several intrusion detection strategies (designed with the memory and processor limitation of current-day NICs in mind) and evaluate the performance of these schemes. We compare the NIC-based schemes with a purely host-based schemes while varying the load on the host processor. We evaluate each scheme based on the quality (how effective are such schemes for detecting intrusions) of detection, and the performance (in terms of bandwidth supported).

A basic anomaly detector framework was implemented on the firewall and host NICs. We implemented the P(SrcIP | DstIP) model [17] for anomalous client detection (described in Section 3.2) on the firewall NIC. A distributed version of the P(SrcIP | DstPort, DstIP) model was implemented on the host NIC. In addition to the above models, entropy was used to modulate the anomaly score. Hybrid approaches for intrusion detection were also implemented. One approach had the firewall host processor executing an association rule based anomaly detection scheme (described in Section 3.4) and discovering frequent itemsets that are downloaded to the NIC. A packet was labeled as anomalous if fewer than a certain threshold of frequent itemsets applied to it. The other approach involves clustering the packets to find anomalies. A simulated traffic workload based on the DARPA 99 dataset was used to evaluate our schemes. When the host is loaded, the performance of the host-based strategy degrades, while the NIC-based strategy continues to perform at a constant rate allowing for an 84% throughput efficiency. The preliminary results we obtain are very encouraging and lead us to believe that such NIC-based security schemes could very well be a crucial part of next generation network security systems.

The rest of this paper is organized as follows. In section 2 we document some of the related work in intrusion detection, NIC-based computing, NIC-based security and data stream processing. Section 3 documents our intrusion detection strategies. We report on the experimental evaluation of above strategies in Section 4. Finally we conclude with some directions for future research in Section 5.

2. RELATED WORK

In this section we present the related work in intrusion detection and NIC-based computing most relevant to our work.

2.1 Intrusion Detection Systems

There are two general approaches to the problem of intrusion detection: signature detection (also known as misuse detection), where we look for patterns signaling well-known attacks, and anomaly detection, where we look for deviations from normal behavior. Signature detection works reliably on known attacks, but has the obvious disadvantage of not being able to detect new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being able to discern intent. It can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms.

Signature detection methods are better understood and widely applied. They are used in both host based systems, such as virus detectors, and in network based systems such as SNORT [28] and BRO [25]. These systems use a set of rules encoding knowledge gleaned from security experts to test files or network traffic for patterns known to occur in attacks. A limitation of these systems is that as new vulnerabilities or attacks are discovered, the rule set must be manually updated. Another disadvantage is that minor variations in attack methods can often defeat such systems.

Anomaly detection is a harder problem than signature detection because while signatures of attacks can be very precise, what is considered normal is more abstract and ambiguous. Rather than finding rules that characterize attacks, we wish to find rules that characterize normal behavior [8]. Since what is considered normal could vary across different environments, a distinct model of normalcy can be learned individually. Much of the research in anomaly detection uses the approach of modeling normal behavior from a (presumably) attack-free training set. Because we cannot predict all possible non-hostile behavior, false alarms are inevitable. Forrest et al. [9], making the connection between anomaly detection systems and biological immunology, found that when a vulnerable UNIX system program or server is attacked (for example, using a buffer overflow to open a root shell), that the program makes sequences of system calls that differ from the sequences found in normal

operation [8] . Forrest used n-gram models (sequences of n = 3 to 6 calls), and matching them to sequences observed in training. A score is generated when a sequence observed during detection is different from those stored during training. Other models of normal system call sequences have been used, such as finite state automata [30] and neural networks [10]. Lane and Brodley [15] use instance-based methods and Sequeira and Zaki [31] use clustering methods for detecting anomalous user commands. Current network anomaly detection systems such as NIDES [1], ADAM [3], and SPADE [5] model only features of the network and transport layer, such as port numbers, IP addresses, and TCP flags. Models built with these features could detect probes (such as port scans) and some denial of service (DOS) attacks on the TCP/IP stack, but would not detect attacks of the type detected by Forrest, where the exploit code is transmitted to a public server in the application payload. Most current anomaly detectors use a stationary model, where the probability of an event depends on its average rate during training, and does not vary with time. However, using the average rate could be incorrect for many processes. Paxon and Floyd [26] found that many network processes, such as the rate of a particular type of packet, have self-similar (fractal) behavior. Events do not occur at uniform rates on any time scale. Instead they tend to occur in bursts. Hence, it is not possible to predict the average rate of an event over a time window by measuring the rate in another window, regardless of how short or long the windows are. An example of how a stationary model fails in an anomaly detector would be any attack with a large number of events, such as a port scan or a flooding attack. While most research in intrusion detection has focused on either signature detection or anomaly detection, most researchers have realized that the two models must work hand-in-hand to be most effective [3, 2].

2.2 NIC-based Computing and Security

Recently there has been a fair amount of activity in the area of NIC-based computing. More closely related to our work is the use of NICs for firewall security [20]. The idea is to embed firewall-like security at the NIC level. Firewall functionality, such as packet filtering, packet auditing, and support for multi-tiered security levels, has been proposed. While most of these ideas are in their infancy, a couple of simple ideas have been commercialized (e.g. 3Com's embedded firewall).

Several other efforts have targeted the use of NIC resources for improving communication and synchronization performance. [22] discusses a NIC based implementation of atomic remote memory operations along with an implementation of locks based on these remote memory operations. An implementation of NIC-based QoS on a Myrinet cluster is discussed in [29]. A NIC-based barrier implementation that is an order of magnitude faster is described in [21]. The SPINE project focuses on mechanisms to improve application performance by downloading pieces of code to the NIC which are then executed in a sandbox [16]. There have also been several other investigations into the issues of improving application performance by using active NIC's [4, 7, 27, 14].

2.3 Data Stream Processing

The past few years have witnessed the emergence of application domains wherein data elements arrive in the form of a continuous stream. Examples of such streaming datasets include stock tickers and click streams. Often, these data streams can be characterized as infinite streams that have no pre-defined size. This requirement has motivated online processing of data streams as and when they arrive and by developing algorithms that bound memory usage. Existing algorithms have been redesigned to process the stream in one pass using a summary structure, which stores an approximate representation of the data stream in memory [13, 18, 11, 19]. When processing network data for network intrusion detection, essentially, we process each incoming packet at a time, and never get a second look. This makes data stream related research highly relevant to NIC-based network intrusion detection. However, the key difference is that NIC-based stream processing is even more constrained when it comes to processing capabilities and memory usage. As a result, several data stream processing algorithms are rendered inapplicable for network intrusion detection under real-time processing requirements.

3. ALGORITHMS

In this section we describe the three basic types of algorithms (anomalous client detectors, port scan detector, and hybrid models) we evaluated. Each of these algorithms were implemented both on the NIC as well as on the host processor. However, they have all been designed keeping the memory and processing limitations of the NICs in mind.

3.1 NIC Programming Difficulties

NICs are typically very limited in their computing resources. They typically have a relatively small amount of memory, and a slow, limited processor. Any algorithm designed to run efficiently on a NIC must take these facts into account. For example, the NIC on which the following algorithms were implemented have no floating-point capabilities, and so any floating-point operations must either be eliminated, or estimated using fixed-point implementations. Likewise, array sizes must be constrained so that they fit within the limited memory available on the NIC.

3.2 Anomalous Client Detectors

3.2.1 P(SrcIP | DstIP) Anomalous Client Detector

The P(SrcIP | DstIP) anomalous client detector algorithm (see Figure 2) is loosely based on one of the models used in the non-stationary application layer anomaly detection (ALAD) algorithm proposed by Chan and Mahoney [17]. The objective of this model is to determine the anomaly score of a given packet based on previous interactions between a particular client and the particular destination host in question. The anomaly score is based on the value of P(SrcIP | DstIP), the probability that a client machine connects to a given host. The set of normal clients for a host are those for which P(SrcIP | DstIP) $>$ threshold(DstIP). A value of P(SrcIP | DstIP) that is lower than the threshold may be indicative of suspicious behavior.

The model is capable of detecting two types of anomalous behavior. One behavior is when a new or existing client (someone the system has not seen before) attempts to connect to a host that it has not connected to before. The second behavior is when the amount of a client's interaction with a particular host radically changes over time. Note that the system can only detect anomalies in the quantity of the interactions between the

```
function DetectAnomalousClient(SrcIP, DstIP)
begin
    SDTable: A two dimensional hash table with one axis indexed by DstIP and the other by
    SrcIP, which holds the number of packets each DstIP receives from each SrcIP.
    DstTable: A one dimensional table holding the number of packets received by each DstIP
    H: Hash function for the SrcIP axis in the SDTable
    DstTable[DstIP]++;
    SDTable[DstIP][H(SrcIP)]++; /* conflict resolution is handled using standard approaches*/
    if ((SDTable[DstIP][H(SrcIP)] / DstTable[DstIP]) > threshold(DstIP)
      return "Normal Client";
    else
      return "Possible Anomalous Client";
end
```

Figure 2: Anomalous Client Detector - P(SrcIP | DstIP) version

host and the client; it cannot detect anomalies in the character of the interactions.

To model the first scenario we need to model an *anomaly score* or *surprise factor*, for such a scenario unfolding. Basically, a new client accessing a well known world wide web portal is less surprising than a new client accessing a particular internal machine that is typically accessed only by a handful of trusted client machines. To model this surprise factor, we rely on incrementally keeping track of a threshold function that is inversely proportional to the entropy of accesses to a particular destination host. The entropy for the web based example (low threshold value) would be much higher than the entropy for the trusted client example (high threshold value). This threshold function (by the very definition of how entropy is computed) is dependent on the number of different client connections for a given host as well as the frequency of client connections. We incrementally maintain the threshold by adopting ideas from our previous research [23]. Note that a destination host which receives connections from a large number of sources (clients), is more likely to be accessed by a new source (client), than by a host which typically receives connections from just a few sources (clients). To model the second scenario, we monitor the rate of change of P(SrcIP | DstIP) over temporal windows. This would allow us to detect large fluctuations in the conditional probabilities which would trigger a possible anomaly alarm.

The basic algorithm stores information in a set of hash tables. Again we use hash tables so that one can save on memory utilization, at a (hopefully small) cost to accuracy of the model. Upon receiving a new packet, it simply computes the conditional probability P(SrcIP | DstIP) and if this conditional probability is less than the threshold determined from the training data, then the algorithm considers the source host to be an anomalous client. In the current implementation of the algorithm the thresholds are adjusted according to the Gini index instead of entropy, as the computation of a Gini index is better suited to the capabilities of the NIC than the computation of entropy is; computing a Gini index requires sqauring probabilities, whereas computing an entropy value requires taking logarithms of the probabilities, a function that is not efficient on the NICs.

3.2.2 P(SrcIP | DstPort, DstIP) Anomalous Client Detector

The P(SrcIP | DstPort, DstIP) anomalous client detector algorithm (see Figure 3) is also loosely based on one of the models used by Chan and Mahoney [17]. The objective of this model is to determine the anomaly score of a given packet based on previous interactions between a particular client and the particular service (port) it is accessing on a given host. This model is designed to be used in a distributed manner: this algorithm runs on the NIC of each host in the network, instead of running on the central firewall NIC. As such, the value of DstIP is constant (it is the IP address of the host), and so the hash table is only two-dimensional. Running this algorithm on the host NICs leaves the firewall NIC free to run other detectors. This algorithm can operate on the firewall NIC, but its memory requirements are much higher, as the hash table must be three-dimensional since it must take into account the value of DstIP, which it takes directly from the packet header.

3.3 Port Scan Detector

The port scan detector algorithm (see Figure 4), uses an array of N bit vectors of length B to keep track of the ports of a given destination machine that have been accessed. Since the memory requirements of keeping track of each of the 65536 possible ports would be too expensive, and intractable, given the NIC's resource capabilities, we apply a many-to-one hash function (f) on each port to reduce the storage requirements. This many-to-one function reduces the set of possible ports to B bits. We maintain N of these B-bit vectors to implement a sliding window, so that one can detect potentially hard-to-detect time-delayed port scans. These N vectors are cycled through depending on the frequency with which packets arrive at the destination machine: After P packets have arrived, a new vector is used.

Once a packet has been received, the algorithm sets the proper bit in the current bit vector and updates its count. If less than F packets have been received since the last check for a port scan, the algorithm finishes with the message that no check was made. If F packets have been received, the algorithm checks for a port scan by proceeding to OR the N vectors. If the number of 1's found in the OR is greater than a given threshold S, it declares the detection of a port scan.

3.4 Hybrid Models For Anomaly Detection

The following models are hybrid models: They use resources at both the host and NIC level. Operations that cannot be performed at the NIC level due to memory or speed constraints can be performed at the host level, and their results can be sent

```
function DetectAnomalousClient(SrcIP, DstPort)
begin
    SDDTable: A three dimensional hash table with one axis indexed by DstPort, one by DstIP, and the third by
    SrcIP, which holds the number of packets each (DstPort, DstIP) pair receives from each SrcIP.
    DstTable: A two dimensional table holding the number of packets received by each (DstPort, DstIP) pair
    H: Hash function for the SrcIP axis in the SDTable
    DstIP = Localhost.IPAddress;
    DstTable[DstPort][DstIP]++;
    SDDTable[DstPort][DstIP][H(SrcIP)]++; /* conflict resolution is handled using standard approaches*/
    if ((SDDTable[DstPort][DstIP][H(SrcIP)] / DstTable[DstPort][DstIP]) > threshold(DstPort, DstIP)
      return "Normal Client";
    else
      return "Possible Anomalous Client";
end
```

Figure 3: Anomalous Client Detector - P(SrcIP | DstPort, DstIP) version

```
function DetectPortScan(DstPort)
begin
      Vector[]: An array of N bit vectors, each of length B bits
      f: function to map port numbers to bit numbers
      F: Number packets received between successive checks
      S: The number of bits that must be 1 for a port scan to be reported
      P: The number of packets per vector
      Current: A global counter that keeps track of the current bit vector
      Count: The number of packets represented by current bit vector
      Result: The result of the detector
      Set bit f(DstPort) of Vector[Current mod N] to 1;
      Count++;
      if (Count mod F = 0)
          Temp := OR all N bit vectors together;
          if (Sum of Ones in Temp > S)
              Result := "Port Scan Detected";
          else
              Result := "No Port Scan Detected";
      else
          Result := "No Check Performed";
      if (Count = P)
          Current++;
          Count := 0;
          Clear Vector[Current mod N];
      return Result;
end
```

Figure 4: The Port Scan Algorithm

to the NIC to aid in detection.

3.4.1 *Anomaly Detection Using Frequent Itemsets*

This algorithm uses the concept of frequent itemsets to discover anomalies. (see Figure 5). This is a hybrid system: The frequent itemsets are computed at the host level and periodically passed down to the NIC, since the NIC itself does not have sufficient resources to compute frequent itemsets. The frequent itemsets are computed using features found in the headers of packets sampled by the host machine.

The intuition behind this algorithm is that since intrusions occur only rarely, by sampling the packets to find the frequent itemsets, few if any intrusion packets will be incorporated into the model. Also, many intrusion packets have little in common with normal packets, and so the number of frequent itemsets found in an intrusion packet will be relatively small. Finally, the size of those itemsets will be relatively small since small itemsets usually have high supports and so these itemsets are shared by all packets, whether they are intrusions or not. Hence, in the algorithm, two thresholds (CountThreshold and SizeThreshold) are used to determine which are intrusion packets and which are not. CountThreshold separates those packets in which only a few itemsets from those in which many are found. SizeThreshold separates those packets with itemsets that are, on average, smaller from those with larger itemsets.

3.4.2 *Anomaly Detection using Clustering*

The packet clustering algorithm (see Figure 6) also takes a hybrid approach to NIC-based intrusion detection. The host based on a random sample of packets from the flow, computes a set of initial clusters. Then these initial clusters are sent to the NIC. The algorithm running on the NIC will then either assign each packet to one of these clusters or if the packet is not close enough to any current cluster, the algorithm will declare this packet as a anomaly and assign an anomaly score to it. We have a heuristic function which gives the anomaly score based on the similarity and dis-similarity measures of the cluster and the packet. Typically the number of clusters formed is very small, so this part of the algorithm can easily be implemented on the NIC. To be up-to-date, at regular intervals the host takes new random samples and recomputes the clusters if need be. Since the network data used has both catagorical and continuous attributes, We use a hybrid version of ROCK [12] to cluster the packets. Our preliminary results show good results in this direction.

4. EXPERIMENTAL RESULTS

In this section we detail the experimental results we obtained. We first start by describing the experimental setup (the machine configurations, data sets used and evaluations performed) for each of the three algorithms. We then examine the qualitative and quantitative aspects of the algorithms. The quantitative study focuses on the comparing the host-based and NIC-based schemes while the quality study focuses on the quality of the algorithms as measured by the accuracy of detection.

4.1 Setup

All host-based experiments, unless otherwise noted, were evaluated on dual-processor 300 MHz Pentium II machines with 128 MB of memory. All NIC-based evaluations were done on Myrinet NICs, each of which had a 66Mhz LANai 4 processor and 1 MB of memory.

To test each algorithm, we used both synthetic and real data sets. The synthetic data sets were generated using a program available at http://www.cis.ohio-state.edu/~otey/NIC/. The real data set is the 1999 DARPA Intrusion Detection Evaluation data set. The size of the packets, unless otherwise noted, is 2048 bytes. Each packet is composed of four protocol bits, four flag bits, four source IP bytes, four destination IP bytes, two source port bytes, two destination port bytes, one TTL byte, two packet size bytes and 2032 bytes for the data. We set it up so that the network packet generator sends roughly 1 packet every 0.001 seconds for a net theoretical throughput of 2MB per second. However, the actual measured throughput is 1796 KB per second.

For evaluating port scans the data set generator generates K independent port scans (that scan different parts of a particular destination host's ports). Note that here we are actually evaluating the case where a single NIC can be responsible for a system area network or a small subnet. The parameters to the data set generator include the number of port scans (K = 5), the length of each port scan (32768 ports), the average distribution of normal packets to port scan related packets (5:1) when a port-scan is in progress, and the total number of network packets (1 million). As for the packets that are not part of the port scan, their destination port numbers are distributed such that 99% of all port accesses are to port numbers below 1024 (reflecting realistic distributions). The algorithmic parameters we used for the port scan were F = 16384 and 32768, B = 8192, P = 32768, N = 4, and S = 3072.

The data set for the anomalous client detector focuses solely on the source and destination IP addresses. All other fields are filled with random numbers. The data set we used consisted of 256 local hosts, the set of valid clients per host ranged from 0-64, and the total set of clients was the remaining valid IP addresses on the Internet (approximately 4.3 billion hosts). The first hundred thousand packets were messages from valid clients (the training data) and the next 1 million packets are testing packets, about 90% of which come from the set of valid clients.

We used the first three weeks of tcpdump data from the DARPA data sets. Weeks one and three contain normal network traffic. Week two contains four days of data containing 34 intrusions (Tuesday's tcpdump file was unreadable). For the anomalous client detectors we used the data from weeks one and two to do testing, and for the frequent itemset-based detector we used weeks one and three to build the itemsets and week two to test it.

4.2 Quantitative Experiments

Before detailing the experimental results on performance, we wanted to quickly get a sense for the individual resource requirements (memory and computational) of the different algorithms.

Figure 7 documents these results. The Y-axis represents wall-clock execution time in seconds on the host processor. As we can see from the figure, the port-scan algorithm at various parameter settings takes a longer amount of time than the

```
function DetectAnomaly(Packet)
begin
    I: The set of frequent itemsets found in the headers of normal packets
    Count: The number of frequent itemsets found in the current packet
    SizeSum: The sum of the sizes of the frequent itemsets found in the current packet
    CountThreshold: Threshold for the number of itemsets that must be found
    SizeThreshold: Threshold for the average size of the itemsets found
     if i is present in Packet
      Count++;
      SizeSum := SizeSum + Size(i);
     end if
    end for
    if (Count > CountThreshold or (SizeSum/Count) > SizeThreshold)
     return "Possible Anomalous Client";
    else
     return "Normal Client";
end
```

Figure 5: Anomaly Detection Using Frequent Itemsets

```
function ClusterPacket(Clusters C, Packet P)
begin
      C: Set of initial clusters given by the Host
      P: The packet to be clustered
      Max: The maximum goodness value for clustering
      Threshold: The threshold value for declaring a packet as part of a cluster
         Max := 0;
         for each cluster c belonging to C do
             if (Closeness ( c, P) > max )
                 max := Closeness(c, P);
         if (max < threshold)
             declare anomaly;
end
function Closeness(Cluster C, Packet P)
begin
      C: Existing Cluster
      far: A measure of farness of P from C
      close: A measure of closeness of P from C
      a(i): The i'th attribute of P
      P(a(i)): Returns the value of the i'th attribute of P
      C(a(i)): Returns the value of the i'th attribute in the summary of C
      h(far,close): Returns a combined measure of closeness and farness of P from C
         far = 0;
         close = 0;
         for each attribute a(i) of P do
             if (P(a(i)) == 1)
                 close += C(a(i));
             else
                 far += C(a(i));
         endfor
        return h(far, close);
end
```

Figure 6: The Clustering based Anomaly Algorithm

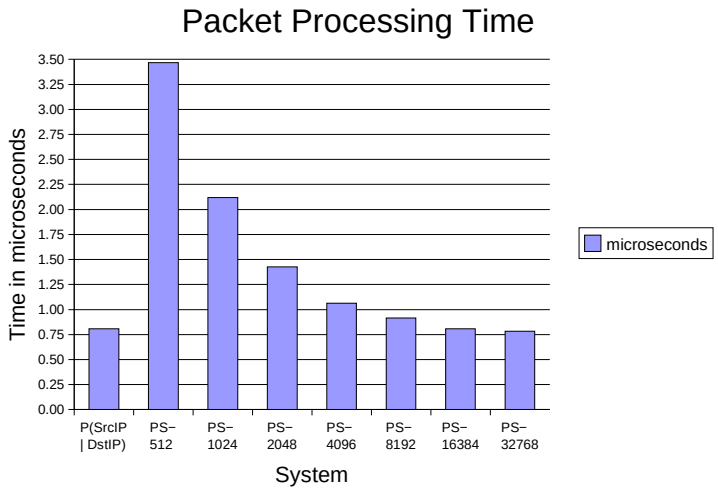


Figure 7: Evaluation of Proposed Method: Similarity plots

P(SrcIP | DstIP) anomalous client detector algorithm. However, for larger parametric settings ($F \geq 16384$), the port-scan algorithm is comparable to the anomalous client detector algorithm.

We next compared the performance of the host-based and NIC-based strategies. The results are documented in Figure 8. The X-axis in all cases documents variation in host load. The Y-axis is the net throughput of valid packets (allowed to filter through). Re-call that the maximum throughput is limited by the incoming data rate which is 1796 KB per second.

For both the P(SrcIP | DstIP) anomalous client detector and port scan algorithms we notice that the NIC-based and Host-based strategies perform comparably until the host becomes overloaded. The NIC-based strategy is unaffected by the load and performs at a relatively constant rate allowing for a 95% throughput efficiency.

4.3 Qualitative Experiments

For the port scan, preliminary results showed that a value of 3072 for S would be suitable given the parameters used to generate the test data. Varying the parameter F in the port scan detection algorithm not only affected the speed of the algorithm but the quality of its results as well. Figure 9 documents the port scan algorithm's behavior when F = 32768. The lower function in the figure represents the test data, and is high when a port scan is in progress and low when one is not. The upper function represents the algorithmic behavior, and is high when the algorithm detects a port scan and low when it does not. The figure shows that there is a lag between when a port scan begins and when it is actually detected by the algorithm. This lag is directly proportional to the parameter F, as seen in figure 10. However, decreasing the value of F makes the detector more sensitive to noise, as can be seen in the table in figure 11. Noise can cause the number of ones in the OR of the bit vectors to rapidly fluctuate above and below the threshold value S, causing the detector to find multiple phantom port scans that in reality correspond to a single port scan.

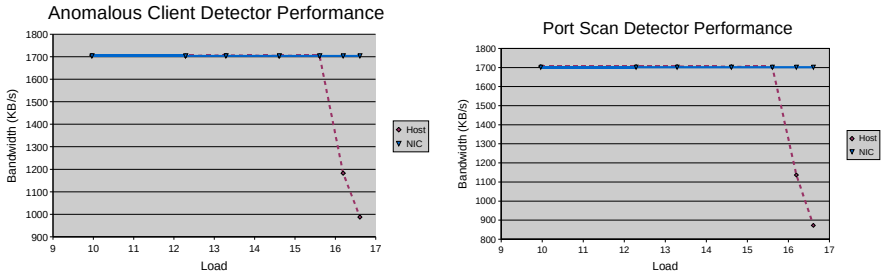
Figure 12 and figure 13 show how different algorithms fared with the DARPA dataset. In all, 18 of the 34 intrusions present in week two were detected. Figure 12 show the results for different versions of the anomaly detectors as well as the frequent itemset-based detector. The P(SrcIP | DstIP) Window algorithm is a modification of the the P(SrcIP | DstIP) algorithm that employs a sliding window technique similar to the port-scan detector algorithm that periodically removes older packets from the hash table. The P(SrcIP | DstIP) Sample algorithm is another modification that only examines a sample of the packets. In this case we sampled 25% of the packets. The P(SrcIP | DstPort, DstIP) algorithm is implemented here in a non-distributed fashion: We use a three-dimensional hash table instead of a seperate table for each internal host. For all of these detectors, the threshold used was a constant value of 0.000015. The frequent itemsets were generated from a 5% sample of the weeks one and three data with a minimum support of 5% and a minimum itemset size of 4. CountThreshold was set to 5 and SizeThreshold was set to 4.05.

In figure 12 one can see that the first four algorithms perform fairly well as far as false positives are concerned. The Sample algorithm does the best in this respect, though it only detects 9 intrusions. In the Venn diagram in figure 13 one can see which intrusions were detected by which algorithms. It is worth noting that the frequent itemset algorithm can only detect denial-of-service (DOS) and probing attacks, since these attacks produce anomalies in the packet headers. Generally, user-to-root (U2R) and remote-to-local (R2L) attacks cannot be detected by this algorithm since their signatures are concealed within the payload portion of the packet, which is ignored by the algorithm. The other algorithms can detect such intrusions, however, because many intrusions originate from a machine that has had little previous contact with the host. Therefore the probabilities will be smaller and the packets will be marked as intrusions.

5. CONCLUSIONS

We present and evaluate a NIC-based network intrusion detection system. We consider embedding both signature detection algorithms as well anomaly detection algorithms in our evaluation.

The quantitative improvements we achieve with this approach relies on the fact that the operating system of the host does not have to be interrupted with the detection process. Thus on heavily loaded hosts admissible network traffic proceeds at a consistent rate provided the computational and memory resources of the NIC is not stretched. Our preliminary empir-



(a) Anomalous Client Detector (b) Port Scan

Figure 8: Quantitative Comparison of NIC-based and Host-Based Intrusion Detection

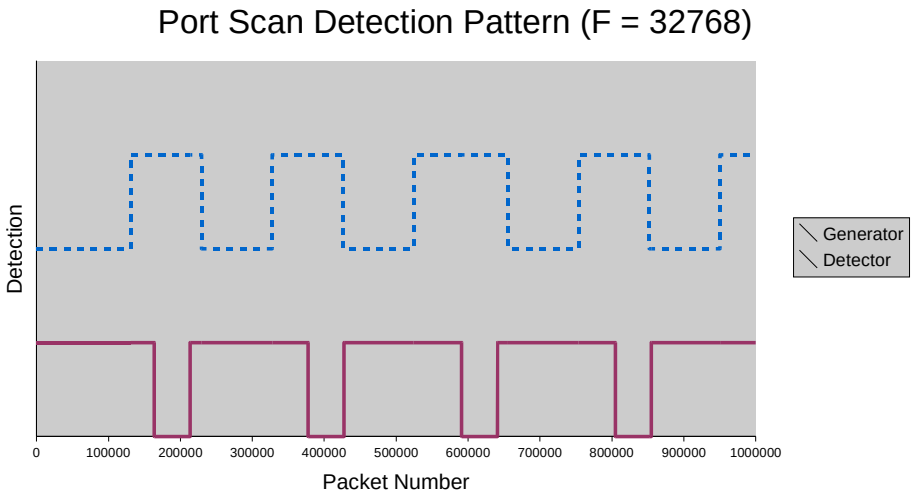


Figure 9: Port Scan Detector Behavior

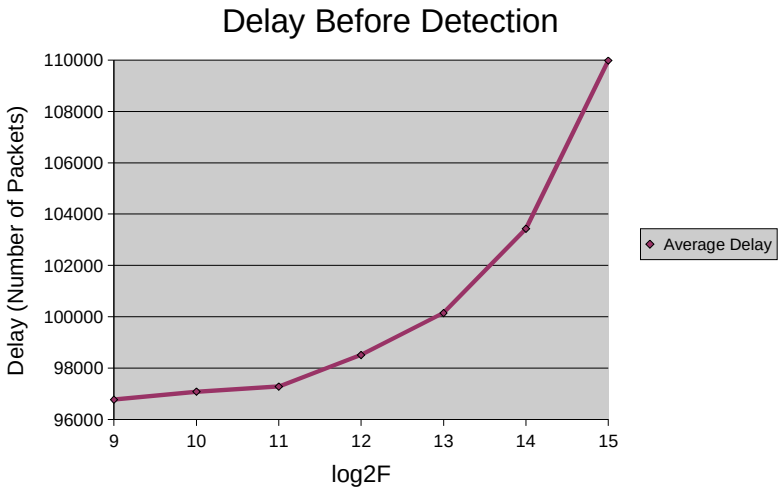


Figure 10: Different Values for F Affect How Quickly Port Scans Are Detected

F	*Number of Scans Detected*
512	7
1024	7
2048	7
4096	6
8192	5
16384	5
32768	5

Figure 11: Different Values for F Affect How Many Port Scans Are Detected

Method	Total Flagged Packets	Total Flagged Week 2 Packets	Total Intrusion Packets	Total Intrusions Detected
P(SrcIP | DstIP)	1021	593	197	10
P(SrcIP | DstIP) Sample	79	63	41	9
P(SrcIP | DstIP) Window	182	30	15	10
P(SrcIP | DstIP, DstPort)	259	182	89	10
Frequent Itemsets	N/A	124434	N/A	10

Figure 12: Results of different algorithms

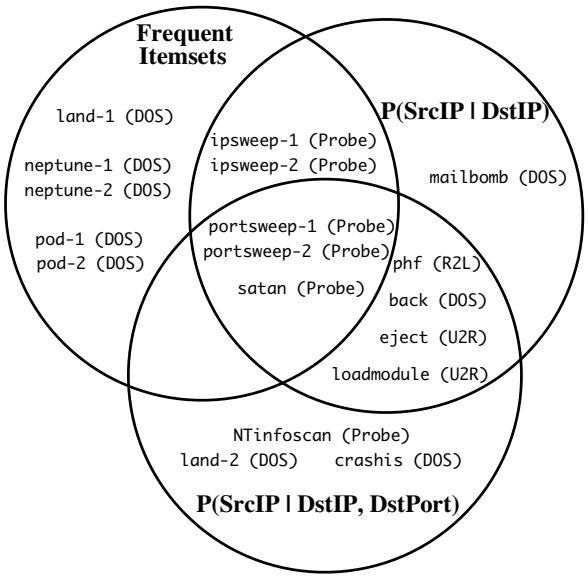


Figure 13: Different intrusions are detected by different algorithms

ical results bear this out. A key element in understanding the tradeoff's involved is the amount of computation and memory utilization involved in the programs. The larger the computation cost, the better the performance of a purely host-based approach.

From the qualitative angle, the downside of using simplified algorithms is that the quality and detection rate are hampered. However, the benefit of having the NIC do the policing is that it can actually prevent network-based intrusions from wrecking havoc on host systems. Since the intrusive packet, if caught, never reaches the host operating system, this approach can detect and prevent, unlike host-based systems. In effect, the NIC acts as a basic shield for the host. If the NIC cannot catch up with the rate the packets are arriving, it can begin dropping the packets (this is the default behavior in the Myrinet NICs), as this may be indicative of a denial-of-service attack. If the NIC were to become overwhelmed by a such an attack, the host would be spared from it. We would prefer to sacrifice only the NIC to the attack rather than the entire host machine.

However, from a technology perspective we are not very far away from 1GHz NIC processors (with appropriately larger memory). With those projected systems we anticipate that NIC-based intrusion detection will do better both from a quantitative standpoint and from a a qualitative standpoint (as we will be able to use less-restrictive algorithms). In terms of ongoing and future work, we are looking into taking the algorithms proposed and porting them onto the newer Myrinet cards featuring faster LANai 9 processors that are now available for use in our group. We are also looking at how to use existing real intrusion data (for example, the KDDCUP and DARPA datasets) to evaluate our algorithms. Finally, we are looking at the problem of implementing incremental techniques[6, 24, 32] on the NIC platform, to update the information used to detect intrusions (as described in Section 3).

6. REFERENCES

[1] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (nides). In *Technical Report SRI-CSL-95-06, SRI*, 1995.

[2] D. Barbara and S. Jajodia, editors. *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[3] D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *Proc. SIAM Intl. Conf. Data Mining*, 2001.

[4] J. Anderson D. Gallatin A. Lebbeck A. Chase and Yocum K. Network i/o with trapeze. In *Proceedings of 1999 Hot Interconnects*, 1999.

[5] Silicon Defence. Spade. In *http://www.silicondefense.com/software/spice/*, 2001.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Micha el Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environmen t. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 323–333, 24–27 1998.

[7] M. Martin R. Owa T. Fiuczynski and Bershad B. On using intelligent network interface cards to support multimedia applications. In *Proceedings of NOSSDAV'98*, 1998.

[8] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of 1996 IEEE Symp. on Computer Security and Privacy*, 1996.

[9] S. Forrest, S. Hofmeyr, and S. Somayaji. Computer immunology. In *Comm. ACM, 4(10):88-96*, 1997.

[10] A. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, 1999.

[11] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *proceeding of the Annual Symp. on Foundations of Computer Science*, 2000.

[12] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proc. IEEE Internationl Conference on Data Engineering*, 1999.

[13] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *proceeding of the seventh International Conference on Knowledge Discovery and Data Mining*, 2001.

[14] Yocum K. and Chase J. Payload caching: High speed data forwarding for network intermediaries. In *2001 Usenix Conference*, 2001.

[15] T. Lane and C. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *ACM Trans-actions on Information and system Security*, 1999.

[16] Fiuczynski M. and Bershad B. SPINE: A safe programmable and integrated network environment. In *Proceedings of the Eighth ACM SIGOPS Workshop*, 1998.

[17] M. Mahoney and P. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *SIGKDD*, 2002.

[18] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *proceeding of the 28th VLDB Conference, Hong Kong, China*, 2002.

[19] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *proceeding of the ACM Intl Conf. on Management of Data*, 1998.

[20] D. Nagle and D. Friedman. Building firewalls with intelligent network interface cards. In *CMU SCS Technical Report CMU-CS-00-173*, 2002.

[21] D. Buntinas D. K. Panda and P. Sadayappan. Fast nic-based barrier over myrinet/gm. In *Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2001.

[22] D. Buntinas D.K. Panda and W. Gropp. Nic-based atomic operations on myrinet/gm. In *SAN-1 Workshop, to be held in conjunction with High Performance Computer Architecture (HPCA) Conference*, 2001.

[23] S. Parthasarathy and A. Ramakrishnan. Parallel incremental 2d-discretization on dynamic datasets. *International Conference on Parallel and Distributed Processing Systems*, 2002.

[24] S. Parthasarathy, M. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. ACM Confereince on Information and Knowledge Management (CIKM), Mar 1999.

[25] V. Paxon. Bro: A system for detecting network intruders in real-time. In *Proc. 7th USENIX Security Symp.*, 1998.

[26] V. Paxon and S. Floyd. The failure of poisson modeling. In *IEEE/ACM Transactions on Networking, 3:226-24*, 1995.

[27] Krishnamurthy R. Schwan K. West R. and Rosu M. A network co-processor-based approach to scalable media streaming in servers. In *Proceeding of the International Conference on Parallel Processing (ICPP'00)*, 2000.

[28] M. Roesch. Snort – lightweight intrusion detection for networks. In *USENIX LISA*, 1999.

[29] A. Gulati D. K. Panda P. Sadayappan and P. Wyckoff. Nic-based rate control for proportional bandwidth allocation in myrinet clusters. In *Int'l Conference on Parallel Processing*, 2001.

[30] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollinen. A fast automaton based method for detecting anomalous behaviours. In *proceeding of IEEE Symposium on Security and Privacy 144-155*, 2001.

[31] K. Sequira and M. Zaki. Admit: Anomaly-based data mining for intrusions. In *SIGKDD Conference*, 2002.

[32] A. Veloso, W. Meira, M. Carvalho, B. Possas, S. Parthasarathy, and M. Zaki. Mining frequent itemsets in evolving databases. SIAM International Conference on Data Mining, 2002.