# PARNIDS: A Scalable Network Intrusion Detection Loadbalancer


## ABSTRACT

*Network intrusion detection systems (NIDS) are becoming an increasingly important security measure. With rapidly increasing network speeds, the capacity of the NIDS sensor can limit the ability of the system to detect intrusions. The PARNIDS parallel NIDS architecture overcomes this limitation by distributing network traffic load over an array of sensor nodes. Based on a custom hardware load balancer and cost-effective off-the-shelf sensors, the system employs novel stateless load balancing heuristics to thwart scalability limitations. It also uses dynamic feedback from the sensor nodes to adapt to changes in network traffic. This paper describes the overall system architecture, discusses some of the critical design decisions and presents experimental results that demonstrate the performance advantage of this approach.*


## 1. INTRODUCTION

With the continuing computerization of our society, network intrusion detection systems have become one of the key tools for a sytem to secure critical data. Complementing firewalls and host-based security tools, network intrusion detection systems *(NIDS)* are usually located at the interface between internal and external networks, where they can observe and analyze all information traveling between the two networks to find potential security breaches. The overall efficiency of the NIDS in detecting and signaling attacks depends not only on the sophistication of the analysis algorithm but also on the system's capacity. To be effective, the NIDS platform must be able to examine network packets at full wire speed, since any significant packet loss can impact the attack detection accuracy.

Network traffic speeds and volume are increasing at an exponential rate. Historically, Ethernet bandwidth has increased tenfold nearly every four years, outstripping the ability of general-purpose computer systems to receive and process every network packet. For instance, many existing hosts are unable to effectively process traffic on a 100 Mbit per second link [12]. Increasingly complex intrusion detection methods only add further to the pressure on NIDS platforms.

The conventional approach of tuning the hardware and software of the NIDS platform to maximize its performance can yield considerable improvements, but falls short in supporting next-generation networks operating at gigabits per second and faster. Custom hardware solutions, on the other hand, are expensive and inflexible. Parallel or distributed network intrusion detection platforms present a viable alternative for high-speed network environments, as they distribute network traffic over an expandable set of sensor hosts. Such systems combine the flexibility and cost-effectiveness of off-the-shelf hardware and software with the performance of custom designs. The key challenge for a parallel NIDS platform is to distribute network traffic over several hosts while avoiding overloading any of the sensors. At the same time, the parallel NIDS platform must be robust against malicious traffic patterns, and must not introduce any new vulnerabilities to the overall detection platform due to the distributed processing that would allow attackers to avoid detection. Finally, the traffic distribution scheme should not negatively impact the sensors' ability to perform intrusion analysis.

This paper first analyzes the requirements for a parallel network intrusion detection platform, and then describes the architecture of a novel, scalable, parallel NIDS platform consisting of a custom hardware load balancer and off-the-shelf sensor nodes. The *PARNIDS* load balancer employs multiple levels of hashing and incorporates feedback from the sensor nodes to distribute network traffic over the sensors without overloading any of them. The system is designed to handle network traffic at 1 Gigabit per second and above with minimal packet loss. Unlike previous connection-based approaches [3][6], the hash-based load balancer does not impose scalability limitations on the number of concurrent connections or on the number of sensors. At the same time, the system minimizes the impact of traffic distribution on the accuracy of each sensor's analysis mechanism by routing packets belonging to the same connection to the same node whenever possible.

To ensure operation at wire speed, the load balancer is implemented as custom hardware rather than software. A Gigabit Ethernet link can carry packets at a rate of one every 742 nanoseconds, leaving insufficient time for most software load balancing implementations. On the other hand, modern field-programmable gate arrays [14] provide enough logic and memory resources to implement sophisticated, pipelined load balancing techniques even at wire speed. At the same time, the ability to reconfigure these devices allows for an incremental design and test methodology and reduces overall development risk. The NIDS sensor nodes, on the other hand, are built from commodity hardware and software, thus leveraging improvements in system performance and intru-

sion detection software techniques, and reducing overall system cost. The overall approach strikes a balance between flexibility in the actual NIDS algorithms and high performance through a small amount of custom logic.

The remainder of this paper is organized as follows. Section 2 details the requirements and describes the design of a scalable NIDS load balancer with a particular emphasis on dynamic overload avoidance and hot spot management. Section 3 presents evaluation data from trace-based simulations and discusses several design tradeoffs and optimizations. Section 4 discusses how the general design maps to a prototype implementation. Finally, Section 5 contrasts this approach with other work in high-speed network intrusion detection and Section 6 outlines plans for future work.

# 2. PARALLEL NETWORK INTRUSION DETECTION

## 2.1 Requirements

Network intrusion detection sensors operate by reading and analyzing all network packets from the link under observation. Intrusion detection algorithms range from signature based analysis [11] to stateful flow analysis [7] and statistical anomaly detection. In all cases it is of critical importance to minimize the number of dropped packets, since any packet may contain valuable information about an attack and may even contain the entire attack.

The primary goal of a NIDS load balancer is to distribute network packets across a set of sensor hosts, thus reducing the load on each sensor to a level that the sensor can handle without dropping packets. However, network traffic characteristics such as average packet size, inter-arrival times and protocols vary over time, usually over the course of a day. For instance, an ISP may experience mostly HTTP traffic during the day, followed by large UDP packets used by file sharing programs in the evenings. Furthermore, network load characteristics may be intentionally manipulated by an attacker to circumvent the NIDS. Consequently, the load balancing approach must be able to quickly adapt to such changes.

Note that, unlike load balancers in other environments such as web servers, distributed systems or clusters, a NIDS load balancer is not concerned with achieving the best possible distribution of work across all nodes. Since the NIDS is not in the active communication path, improving its throughput beyond the offered network load does not result in any performance improvements. It is sufficient to assure that no sensor's load exceeds its capacity.

On the other hand, the analysis performed by the NIDS software running on the sensor hosts places some restrictions on how network packets are distributed. Stateful, flow-based analysis requires that at least one sensor observes the complete set of packets in a flow. In the case of TCP, a flow simply corresponds to a TCP connection, but even datagram protocols such as UDP often show connection-like behavior. For instance, each NFS client maintains a logical connection to the server, even though the underlying protocol is based on datagrams and is inherently connection-less. To facilitate the stateful analysis of such network flows, the load balancer must strive to forward related packets to the same sensor. As a result, while a simple round-robin distribution of network packets would distribute processing load evenly across the sensors, it is not a viable solution as it scatters packets belonging to the same flow over many sensors. A static assignment of flows to sensors, on the other hand, does not take into account differences in packet or data rates between different flows and can result in concentrations of high-intensity flows overloading some sensor nodes.

Finally, the load balancer should not introduce any additional vulnerabilities to the NIDS. Existing single-stream intrusion detection systems can be overloaded with a flood of small packets, similar to a conventional denial-of-service attack. During the overload period, the NIDS is less likely to detect the actual attack. While a load balancer directly addresses the capacity problem, it may introduce new vulnerabilities through the load balancing algorithm. For instance, if the load balancer maintains a table of flows to facilitate flow-based load distribution and analysis, attackers can overflow the forwarding table and render the load balancer ineffective. A good load balancing algorithm must thus minimize or eliminate such vulnerabilities.

## 2.2 Scalable Load Balancing

The PARNIDS parallel network intrusion detection platform uses a hash-based packet distribution scheme to address the challenges of a high-speed distributed NIDS. Network packets belonging to the same flow or connection are uniquely identified by the tuple consisting of source and destination IP addresses and port numbers. To distribute packets over a set of sensors, the load balancer hashes these fields into a table, where each table entry or hash bucket is associated with a specific sensor. This approach addresses several of the challenges outlined above. It does not limit the maximum number of concurrent flows or connections, since the hash table size is fixed and independent of the number of flows. At the same time, the design is sufficiently flexible to support varying numbers of sensors. Most importantly, network packets belonging to the same flow always hash to the same value and are thus always forwarded to the same node, but without incurring the cost of keeping track of individual flows.

## 2.3 Dynamic Feedback

While the basic hash-based packet distribution scheme combines scalability and flexibility and does not interfere with flow-based analysis, it is not able to dynamically adjust to traffic changes. In fact, the hashing scheme assumes a uniform distribution of the values in the header fields being hashed. In many installations, such uniformity is not present. For instance, a university may receive more web requests originating from the country it is located in, thus skewing the source IP address range. To tolerate such non-uniformities, the load balancer must be aware of the excess load on any one sensor node and must be able to adjust the distribution scheme accordingly.

Each network packet imposes a certain load on the sensor. This cost not only depends on the fixed interrupt and system call overhead, but also on the size of the packet and the actual payload, making it difficult for the load balancer to accurately determine sensor load based solely on the num-

ber of packets or bytes forwarded. Thus, sensors themselves must observe their own input buffer utilization and issue flow control messages to the load balancer when the buffer reaches a certain threshold, so that the packet distribution scheme can be modified before packets start being dropped. Upon receipt of a flow control message, the load balancer moves some of the hash table entries currently assigned to that sensor to a different node. This approach, shown in Figure 1, provides a simple and effective means of incorporating dynamic feedback into the basic hash-based load balancing scheme without violating the requirement of a stateless design. However, an adjustment in response to a flow control message disturbs the flow-based analysis as it effectively breaks network flows as it moves them to different sensors. In general, this disturbance is preferable to the loss of packets that would occur if no adjustment was made.

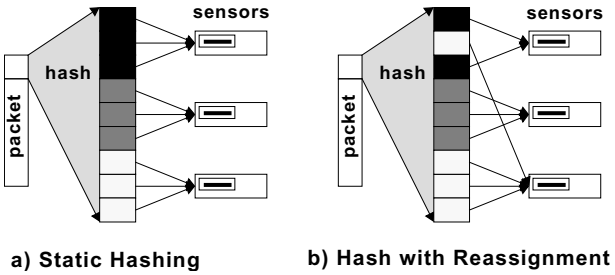


**Figure 1: Dynamic Reassignment of Hash Buckets**

With a given hash table size, the average number of buckets assigned to any sensor is inversely proportional to the number of sensors. For small numbers of sensors, it may be necessary to move a significantly larger number of buckets to achieve a load balancing effect. To avoid this problem, the hash table size scales in powers of two, proportional to the number of sensors. This solution keeps the average number of buckets per sensor within a fixed range, and makes load balancing decisions more predictable.

## 2.4 Hot Spots and Multi-level Hashing

While the dynamic assignment of hash buckets to sensors is able to adjust to many types of changes in network traffic, it is not immune to the load offered by a small number of high-bandwidth flows. It is possible, particularly when confronted with a sophisticated attacker, that a number of such flows hash to the same table entry, and thus are forwarded to the same sensor. In this case a simple reassignment of hash buckets will not address the problem sufficiently, since a single hash bucket may overload any sensor it is assigned to.

Instead, the PARNIDS load balancer promotes such high-intensity traffic to an additional level of hashing. Essentially, rather than routing the packets to the sensor indicated by the hash bucket, a secondary hash function is applied to the packet to determine the target sensor, as shown in Figure 2. If properly selected, the secondary hash function then distributes the packets evenly over all sensor nodes. This approach can be taken further with additional levels of hashing to reduce the effects of a hot spots even more.

A detailed analysis of a number of traces captured on a University campus Internet connection shows that even simple XOR-based hash functions can achieve a relatively even distribution across the hash buckets. Varying the way in which individual bits are combined in the hash functions is an effective way to produce different hash functions. Adding the two IP addresses or port numbers before hashing results in additional variations.

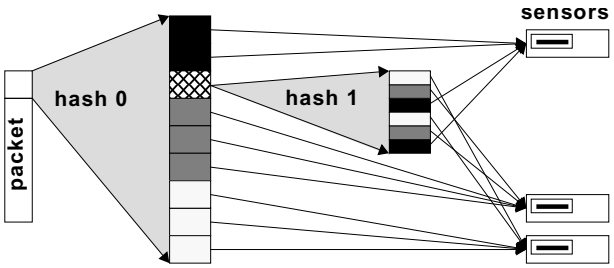


**Figure 2: Multiple Levels of Hashing**

Ultimately, the worst case scenario from the point of view of a distributed NIDS like PARNIDS is an intense denial-of-service attack against a single port on a single target IP from a single source IP and a single source port. In this case no hashing function that uses only the connection ID is able to distribute these packets over multiple sensors. As a final measure against such high-intensity traffic, the load balancer performs a round-robin distribution of individual packets once all levels of hashing are deemed to be insufficient. However, given the heterogeneous and distributed nature of the Internet, and the resulting sharing of bandwidth, it is unlikely that a single attack flow can overload a reasonably capable NIDS sensor.

The load balancer must periodically re-evaluate all promoted buckets. If traffic characteristics have changed and a bucket is no longer a hot spot, routing for this entry falls back to the original destination. This avoids the scenario where all buckets are eventually promoted, forcing a round-robin distribution scheme on all packets. To reduce the likelihood of an attacker exploiting the load balancing algorithm to avoid detection, buckets are re-evaluated after a random time-out value of several seconds.

The main challenge in applying this multi-layered hashing approach is twofold: how to determine when it is necessary to promote a hash bucket to the next level of hashing, and which hash bucket to promote. Ideally, if the load offered by a single bucket exceeds the capacity of the target sensor, the bucket needs to be promoted. However, accurately measuring the processing load that packets exert is impractical as it depends not only on the rate and size of packets but also on the algorithmic complexity of the payload.

A realistic alternative is to compare the packet rate of individual buckets with the average packet rate of all buckets currently associated with a sensor. Buckets that exceed a specific relative threshold are subject to promotion instead of reassignment.

Figure 3 demonstrates this heuristic approach. Both graphs show a sorted histogram of bucket load for a given sensor.

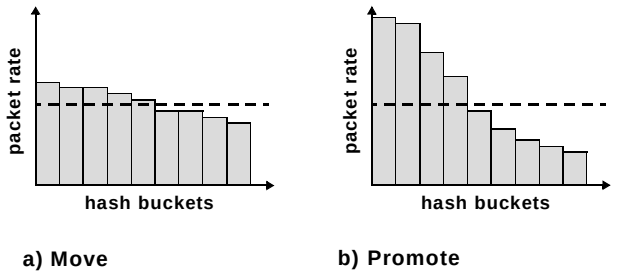


**Figure 3: Chosing between Moving and Promotion of Buckets**

The dashed line corresponds to the average load. The histogram in subfigure *a* shows a histogram with only a small variation of bucket loads. In this case, to address such a sensor's overloading problem, it is sufficient to move some of the hash buckets to another, less utilized sensor. The histogram in subfigure *b* exhibits a larger variation, suggesting that the network traffic associated with those few intense buckets may be primarily responsible for overloading that sensor. In this scenario, the load balancer promotes the top hash buckets to the next level of hashing to further spread the network traffic among the sensors. Note that this heuristic only approximates the ideal decision. First, the load balancer estimates the packet load associated with each hash bucket based on packet rates. Second, the decision whether to move or promote does not consider the shape of the histogram, but only compares the highest intensity buckets with the average load. On the other hand, this approach does not require the costly analysis of all hash buckets, and does not rely on overly detailed feedback information from the sensors. In fact, a sensor does not need to be aware of which hash buckets are currently assigned to it. Finally, this heuristic reduces the required storage on the load balancer, since only a small number of high-intensity hash buckets need to be tracked for each sensor, rather than the complete set.

Alternatively, sensors can guide the load balancer decision based on the packet buffer utilization. A sensor with a highly utilized packet buffer is more likely to drop packets and requires a more drastic load balancing response in the form of promoting buckets to the next hash level. In the case of a more lightly utilized buffer, even though it exceeds the flow control threshold, moving a number of buckets away from the overloaded sensor may still be sufficient to address the overload. Both schemes are quantitatively evaluated in the following section.

# 3. EVALUATION

## 3.1 Simulation Environment

To evaluate the performance of the NIDS load balancer, and to further refine the design and explore tradeoffs, a trace-based simulator is being developed. Trace-based simulation is the ideal tool for this purpose, as network traces can provide realistic workloads to the simulator without needing the simulator to go through the overhead of fully simulating traffic sources. This setup also closely corresponds to a real implementation where a NIDS platform is offered a certain traffic load without being able to influence it. The simulator implements a detailed and functionally accurate model of the load balancer operation, including the hash functions and tables, responses to flow control messages and promotion/demotion of hash buckets.

A configurable number of sensor nodes are modeled as finite packet buffers that drain packets at a rate based on packet size. When the buffer utilization reaches a configurable threshold, a flow control message is issued to the load balancer. The model used for this particular evaluation consists of 12 simulated sensors, each with packet buffers of the Linux default size of 64 kbytes. Flow control messages are generated when the buffer reaches 30% utilization, equivalent to 19660 bytes.

Note that the simulator does not model the actual intrusion detection software, as these details have no impact on the load balancer and would merely provide more accurate packet drain speeds. Instead, the main focus of the tool is to characterize the load balancer behavior and to refine design decisions such as suitable traffic intensity measures, time-out values and hash table sizes.

Two different network traces, both captured on the campus Interet connection of a major University, are used for the following evaluation. An older trace, representing a 45 MBit per second link, contains 1,402,229,593 packets captured over a 21 hour period on December 1, 2002. The second trace corresponds to a 200 MBit per second link and contains 38,599,994 packets, covering approximately fourty minutes of real traffic, captured on June 9, 2004.

## 3.2 Dynamic Feedback

One of the innovations of the PARNIDS loadbalancer is the dynamic feedback from the sensors to handle potential overload situations. To demonstrate the need for such feedback, Figure 4 shows the packet loss of individual sensors over time for the faster trace. The top graph shows a static hash-based traffic distribution scheme without dynamic feedback or adjustment. The bottom graph shows the same distribution scheme, but with dynamic adjustment employing both the moving and promotion of hash buckets.

Clearly, dynamic feedback is able to drastically reduce the number of lost packets. Without feedback, a total of 498,995 packets is dropped, while feedback reduces the total to 46,208. Note that these numbers correspond to relatively small drop rates of 1.3 and 0.12 percent, respectively. However, the NIDS loadbalancer must be designed for worst-case traffic scenarios, in which the loadbalancer itself is under attack. The traces used here do not contain such malicious traffic, hence the packet drop rates are smaller. Nevertheless, the effects are sufficiently strong to guide the design of the loadbalancer.

The feedback-directed loadbalancer is able to completely eliminate the packet loss spike encountered by sensor 8, as well as most other spikes in the second half of the trace. Only the initial spike incurred by sensor 6 remains, but it has been moved to a different sensor node as a response to feedback. The remaining packet loss indicates, however, that the scheme is not yet able to completely eliminate sensor overload, for a number of reasons. During the time it takes

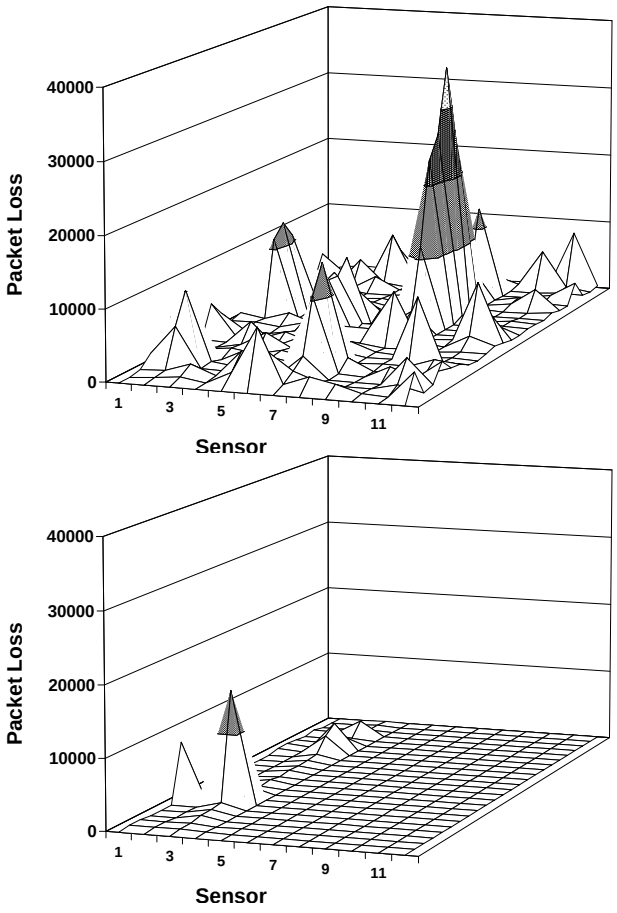


**Figure 4: Number of Packets Dropped per Sensor over Time**

to generate the flow control message and to adjust the hash bucket distribution, additional packets are forwarded to the sensor. If the remaining packet buffer is insufficient to hold these packets, some packet loss will be incurred. Lowering the flow control threshold provides more headroom to tolerate the response latency, but it can also increase the number of flow control situations, thus disrupting more flows. Furthermore, system parameters such as the number of hash buckets that are moved or promoted for each flow control message, as well as the rules guiding the decision whether to move or promote, have a direct impact on the loadbalancer performance. Some of the more important parameters are evaluated in the following section.

## 3.3 Flow Control Response

Perhaps the most critical design decision for the scalable loadbalancer is the heuristic used to determine which hash buckets are either moved to another sensor node, or promoted to the next level of hashing. While neither scheme has a clear advantage over the other in terms of minimizing packet loss, promotion is restricted by the number of hash levels supported by the loadbalancer. On the other hand, there is no limitation on how many buckets are moved, or how often.

Figure 5 summarizes load balancer performance in terms of

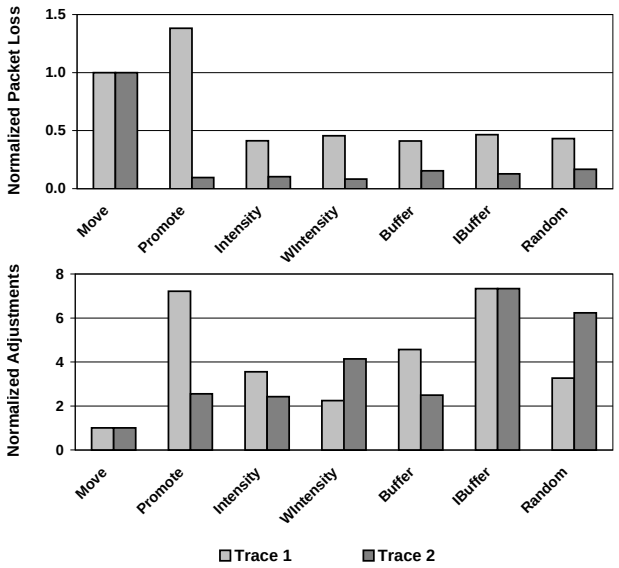


**Figure 5: Normalized Packet Loss and Loadbalancer Operations for Different Loadbalancing Heuristics**

packet loss for both traces and for a number of different heuristics. Packet loss is normalized to the always-move performance to enable a comparison between the two traces.

The static *always-move* and *always promote* schemes completely avoid the decision between moving or promoting hash buckets and simply apply only one of the two flow control response mechanisms. Overall, these schemes do not perform well compared to the other heuristics. The always-promote heuristic achieves mimimal packet loss for the second trace, but at the same time it is the worst-performing for the first trace.

The *intensity* scheme, as described above, observes the number of packets forwarded through each hash bucket and compares that traffic intensity to the average intensity for all buckets currently assigned to a sensor. If a particular bucket receives more than 20 times the average number of packets, it is promoted to the next hash level. This scheme attempts to estimate the load that each hash bucket exerts on the sensor. If the packet load exceeds a significant fraction of the sensors capability, it is promoted rather than reassigned to another sensor. The *WIntensity* approach is a variant of the previous scheme that measures packet rates as a weighted average rather than a simple packet count over a fixed-interval. *Buffer* refers to a heuristic that uses sensor packet buffer utilization to decide between moving and promoting. It encodes additional information about a sensors packet buffer in the flow control message. Generally, a flow control message is generated if the packet buffer is more than 30 percent full. If the buffer is more than 32.5 percent full, the sensor requests a promotion of the most intense hash buckets, otherwise the buckets are reassigned to the least busy sensor. *IBuffer* is the inverse of this scheme, it promotes buckets if the packet rate is below 32.5 percent, otherwise it moves them to another node. Finally, *random* probabilistically decides between moving and promoting, with a 50 percent likelyhood for either decision.

Overall, the heuristics based on packet rates or buffer utilization result in more predictable and consistent behavior and achieve better packet loss rates. In fact, the performance difference between these approaches is relatively minor, suggesting that perhaps the detailed implementation has little impact. Interestingly, the relatively simple random scheme achieves comparable packet loss rates at a lower hardware cost. In addition, this scheme introduces another degree of randomness into the loadbalancer, thus making it more difficult to predict and circumvent by an attacker.

To further evaluate the tradeoff between the various heuristics, the bottom graph in Figure 5 reports the number of hash buckets affected by a loadbalancer operation, either through moving or promotion. This metric provides an indication of how disruptive a loadbalancer adjustment is on the stateful analysis of the affected NIDS sensor, since overall loadbalancer performance is a combination of packet loss rates and the number of hash bucket adjustments. Again, results are normalized to the performance of the always-move scheme.

The graph shows that the performance improvement of the more sophisticated schemes comes at the expense of more frequent adjustments to the hash buckets. Furthermore, the inverted buffer-utilization scheme as well as the randomized scheme require a greater number of hash bucket adjustments to achieve packet loss rates comparable to the intensity heuristic. Overall, the intensity-based approach provides the best combination of packet loss and loadbalancer adjustments. The following sections investigate additional parameters to tune the performance of this heuristic.

## 3.4 Response Intensity

A second critical parameter in the loadbalancer design is the number of hash buckets involved in a flow control response. Moving too few buckets does not reduce the sensor load sufficiently to avoid packet loss. Although the overloaded sensor can issue additional flow control messages, the packet buffer may overflow by the time the loadbalancer has reacted. On the other hand, moving too many hash buckets can overload the new sensor to which these buckets are assigned. Hence, moving the appropriate number of hash buckets is important to minimize packet loss. Note that this aspect is somewhat less critical when promoting hash buckets, since promoting too many buckets does not negatively impact packet loss rates.

Figure 6 shows the packet loss when moving or promoting different numbers of hash buckets. Results are normalized to the packet loss when adjusting 32 hash at a time. In addition, the number of hash buckets affected by loadbalancing operations is shown, again normalized. The packet loss rate indicates that moving or promoting six hash buckets for each flow control message minimizes packet loss. However, the total number of load balancer adjustments, and thus the number of disrupted network flows, increases proportionally to the number of buckets affected. To further explore this tradeoff, Figure 7 shows the weighted sum of packet loss and hash bucket adjustments. Two overall performance measures are shown for each trace. The first metric gives packet loss a weight of ten compared to a bucket adjustment, while the second metric assigns a weight of 100

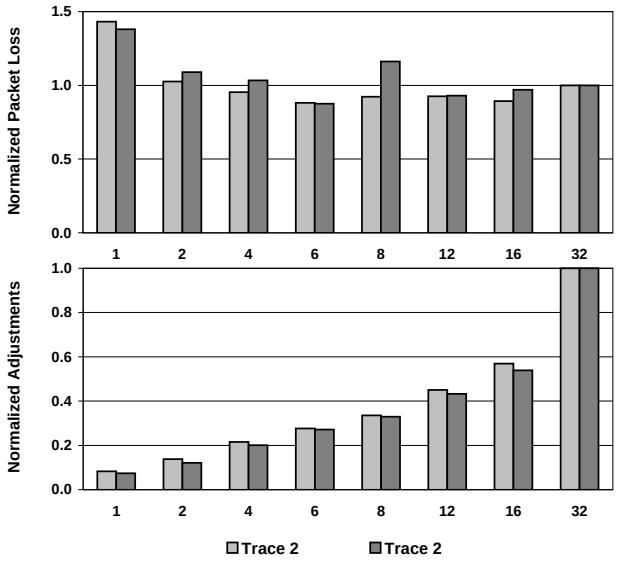


**Figure 6: Loadbalancer Performance when Moving or Promoting Different Numbers of Hash Buckets**

to a lost packet. The latter approach optimizes for packet loss at the expense of increased loadbalancer activity. This may for instance be desirable if the NIDS sensors employ only basic stateful analysis, and reassigning a network flow to a sensor has minimal impact on detection accuracy.

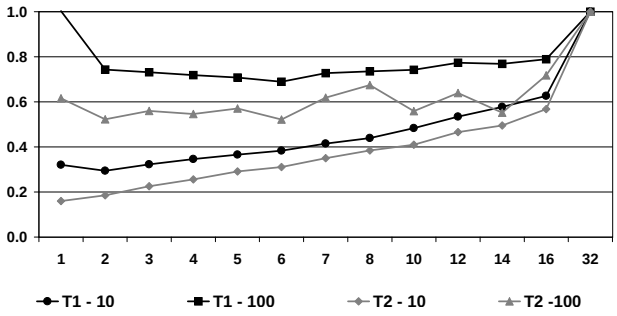


**Figure 7: Loadbalancer Performance Tradeoff when Adjusting Different Numbers of Hash Buckets**

The graph shows how the tradeoff changes as the significance of packet loss changes. For a packet loss weight of ten, adjusting only one or two hash buckets provides optimal performance. On the other hand, when minimizing packet loss is considered critical, adjusting more buckets further improves performance. These results suggest that the overall loadbalancer performance is indeed a tradeoff between packet loss and flow disruption. By adjusting the number of hash buckets that is either moved or promoted, it is possible to trade one measure for the other and thus optimize performance based on the needs of a particular installation.

A third design parameter of importance is the decision at which traffic intensity to promote a hash bucket rather than moving it. The intensity-based scheme compares the packet rate of the most intense buckets with the average rate to estimate whether a particular bucket overloads the sensor. If the packet rate of a particular hash bucket exceeds the

threshold, it will be promoted to the next hash level, otherwise it is moved. Raising this threshold further makes the loadbalancer less aggressive in promoting and may increase the number of packets dropped. Lowering the threshold reduces the risk of dropped packets, but may promote too many buckets and thus may exhaust the available levels of hashing more quickly.

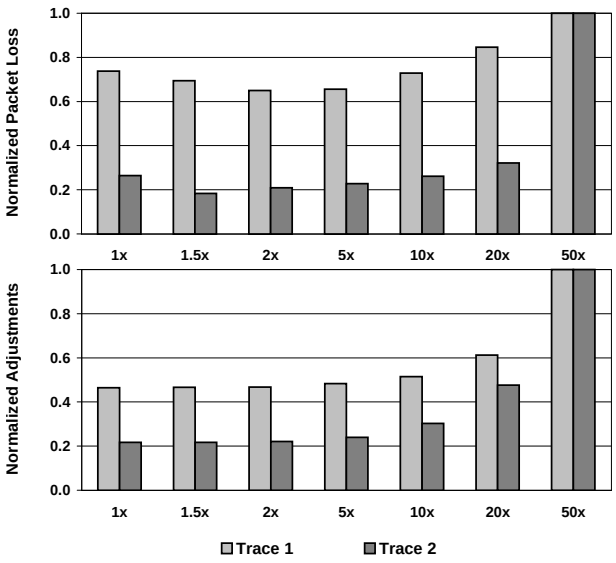


**Figure 8: Loadbalancer Performance when Changing the Threshold to Promote Hash Buckets**

Figure 8 reports the loadbalancer performance for varying promotion thresholds, again normalized to the performance using a 50-times threshold. Promotion threshold is given as a factor applied to the average rate. For instance, a threshold of 10x means that buckets are promoted if the intensity is greater than ten times the average rate. Increasing the threshold improves packet loss rates up to a point. Extremely low threshold effectively implement the always-promote heuristic, since almost all hash buckets will exceed the threshold and will be promoted. As seen previously, this approach results in unstable and extreme performance and is not well suited as a loadbalancer heuristic. However, raising the threshold further increases the packet loss since high-intensity buckets are more likely to be moved rather than promoted, thus only overloading the new sensor node. Interestingly, the number of hash buckets affected by load-balancing does not change significantly for all but the highest thresholds.

# 4. PROTOTYPE IMPLEMENTATION

## 4.1 Prototype Architecture

The scalable distributed NIDS architecture is currently being implemented as a fully-functional prototype system. This effort ensures that the design decisions described in this paper result in a truly scalable system capable of processing packets at wire speed, and also facilitates extensive performance characterizations under realistic network conditions. The prototype system consists of a Xilinx Virtex-II FPGA hosted in the PCI slot of a Linux PC, a commodity switch that implements the data plane of the load balancer, and a collection of rack-mount systems as sensor nodes, as shown in Figure 9.

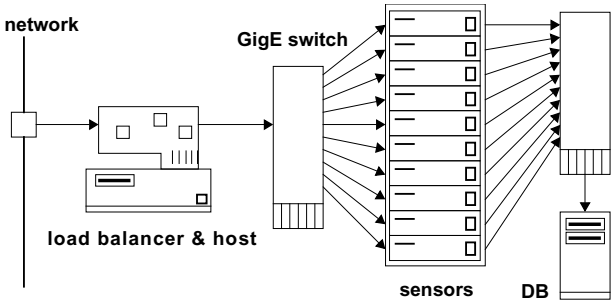


**Figure 9: Prototype System Architecture**

For each packet, the control logic determines the destination sensor and rewrites the destination MAC address accordingly. The external switch then forwards the packet to the associated sensor. Splitting the actual load balancer into the FPGA-based control logic and a data plane reduces the number of network ports required on the FPGA board, and also reduces the logic design effort. Sensor hosts are running the open-source Snort network intrusion detection software [11]. A custom kernel module implements a raw packet interface that monitors its buffer utilization and issues flow control messages as necessary. A second switch routes alert messages from the sensors nodes to a database system for further analysis and long-term storage.

## 4.2 Implementation

The load balancer control logic is implemented in a Xilinx XC2V6000 FPGA and is hosted on a PCI board that also provides two Gigabit Ethernet interfaces. The Virtex FPGA integrates 65536 flip flops and associated 4-entry lookup tables, as well as 144 blocks of 18-kbit dual-ported SRAM. The microarchitecture of the load balancer control is shown in Figure 10.

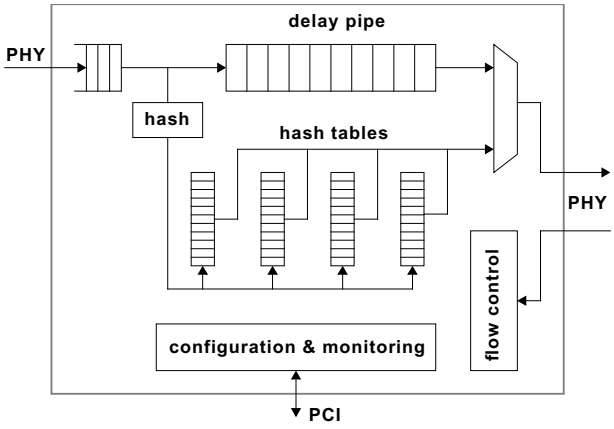


**Figure 10: Load Balancer Microarchitecture**

When receiving a packet, the load balancer first synchronizes it to the clock of the outgoing PHY interface through an asynchronous FIFO. It then latches and decodes the IP header as well as portions of the layer-3 header, calculates the hash indexes and performs the first lookup. Four hash

tables of 4096 entries each are implemented in the on-chip SRAM blocks. Each entry contains a sensor number, a traffic intensity measure and a bit indicating that a bucket is elevated to the next level of hashing. If necessary, the routing logic performs further lookups in the additional hash tables, before producing a sensor number.

While the routing decision is computed by the control state machine, the original packet passes through a delay pipeline from which it emerges at the same time the routing is complete. At this point, the original destination MAC address is replaced by the MAC address of the target sensor. Note that the payload of the packet remains unaffected by the routing, only the physical layer MAC address that is not used by the NIDS software is modified. The outgoing network port then recalculates the Ethernet checksum and transmits the packet to the external switch. Flow control packets arrive on the receive-port of the outgoing PHY interface. These frames are also latched and decoded, and trigger the required adjustments in the routing stage.

The internal data path is 16 bits wide and runs at 62.5 MHz. A minimum-size network packet requires 32 cycles to transmit, plus 4 cycles of preamble. Combined with a 6-cycle minimum inter-packet gap, the load balancer must thus be able to handle a new packet every 42 cycles. Since the system must sustain network traffic at wire speed to be immune to denial-of-service attacks, no on-chip buffering is implemented. Instead, all control logic is pipelined as necessary to achieve the required packet rate.

In addition to the actual load balancing logic, the FPGA also implements a detailed performance monitoring logic. For each sensor, the performance monitor records the number of packets and the number of bytes forwarded, as well as the number of flow control messages received. This data, along with other summary statistics such as total packets and number of elevated buckets, is made available via the PCI interface. Monitoring software running on the host system reads this data and presents it in graphical form, to aid in detailed performance evaluation. Note that the host system is only involved in configuring and monitoring the load balancer, all network traffic flows only through the FPGA.

During startup, the load balancer issues a broadcast UDP packet, requesting a notification message from each attached sensor. Sensors signs on to the load balancer by providing their MAC and IP addresses, which are stored in a table that associates sensor numbers with MAC addresses and vice versa.

At the time of this writing, all peripheral logic surrounding the actual routing logic is implemented and tested. The board is correctly receiving and forwarding Ethernet frames. Sensor nodes sign on with the board, and flow control packets are received and decoded. Ongoing work focuses on identifying implementations of the basic routing algorithm that are suitable for hardware implementation. For instance, since the hash tables are implemented as SRAM arrays for space-efficiency, they support only two accesses per cycle. Checking timeout values in all table entries and periodically clearing the intensity measures requires iterative logic that competes with regular lookup operations.

# 5. RELATED WORK

Network intrusion detection is an active field of research, constantly developing new approaches and techniques. The PARNIDS project leverages these improvements by utilizing off-the-shelf sensor hardware and software. The key contribution of this work, the scalable load balancing approach, provides a significant capacity improvement, independent of and orthogonal to any improvements in the actual NIDS sensor software or hardware. While the prototype system is currently using Snort [13][11], other approaches such as protocol analysis [8] benefit equally from the parallel system organization.

Previous work in parallel network intrusion detection has employed flow-based distribution strategies [3][6]. Similar to conventional routers, these systems maintain tables of established connections to route packets to the appropriate sensor nodes. This approach supports flow-based intrusion analysis at the sensors, but it makes the overall system vulnerable to denial-of-service attacks. The PARNIDS architecture does not exhibit this scalability limitation. In addition, existing systems have not used dynamic feedback to improve system robustness and adaptability.

The distribution of network traffic over clusters of nodes is commonly used in network services such as web servers [2][4]. Such load balancers often maintain per-flow state to ensure that connections are not disrupted. This approach is acceptable in these systems since it is possible to reject new connection requests. A NIDS platform, on the other hand, is not part of the actual network conversation and has no means to throttle network traffic, and therefore must be able to handle the maximum possible load.

Network intrusion detection performance has been measured extensively, both in terms of capabilities [5][9] and capacity [10][12]. The work presented here is motivated by the realization that increasing network speeds make intrusion detection systems vulnerable to overload scenarios. It directly addresses the capacity bottleneck, while also developing and refining methods to evaluate NIDS performance.

# 6. CONCLUSIONS AND FUTURE WORK

Network intrusion detection is one of several important security measures commonly employed to secure critical data. With increasing network speeds, the capacity of the NIDS platform is becoming a bottleneck that can compromise the effectiveness of the entire system. Parallel NIDS platforms are a viable solution to address this problem.

This paper discusses the unique requirements for a parallel network intrusion detection platform, and then describes a cost-effective yet scalable solution. A custom NIDS load balancer distributes the processing load over an array of sensor nodes to minimize packet loss. It employes a scalable multi-level hashing technique to minimize NIDS vulnerabilities and to adjust to changing network traffic characteristics. The main contribution of this approach is the design of a loadbalancing approach that does not maintain per-connection state while still supporting stateful flow-based intrusion detection. Furthermore, the PARNIDS loadbalancer incorporates dynamic feedback from sensor nodes to adjust to changes in network traffic and processing load,

thus further minimizing packet loss.

Evaluation results demonstrate the performance potential of this approach. The best loadbalancing heuristic not only depends on the packet loss rates but also on the number of loadbalancing operations required to achieve this performance, since each adjustment potentially disrupts network flows from the sensors point of view. An equally important consideration is the implementation complexity of different heuristics. Under these considerations, a heuristic that estimates the processing load that hash buckets exert on the sensor nodes outperforms all other schemes. Additional parameters for this approach also have a noticable impact on overall performance, allowing the fine-tuning of packet loss or loadbalancer activity to meet the needs of particular environments.

Future work includes the design and investigation of techniques to make the load balancing heuristics less deterministic and predictable, thus further reducing the vulnerability of the NIDS platform to sophisticated evasion techniques. For instance, the load balancer can randomly choose one of several hash functions when promoting hash buckets to the next level, or can vary the number of buckets being moved or promoted when responding to flow control feedback.

Other avenues for future work include the development of techniques to transfer flow information maintained by individual NIDS sensors in the event that the loadbalancer moves a flow to another sensor. Furthermore, the correlation of observed events will become more important and parallel NIDS architectures must include light-weight communication mechanisms to facilitate the exchange of such information.

The authors would like to thank the entire PARNIDS team for their hard work on various aspects of this project. This material is based upon work supported by the National Science Foundation under Grant No. 0231535. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## 7. REFERENCES

[1] tcpdump/libpacp. URL: http://www.tcpdump.org.

[2] O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang. One-ip: Techniques for hosting a service on a cluster of machines. *Computer Networks and ISDN Systems*, 29(8-13):1019 – 1027, 1997.

[3] S. Edwards. Vulnerabilities of network intrusion detection systems: Realizing and overcoming the risks. the case for flow mirroring, 2002.

[4] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *ACM Symposium on Operating Systems Principles*, pages 78–91. ACM Press, New York, N.Y., 1997.

[5] J. Haines, R. Lippmann, D. Fried, J. Korba, and K. Das. 1999 darpa intrusion detection system evaluation: Design and procedures. Technical Report 1062, MIT Lincoln Laboratory, Boston, Mass., 2001.

[6] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *IEEE Symposium on Security and Privacy*. IEEE CS Press, Los Alamitos, Calif., 2002.

[7] Network ICE Corp. *Protocol Analysis vs. Pattern Matching*, 2000.

[8] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[9] N. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996.

[10] M. Ranum. *Experiences Benchmarking Intrusion Detection Systems*. Network Flight Recorder Security, Inc.

[11] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Usenix LISA '99 Conference*. Usenix Society, Berkeley, Calif., 1999.

[12] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Sixth International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, pages 155–172. Springer-Verlag, Berlin - Heidelberg - New York, 2003.

[13] Sourcefire Network Security Inc. *Snort 2.0 - Detection Revisited*, 2002.

[14] Xilinx Corporation. *Virtex II Datasheet*, 2001. DS-031-1.