In: *Network Processor Design: Issues and Practices, Volume 2*, Eds. P. Crowley et al., chapter 12, pp. 243-272, Morgan Kaufmann Publishers, San Francisco, 2004.

High-speed Legitimacy-based DDoS Packet Filtering with Network Processors: A Case Study and Implementation on the Intel IXP1200

Roshan Thomas[*], Brian Mark[+], Tommy Johnson[*], James Croall[*]

[*]Network Associates Laboratories
Network Associates, Inc.
1145 Herndon Parkway, Suite 500
Herndon, VA 20170
{rthomas, jcroall, tjohnson}@nai.com

[+]Dept. of Electrical and Computer Engineering
George Mason University
4400 University Drive
Fairfax, VA 22030
bmark@gmu.edu

Abstract

We present an interesting application of network processors in the field of network security. Specifically, we report on the design and implementation of a high-speed prototype to provide packet-filtering functions to mitigate distributed denial of service (DDoS) attacks that target network resources. The effects of DDoS attacks are felt when network or host resources are used illegitimately at the expense of legitimate services. Our approach thus relies on administering a variety of legitimacy tests in real-time to incoming packets so as to determine their degree of legitimacy followed by appropriate filtering and traffic management. To be practical, such tests and filtering have to be done in a manner that can sustain high throughputs and this presents a variety of technical and research challenges. We collectively refer to our approach and set of technologies as "NetBouncer". NetBouncer represents an interesting case study for the application of network processors as it presents much more complex functionality when compared to traditional packet processing applications and devices such as routers and firewalls. Our implementation experience thus far has given us important insights into the possibilities as well as the limitations of network processors for building applications that go beyond simple IP packet forwarding.

1. Introduction

Distributed denial of service (DDoS) attacks have presented a growing concern to the security and networking communities as well as the industry at large. These attacks first received wide attention in February 2000 when some well-known web sites (such as Yahoo, Amazon, CNN) and electronic commerce companies were attacked. The DDoS problem is considered by many to be one of the most difficult to solve and the situation is exacerbated by the reality that these attacks are increasing in sophistication with many hacker sites now distributing free attack toolkits.

In this paper, we present the development of a high-speed packet filtering solution using network processors so as to provide a defense against DDoS attacks. Collectively, our solution is referred to hereafter as "NetBouncer". At its core, the DDoS problem is fundamentally an availability problem that arises from the "illegitimate" use of resources. Thus, central to the NetBouncer approach to DDoS mitigation is the ability to distinguish legitimate traffic from illegitimate traffic and the ability to prioritize processing of legitimate traffic in a manner that gives legitimate packets the highest priority for service. A successful NetBouncer solution is independent of individual DDoS attack and vulnerability specifics.

In its current form, NetBouncer technology consists of high-speed packet processing and filtering devices. From a high level concept of operations standpoint, the working of a NetBouncer device is very simple. On receipt of an incoming packet, a device has to make one of three decisions: (1) accept and schedule transmission of the packet if it is determined to be legitimate; (2) discard the packet or (3) challenge the sender of the packet to prove legitimacy of the packet. To enable filtering of incoming packets, a NetBouncer device maintains a large legitimacy list of identifiers of clients that have been proven to be legitimate. If packets are received from a client (source) not on the legitimacy list, a NetBouncer device will proceed to administer a variety of legitimacy tests to challenge the client to prove its legitimacy. If a client can pass these tests, it will be added to the legitimacy list and subsequent packets from the client will be accepted until a certain legitimacy window expires. Once accepted, the transmission of legitimate packets is controlled by a traffic management subsystem

that applies various bandwidth allocation and rate limiting schemes to ensure that legitimate clients do not abuse bandwidth consumption and that target servers cannot be overwhelmed even by what appears to be legitimate traffic.

In theory, the use of legitimacy tests on incoming packets allows us to prevent illegitimate use of network bandwidth. However, to be practical, NetBouncer has to operate in a manner that can meet the scalability and performance needs of high bandwidth, real-world commercial and military environments. In particular, our objective is to create a solution that is easy to integrate and has low technology insertion cost, scalable in terms of network topology complexity and network speeds, imposes minimal administrative overhead, and requires minimal collaboration and information exchange across organizational network infrastructures and Internet service provider (ISP) networks.

Meeting the above needs poses several research, design and architecture challenges. As such, the NetBouncer approach and design incorporates several innovative elements including:

- Novel techniques to test for the legitimacy of network traffic using state-safe[1] legitimacy tests and the subsequent enforcement of access controls on traffic.
- Algorithms to enable efficient look-up and updates of very large legitimacy lists.
- Quality-of-service (QoS) related traffic management schemes to provide rate limiting and bandwidth management for various classes of traffic based on client legitimacy and service priorities.
- Hardware-assisted high-speed packet processing techniques and architectures using network processors to implement the above functions, so as to achieve high performance.

The focus of this paper is on the design and implementation of a NetBouncer high-speed prototype device using the Intel IXP1200 network processor as this will be of most interest to the NP workshop attendees. A much broader overview of the NetBouncer research project including research topics such as traffic management are beyond the scope of this paper but are discussed in [23]. The use of programmable hardware such as the Intel IXP1200 network processor chip was motivated by several factors including the promise of building a low-cost rapid prototype that can operate at high line rates. A Linux-based software prototype of NetBouncer implemented on conventional PC hardware and software could only sustain about 10Mbps of throughput.

It is our belief that the NetBouncer approach and its implementation using network processor technology represents a case study that highlights the benefits as well as the limitations of low-end network processors which will be of interest to the broader NP research community. In particular, NetBouncer functionality places unique demands on packet processing that go beyond traditional packet processing applications such as IP routing (as reported in [20]), packet switching and forwarding, as well as established security applications such as firewalls and virtual private networks (VPNs). For example, the use of legitimacy tests to challenge incoming packets introduces substantial complexity and additional data paths in the packet processing architecture when compared to common IP routing and forwarding applications. Another complication is the use of cryptographic techniques to authenticate and validate messages exchanged to administer legitimacy tests.

The maintenance of large legitimacy lists is also complicated. It is not unrealistic to think of a large e-commerce site possibly utilizing a legitimacy list of about half a million entries. This is in contrast to routers where the routing tables are typically of the order of about 80,000 entries. Also, when compared to routing table entries that evolve slowly, the legitimacy list may be updated very frequently, on the order of several hundred updates every minute. Further, the amount of memory and CPU cycles available for list storage and manipulation may be limited on a network processor. Collectively, these represent interesting challenges and call for memory-efficient schemes for representing large tables and for algorithms that can simultaneously be efficient at processing lookups as well as table updates. Another challenge is the maintenance and tracking of legitimacy. This requires that we maintain legitimacy state at several level of abstraction including individual packets, flows and application sessions and understand how states at these different levels are related. Maintaining and consulting such state information introduces additional overhead on packet processing. Lastly, the integration of traffic management schemes that incorporate legitimacy weights into service priorities and queuing disciplines present interesting problems (although we do not cover this topic in this paper).

The rest of this paper is organized as follows. Section 2 discusses some background material on our approach to DDoS mitigation using legitimacy tests and the management of legitimacy lists. Section 3 elaborates on our prototype built using the Intel IXP1200 network processor. Sections 4 and 5 cover our performance testing experiments. Section 6 discusses lessons learnt and

[1] The concept of state-safety can be informally defined as a property that guarantees that the consumption of state at a NetBouncer device cannot be directly induced by the proportion and rate of illegitimate traffic.

architectural directions that we desire in network processors and section 7 concludes the paper.

2. Background: Legitimacy Tests and Legitimacy List Management

2.1. Client-based Legitimacy Tests and DDoS Mitigation

In response to the growing DDoS problem, we have seen the emergence of a variety of vendor-supplied as well as research-oriented solutions. Typically, these are based on a variety of techniques such as traffic sampling and analysis using statistical techniques and traffic profiles [2, 3, 4, 5, 6, 7, 8], use of DDoS attack signatures to provide DDoS specific traffic diagnosis [2, 3], rate limiting mechanisms [4], as well as better connection and time out management and congestion control [15]. Other techniques use network-wide automated tracing and blocking of attacks as close to the attacker as possible [18, 19].

The NetBouncer approach differs from the above solutions in that it is based on distinguishing legitimate traffic from illegitimate traffic. A NetBouncer device is intended to provide localized DDoS protection close to a potential victim but upstream of the local chokepoint. If we examine the issue of legitimacy more closely, we come to the realization that determining if traffic is legitimate, in turn, requires us to determine if the origin of the traffic (the client) itself is legitimate in relation to the target of the traffic (such as a server). In general, determining legitimacy may require us to abstract and analyze the traffic at one or more levels of the protocol stack. The NetBouncer approach relies on a series of legitimacy tests, with each test targeted for a particular type of traffic recognized at a specific protocol layer. The source of the traffic and the related notion of the client identity that is validated (also called the legitimacy identifier) will depend on the protocol layer (e.g., network, application etc.) and the application or service (ftp, real video, etc.) for which a test is administered.

Thus, it makes sense for a legitimacy test at the network layer to determine the validity of a host or router as identified by its IP address. At the transport layer, the legitimacy tests may be aimed at validating TCP connections. At the application layer, our legitimacy tests will attempt to determine valid application sessions and user processes and identifiers. Depending on the circumstances and the application, we may apply in succession, a combination of tests for various protocol layers and application level notions such as sessions.

Our ongoing research has led us to identify at least three categories of legitimacy tests. Specifically, these include packet-based tests, flow-based tests and service-based tests. Informally, a legitimacy test ***t*** is a tuple <assertion, legit-id, pre-scope, pre-state, post-state, post-scope>. The objective in administering a test is to examine a set of entities (called the pre-scope) such as a packet, protocol flow or application session in a manner that can validate a legitimacy assertion for a given legitimacy identifier (legit-id) such as a source-IP address, session-id, etc. The pre-state is the state that is used to examine and validate the assertion and the post-state is used to process legitimate packets after legitimacy has been established. The post-state specifies the set of entities to which legitimacy is to be associated with.

2.1.1. Packet-based Tests

The distinguishing characteristic of a packet-based legitimacy test is that the pre-state and post-state that need to be examined are fully self-contained in an individual packet. In other words, a decision as to whether a packet can be passed is made solely on examining the contents of the packet. In particular, there is no information from previously seen packets that needs to be retained and consulted. We define a "flow" as a stream of packets from a particular protocol connection/session (at layer 4 in the network stack) and thus having the same (<source-address, source-port> <destination-address, destination-port>) properties. In contrast to a packet-based test, for a flow-based test, the post-state is not contained within individual packets and more significantly a single post-state applies to the entire stream of packets that belong to a flow. In contrast to packet-based and flow-based tests, the category of service-based tests is relevant for higher-level (above layer 4) services and applications. In particular, these tests understand application-level abstractions and structures as well as session semantics. Testing for legitimacy requires an understanding of the structure and semantics at the application-level. Thus a packet-based and flow-based examination of traffic at the level of IP packets and flows will generally not provide conclusive evidence of the legitimacy of application traffic. In an application-level test, the pre-state may be spread across several IP packets and applying the post-state may require examination of application-level headers.

A simple example of a packet-based test is one that attempts to validate for every incoming packet the assertion : "*There exists a valid and live host at the claimed source IP address of the incoming packet* ". One way to realize such a test is through the creative use of ICMP echo messages (and hence referred to as the "ICMP Ping test"). This is useful in filtering DDoS traffic as many DDoS attacks use spoofed and bogus source IP addresses so as to make attack tracing difficult. One way to implement such a test is for NetBouncer to intercept every incoming packet and look up its source IP address on the legitimacy list. If no match is found, NetBouncer can challenge the legitimacy of the source IP address by

issuing an ICMP echo message. However to avoid storing any state in NetBouncer so as to make the test state-safe, the original incoming request packet is encapsulated in the payload of the outgoing ICMP echo request. If the original sender of the packet is a valid host, NetBouncer can expect to get back an ICMP echo reply packet. However, we need to authenticate such replies and verify their integrity. To enable this, the payload of the ICMP echo request also includes a hashed message authentication code (HMAC) computed using a keyed hash function and taking as input a packet tag, the incoming request packet, source IP address, payload-length, expiry-time, and a nonce (randomly generated number). If an ICMP echo reply is received and the HMAC can be verified, the authenticity and integrity of the ICMP echo reply is verified and the source IP address of the extracted incoming request packet is added to a legitimacy list consisting of validated source addresses. The HMAC is an example of the pre-state.

2.1.2. Flow-based Tests

A good example of a flow-based test is one that can be used to validate TCP connection requests for their legitimacy. Such a test provides a defense against denial-of-service attacks based on TCP SYN floods where an attacker repeatedly issues TCP connection requests but never responds to the handshaking sequence required to complete the connection request. This typically results in the overflow of the connection table at various hosts due to the state reserved for half-open connections, the dropping of existing TCP connections and the inability for new legitimate TCP connection requests to complete. This DDoS condition can be attributed to the fact that in many traditional TCP implementations, as soon as the first TCP SYN packet is received, state is allocated for the connection. This violates state-safety.

We now describe a flow-based test to validate TCP connection requests, referred to hereafter as the "TCP SYN Cookies" test. This test adapts the idea of SYN Cookies described in [1]. To implement such a test, a NetBouncer device basically intercepts TCP connection establishment requests. To elaborate, NetBouncer intercepts SYN packets and generates SYN/ACK packets that store a cryptographic checksum (cookie) of private rotating keying material, various fields in the TCP/IP header and a NetBouncer generated sequence number (this cookie is effectively the pre-state). The SYN/ACK is then returned to the source address of the SYN packet. When the ACK (which is the response from the client to the SYN/ACK) arrives at the NetBouncer, if the cookie (pre-state) verifies, state is instantiated for the connection, and the original SYN along with client supplied sequence number is recreated based on the information in the SYN/ACK packet and the stored cookie and forwarded to the original destination server. However, the original destination will then return a SYN/ACK server sequence number of its own, and NetBouncer will complete the 3-way handshake. NetBouncer will also retain the offset between its sequence number and the server sequence number, allowing it to transform the sequence number on each packet on the TCP connection from and to the server through NetBouncer. This offset and related information form the post-state for the test.

2.1.3. Service-based Tests

In contrast to packet-based and flow-based tests, the third category of service-based tests is relevant for higher-level (above layer 4) services and applications. In particular, these tests understand application level abstractions and structures as well as session semantics. Testing for legitimacy requires an understanding of the structure and semantics at the application-level. In reality, this category of tests may be thought of as consisting of two subcategories. The first is what we call *structured composite services* (SCS). These consist of services and protocols such as the real-time streaming protocol (RTSP). The structure of an RTSP session can be thought of as a composite one consisting of may underlying lower level protocol sessions and connections (including TCP, UDP and RTP, etc.). However, the exact structure is fixed by the RFCs and standards. The second subcategory consists of *ad-hoc composite services* (ACS). The structure of ACS services is also a composite one but varies from one environment to another rather than being defined by a standard. A simple example of an ACS service would be one where a user clicks on a URL at a web site and the server subsequently downloads one or more applets to the client machine. These applets may subsequently initiate additional network connections and services depending on the application logic and transaction structure. So the critical challenge here is to understand how legitimacy state is preserved and tracked through various component sessions and application interactions.

A simple example of a service level test is one that tries to determine if a human user really exists at the client end when a service request is issued. The need for service level tests arises from the observation that as validation methods and countermeasures against DDoS attacks become more powerful, it is only natural to expect attacks to move up from the protocol level (layer 3 and layer 4) to the application and service levels. During a DDoS attack, it is critical to distinguish attackers from non-threatening users. Since most DDoS attacks are composed of a handler guiding a large number of automated attack agents [11], we can use the non-intelligence of the digital agents to distinguish them from intelligent human users. The key idea is to interrupt an application/service session and challenge the client host with a question only a human user can answer. In our

current hardware prototype of NetBouncer, we have completed an initial implementation of this test at the application level and integrated it with HTTP/HTML. In this implementation, a user (client) initiating an HTTP (web) request from a web browser is confronted with a question (i.e., a puzzle). If the user can supply the correct answer, he is added to the legitimacy list. The test works by NetBouncer intercepting the connection establishment TCP SYN packet from the client and completing the TCP 3-way handshaking procedure. When the client subsequently follows up with an HTTP `get` request, NetBouncer will issue a challenge by posting a puzzle in an HTML form. However, the correct answer to the puzzle is cryptographically sealed and sent with the form so as to preserve state safety. If the client responds correctly, as verified by comparing the user supplied answer with the cryptographically sealed and extracted answer, NetBouncer will then send an HTTP `refresh` request to the client's browser and this will result in the client issuing a second HTTP `get` request that NetBouncer routes directly to the server.

Due to space constraints, we omit a full exposition of the details of the above tests in terms of the various message exchanges and cryptographic operations, but details are given in [23]. We are currently developing a formal framework to model legitimacy tests and reason about their correctness and safety properties. We are currently investigating popular services and protocols such as FTP, RTSP, H323 etc. in order to understand the complex structure of application sessions and more importantly studying how legitimacy can be tracked and traced so as to provide legitimacy-based filtering of application traffic.

2.2. Approaches to Legitimacy List Management

Having discussed legitimacy tests, we now turn our attention to the organization of legitimacy information.

2.2.1. Organizing legitimacy state information

Legitimacy state information serves two purposes. First, it is consulted to quickly determine if an incoming packet is legitimate. Second, it is used to provide any required processing on a legitimate packet such as header modification as well as scheduling for traffic management purposes. Also, the need to support packet-based, flow-based and service-based tests requires that we maintain several related pieces of information on the incoming traffic. Briefly, these include:

- The identifier (IP address) of the source of the traffic.
- All incoming protocol-level flows originating from a host.
- All higher-level application sessions (services) emanating from a host.
- For each protocol-level flow, a link to the application-level information for the instantiated flow.

To support fast lookups, we have divided the legitimacy information and the lookup problems into two parts. The first part consists of extracting the source address of the incoming packet and checking a host lookup table (HLT) of IP addresses to see if there is a match. The second part consists of traversing and checking a more elaborate structure called a legitimacy state entry (LSE) for more specific information regarding flows and application sessions in relation to the matched host. The LSE in turn consists of two substructures. The first is called the flow-tree (FT) and is basically a binary tree of nodes with each node representing a single flow from a host. The second substructure is a linked list structure called the application list (AL) where each node represents a unique application or service. An application object serves to bind together all individual flows for a single application session. The relationship between flow objects and application objects is many-to-one and is represented by a back pointer from every flow object to its application object.

Several alternate approaches can be used to provide fast lookups into the host lookup table (HLT). Figure 1 shows a representation of legitimacy state information with the lookup on the HLT based on a hash table. Thus every bucket may lead to linked list of host objects with each host object supporting an LSE. The basic sequence for traversing legitimacy information is as follows:

- Extract client's IP (source) address from the packet and find a match in the host lookup table.
- Extract the flow tuple consisting of <protocol, source address, source port, target IP address, target port> and find a match in the flow-tree.

Traversal of the flow-tree is based on a binary search of a string representing the flow tuple. From every host node, we have a flow pointer which points to a tree of all flow nodes for that host. The links from the objects in the flow-tree to objects in the application-list are provided to implement higher-level service-based legitimacy tests, as well as QoS and traffic management on an application and per host basis.

2.2.2. Efficient lookup and update strategies

The lookup function on the list of legitimate host IP address in NetBouncer differs from the table lookup function implemented in conventional IP routers in the following aspects:

- The lookup is based on the full source IP address, rather than a prefix of the destination IP address.

- New legitimate clients are added on a much faster time-scale to the legitimacy list compared with typical IP routing tables.

Also, there is limited SRAM and SDRAM memory on a network processor such as the IXP1200 and thus the lookup schemes have to be space efficient. These requirements make the design and implementation of the legitimacy list a challenging task. A NetBouncer legitimacy list is expected to be significantly larger than typical IP routing tables (on the order of few hundred thousand entries). On the other hand, we remark that longest prefix matching is not a required feature of the NetBouncer lookup function.

Current approaches to fast route-table lookup include hardware-based solutions, software-based solutions, and hybrid solutions. Specialized CAM (Content Addressable Memory) hardware engines [16, 25] can achieve the best performance in terms of lookup speeds. However, large CAMs are very expensive and the cost grows significantly as the number of entries in the table increases, making them an infeasible approach for handling the large legitimacy list requirements of NetBouncer. Other approaches to table lookup include the use of hash tables [24] and variations on binary search trees, also called tries [12, 17, 21]. These structures do not require specialized hardware and are suitable for implementation on network processors such as the Intel IXP1200.

The hash table approach employs a hash function, which maps the host IP address into one of N hash buckets. Each bucket has an associated linked list containing host objects that have been hashed to the same bucket. In the trie approach, the computational complexity of a lookup or insert operation is generally logarithmic in the number of entries. Valid entries in the table correspond to leaf nodes in the trie. The non-leaf nodes of the trie, which are required in the search process, can incur a substantial memory space overhead. Paths in the tree that do not have branches can be compressed into single nodes, resulting in the so-called Patricia trees. The LC (Level-Compressed) trie proposed in [17], further employs "level compression," whereby the i highest complete levels of the trie are replaced by a single node with 2^i branches; this replacement is performed recursively on each subtree. LC-tries generally have average depths that are significantly smaller than those of Patricia tries, resulting in smaller lookup times and less memory to store the lookup structure. On the other hand, the compact structure of the LC-trie makes it comparatively difficult to perform insertions or deletions in an efficient manner. Doeringer et al. [12] propose a variant of a binary trie structure that accommodates dynamic updates, but incurs a relatively large space overhead and does not exploit level compression.

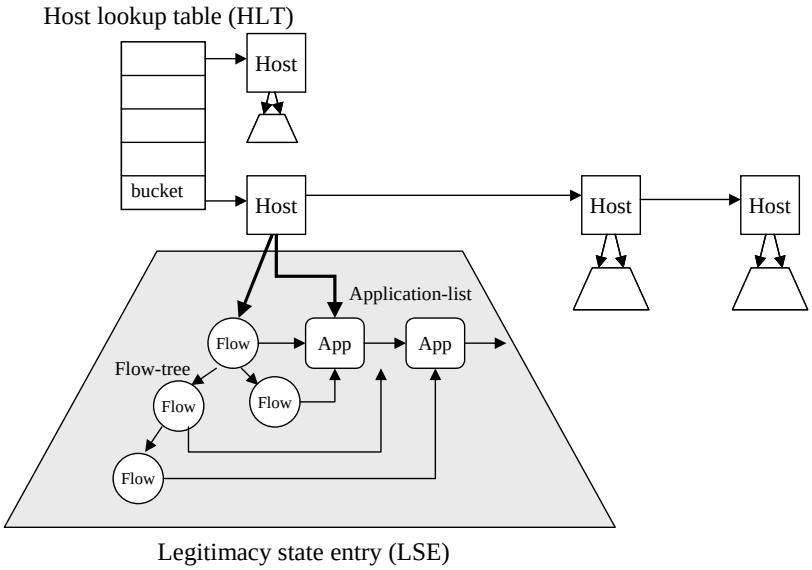

Figure 1. Organizing legitimacy state information with a hash table.

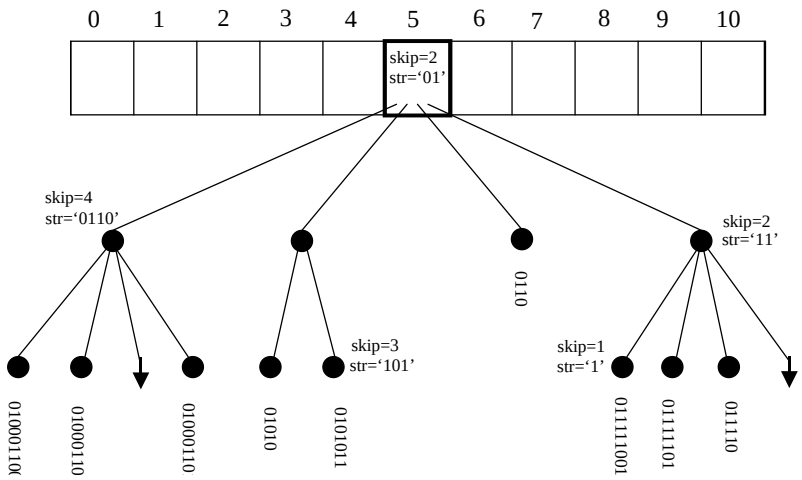

Figure 2. Illustrating the hash-trie structure

2.2.3. Hash-trie structure

To meet the requirements of legitimacy list management, we are developing a novel structure called the "hash-trie," which combines features of hash tables and LC-tries in order to perform efficient lookups and updates. Please note that this has not yet been implemented on our prototype. The hash-trie consists of a hash table in which collisions at each bucket are resolved by a variant of the trie stucture, which we call the "LC-*m* trie." The hash-trie allows us to exploit the efficient hash function provided by the IXP1200 and avoids the potentially time-consuming linear searches that may occur in the conventional hash table structure. The LC-*m* trie exploits path compression and up to *m* levels of level compression, but unlike the LC-trie, updates can be performed efficiently.

In the LC-*m* trie, a node contains pointers to up to 2^m child nodes. Each node contains a "skip" parameter for path compression, and a "branch" parameter, where the value of "branch" can be at most *m*. Thus, up to *m* levels in an equivalent binary subtrie can be (level) compressed into a single node in the LC-*m* trie. To facilitate update operations, each node also contains a parameter "str," which contains the actual bit sequence that was skipped in

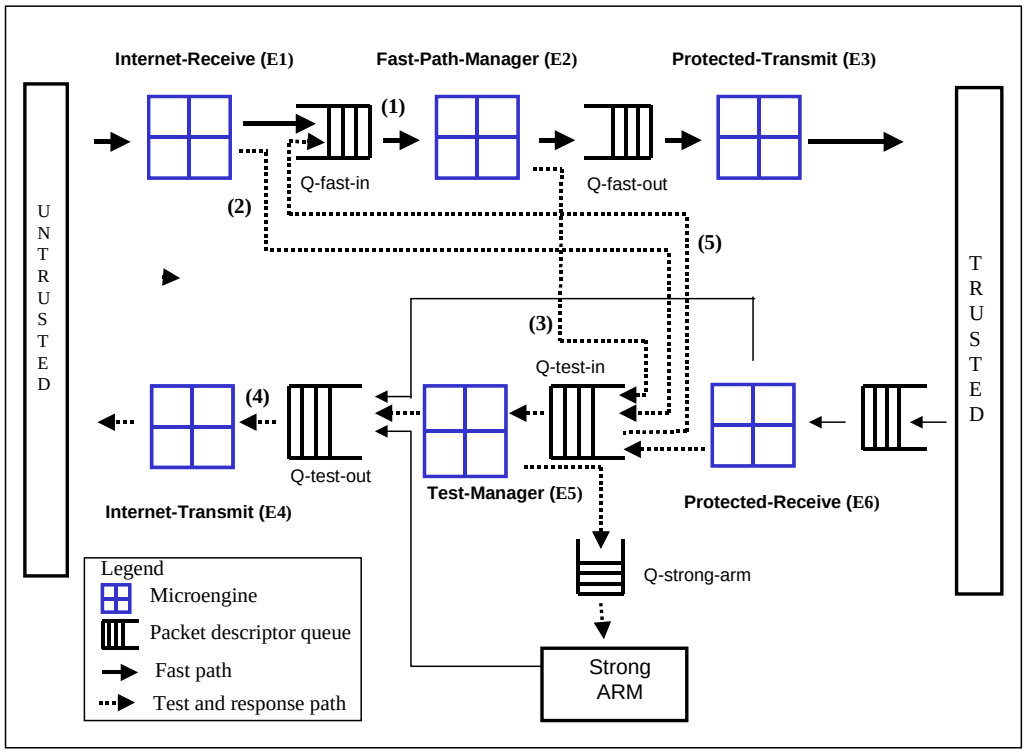

Figure 3. Architecture of the NetBouncer prototype

the path compression process. In the LC-*m* trie, the computational costs of lookups, insertions, and deletions are determined by the depth of the trie. Insertions and deletions incur a constant additional computational overhead compared to lookups.

A simple example of a hash-trie structure is illustrated in Figure 2. The hash table consists of *N=11* buckets, labeled from *0* to *10*. Bucket *5* in the hash table is highlighted to illustrate an example of an LC-*2* trie. Each of the leaf nodes in the trie corresponds to a binary bit sequence indicated in the figure. The maximum number of child nodes, although some of the child pointers in a node may be null, as indicated in the figure by a downward arrow symbol. Nodes with non-zero skip values are labeled with the values of the 'skip' and 'str' parameters. Observe that nine entries are hashed to bucket *5* in this example, although the maximum depth of the corresponding LC-*2* trie is only three.

3. Prototype Architecture on the Intel IXP1200 Network Processor

3.1. Overview of the Intel IXP1200 Network Processor

The IXP1200 chip and network processor development system consists of a StrongArm processor and six micro-engines. The Intel StrongArm Core processor, is a 32-bit RISC processor currently available at an operating frequency of 200 MHz and acts as the master general-purpose processor and supports the control plane. It is responsible for performing the bulk of the signaling and network management functions comprising the control plane. The actual movement of data and packet processing operations is performed by six 32-bit RISC Micro-engines running at 200MHz and acting as specialized slave processors. Each micro-engine can execute four parallel threads by means of four independent program counters, zero overhead context switching and hardware semaphores. In addition, the Micro-engine contains an ALU and shifter, 4 Kbytes of RAM control store, 128 32-bit general-purpose registers and 128 32-bit transfer registers for accessing the SRAM and SDRAM units. The IXP1200 also contains a hash unit for generating 48- and 64-bit hash keys. The IXP1200 supports two Gigabit Ethernet ports and eight fast Ethernet (100 Mbps) ports. For our current prototype, only the Gigabit ports are used.

3.2. Prototype Architecture

3.2.1. Fast path and test path packet processing

The details of the architecture for our current hardware prototype are shown in figure 3. Our prototype uses only the two Gigabit ports. The path labeled (1) and shown by thick arrows is the *fast path* in the architecture - the path through which legitimate packets are processed and transmitted. The paths labeled (2), (3) and (4) and as shown by dotted arrows are the *test paths* - the paths through which packets that failed legitimacy, as well as test packets associated with sending legitimacy tests, are processed and transmitted. The figure also shows the careful allocation of various NetBouncer packet filtering and legitimacy administration functions to the available micro-engines (labeled E1 through E6). This is basically a six-stage pipelined architecture. Packets are received from the media access control (MAC) layer, processed, and transmitted. To enable this, a variety of queues are used. As a packet is received from the MAC layer, it is stored in SRAM. However, vital information from the packet (such as critical header information) is extracted and stored in a structure called a *packet descriptor*. The packet descriptor also stores a pointer to the physical SRAM address where the packet is stored. The queues are used to pass the packet descriptors from one micro-engine to another as processing of the packet progresses in the

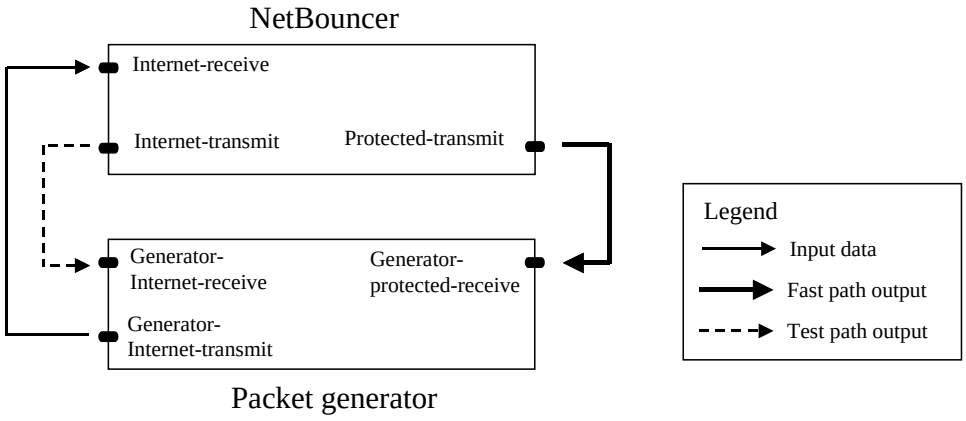

Figure 4. The test configuration consisting of NetBouncer and a packet generator

pipeline. This is more efficient than passing the actual packet from one queue to another.

Let us look at the details of how packets are processed on the prototype. We start with the fast path in the architecture. Processing starts with the "Internet-Receive" micro-engine (E1**)** reading incoming packets from the MAC layer and depositing their packet descriptors into the queue labeled Q-fast-in. A second micro-engine labeled "Fast-Path-Manager" (E2**)** dequeues these descriptors, extracts the source address and checks the legitimacy list to see if the packet is coming from a legitimate source. If the source address is on the legitimacy list, we consider the packet to have passed the legitimacy test and the corresponding packet descriptor is queued into the queue labeled Q-fast-out. A third micro-engine labeled "Protected-Transmit" (E3) consumes descriptors from this queue and uses the pointer in the descriptor to extract the packet contents and transmit them to the appropriate destination on the trusted side.

Now if a packet is not on the legitimacy list, its descriptor is queued into the queue labeled Q-test-in and the packet is then processed through the test path. A micro-engine labeled "Test-Manager" (E5) empties entries from this queue and issues one or more legitimacy tests. Each legitimacy test challenge packet is queued into the queue Q-test-out from where the micro-engine "Internet-transmit" **(**E4) transmits the challenge back to the client source on the untrusted (Internet) side of the network. Now our design accommodates the scenario where if a legitimacy test is too complicated for data plane processing in a micro-engine, the appropriate descriptor is forwarded to the StrongArm processor for further processing via the queue labeled Q-strong-arm. The StrongArm may do more complex processing such as those involving very sophisticated legitimacy tests. The micro-engine labeled "Protected-Receive" **(**E6) processes packets from trusted side of the network such as general traffic from server machines. It may forward packets directly to the queue Q-test-out for subsequent transmission by the Internet-transmit micro-engine or alternatively send some packets to the Test-manager micro-engine for further processing and tagging. The latter path is taken for some legitimacy tests (such as an anti-Smurf test) that require certain outgoing packets such as outgoing ICMP-echo requests, to be cryptographically sealed and tagged so that subsequent ICMP-echo replies can be verified for these tags. This can be used to defend against smurf-style amplification attacks that flood servers with ICMP-echo replies (details are given in [23]).

3.2.2. Response path packet processing

There also exist response data paths within the architecture. These are the data paths use for processing responses to challenges issued as part of legitimacy tests. Responses can be fed to the Test-Manager" (E5) through one of two paths, also labeled (2) and (3) in Figure 3. The response path labeled (2) is taken if the Internet-receive micro-engine (E1) receives a packet identified as a response from a client to a previously issued challenge and if it is determined that the challenge can be validated without consulting stored legitimacy state. In other words, the packet descriptor is put directly into the queue Q-test-in, thereby avoiding the Fast-Path-Manager" (E2) micro-engine. On the other hand, the second path, labeled (3), is taken when legitimacy information needs to be consulted with the help of microengine E2. Path (2) is basically an optimization to achieve speed up response processing by exploiting certain properties of legitimacy tests, such as if the pre-state of the test is self contained in a packet. Path (2) is generally possible for packet-based legitimacy tests such as the ICMP Ping test. The actual speed up is obtained by eliminating the Fast-Path-Manager micro-engine from the processing pipeline. In either case, if legitimacy can be validated, the path labeled (5) is used by the Test-manager micro-engine to send updates to the legitimacy table through the Fast-Path-Manger.

3.2.3. Use of Cryptography and digital signatures

As mentioned earlier, legitimacy challenges issued to clients are cryptographically sealed and responses from clients are cryptographically verified. Implementing cryptographic functions such as a Hashed Message Authentication Code (HMAC) function, on the IXP1200, requires careful thought due to the limited instruction store and memory and other efficiency concerns. Common schemes such as MD5 and SHA1 are too complex and inefficient. So we have chosen to implement an alternate algorithm called Michael [13] to generate HMACs for the various legitimacy test packets. Michael is simple to implement and offers acceptable tradeoffs between performance and security strength given NetBouncer's design intention to change keys frequently. By default, Michael generates 64-bit authentication tags. NetBouncer performs some modest post-processing of the Michael output by applying the XOR operation to the two 32-bit halves to generate a final 32-bit result that is encapsulated in outgoing legitimacy test packets.

4. Performance Analysis Experiments

4.1. Test Configuration

Figure 4 shows our test configuration. The first component is a NetBouncer prototype based on the Intel IXP1200 network processor evaluation system with two one gigabit Ethernet ports with each port capable of transmitting and receiving packets. The second component is an IXP1200 system configured to generate packets at one gigabit. The input stream, fast path output and test path output were set up as shown in the figure.

NetBouncer itself did not modify incoming or outgoing packets for the purpose of instrumentation with any information such as timestamps as doing so would reduce the true throughput of the hardware. Instead, all instrumentation was done when packets were transmitted and received by the packet generator.

4.2. Test Methodology

We conducted two broad classes of experiments. The first set of experiments was aimed at getting bounds on throughput and latencies and packet rates in our architecture. These experiments were thus conducted with the legitimacy list lookup function bypassed. In particular, we wanted to measure throughput and latency as a function of packet size as packet sizes directly affect packet rates. Packet sizes were varied starting at 64 bytes and going all the way to 1496 bytes in increments of 8 bytes. For each packet size, a set of three runs (trials) was conducted and plotted where a run consisted of generating packets for a sample interval of 5 seconds. After the three runs at a packet size, the packet size was varied by 8 bytes and a "settling time" of 2 seconds was allowed during which the flow of packets was not measured. The second set of experiments measured throughput and latency for 128, 600 and 1200 byte packets (referred to as small, medium and large packets) in a configuration where every data packet resulted in lookup on the legitimacy list. The experiment included the "all-good" configuration to test the fast path where every incoming packet was legitimate and an "all-bad" configuration with only illegitimate packets so as to exercise the test path. Another parameter that was varied was the configuration of the legitimacy list - one used a hash-based host lookup table (HLT) with 256 buckets (called the small hash table) and the other used 4192 buckets (large hash table). The hash unit of the IXP1200 was used to implement the hash function. By comparing our results with the bounds obtained in the first set of experiments, we could compute the overhead incurred by the lookup function. Once again, three trials of 5-second samples were used for each variation.

Measuring Latency. Latency was measured by inserting the current value of the IXP chip's cycle counter as a timestamp into the Ethernet addresses within the packets being sent by the packet generator. This avoided modifying the format of the packets sent as well as any increase in the packet size due to our tests. After a packet was sent and eventually processed and transmitted by NetBouncer back to the packet generator's Generator-protected-receive port, the cycle counter was read again and the difference between the send and receive timestamps computed.

Measuring Throughput. To measure throughput, we read a hardware counter on the Ethernet interface card of the packet generator at the end of each 5-second sample interval after having packets cycled through NetBouncer. It is to be noted that the throughput experiments did not have packets time stamped for latency measurements by the generator so as to avoid the overhead of reading the cycle counter when generating packets. Doing so would reduce the throughput of the generator.

5. Performance Results

5.1. Throughput on the Fast Path and Test Path

Figures 5 (a) and 5 (b) show the throughput results for the first set of experiments where the lookup function was bypassed. The incoming throughput observed at the Internet-receive port on NetBouncer ranged from about 990 Mbps for packet sizes of 1496 bytes to about 707 Mbps for small packet sizes of 64 bytes. The throughput observed on the fast-path (Protected-transmit port) very closely matched the incoming rate for packet sizes greater than 248 bytes. Below a packet size of 248 bytes there was a sharp drop in throughput with the low end around 339 Mbps for 64-byte packets. In general, the lower throughput for small packet sizes can be attributed to the fact that with smaller packet sizes, the packet frequencies increase and the overhead per packet is thus greater. Depending on how much processing a packet incurs, NetBouncer can see packet rates ranging from about 64,000 to 166,622 packets per second for packet sizes of 1400 bytes and rates ranging from 492,000 to 1,432,978 packets per second for a packet length of 64 bytes.

We next studied throughput as a function of packet size for the test path with and without the use of HMAC for the challenge packets (see Figure 5(b)). When HMACs are used, there is a significant reduction in throughput when compared to no HMACs being used but this difference decreases steadily as packet sizes increase. This is an indication that the HMAC overhead is per packet and not per byte. At about 1400 bytes, the throughputs with and without HMACs are very close. When HMACs were not used and the packet size was below 305 bytes, the throughput degraded almost linearly and approached about 400Mbs. This is a reflection of the fact that the illegitimate packet rate exceeded the hardware's capability to issue test challenges. Further optimization of our code could improve this situation.

Figures 6 (a) and (b) show throughput results for our second set of experiments where the lookup function was incorporated. The figures show throughput plotted as a function of the size of the legitimacy list. The size of the list was varied from about 100 entries to about 4200 entries where the latter represents the maximum we could store given the memory constraints of the Intel IXP1200. Throughput was studied for 128 (small), 600 (medium), and 1200 (large) byte packets and for legitimacy list

implementations of hash table configurations consisting of 256 (small) as well as 4192 (large) buckets. For the fast path, we see in Figure 6 (a) that for the large packet size, the throughput is steady at an average rate of 990 Mbps. The throughput characteristics for the medium packet size are nearly the same as for the large packet size. But with the small packet size of 128 bytes, throughput drops to an average rate of about 198 Mbps. This can be attributed to the increased per packet overhead at higher packet rates (as explained when we discuss latency below). In this case, NetBouncer is not able to keep up with the incoming packet rate, which is highest for the small packet size. The hash table configuration has very little impact on the throughput for the large and medium packet sizes. For the small packet size, a small gradual dip in throughput from about 325 Mbps to about 285 Mbps can be observed on the fast path when the small hash table is used. No such dip occurs when the large hash table is used.

To study the test path, NetBouncer was fed a set of addresses that were all illegitimate by subjecting it to the well-known TCP attack using SYN floods. Each incoming SYN packet was thus challenged with a legitimacy test. As shown in Figure 6 (b), we see that there is a dramatic reduction in overall throughput on the test path when compared to the fast path. Throughput is about 50 Mbps for the large packet size, 98 Mbps for the medium packet size, and about 140 Mbps for the small packet size. Also notable is the inversion of throughput with respect to the packet size when compared to the fast path. In other words, smaller packet sizes result in higher throughputs. This can be attributed to the fact that the larger the incoming packet, the longer the duration that the Test-Manager micro-engine has to wait to issue a challenge. This decreases the number of challenges issued per unit of time and thus lowers the throughput on the test path.

5.2. Latency on the Fast and Test Path

Figures 7 (a) and (b) show the packet latency through NetBouncer for the fast and test paths. The latency numbers were obtained by first measuring the total packet delay from the transmit to the receive ports of the packet generator with NetBouncer in the loop and then subtracting out the packet delay measured when NetBouncer was replaced by a short-circuit. We observe two key facts. First, latency is largest for the trials with the small packet size (128 bytes) on both the fast and test paths. For the medium and large packet sizes, the latency numbers were markedly smaller. Second, one can observe the dependence of latency on the number of buckets used for the hash-based lookup, particularly for the small packets when the small hash table was used. On the fast path, the latency for small packets using the small hash table ranged from about 25 microseconds when the legitimacy list contained less than 3000 entries to over 200 microseconds when the list contained over 4000 entries. The dependence of latency on the legitimacy list for the small hash table can be clearly seen on the test path results shown in Figure 7 (b). For the large packet sizes, the latency increases nearly linearly from about 2 to around 8 microseconds. The corresponding curve for the medium packet sizes is also approximately a line, with slightly higher slope. For the small packet sizes on the test path, the latency also shows an approximately linear growth from a latency of about 7 microseconds to about 19 microseconds. In Figures 7 (a) and 7 (b), when the large hash table is used, there is little or no dependence of the latency on the number of legitimate entries.

5.3. Analysis and Discussion

Focusing on the fast path results with lookup presented in Figures 6 and 7, we note that there was no packet loss within NetBouncer for the large and medium size packets. More precisely, the incoming and outgoing packet rate to NetBouncer for the large packets was about 99,000 packets per second (i.e., 99 Kpps), while the incoming/outgoing packet rates for the medium size packets were 192 Kpps. For the small packets, the incoming rate was about 570 Kpps, while the outgoing rate was only about 275 Kpps (and actually slightly lower than this when the small hash table was used). Therefore, nearly 52 percent of the small packets are dropped by NetBouncer in this experiment. One can conclude from this that the maximum packet processing rate of our NetBouncer prototype on the fast path is about 275 Kpps. On the test path, the maximum packet rate is slightly higher at 280 Kpps. This apparent contradiction is explained in the next paragraph.

The latency figures for the test path are markedly smaller than those for the fast path, which seems to contradict the nomenclature of "fast path" vs. "test path." This can be explained as follows. The test that is being applied is the TCP SYN Cookies test (see section 2.1), which requires relatively little processing to issue. More importantly, the size of the challenge packet is only 60 bytes (i.e., less than half the size of the small packet), which corresponds to the minimum Ethernet frame size. Therefore, the packet transmission time required for the challenge packet is significantly smaller than for the incoming packets of the three sizes considered. We found that in our experiments, packet transmission time was a dominating factor in the latency of packet processing through NetBouncer. In our NetBouncer prototype, the maximum number of entries on the legitimacy list was limited to less than 5000 entries. In a real implementation, we would expect NetBouncer to handle a list size about two orders of magnitude larger. In

our experiments, the lookup time was not a factor when the large hash table was used. For the small hash table, however, the latency figures clearly showed a linear growth rate as the legitimacy list size was increased. For larger list sizes, the lookup time is expected to become a dominating factor in the packet processing time. This strongly suggests the need for a more efficient lookup structure, such as the Hash-Trie described in section 2.2.3 as a replacement for the hash and linked-list scheme we currently used.

The performance experiments we have conducted so far do not include measurements on updates to the legitimacy list. This is attributed to the fact we found it difficult to set up tests that will generate steady state conditions since the maximum legitimacy table size of about 4000 entries would be populated in a fraction of a second at the incoming line rate of 900Mbs. To get reliable readings on metrics such as the number of updates sustainable per second, we need to maintain steady state with legitimacy tests and response validations for some reasonable sample intervals. We hope to address this issue in the future so that we can evaluate the time and space efficiency tradeoffs of the Hash-trie structure when the legitimacy list incurs a high rate of updates. Also, the tests that generated graphs in figures 6 and 7 did not involve cryptographic operations for the HMAC function. Intuitively, combining HMAC along with the lookup function will lead to further throughput degradation on the test path. We hope to study this in the future.

6. Lessons Learned on Architectural Directions for Network Processors

In the course of developing the NetBouncer prototype on the IXP 1200, we have seen the benefits, as well as the limitations of low-end network processors. We now reflect on some of the lessons learned and discuss desirable features that would have simplified the implementation and increased efficiency and performance. We note that NetBouncer functionality goes significantly beyond pure IP routing and forwarding and represents a good case study of a complex and practical application that places unique demands on packet processing. These requirements can provide a set of guidelines for supporting advanced features in next generation network processor technology. The complexity of NetBouncer-style packet filtering can be attributed to the fact that complex legitimacy state needs to be maintained and challenge packets need to be sent back to the client machines. The administering and verification of legitimacy tests in real-time introduces a variety of data paths and pipelines on the IXP 1200 implementation.

A challenging aspect of programming the IXP 1200 was constructing the pipeline of tasks and distributing them among the microengines. This is partly due to the programming model for the IXP family of network processors, which calls for the application designer to be aware of internal details such as the pipeline and queue structures, as well as the careful allocation of functionality across the various processors (microengines). The IXP 1200 does not provide explicit mechanisms for queueing packets between the pipeline stages. Hence, we resorted to using the generic queue structures provided by a third-party application development environment for the IXP 1200, namely Teja [22]. If the queueing structures could support priority schemes, this would allow NetBouncer to make more efficient use of the IXP microengines. For example, we would like to sort the queue of challenge packets according to some service priority so that higher priority challenges are issued more promptly. The IXP transfer register scheme allows only eight 32-bit words of a packet to be read into the IXP at a time. This makes it rather cumbersome to access arbitrary fields in an entire packet and slows down packet classification and other packet processing tasks. In particular, if the string to be searched is not aligned on a 32-bit word boundary, additional low-level bit manipulations are required to extract the string. The ability to efficiently perform "grep"-like or string-search operations would allow us to recognize application-specific messages. This would make the implementation of higher level service-based legitimacy tests much more efficient.

The pipelined architecture of the IXP 1200 can result in performance bottlenecks as the computational loads incurred on the microengines can vary considerably based on the incoming traffic mix and DDoS attacks in progress. Mechanisms to perform dynamic load balancing across microengines could lead to significantly better performance by reallocating legitimacy testing and cryptographic operations across idle micro-engines so as to balance the microengine utilizations. On the other hand, dynamic load balancing would be difficult to implement on the current IXP architecture due to the lack of common register space and control stores across microengines. Moreover, load balancing would require an architecture that could dynamically monitor microengine utilization and allow relatively idle microengines to assume additional tasks with low overhead.

In implementing the NetBouncer architecture on the IXP 1200, we found that several data-intensive tasks might be performed better by a co-processor or specialized hardware engine. NetBouncer performance could be greatly enhanced by a packet classifier that could parse incoming packets, extract the fields of interests, and hand these fields to the microengines further down in the pipeline. This would avoid the need

to move the entire packet into the IXP and would free up the microengines to perform other NetBouncer tasks. Moreover, a packet classifier could assign priorities to packets that could be used to reduce the burden on heavily loaded microengines during periods of congestion within the IXP.

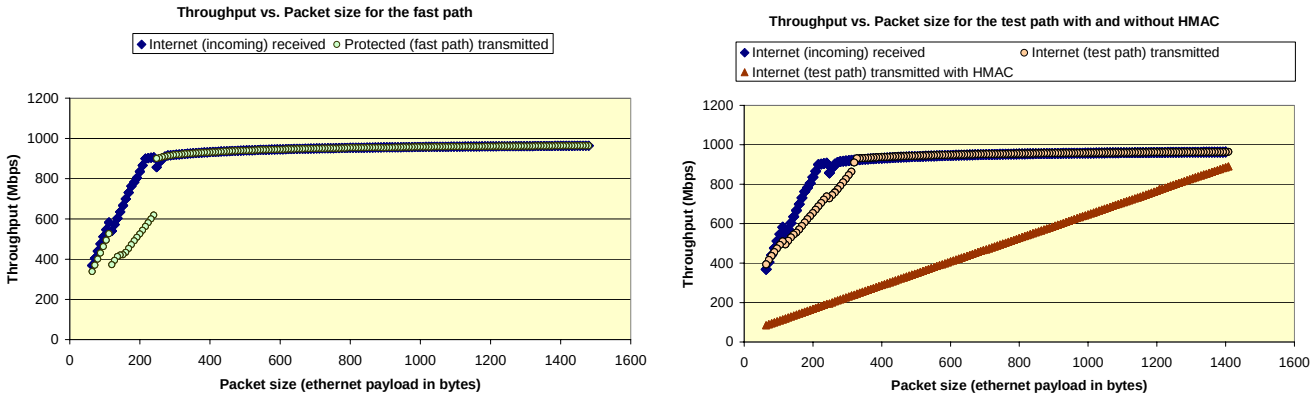

Figure 5 (a) and (b). Throughput for the fast path and test path with no lookup function.

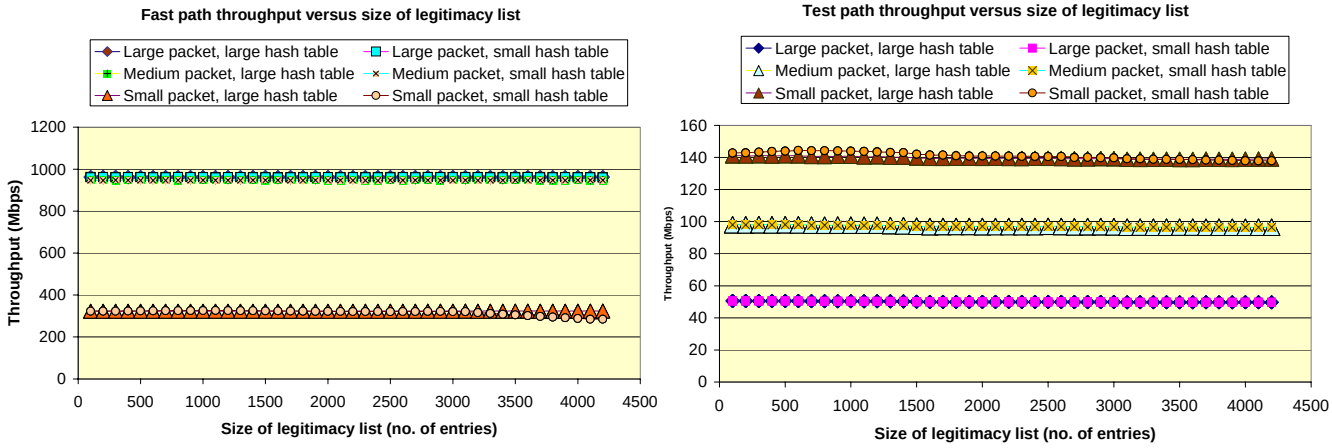

Figure 6 (a) and (b). Throughput as a function of legitimacy table size for the fast path and test path for 128 and 1200 byte packet sizes and for small and large hash tables (256 and 8192 buckets)

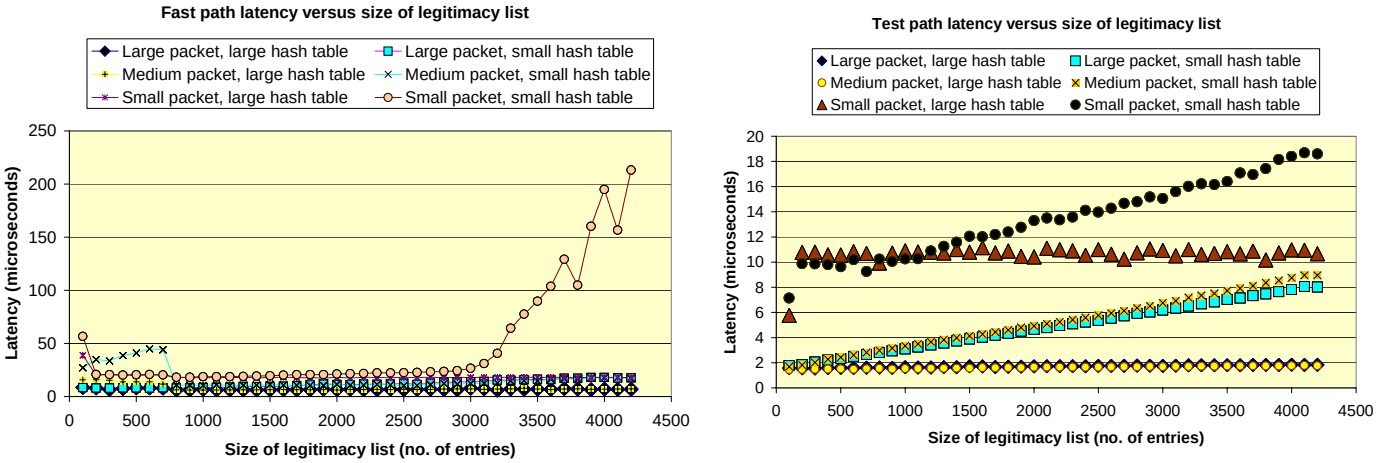

Figure 7 (a) and (b). Latency as a function of legitimacy table size for the fast path and test path for 128 and 1200 byte packet sizes and for small and large hash tables (256 and 8192 buckets).

General packet classification is cumbersome to implement on the IXP 1200. Although classification of HTTP packets is relatively easy to perform on the IXP, parsing of more complex protocols is more difficult, especially when parameters such as UDP port numbers are dynamically assigned as in RTP. For the service-based legitimacy tests, it would be desirable to have a general packet classifier that could parse HTTP style headers.

Another desirable function that could be implemented in hardware or in a co-processor is a generic checksum engine. For example, to rewrite a TCP header, NetBouncer would extract the TCP checksum, send the checksum engine the fields that were to be changed, change the fields, send the new values of the field, and finally query the new checksum value. A packet classifier and checksum engine could be combined to create a packet coprocessor that would process packets at the ingress/egress of NetBouncer. Other common packet processing functions that could be offloaded to the packet co-processor might include general hash functions and encryption/decryption operations. This is indeed the direction taken by more advanced network processors.

A general hardware lookup engine would also be a desirable feature that would simplify the NetBouncer architecture. While we devised an efficient lookup engine that could be implemented directly on the IXP, we feel that the resources of the IXP could be much better utilized to implement the NetBouncer functionality if the lookup function were offloaded to a specialized search engine. For flow lookup, such a search engine should allow an arbitrary block of data to be allocated to each flow. Other items on our hardware "wishlist" that would simplify and improve the performance of a NetBouncer implementation on a network-processor based system include a flow control engine and a TCP engine. The flow control engine would provide flow management functionality such as packet buffering, packet dropping, and setting flow parameters.

A TCP engine would allow NetBouncer to effectively terminate and source a TCP connection. This would allow NetBouncer to see a stream of bytes arriving on a TCP flow rather than packets, which would simplify the implementation of service-based legitimacy tests such as the Turing test.

7. Conclusions and Future Work

We have presented the results of the NetBouncer research effort to build a high-speed DDoS traffic filtering appliance using the Intel IXP1200 network processor. The unique feature of the NetBouncer approach to DDoS mitigation is the use of legitimacy tests to distinguish and filter incoming illegitimate traffic. These tests exist at various levels of abstraction in the protocol stack and have to be administered on every incoming packet from untrusted clients in a manner that can maintain high line rates.

At this stage of our research project, we have seen the benefits, as well as the limitations of low-end network processors. Our IXP1200 prototype has clearly outperformed earlier software (Linux-based) versions of NetBouncer implemented on mainstream PC processors. However, initial tests of the prototype have also exposed the limitations of low-end network processors such as the IXP1200 in supporting functions beyond simple packet routing and forwarding. In a nutshell, the IXP 1200 was not sufficient to implement the desired capabilities of NetBouncer.

The complexity of NetBouncer-style packet filtering can be attributed to many functions. First, NetBouncer functionality calls for packet contents to be looked up in a legitimacy structure that is much more complex than IP routing tables. This structure is required to maintain and track legitimacy state across a variety of traffic-related abstractions and may incur a very high frequency of updates. We also failed to support the requirement to store at least a few hundred thousand entries in the legitimacy list due to memory constraints. Secondly, the administering and verification of legitimacy tests in real-time introduces a variety of data paths and pipelines on a network processor architecture such as the IXP1200. In particular, this is required to support the sending of challenge packets back to client machines and the subsequent verification of client responses. This complexity is also partly due to the programming model for the IXP family of network processors, which calls for the application designer to be aware of internal details such as the pipeline and queue structures, as well as the careful allocation of functionality across the various processors (micro-engines). From a security standpoint, we note that even a lightweight HMAC function for cryptography increased the per-packet overhead resulting in considerably reduced throughputs and increased latencies.

A critical feature that we desperately desired was the need to provide dynamic load balancing as illegitimate and legitimate traffic loads vary considerably based on the incoming traffic mix and the DDoS attacks in progress. This could lead to better performance by reallocating legitimacy testing and cryptographic operations across idle micro-engines.

In the future, we hope to experiment with the more powerful IXP 2400 or IXP 2800 processors, as well as explore network processors with alternate programming models such as the run-to-completion models (e.g., IBM and AMCC processors) that provide an abstract, single and unified model to the application developer. These processors also provide cryptographic co-processors that should improve performance.

Acknowledgement

The NetBouncer project is funded by the Defense Advanced Research Projects Agency (DARPA) under contract N66001-01-C-8046. We are grateful to Dr. Doug Maughan and the Fault Tolerant Networks program for their support.

References

[1] D.J. Bernstein, SYN Cookies, http://cr.yp.to/syncookies.html

[2] Peakflow/DoS, Arbor networks, http://www.arbornetworks.com/up_media/up_files/Pflow_Enter_datasheet2.1.pdf

[3] DDoS Enforcer, Mazu Networks, http://www.mazunetworks.com/solutions/product_overview.html#300

[4] S. Capshaw, Minimizing the Effects of DoS Attacks, Application Note, Juniper networks, November 2000, http://www.juniper.net/techcenter/app_note/350001.pdf

[5] Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks, White Paper, CISCO Systems, http://www.cisco.com/warp/public/707/newsflash.html

[6] Characterizing and Tracing Packet Floods Using Cisco Routers, Technical Note, CISCO Systems, July 1999, http://www.cisco.com/warp/public/707/22.html

[7] Configuring TCP Intercept (Prevent Denial-of-Service Attacks), CISCO Systems, http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/scdenial.htm

[8] Cisco IOS Netflow, http://www.cisco.com/warp/public/732/Tech/nmp/netflow

[9] D. Dean and A. Stubblefield, Using client puzzles to protect TLS, *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C, August 13-17, 2001.

[10] M. Degermark, A. Brodnik, S. Carlsson and S. Pink, Small forwarding tables for fast routing slookups, *ACM Computer Communication Review*, 27 (4), October 1997, pp. 3-14.

[11] S. Dietrich, N. Long and D. Dittrich, Analyzing Distributed Denial of Service Tools: The Shaft Case, *Proceedings of the LISA XIV*, New Orleans, LA, December 3-8, 2000.

[12] W. Doeringer, G. Karjoth, and M. Nassehi. Routing on longest-matching prefixes. IEEE/ACM Trans. on Networking, 4(1), February 1996, pp. 86-97.

[13] N. Ferguson, Michael: an improved MIC for 802.11 WEP, IEEE P802.11, Wireless LANs, MacFergus, Bart de Ligtstraat 64, 1097 JE Amsterdam, Netherlands, January 17, 2002.

[14] D. P. Morrison, Practical algorithm to retrieve information coded in Alfanumeric, *Journal of ACM* 15, 4 (October 1968), pp. 514-534.

[15] R. Mahajan, S. Bellovin, D. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, Controlling High Bandwidth Aggregates in the network, AT&T Center for Internet research at ICSI (ACIRI), DRAFT, February 5, 2001, http://www.research.att.com/~smb/papers/ddos-lacc.pdf

[16] A. McAuley and P. Francis, Fast routing table lookup using CAMs, *In Proceedings of INFOCOM*, March-April, 1993, pages 1382-1391.

[17] S. Nilsson and G. Karlsson, Fast Address Lookup for Internet Routers, *In Proceedings of the International Conference on Broadband Communication*, April 1998.

[18] K. Park and H. Lee, On the Effectiveness of Route-based Packet Filtering for Distributed DoS Attack Prevention in Power-law Internets, in *Proceeding of the ACM SIGCOMM '01*, pp. 15-26.

[19] D. Schnackenberg, K. Djahandari, and D. Sterne, Infrastructure for Intrusion Detection and Response, *Proceedings of the DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, January 2000.

[20] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2001

[21] D.E. Taylor, J.W. Lockwood, T.D. Sproull, J.S. Turner, D.B. Parlour, Scalable IP Lookup for Programmable Routers, *Proceedings of the. IEEE Infocom'02*, pp. 562-571, 2002.

[22] Software Development Platform for Network Processors, Teja Technologies, http://teja.com/content/teja_np_datasheet.pdf

[23] R. Thomas, B. Mark, T. Johnson, J. Croall, NetBouncer: Client-legitimacy-based High-performance DDoS Filtering, To appear in the Proceedings of the *Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C, April 22-24, 2003.

[24] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP routing lookups. *ACM Computer Communication Review*, 27(4), October 1997, pp. 25-36.

[25] C.A. Zukowski and T. Pei. Putting routing tables into silicon., *IEEE Network*, January 1992, pp. 42-50.