2010 Fourth International Conference on Network and System Security

DIPLOMA: Distributed Policy Enforcement Architecture for MANETs

Mansoor Alicherry Angelos D. Keromytis
Department of Computer Science
Columbia University

***Abstract*—**
Lack of well-defined defense perimeter in MANETs prevents the use traditional firewalls, and requires the security to be implemented in a distributed manner. We recently introduced a novel deny-by-default distributed security policy enforcement architecture for MANETs by harnessing and extending the concept of *network capabilities*. The *deny-by-default* principle allows compromised nodes to access only authorized services, limiting their ability to disrupt or even interfere with end-to-end connectivity and nodes beyond their local communication radius. The enforcement of policies is done hop-by-hop, in a distributed manner. In this paper, we present the implementation of this architecture, called DIPLOMA, on Linux. Our implementation works at the network layer, and does not require any changes to existing applications. We identify the bottlenecks of the original architecture and propose improvements, including a signature optimization, so that it works well in practice. We present the results of evaluating the architecture in a realistic MANET testbed Orbit. The results show that the architecture incurs minimal overhead in throughput, latency and jitter. We also show that the system protects network bandwidth and the end-hosts in the presence of attackers. To that end, we identify ways of creating multi-hop topologies in indoor environments so that a bad node cannot interfere with every other node. We also show that existing applications are not impacted by the new architecture, achieving good performance.

Keywords: MANETs, Capabilities, Distributed firewalls

I. Introduction

Due to lack of well-defined perimeter nodes where access control could be enforced, MANETs require a distributed policy enforcement architecture, which is different from traditional networks. Recently, we proposed a deny-by-default architecture [6] that enforces trust relationships and traffic accountability between mobile nodes through a distributed policy enforcement scheme for MANETs. In that architecture, we extended the network capability framework [8] and tailored it to the resource-constrained MANET environment. A capability is a token of authority that has associated rights. The capabilities propagate both access control rules and traffic-shaping parameters that should govern a node's traffic. In the deny-by-default, model nodes can only access the services and hosts they are authorized for by the capabilities given to them. The enforcement of the capability is done in a distributed manner by all the nodes in the path from the source to the destination. Compromised or malicious nodes cannot exceed their authority and expose the whole network to an adversary. Upon detection, we can prevent a compromised node from further attacking the network simply by revoking its capabilities. Moreover, that architecture helps mitigate the impact of denial of service (DoS) attacks because excess or unauthorized packets are dropped closer to the attack source. Thus, we avoid unnecessary data processing and forwarding at the target node and the network itself.

We look at aspects of implementing the architecture in real systems. We name our architecture **DIPLOMA**, which stands for **DI**stributed **P**o**L**icy enf**O**rce**M**ent **A**rchitecture. We adapt the original proposal for the real implementation, so that it provides good performance and is effective against attacks. We implement DIPLOMA on Linux systems as a user level protocol engine that interfaces with the rest of the packet processing system through *netfilter-queue* APIs. Our implementation does not require any changes to the existing applications. However, the applications see the benefit in terms of receiving only the authorized traffic, and being able to send the allocated bandwidth even in the presence of rogue nodes that are trying to send large amount of traffic. Our implementation uses a simple way of representing and enforcing bandwidth constraints using the token bucket parameters. In DIPLOMA, we are doing a clean-slate design. In our design, we leave the IP layer intact; hence, we can make use of existing routing and packet forwarding capacities of the nodes.

Furthermore, we propose a novel signature scheme to integrity protect the packets, and drop the tampered packets closer to the source. This is a combination of RSA signature and SHA-1 hashing scheme, and significantly improves the processing time per packet from the earlier proposal. The experimental results confirm that it can work well in practice.

We also conduct extensive experiments to evaluate the performance and the effectiveness of our system. To that end, we implement our system on the Orbit-lab test-bed [3]. Most of the indoor wireless testbeds create multi-hop topologies using MAC address filtering. A major problem with that approach is that an attacker can get unfair share of the channel, affecting all the other nodes in the topology. We propose a novel solution for creating multi-hop topologies on indoor environments, in which an attacker cannot cause unlimited damage to the nodes in its proximity by hogging the channel.

Our experiments show the effectiveness of the new authentication scheme, and show that the overhead is minimum on throughput, latency and jitter for DIPLOMA. Our experiments on attack resiliency show that the good nodes and the bandwidth of the network can be protected from the attackers. We also show that our system works well in the presence

978-0-7695-4159-4/10 $26.00 © 2010 IEEE
DOI 10.1109/NSS.2010.27

IEEE computer society

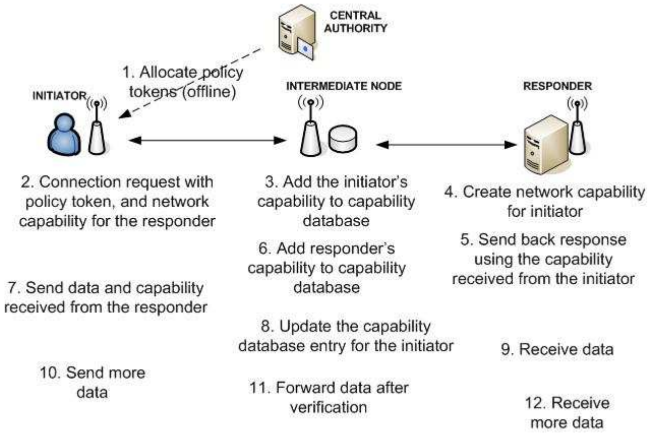
CENTRAL AUTHORITY
1. Allocate policy tokens (offline)
INITIATOR
INTERMEDIATE NODE
RESPONDER
2. Connection request with policy token, and network capability for the responder
3. Add the initiator's capability to capability database
4. Create network capability for initiator
5. Send back response using the capability received from the initiator
6. Add responder's capability to capability database
7. Send data and capability received from the responder
8. Update the capability database entry for the initiator
9. Receive data
10. Send more data
11. Forward data after verification
12. Receive more data

Fig. 1. System overview

of multiple flows. We also show that existing applications like FTP and *wget* work without any changes, achieving good performance.

Previously we provided the evaluation of the deny-by-default architecture using simulations [7]. The simulations were conducted by implementing the architecture on the GloMoSim [1] simulator, and adding the additional packet processing capability. The simulations reported 5% overhead in throughput. Our current implementation reports about 19.8% to 23% reduction in the throughput. This is because the simulator is a discrete event simulator that deals with only certain packet events. It cannot accurately model the processing at the nodes, and the load the nodes may encounter due to that processing. It also cannot accurately model packet delays and losses due to characteristics of the environment.

We describe the DIPLOMA architecture in Section II, and its implementation using the netfilter framework in Section III, and present the packet signature optimization in Section IV. We describe our experimental methodology and results in Section V. Section VI discusses related work.

II. System Architecture

In our architecture, one or more pre-defined nodes act as a *group controller* (GC), which is trusted by all the group nodes. A GC has authority to assign resources to the nodes in MANET. This resource allocation is represented as a credential (capability) called *policy token*, and it can be used to express the services and the bandwidth a node is allowed to access. They are cryptographically signed by the GC, which can be verified any node in the MANET.

When a node (initiator) requests a service from another MANET node (responder) using the policy token assigned to the initiator, the responder can provide a capability back to the initiator. This is called a *network capability*, and it is generated based on the resource policy assigned to the responder and its dynamic conditions (*e.g.,* level of utilization).

Figure 1 gives a brief overview of our system. All nodes in the path between an initiator to a responder (*i.e.,* nodes relaying the packets) enforce and abide by the resource allocation encoded by the GC in the policy token and the responder in the network capability. The enforcement involves both access control and bandwidth allocation. A responder accepts packets (except for the first) from an initiator only if the initiator is authorized to send, in the form of a valid network capability. It accepts the first packet only if the initiator's policy token is included. An intermediate node will forward the packets from a node only if they have an associated policy token or network capability, and if they do not violate the conditions contained therein. Possession of a network capability does not imply resource reservation; they are the maximum limits a node can use. Available resources are allocated by the intermediate nodes in a fair manner, in proportion to the allocations defined in the policy token and network capability.

The capability need not be contained in all packets. The first packet carries the capability, along with a transaction identifier (TXI) and a public key. Subsequent packets contain only the TXI and a packet signature based on that public key. In Section IV, we present a scheme, to avoid the public key based packet signature computation for majority of the packets, still maintaining the packet integrity. Intermediate nodes cache policy tokens and network capabilities in a *capability database*, treating them as soft state. A capability database entry contains the source and the destination addresses, TXI, the capability, public key for the packet signature and packet statistics. Capability retransmissions update the soft state of intermediate nodes when the route changes due to node mobility. The soft state after a route change is also updated using an on-demand query for the capability database entry from the upstream nodes.

III. Implementation details

We implement DIPLOMA in Debian Linux system running 2.6.30 kernel by creating a user space DIPLOMA engine that interacted with Linux packet processing system using netfilter queue framework. In our implementation, the user applications do not require any change, and all the DIPLOMA related processing is done by the DIPLOMA engine. We use a popular implementation of AODV from University of Uppsala (AODV-UU) for MANET routing [4]. Here, we describe the details of our implementation.

A. Netfilter overview

Linux kernel provides a series of hooks for intercepting and manipulating packets in various points in the protocol stack, called the *Netfilter* framework [2]. Figure 2 shows the netfilter framework and how the DIPLOMA fits into it. There are five types of hooks [14]. When the packet enters the system from the network it passes through the PREROUTING hook. Then the packet goes through the routing module. If the packet is destined to that host, then the packet goes through the INPUT hook, before it is passed on to any local process. If the packet is destined for another network interface, then it goes through the FORWARD hook. Then the packet is send to the POSTROUTING hook, before it is put in the wire again. If

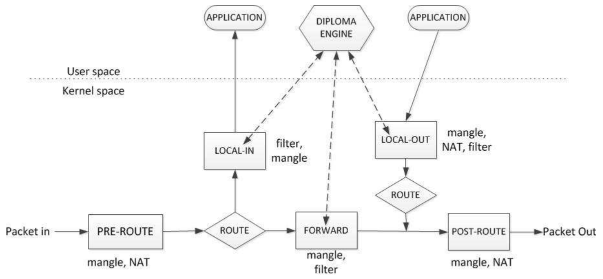

Fig. 2. DIPLOMA on Netfilter Architecture

the packet is created locally, then the OUTPUT hook is called before taking any routing decision. The packet will also go through the POSTROUTING hook after the routing is done, before it leaves on the wire.

A kernel module can register to receive the packets on one or more of these hooks by providing the netfilter with a callback function. The callback function is invoked when a packet hits the registered hook, and can optionally modify the packet and send a “verdict” on the packet. There and five verdicts: continue the normal processing (NF_ACCEPT), drop the packet (NF_DROP), the module has consumed the packet (NF_STOLEN), queue the packet for user space handling (NF_QUEUE), or call the hook again (NF_REPEAT).

A set of tables has been built on top of the netfilter for packet selection. A user level program called *iptables* allows system administrators to configure these tables. The tables can be used to filter packets (‘filter’ table), perform a network address translation (‘nat’ table), or do a pre-route packet mangling (’mangle’ table). Figure 2 also shows which tables are present with each of the netfilter hooks. Using iptables, one can also send the packet to the user space.

A user-level library called *netfilter-queue* provides APIs for manipulating packets that has been queued by the kernel filter. This API can receive the queued packet to the user space, manipulate the packet, and provide a verdict on the packet. This library can be used in conjunction with iptables to implement any user level protocol processing.

B. DIPLOMA Implementation

DIPLOMA is implemented as a user-level daemon (called DIPLOMA engine), using the netfilter-queue library. At the system startup, iptables rules are added to the mangle table, so that all the packets that are leaving, entering or transiting through the system on selected interfaces are received by the engine. All the processing related to the DIPLOMA protocol are handled solely inside the engine. The application processes do not have to be aware of these processing, and our implementation does not require any changes to the applications. The DIPLOMA engine performs the following packet processing operations:

- It adds DIPLOMA headers and the packet signatures to all the outgoing packets from the system. It also enforces the outgoing bandwidth constraints.
- It removes the DIPLOMA headers of the incoming packets, before the packet is given to the application.
- It verifies the packet signatures and the bandwidth usage of the transiting and incoming packets.
- It handles all the DIPLOMA protocol packets and their processing, including the capability establishment, capability refresh, error handing, *etc.*

All the data packets transiting through the network contain a new header, called *capability header*. This is placed between the IP header and the transport header (TCP/UDP). The protocol field in the IP header is changed to indicate the presence of the new header. The capability header contains the type of the packet, a transaction identifier to identify the capability associated with the packet, a sequence number, and the signature for the packet. The size of the capability header in our implementation is 36 Bytes. The capability header is added by the DIPLOMA engine at the source node after receiving the packet from the application. The header is removed by the engine at the destination, before the application receives the packet.

1) Capability Establishment: Capability establishment is the process of establishing the mapping between the transaction identifier, and the actual capability and the keys for the packet signatures, on the path from the source to the destination for any flow. The intermediate nodes store the relevant information in a data structure for easy lookup and update. Once the capability is established the source node does not include the actual capability in the data packets. Instead, the transaction identifier and the packet signatures are used by the intermediate nodes to verify the packets.

2) Sending data: When an application sends a packet, it is first received by the kernel protocol stack. The packet hits the OUTPUT hook, and the packet is queued by the iptables mangle rule for user space processing. The DIPLOMA engine receives the packet using netfilter-queue API. If the packet is the first packet in the flow, a capability needs to be established for that flow. The DIPLOMA engine looks for a matching policy token in its possession. If there is a matching policy token, then it queues up the packet and initiates the capability establishment protocol. Once the capability establishment is complete, the engine adds a capability header to the original packet and sends the modified packet with an ACCEPT verdict to the kernel. The kernel continues the normal packet processing for the packet, and sends it to the corresponding network interface for transmission. If the application sends more packets while establishing the capability, those packets are also queued by the DIPLOMA engine. If the engine receives a packet from an application for which it does not possess a policy token or a network capability, it gives a DROP verdict and the packet is dropped by the kernel.

The DIPLOMA engine also enforces the bandwidth constraints of the capability associated with the outgoing packet. If the sender does not enforce these bandwidth constraints, the packet will eventually get dropped at the next hop.

3) Forwarding data: All the transit packets are queued by the kernel filters and send to the DIPLOMA engine using iptables rule on the FORWARD chain on the mangle table. The engine in the forward path verifies the data packet against the capability and gives a verdict. The verification involves checking whether there is a capability associated with the packet using the source address and the transaction identifier, whether the packet signature is correct, and if the packet is in conformance with the bandwidth constraints. There is no modification of the packets by the engine during this process.

The bandwidth allocated to a capability is enforced using a *token bucket algorithm*. The capability contains two bandwidth parameters: the *rate* and the *burst size*. The rate is number of bytes per second it is allowed to transmit, and the burst size is the size of the bucket. Whenever the engine sees a packet, it first updates the available tokens in the bucket based on the rate and the bucket size, and then removes the number of tokens equal to the size of the packet. If the available token is less than the size of a packet, then the packet is dropped. The engine updates the token bucket only when there is a corresponding packet. It keeps track of the time the token bucket was last updated to perform the proper accounting.

4) Receiving data: All the incoming packets are queued by the kernel filters for the DIPLOMA engine processing by adding iptables rules on the INPUT chain on the mangle table.

When the packet is received by the engine, it first verifies that the packet is in conformance to the capability, similar to the packet forward path. If it is not conformant, the packet is dropped by giving a DROP verdict. Otherwise the capability header is removed from the packet, the IP header is modified to reflect the correct network protocol and size, and the modified packet is send back to the kernel with an ACCEPT verdict. From now on, the normal packet processing happens inside the kernel and the packet is send to the user application.

5) Fragmented packets: The packets received on the OUTPUT hook by the DIPLOMA engine are usually of the MTU size. The DIPLOMA engine adds a capability header to the packet, making the size of the packet more than the MTU size. When this modified packet is send back to the kernel with an ACCEPT verdict, the protocol stack will fragment the packet. These fragments pose two issues for the DIPLOMA. Firstly, the fragments cannot be associated with the capability, as they do not contain the transaction IDs. Secondly, the signature is computed for the whole packet by the DIPLOMA; intermediate nodes will be unable to verify the fragment signatures.

The fragmentation problem is arising because our implementation of the DIPLOMA engine is separate from the network stack. If the engine was integrated into the network stack, then it can take care of this issue without causing the fragmentation. Reducing the MTU size of the system also does not solve this problem, as the network stack will end up fragmenting the packets using the new MTU size.

We address the problem in two ways. Firstly, we forward a fragmented packet at the intermediate node only after receiving all the fragments and verifying them against the capability. Secondly, for TCP packets, we reduce the maximum segment size (MSS) of the packet by the capability header size, using iptables rules. Hence, the TCP stack will packetize the data such that there is enough room for the capability header without exceeding the MTU.

IV. Packet signatures

In a MANET, since there are no dedicated nodes for routing, it is easy for rogue nodes to inject packets into an existing flow. Hence we require an integrity check for the packets to save the precious bandwidth resources and to avoid unnecessary packets reaching to the destination hosts. To integrity protect the packets, we require an asymmetric scheme where the sender is able to perform the signature operation that the intermediate node can only verify, but not sign.

In our earlier proposal, we used RSA signatures with small keys (256 or 512 bytes) to sign the packets. Though, these keys can be broken in short time, the nodes refreshes the keys frequently rendering the key breaking useless to the attackers. Use of RSA has another property that is useful for our architecture. The generation of signatures require much more processing than the verification, making the packet forwarding task much easier to perform compared to sending. Other asymmetric algorithms like ECDSA, though require much smaller key sizes, do not possess this property.

Although the RSA key sizes used are small, signing individual packets is still a bottleneck if the sender transmits many packets. We implement a combination of the packet hash and the signature to overcome this bottleneck.

When a sender has to send large number of packets, say P packets, then it computes the RSA signatures only for the first packet. All the remaining $P-1$ packets only contain their hashes. The first packet's header also contains these $P-1$ packet hashes. The RSA signature for the first packet is computed on its data and these hashes. In this scheme, P is called the **block size (P)**. The first packet is marked as of type *DATA-FIRST* and the remaining $P-1$ packets are marked of type *DATA-NEXT*. The sender always sends the DATA-FIRST packet before any of the DATA-NEXT packets in the block.

When an intermediate node receives a DATA-FIRST packet, it verifies the packet against its capability for the bandwidth usage and the signature. It also retrieves the hashes of the subsequent DATA-NEXT packets from this packet, and saves in its memory. The packet is forwarded as any other packet, without waiting for the subsequent DATA-NEXT packets. When the intermediate node receives a DATA-NEXT packet, its hash is compared with the hashes stored for that flow. The packet is accepted for forwarding only if there is a match for the hash, and the packet satisfies the bandwidth constraints.

If the number of packets the source want to send is less than P, then the DIPLOMA engine should not to wait indefinitely for the packet block to fill before sending the first packet. To handle this case, the engine waits only for a certain time period called **block timeout (T)**, before it sends out the first packet. T is typically few tens of milliseconds. When the timer ticks in, the engine uses the available packets to form a block.

The parameter T is useful for the dynamic content and the last few packets of the static content. If the dynamic content generated is small, then all the packets that are generated in T time are send as a block. When the dynamic content is large, the block may get filled before the block timeout. In that case, the packets are send as soon as the block is filled.

A. Priority for DATA-FIRST packets

Since each node implements the token-bucket algorithm independently, in a distributed manner, it is possible that the available tokens in the nodes are not synchronized. Different packet processing and the propagation delays can also cause this issue. When a sender sends at the rate of its allocated capacity, it is rarely possible that a few packets may not have enough available tokens at some intermediate nodes. If the packet to be dropped is a DATA-FIRST packet, then all the subsequent DATA-NEXT packets in the block will also get dropped due to hash verification failure. To overcome this problem, our implementation accepts a DATA-FIRST packet even when there is not enough available tokens. This will make the available token negative. To prevent any misuse, all the subsequent packets, including DATA-FIRST packets, are dropped till the available token becomes positive again.

B. Block size and block timeout tradeoffs

There are tradeoffs between choosing a large block size, vs a small block size. The extra capability header used for the DATA-FIRST packet increases by 20 bytes for every increase in the block size. If one wants to remove the fragmentation processing for TCP, by setting the maximum segment size (MSS), then the MSS reduces by 20 bytes for every increase in the block size. This reduces the amount of data bytes per packet, increasing the percentage overhead of the headers in the packet. Another issue with the large block size is that the loss of a DATA-FIRST packet will render the system to drop all the DATA-NEXT packets in that block. Thus, a larger block size may cause higher packet loss. The processing cost of sending the packet can be divided into three components:
p_s - signature computation cost
p_h - hash computation cost
p_o - other processing costs including protocol processing, scheduling of engine *etc.*
Per packet processing cost = $p_o + p_h + \frac{p_s}{P}$.

The advantage of using larger block size is lower processing at the sender. This is because there is only one RSA signature operation per block, which is the predominant transmission cost. If the sender has many packets to send, then a larger block size helps it achieve higher send bandwidth.

There is a similar tradeoff for the block timeout. Recall that the DIPLOMA engine waits the minimum of block timeout and the time to fill the packet block, before it sends the packet block. If the timeout is small, then the engine will end up sending partially filled blocks. If the timeout is large, then the packet block likely will get filled before the timeout, reducing the processing time per packet at the sender. The disadvantage of large block timeout is larger packet latency. If there are not enough packets to send, then the latency of the DATA-FIRST packets will be at least the block timeout. Large block size and block timeout tend to send the packets as bursts; this may reduce the wireless link utilization.

V. Experimental Evaluation

In this section we describe the test setup, the experiments conducted to study the effectiveness of DIPLOMA, and the results and the analysis of those experiments. We study the throughput of the systems running DIPLOMA, and identify the block sizes that provides the optimal throughput. We also study the effectiveness of DIPLOMA in enforcing the bandwidth constraints. We also measure the latency and jitter for these systems. We also show how the DIPLOMA can protect the end-hosts and the network bandwidth in the presence of attackers. Finally, we show that DIPLOMA can work well in the presence of multiple flows.

A. Testbed

We implement the DIPLOMA engine as described in Section III in Linux systems running Debian Linux with kernel 2.6.30. We run the resulting system on multiple nodes in the Orbit lab [3] wireless testbed.

Orbit is an indoor wireless testbed consisting of 400 nodes arranged as a 20x20 grid on a physical area of (20m x 20m). Each node contains 1-GHz VIA C3 processor, 512 MB RAM, a 20 GB hard disk, two wireless mini-PCI 802.11 a/b/g interfaces, and two 100BaseT Ethernet ports. Most of the cards are Atheros AR5212-based cards, although there are a few Intel Pro-wireless 2915-based cards as well. In our experiments, we use only the nodes that have Atheros cards. We use only the wireless interfaces for the experimental traffic.

Since the DIPLOMA engine is a user-level process, all packets are queued for user-level processing before transmission. To make a fair comparison, we also do a similar queuing of the packet to a user level process on systems not running the DIPLOMA (called *original*). The user level program gives an ACCEPT verdict on all the packets, without any processing.

1) Topology creation: Since the Orbit grid is housed in a relatively small physical area, every node is reachable by every other node on wireless links. The large range (about 300ft) of 802.11 makes it very difficult to create true multi-hop topologies in indoor environments. A solution used is to create a network layer topology by filtering out packets of the non-adjacent senders based on their MAC addresses, at the receivers. Another approach is to generate large noise [12]; so that the receivers farther from the sender will not have enough signal to noise ratio (SNR) to decipher the signal. Orbit has four noise generator antennas for this purpose. Unfortunately, only limited topologies can be created with this approach, and it requires extensive trial and error to find the right noise levels.

MAC address filtering based topologies are not useful for studying the security properties of a system like DIPLOMA. This is because a bad node can cause damage in its communication radius, even if it is not confirming to the protocol. In an indoor wireless setting like Orbit, the communication radius is the whole grid.

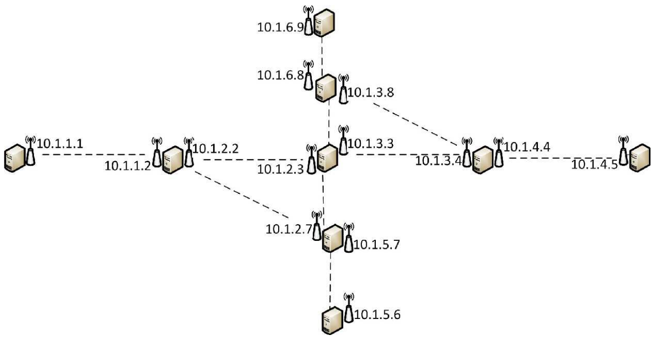

Fig. 3. A Multi-hop topology

An use a different approach for creating the multi-hop topologies; we use different non-overlapping channels for each link. Since orbit has two 802.11 antennas at each node, the intermediate nodes switch the channels when forwarding the packet. We primarily used 802.11a channels in our experiments, since they are not overlapping. We have also used non-overlapping 802.11g channels (channels 1 and 11). Even though we cannot create all the possible topologies similar to the MAC address filtering, it is possible to create a good number of multi-hop topologies.

Figure 3 shows an example multi-hop topology. For the simplicity of discussion, let the nodes be numbered $1, 2, 3, \ldots$. Let the useful channels also be numbered $1, 2, 3, \ldots$. We assign an IP address *10.1.c.i* for an interface on the node i that is transmitting on the channel c. Two nodes can communicate directly, if they have antennas with a common channel. In our example, two antennas share a channel if they are on the same 255.255.255.0 subnet. In the figure, the nodes are numbered first from left to right (1 to 5), and then from bottom to top (6 to 9). The connectivity is shown using dashed lines. Note that nodes 2, 3 and 7 have pairwise connectivity due to sharing of channel 2. In general, the connectivity graph is a set of cliques, one for each channel.

We use the popular Linux AODV implementation, AODV-UU [4], for routing. We modify it to handle multiple interfaces, as it did not have the support of the same.

B. Throughput

In this section we study the throughput of TCP and UDP at different packet block sizes. In this set of experiments, we create a linear (line) topology of multiple nodes, and send high data rate traffic using *iperf*. For the DIPLOMA scheme, we allocate unlimited bandwidth on the capability.

1) Block size: We study the effect of block size on the throughput for TCP and UDP, and select the best block size for further experiments. Recall that the larger block sizes require smaller processing at the sender, but it also has higher per packet overhead and larger penalty on packet losses. The results are average of four iperf for 10 seconds each. The block timeout is 25 ms. The results are similar for 10 ms block timeout.

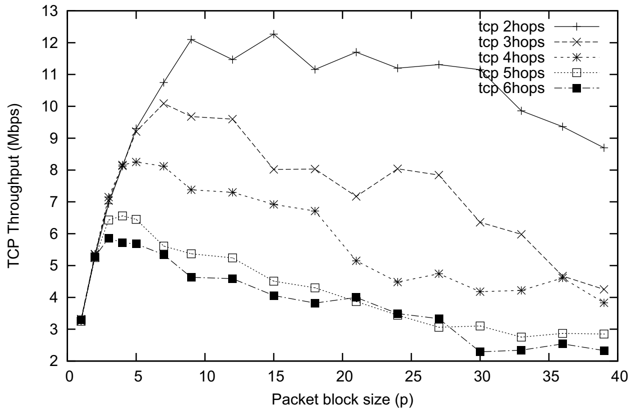

Fig. 4. TCP throughput for different block sizes

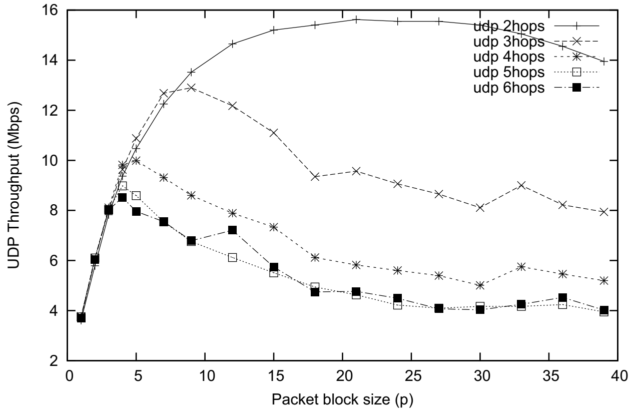

Fig. 5. UDP throughput for different block sizes

Figure 4 gives the TCP throughput for different block sizes on paths of increasing hop lengths. The labels on the plots indicate the path length. For smaller block sizes, the throughput increase linearly. The throughput at that point is limited by the sender's ability to pump traffic, which is limited by the signature computation. As we increase the block size, the processing needed per packet at the sender decreases, and the send rate catches up with the available bandwidth. The end to end bandwidth decreases as the hop count increases. Hence, the linear increase in the bandwidth with the block size stays for longer for smaller hop counts. Once the throughput reaches a maximum point for a given hop length, further increase in the block size decreases the throughput. This is due to two factors. Firstly, the header size increases as the block size decreases, reducing the MSS of the TCP. Second, packet loss of a DATA-FIRST packet will force the entire packet block to be retransmitted, loosing lot of usable bandwidth. The best packet block size for the TCP is between 4 and 9. The best block size decreases as the number of hops increases, which can also explained using a similar argument.

Figure 5 gives a similar result for UDP. Here, the sender was pumping traffic at 20 Mbps. The results for UDP are

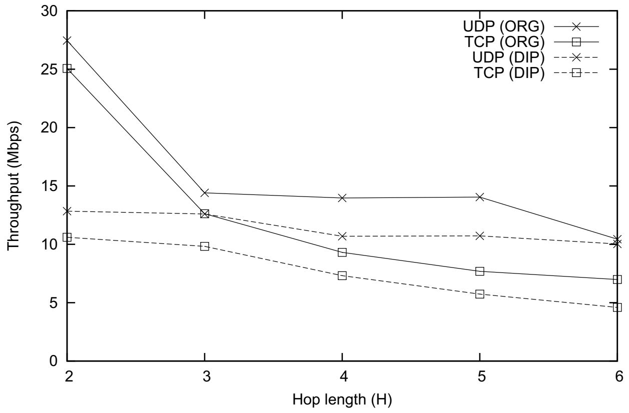

Fig. 6. TCP and UDP throughput comparison

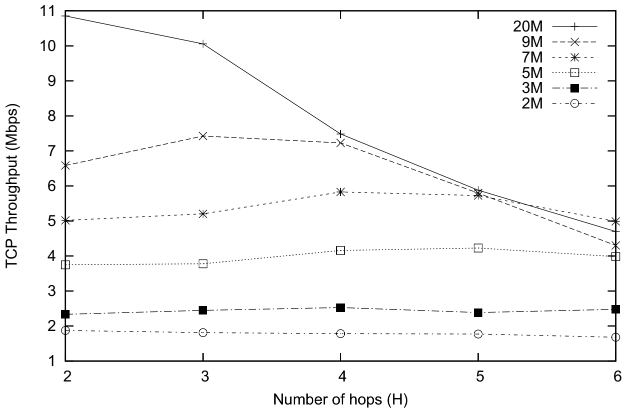

Fig. 7. Bandwidth enforcement for TCP

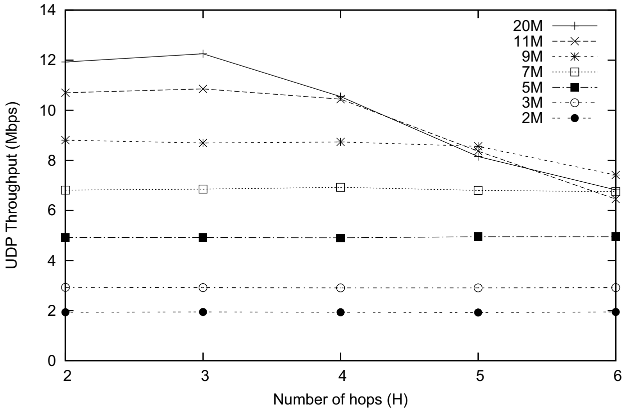

Fig. 8. Bandwidth enforcement for UDP

similar to that of TCP, except that the UDP has higher bandwidth. This is because UDP does not have to deal with the complexities of TCP like guaranteed delivery. Because of the higher bandwidth, the block size at which it achieves maximum throughput is also larger. The block size between 5 and 10 gives the maximum throughput for hops 3 to 6.

We use a block size of 7 for our experiments, as it is a good compromise for UDP and TCP, and for different hop counts.

2) Comparison with original scheme: Now we compare the throughput of the system with and without DIPLOMA. For the DIPLOMA, we use the block size of 7 and allocate unlimited bandwidth to the capabilities.

Figure 6 shows the iperf throughput for TCP and UDP for both the DIPLOMA and the original schemes for various hop lengths. The DIPLOMA throughput is about 23% lower for TCP and 19.8% lower for UDP for hop lengths between 3 and 5. This is because of the extra headers and the extra processing required for the DIPLOMA. For 2 hop distance the bandwidth for the original scheme is substantially higher than that of the DIPLOMA scheme. This is because, at 2 hops the available bandwidth is high, and hence the bottleneck becomes the processing delay. Note that these experiments are conducted at ideal condition, where the nodes were extremely close to each other (1m apart), and hence the performance of the wireless is maximum. In realistic settings, where the nodes are fairly apart to necessitate multi-hop routing, the available bandwidth will not be that high. When the available bandwidth is low, the processing delay is no longer the bottleneck. In those cases we expect the DIPLOMA scheme's headers to be only the factor contributing to its overhead.

We also study the system with the real file transfer application by sending files of size 20 MB. The results are very similar to TCP throughput results. At hop lengths between two and six, the DIPLOMA throughput is between 28% and 34% lower than the original.

C. Bandwidth Enforcement

In this subsection, we study the effectiveness of the bandwidth enforcement capabilities of our implementation. We use a linear topology for these sets of experiments. We allocate different bandwidth on the capabilities and measure the end to end throughput using iperf. Recall that we use the token bucket parameters of rate (in bytes per second) and bucket size (in bytes) to represent the bandwidth constraints. In our experiments we have used the bucket size as $rate/4$, which allows for a burst equivalent to $250\ mS$.

Figure 7 shows the TCP throughput for various bandwidth allocation for capabilities on paths of different hop lengths. The labels on the plots indicate the allocated bandwidth to the capability. The system is able to enforce the bandwidth constraints, as long as there is sufficient available bandwidth. The throughput seen by the TCP is lower than the allocated bandwidth due to capability header overhead. The block size used was 7, which could cause the DATA-FIRST header to be as large as 152 bytes. Even though DATA-NEXT header is smaller (40 bytes), the MSS of TCP was set to 1308 (MTU - TCP/IP header - 152). When the allocated bandwidth is more than the available bandwidth, then the flow receives all of the allocated bandwidth. The plot labeled 20 Mbps is practically the maximum throughput achievable by the system, because of the limited available bandwidth.

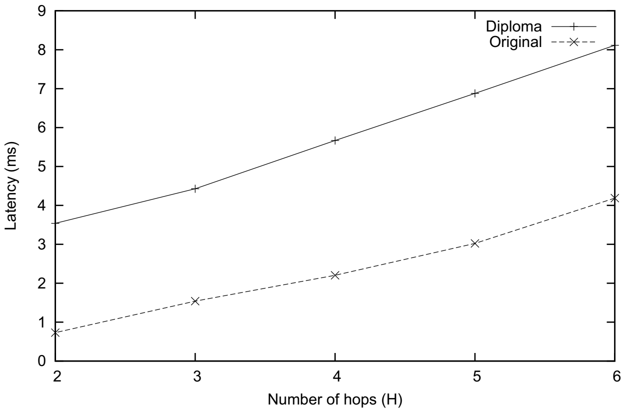

Fig. 9. Packet latency for DIPLOMA and original schemes

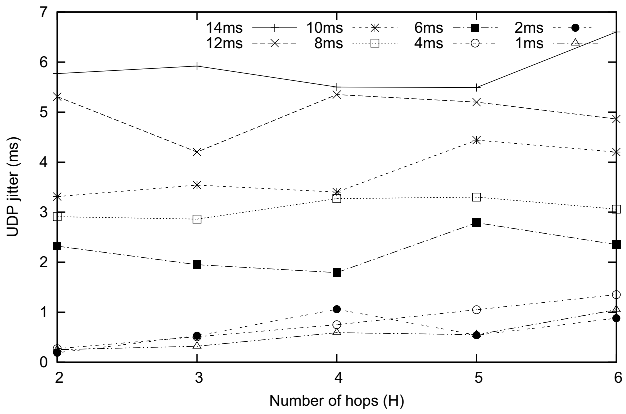

Fig. 10. Packet jitter for DIPLOMA and original schemes

Figure 8 shows similar results for UDP. The enforcement on the UDP traffic gives close to the allocated bandwidth uniformly across the hops. This is because, unlike TCP, there is no flow control in UDP and excess bandwidth is dropped closer to the source. There are enough tokens available at the intermediate nodes, for the packets that are allowed to leave the source, except for the synchronization case (Section IV-A). As expected, the throughput drops for larger bandwidth or hop lengths, when the available bandwidth is less than the allocated bandwidth.

D. Latency and Jitter

We now study the packet latency and jitter for the DIPLOMA, and compare it with the original scheme.

1) Packet Latency: To measure the latency, we use *ping* command to send ICMP echo request/reply. Ping gives the round trip latency, which involves the capability processing delay at both the ends. One important parameter that affects the latency is the block timeout, which is the amount of time the sender has to wait for the packet block to fill before sending the packet. To remove the effect of packet block time, we use the packet block size of one. In this case, the engine sends the packet as soon as it receives the packet.

Figure 9 shows the average round trip latency reported by ping for 20 packets for different hop lengths. As expected, the latency increases close to linear, as the hop length increases. The average latency for the DIPLOMA is 2.8 ms to 3.9 ms higher than the original scheme. This is mainly coming because of the cryptographic operations at both the ends. The latency difference increases as the hop count increases, as there is some processing involved at each intermediate nodes.

2) Packet Jitter: Packet jitter is an important factor in some of the real world applications like Voice over IP (VOIP). Jitter is defined as the average deviation from the mean latency. To measure the jitter, we send UDP packets at low rate (1 Mbps) using iperf. The iperf for UDP reports the jitter values. One important parameter that affects the jitter in DIPLOMA is the block timeout; the packet may wait for the timeout period before the engine sends it, if the block is not full. This is especially true for low bandwidth applications. For small bandwidth latency and jitter critical applications, it is beneficial to use small block timeout, since the the signature operation will not be a bottleneck.

Figure 10 shows the jitter for different block timeout as a function of hop length. The labels on the plots indicate the block timeout. The block size in these experiments was 7. As expected the jitter increases with increase in the hop count or the block timeout. The maximum jitter value for up to six hops is 1.35 ms, for a block timeout of 4 ms. The maximum jitter value in this set of experiments is only 6.60 ms; this occurs on a six hop path at the block timeout of 14 ms.

E. Attacker resiliency

A major goal of the DIPLOMA is to protect the end-host resources, including the protection against the denial of service attacks, by dropping the unauthorized traffic closer to the source. In this section, we study the resiliency of DIPLOMA towards any attacker that does not conform to the protocol and the bandwidth allocation.

To study the attacker resiliency, we use a modified topology on the Figure 3 as follows; we add two more nodes to the right of node 5 (call it 10 and 11), and remove the nodes 8 and 9. We conduct two sets of experiments here. In the first experiment, the good node and the attacker share the channel. In this case the good node is the node 2 and the attacker is the node 7. In the second experiment, there is no sharing of channel between the attacker and any node in the path from the good node to the receiver. In this case, the good node is node 1 and the attacker is the node 6. In both sets of experiments, the attacker is allocated only 1 Mbps bandwidth, but the good node is allocated either 2Mbps, 3 Mbps, or 5 Mbps. The attacker sends as much data as it wants, but the good node conforms to the allocated bandwidth.

1) Attacker and the good sender sharing the spectrum: Figure 11 shows the iperf bandwidth for both the good node and the attacker for both the schemes, when the allocated bandwidth to the good node is 5 Mbps. As it can be seen, for the DIPLOMA scheme, both the nodes receive the allocated

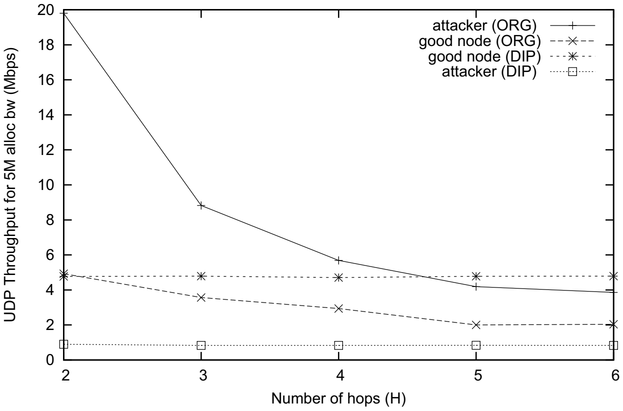

Fig. 11. Resilience to an attacker who is sharing the spectrum

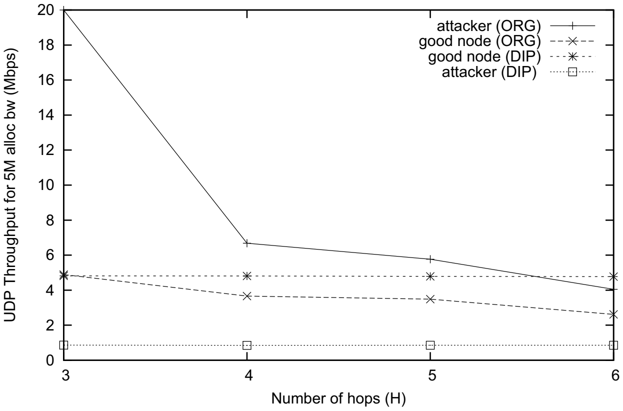

Fig. 12. Resilience to an attacker not sharing the spectrum

bandwidth. In spite of attacker sending large bandwidth, the 2^{nd} hop node drops any excess bandwidth. This essentially protects the end host (receiver), from receiving the unnecessary traffic. The spectrum sharing did not have an effect here, as the good node was able to deliver the allocated bandwidth to the 2^{nd} hop node. In the original scheme, the node that sends the maximum traffic (i.e. the attacker) gets the maximum bandwidth, at the expense of the other nodes.

2) Attacker and the good sender not sharing the spectrum: Figure 12 shows the similar results when the attacker and the good sender are not sharing the spectrum. Here the good node and the attacker get the allocated bandwidth for the DIPLOMA scheme. The attacker hogs most of the bandwidth in the original scheme. It is also interesting to see that the good node in the DIPLOMA scheme ends up getting more bandwidth than the attacker gets in the original scheme for six-hop path. The same thing happens at the hop 5 for the shared spectrum case.

F. Multiple flows

Now we study the performance of the system for multiple flows. We use the topology given in Figure 3 with two source nodes and two destination nodes. Each of the source nodes send UDP iperf traffic of varying bandwidth to each of the destination nodes. Hence, there are four parallel flows in the system. The source nodes are nodes 1 and 6 in the figure. The destination nodes are either nodes 4 and 8 (i.e. 4 hop paths), or nodes 5 and 9 (i.e. 5 hop paths). Figure 13 shows the resulting bandwidth for both the schemes for 4 hop paths, for varying UDP bandwidth. The label on the x-axis shows the source node and the requested UDP bandwidth. Each bar shows the received bandwidth at both the destinations. For the DIPLOMA scheme, the capabilities have the same allocated bandwidth as the requested UDP bandwidth. The results are similar for five hop paths, but with lower bandwidth.

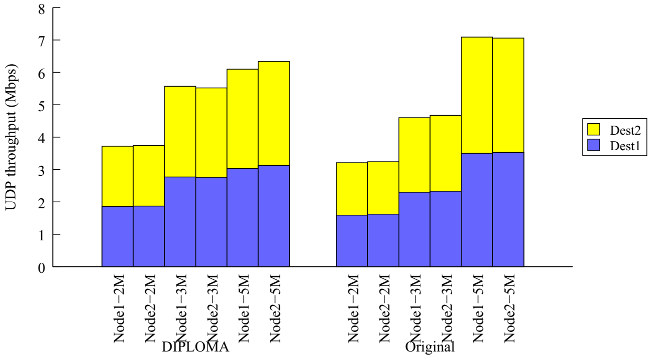

Fig. 13. UDP throughput when there are multiple flows in the system

It can be seen from the histogram that both the schemes receive roughly same amount of bandwidth for all the four flows. The DIPLOMA receives slightly higher bandwidth for the requested bandwidth of 2 Mbps and 3 Mbps. At 2 Mbps speed, the packet loss are minimum, and the send rate by iperf is higher for the DIPLOMA. This may be because the iperf datagram size for the DIPLOMA was 1430 bytes, whereas the size was 1470 for the original. When the requested bandwidth is 5 Mbps, the DIPLOMA and the original receive only 3.11 Mbps and 3.53 Mbps respectively due to the limited available bandwidth.

VI. Related Work

The concept of capabilities was used in operating system for securing resources [15]. Follow-on work investigated the controlled exposure of resources at the network layer using the concept of "visas" for packets [9], which is similar to network capabilities. More recently, network capabilities were proposed to prevent DoS in wired networks [8]. We extend the concept to MANET and use it for both access control rules and traffic shaping parameters [6]. Previous work on distributed firewalls [10] focused on wired fixed-network environments, attempting to protect only end-hosts, using a host-based solution.

Signing and verification of packets between a sender and a receiver were commercially available in early 1990s. Novell's Netware 3.11 and 4.x supported *NCP Packet Signature Option*, where a unique signature was appended to each packet sent between the client and the server [13]. Mitigating DoS attacks by including a message authentication code and the certificate of the sender for each packet has been previously proposed [16]. That work does not study the high overhead associated with sending a large signature or a large certificate on each packet. The authors use game theory to study the problem of dealing with selfish nodes that do not verify the packet signatures, using incentives and punishments. This mechanism or any other reputation based mechanism [11] can also be used in our scheme to deal with selfish nodes.

HEAP [5] mitigates various MANET attacks from outsider nodes by doing a hop-by-hop packet authentication using HMAC. MACs (end-to-end or hop-by-hop) cannot deal with insider attacks. They also cannot provide access control unless different MAC keys are used for different policies. MACs allow rogue nodes to "hide" since MACs are repudiable as all the intermediate nodes in the path between a sender and a receiver need to know the key. Only public key mechanisms allow packet source validation by all intermediate nodes.

VII. Conclusions and Future Work

We implemented a novel distributed deny-by-default architecture for enforcing the security policies in MANETs, in real systems running Linux. The architecture, which is based on the concept of network capabilities, can protect both the end-host resources and the network bandwidth from denial of service attacks, as well as limit the exposure of the MANET to compromised and malicious nodes. We showed that the system is easy to implement, does not require changes to the existing applications, and works well in practice. We presented a packet signature scheme that reduces the public key signature operation, but still provides verifiable integrity protection for the packets. We evaluated the system on a real MANET testbed Orbit. We showed that the impact of the scheme is minimal on throughput, latency, and jitter. We also showed that the scheme allocates resources in a fair manner even in the presence of attackers, and can protect the end-hosts from the attackers. We also showed that the existing applications achieve good performance without modifications. For our future work, we plan to extend the DIPLOMA for multicast traffic, and for protecting routing protocols.

Acknowledgements

This work was supported in part by the National Science Foundation through Grant CNS-07-14277 and by ONR through Grant N00014-09-10757. Any opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF, ONR, or the US Goverment.

References

[1] GloMoSim simulator. *http://pcl.cs.ucla.edu/projects/glomosim/*.

[2] Nefilter/IPtables project home page. *http://www.netfilter.org*.

[3] Orbit Lab Test-bed. *http://www.orbit-lab.org*.

[4] Uppsala University AODV implementation. *http://core.it.uu.se/core/index.php/AODV-UU*.

[5] R. Akbania, T. Korkmaz, and G. Raju. HEAP: A packet authentication scheme for mobile ad hoc networks. *Communications and Networking Simulation Symposium*, 2007.

[6] M. Alicherry, A. D. Keromytis, and A. Stavrou. Deny-by-Default Distributed Security Policy Enforcement in Mobile Ad Hoc Networks. *SecureComm*, September 2009.

[7] M. Alicherry, A. Stavrou, and A. D. Keromytis. Evaluating a Collaborative Defense Architecture for MANETs. *IEEE Workshop on Collaborative Security Technologies (CoSec)*, December 2009.

[8] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. *Proc. of Hotnets-II*, 2003.

[9] D. Estrin, J. C. Mogul, and G. Tsudik. Visa protocols for controlling interorganizational datagram flow. *IEEE Journal on Selected Areas in Communications*, May 1989.

[10] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS)*, pages 190–199, November 2000.

[11] J. Jaramillo and R. Srikant. Darwin: Distributed and adaptive reputation mechanism for wireless ad-hoc networks. *MOBICOM*, 2007.

[12] S. Kaul, M. Gruteser, and I. Seskar. Creating Wireless Multi-hop Topologies on Space-Constrained Indoor Testbeds Through Noise Injection. *IEEE Tridentcom*, 2006.

[13] R. Lee. Netware 4.x performance tuning and optimization: Part 3. *http://support.novell.com/techcenter/articles/ana19931001.html*, October 1993.

[14] R. Russell and H. Welte. Linux netfilter hacking howto. *http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html*, 2002.

[15] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *ACM Trans. on Computer Systems*, 12, February 1994.

[16] X. Wu and D. K. Y. Yau. Mitigating Denial-of-Service Attacks in MANET by Distributed Packet Filtering: A Game-theoretic Approach. *ASIACCS*, March 2007.