# Classification of Firewalls and Proxies

By

Dhiraj Bhagchandka

Advisor: Mohamed G. Gouda (gouda@cs.utexas.edu)

Department of Computer Sciences

The University of Texas at Austin

Computer Science Research and Writing

CS 379H

Spring 2003

# Abstract

The rapid proliferation of the Internet has necessitated the use of computer hardware and software systems that can provide security to the computer network of an organization. Firewalls and proxies constitute an important part of such network security systems. In this paper, we seek to design a novel classification of the firewalls and proxies. We discuss the classification and the applications of firewalls in section 2, followed by a model of the static packet filter firewall in section 2.4. The classification and the applications of proxies are discussed in section 3 along with a model of the FTP proxy in section 3.4.

# 1. INTRODUCTION

In order to develop a comprehensive understanding of the firewalls and proxies, it is instructive to understand the structure of a data packet. Each data packet is of the form shown in Figure 1. In this form, a packet has three headers:

i. Network header
ii. Transport header
iii. Application header

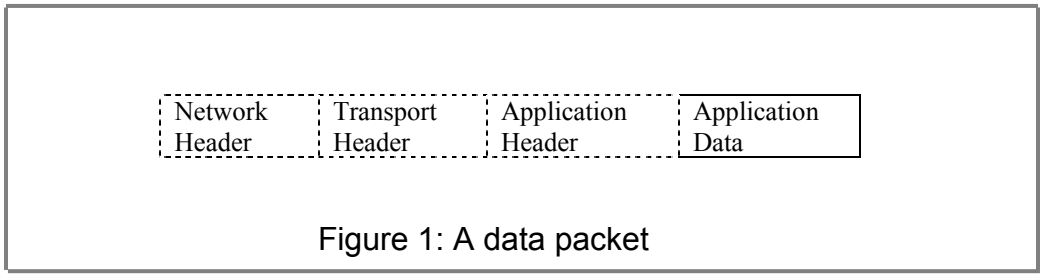


Figure 1: A data packet

Note that the headers of a data packet correspond to the three well-known layers of the protocol hierarchy in the Internet [4]:

i. Network layer
ii. Transport layer
iii. Application layer

Figure 2 shows the hierarchy consisting of these three layers along with the protocols associated with each layer. The network layer consists of only one protocol named the Internet Protocol or IP for short. The transport layer consists of two protocols: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

The application layer consists of several protocols, e.g. the File Transfer Protocol (FTP), the Hyper Text Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP).

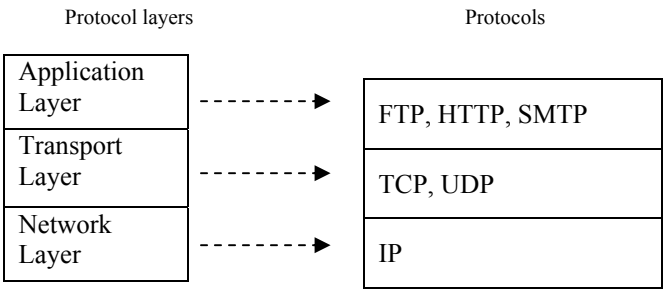


Figure 2: Protocols employed in each protocol layer of the Internet

Our classification methodology is analogous to the structure of the Internet protocol hierarchy shown in the above figure. We next discuss the classification and the applications of firewalls followed by a model of the static packet filter firewall.

# 2. FIREWALLS

## 2.1 Definition

A Firewall is a system of computer hardware and software that provides security to the computer network of an organization. It is generally implemented at every point of contact between the network and the outside Internet, and as such it filters all data packets going in or coming out of the network.

A common practice is to let the flow of outgoing data packets pass, unrestricted and unchallenged, while filtering the flow of incoming packets according to a predefined set of rules, called the rule base. A data packet that matches a rule in the rule base is allowed to pass through the firewall into the network; the remaining packets are considered unsafe for the network and hence, are discarded by the firewall.

## 2.2 Classification

When a firewall receives a data packet, it usually checks one or more headers in the packet. Firewalls, depending on the packet headers they check, may be classified as follows:

a. The Static Packet Filter checks the network header and a part of the transport header.

b. The Packet Filter checks the network header and a part of the transport header.

c. The Connection Filter checks the network header and the transport header.

d. The Application Filter checks the network header, the transport header and the application header.

e. The Application Proxy checks the network header, the transport header and the application header.

We next present a detailed description of each class of firewall. Section *2.2.1* discusses the static packet filter, section *2.2.2* discusses the packet filter, section *2.2.3* discusses the connection filter, section *2.2.4* discusses the application filter and section *2.2.5* discusses the application proxy.

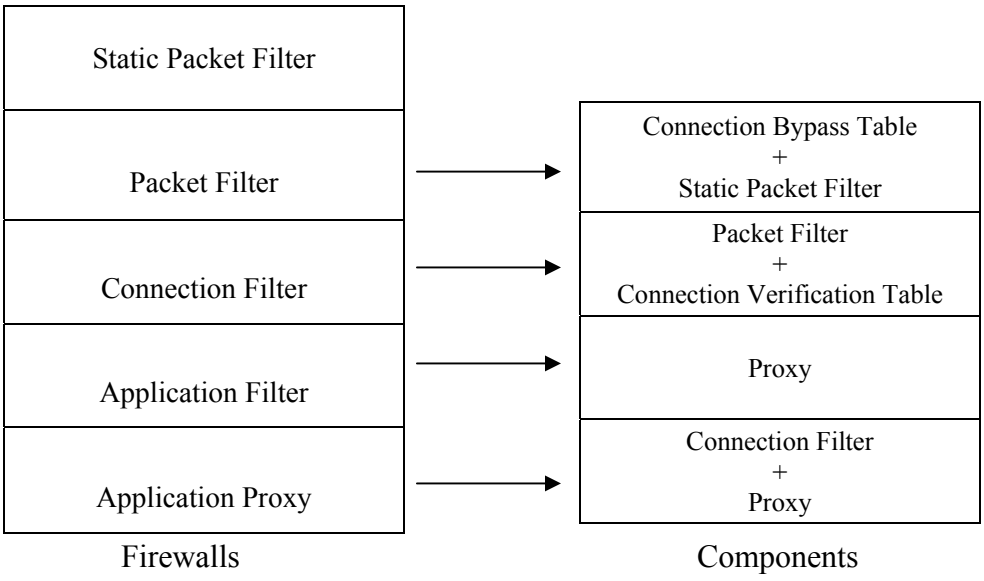


Figure 3: An overview of the firewall classification

### *2.2.1 The Static Packet Filter*

The static packet filter is one of the simplest and oldest firewall architectures [1]. It examines the network header, and a part of the transport header, of the data packet.

The static packet filter checks the following fields in the network header:

a. source IP address
b. destination IP address

It also checks the transport header for the following fields:

a. source port number
b. destination port number

The static packet filter also determines the protocol (TCP or UDP) of the data packet. The values of the above mentioned fields, along with the protocol specification, are used by the static packet filter to determine whether to permit the data packet into the network, or to discard it at the point of entry.

In order to implement the above functionality, every static packet filter maintains a set of packet filtering rules. A packet filtering rule specifies the filtering policy for a data packet, i.e. whether to admit the packet into the network or to discard it at the firewall. The operation of a rule is based on the data packet's source and destination IP addresses, the source and destination port numbers, and the transport protocol of the packet. In section 8, we present a comprehensive discussion of a static packet filter. In that section, we also present the model of a static packet filter firewall which we have designed to protect the network of an arbitrary university.

In any organization, the number of packet filtering rules in a static packet filter may increase dramatically. In order to design and implement them correctly, the packet filtering rules are often classified in three broad categories:

A. Rules which explicitly allow a packet to pass the firewall into the network.
B. Rules which explicitly deny a packet to pass the firewall into the network.
C. Default rules are applied to packets that do not match any other rule in the set. The default rules may either allow a packet the pass into the network, or deny the packet to pass the firewall into the network.

There are two approaches on the use of the default rule (the 3rd category) by the static packet filter. (i) A lenient approach allows a packet to pass through the firewall into the network, unless it is explicitly denied by a prior rule. (ii) A stringent approach denies a packet to pass the firewall, unless it is explicitly allowed by a prior rule. In our example in section 8, we present two versions of the academic firewall. The first version is based on the lenient approach, while the second version is based on the stringent approach.
A static packet filter is typically very fast compared to other firewall architectures. It is flexible, cheap and easy to implement. However, since it only looks at the IP addresses (source and destination), the port numbers (source and destination) and the transport protocol, it provides the lowest level of security amongst all other firewalls [3]. Some of the shortcomings of a static packet filter are as follows:

A. A static packet filter examines the IP addresses in a data packet, and matches them to the rules of a table. However, it does not know the difference between a legitimate and a forged IP address. This gives rise to a very critical threat called *IP address spoofing*. IP address spoofing allows a malicious user to insert his packets in the connection stream after replacing the original source IP address

with the IP address of a trusted host on the network. When such a packet comes across the static packet filter, it is allowed to pass into the network. This packet may carry malicious data and hence, is a security threat to all computers in the network of the organization.

B. Many services, like FTP, in operation today require the use of dynamically allocated ports for responses. So an administrator of the firewall has to open an entire range of ports (usually in the higher range) to acknowledge incoming connections. However, this renders the port filtering useless and is thus, a security vulnerability at the firewall.

### *2.2.2 The Packet Filter*

The packet filter is an extension of the static packet filter. The packet filter examines the network header and a part of the transport header of the data packet. It checks the source and destination IP addresses in the network header. It also checks the transport protocol (TCP or UDP) of the data packet, and the source and the destination port numbers in the transport header. In order to provide state awareness, the packet filter also maintains a table of connection streams called the Connection Bypass table.

All data packets, which have the same IP addresses (source and destination), port numbers (source and destination) and transport layer protocol (TCP or UDP), form a unique connection stream [4]. A packet filter serves to associate each packet with it's connection stream. When a data packet reaches the packet filter, the latter compares the incoming packet's connection stream with the list of connection streams that it maintains in the connection bypass table. If the data packet is found to arrive on a previously encountered connection stream (for which an entry has already been made in the table), it is allowed to pass into the network without any further inspection (i.e. it bypasses all subsequent filtering rules) [1].

However, if the data packet arrives on an unknown connection stream, it is first verified according to the static packet filtering rules. Only if the data packet is found to be permissible according to those rules, the packet filter then allows the packet to pass into the network. Simultaneously, the packet filter also generates an entry, corresponding to the connection stream of that data packet, into the connection bypass table. This allows all future packets coming on that connection stream to pass into the network, bypassing all filtering rules. In simple terms, the packet filter is aware of the difference between a new and an established connection.

Therefore, the structure of a packet filter can be represented as:

Connection Bypass Table
+
Static Packet Filter

The connection bypass table is the most important aspect of the packet filter. The construction of the table therefore, requires special consideration. A packet filter generally implements the connection bypass table as a hash table, which typically resides in the RAM. However, a fixed sized hash table can be a prime candidate for a *resource starvation attack*. In this kind of attack, a malicious user can open thousands of simultaneous connections to a host behind the firewall. This would exhaust the connection bypass table, allowing the user to insert malicious packets into the network through the firewall. A packet filter can take one of the following actions when all entries in the connection bypass table are exhausted [3]:

A. The packet filter can block all new connection streams from generating an entry in the connection bypass table.

B. The packet filter can delete the least recently used (LRU) entry, assuming that it is a dead connection. This alternative is more pragmatic than the previous option.

C. The packet filter can do a random early detection (RED) that randomly selects and drops packets when table starts to get full.

D. The packet filter can time out entries in the connection bypass table.

The benefit of the packet filter over the static packet filter is the drastic reduction of the filtering overhead associated with each packet in the connection stream. This overhead is quite significant, as hundreds of packets may arrive on each connection stream. In the case of a packet filter, only the first packet for each connection stream will be verified. Therefore, the latency for all subsequent packets is reduced to a great extent.

Since the packet filter is quite similar to a static packet filter, it inherits all the shortcomings of the latter. In addition, a packet filter has the following pitfalls [3]:

A. A common form of attack against this kind of firewall is the resource starvation attack, which basically results from poor implementation of the hash table for the connection bypass table. It is recommended that firewall should follow RFC (Request For Comment) recommendations in the establishment of a connection, and it should open the connection only after the three way handshake is complete [1].

B. The most critical shortcoming of a packet filter is that all subsequent packets of a connection stream are allowed to pass without further inspection. This might allow attackers to hijack connections by inserting their own malicious packets in the connection stream.

### *2.2.3 The Connection Filter*

The Connection Filter validates a connection before allowing the packet to pass into the network. It examines the network header and the transport header of the data packet. When a packet arrives at the connection filter, the latter examines certain fields within the network and the transport headers of the packet.

The fields examined within the network header are:

a. Source IP address
b. Destination IP address

The fields examined within the transport header are:

a. Source Port number
b. Destination port number

For TCP connections, in addition to the above fields the connection filter also examines the SYN and the ACK flags, and the sequence numbers involved in the TCP handshaking process [3]. This helps the connection filter to verify the validity of the connection that is being established.

The connection filter maintains a Connection Verification table for the above purpose. An entry specifies the state of the flags (i.e. SYN and ACK flags) and the sequence number of the last packet arriving in the connection stream. The structure of the connection filter can therefore, be expressed as:

Packet Filter
+
Connection Verification Table

After a packet has been verified by the packet filtering component of the connection filter, it is then verified against the connection verification table (in case of TCP packets). Basically, the Connection Filter verifies the legitimacy of the TCP handshaking procedure by examining the state of the flags and the value of the sequence numbers in the packets. If the handshaking procedure is found to be appropriate, the connection filter allows the remaining packets of the connection stream to pass into the network. However, for each of those remaining packets the connection filter verifies the sequence numbers and makes sure that the SYN is appropriately un-flagged.

A connection filter is a step up from the packet filter, in the sense that it verifies legitimate connection establishment by the outside user to a host inside the firewall. Although it suffers from many shortcomings of the packet filter, it provides security from malicious hackers who might hijack connections by inserting their own packets into the connection stream.

### *2.2.4 The Application Filter*

The Application Filter is a firewall that analyzes the content and structure of the data payload before forwarding the packets inside the firewall. Because the application filter examines the entire data, it has to sever the connection between the outside host (e.g. a client requesting the connection) and the inside host (e.g. a server responding to the connection request). The application filter examines the network header, the transport header, and the application header of the data packet.

The application filter runs proxies to analyze the content of a data packet. This proxy simply observes the connection passing through the filter, and looks for abnormalities in these connections. The application filter would allow the packets generated by only those services (like HTTP, FTP) whose proxies are running on it. For example, if an application filter ran the FTP and HTTP proxies, only packets generated by these services could pass through the firewall. Packets of all other applications would be dropped [3]. An application filter is rarely implemented as a firewall by itself, and often is bundled up with other categories of firewalls.

### *2.2.5 The Application Proxy*

The Application Proxy is the most complex and secured firewall amongst all the other firewall architectures [1]. The application proxy examines the network header, the transport header, and the application header of a packet.

Fields examined within the network header:

a. Source IP address
b. Destination IP address

Fields examined within the transport header:

a. Source port number
b. Destination port number

Fields examined within the application header:

The various fields of the header of an application protocol like HTTP, Telnet, etc.

Since the application proxy examines data payload at the application layer, it severs the connection between the outside and the inside host. It therefore, establishes two separate connections: a connection between the outside host and itself, and another connection between itself and the inside host. The application proxy runs a proxy on top of a connection filter. The structure of an application proxy can therefore, be represented as:

Connection Filter
+
Proxy

An important difference between the application filter (discussed in section *2.2.4*) and the application proxy is that the latter does not simply observes the packets passing through it. It also interprets the intended action of every data packet. If the action is not legal according to the security policy of the firewall, the packet is dropped. However if the action is found to be legal, the application proxy reconstructs a new data packet according to the intentions of the original. In theory, nothing will pass through the application proxy unchanged unless it is in exactly the format that the proxy would use to write it [1,3]. Therefore, only the data which found acceptable to the application proxy is copied from the original packet to the new packet constructed by the proxy. After the recreation, the new packet is sent to the host inside the firewall while the original packet is simply dropped.

The application proxy is the most secured firewall among all other firewall architectures [1]. The reconstruction of the packet protects all hosts inside the firewall from an entire class of covert channel attacks. However, this is also the most expensive and complex firewall. The examination at the application layer greatly increases the latency of each packet.

## 2.3 Applications of Firewalls

A firewall is primarily used to provide security to the computer network of an organization. Since a firewall may be used to prevent a large class of network attacks, we may classify the applications of firewalls as follows.

1. *Filtering*: Firewalls can filter a data packet at various layers of the Internet protocol hierarchy. A firewall is generally deployed at the perimeter of the computer network of an organization. It can, therefore, examine every data packet that goes in or comes out of the computer network.

   This feature allows a firewall to deny the malicious data packets from entering the computer network of the organization.

2. *DOS prevention*: A firewall is often used to prevent *Denial-of-Service* attacks launched against the hosts in the computer network of the organization. Such a firewall enforces a maximum on the number of simultaneous connections that can be opened on any host at a given point of time.

   At this stage, if a user in the outside Internet desires to open a new connection on the host, the firewall simply rejects the data packet from entering the computer network.

3. *Detection of IP address spoofing*: A firewall can be configured to detect and filter a data packet that has an incorrect source IP address. In order to check the legitimacy of the source IP address, the firewall connects to the DHCP server of the source host and verifies that the hardware address i.e. the MAC address of the source host corresponds to its IP address [3].

## 2.4 A model of the static packet filter firewall

We next present a model of the static packet filter firewall which monitors the network of an arbitrary university.

In the following example, the university is assigned a class B network address: 128.24.0.0/16. A class B network address (which is of the form "IP address/mask") is used to specify a network of machines, all of which share the same 16 bits in the first part of their IP address. In our example, all machines in the university have an IP address of the form '128.24.x.y' (0 <= x,y <= 255). Therefore, the university can assign $2^{16}$ or 65,634 different IP addresses i.e. it can have 65,634 machines on its network, each assigned a different IP address.

We further classify the machines in this university in different groups: the research machines, the teaching machines, the faculty machines and the administrative machines. Each group of machines is also assigned one or more class C network address. A class C network address (which is of the form "IP address/mask") is used to specify a network of machines, all of which share the same 24 bits in the first part of their IP address e.g. if the class C network address assigned to a particular group is 128.24.8.0/24, then all hosts in that group are assigned an IP address of the form "128.24.8.x" (0<= x <= 255). Therefore, each group can assign $2^8$ or 256 different IP addresses i.e. it can have 256 machines on its network, each assigned a different IP address. We now detail the specifications of each group, including their Class C network addresses and a description of the key functions that they provide in the university.

A. The research machines: The research machines are used for pursuing research work within the university. We assign five class C network addresses to the group of research machines. These network addresses are as follows:

   i) 128.24.0.0/24
   ii) 128.24.1.0/24
   iii) 128.24.2.0/24
   iv) 128.24.3.0/24
   v) 128.24.4.0/24

   Note that this group can have (256 * 5) or 1280 research machines, where each machine is assigned a unique IP address.

B. The teaching machines: The teaching machines are used by students of the university for various academic purposes. In order to acknowledge this requirement, we assign five class C network addresses to the group of the teaching machines. These network addresses are as follows:

i) 128.24.5.0/24
ii) 128.24.6.0/24
iii) 128.24.7.0/24
iv) 128.24.8.0/24
v) 128.24.9.0/24

Therefore, the university can have 1280 teaching machines in its network.

C. The faculty machines: The faculty machines are used by the faculty and staff members of the university. In order to accommodate this requirement, we assign five class C network addresses to the group of faculty machines. The class C network addresses are as follows:

i) 128.24.10.0/24
ii) 128.24.11.0/24
iii) 128.24.12.0/24
iv) 128.24.13.0/24
v) 128.24.14.0/24

This allows the university to support 1280 faculty machines in its network.

D. The administrative machines: The administrative machines in the university are used for various administrative purposes like maintaining student records, disseminating information to other universities, etc. We assign five class C network addresses to the group of administrative machines in the university. These network addresses are as follows:

i) 128.24.15.0/24
ii) 128.24.16.0/24
iii) 128.24.17.0/24
iv) 128.24.18.0/24
v) 128.24.19.0/24

Thus, the university can have 1280 administrative machines in its network, assigning each machine a unique IP address.

Figure 4 shows the network of the university with the four groups of machines. In this figure, 'R1' is a router which is deployed by the university. R1 receives all incoming packets to the university from the outside network, and also routes all outgoing packets from the university to the outside network.

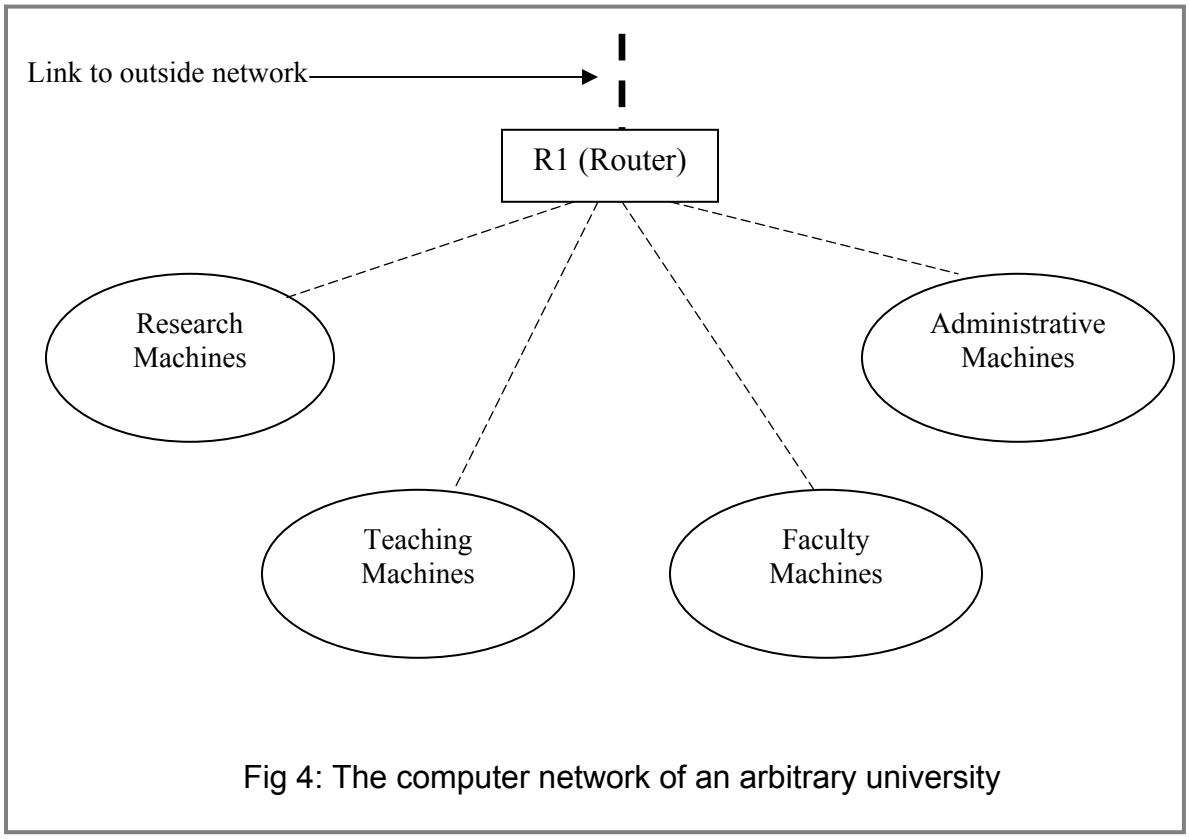


Fig 4: The computer network of an arbitrary university

Our firewall follows a lenient policy by allowing all data packets, which have not been explicitly disallowed, to enter the network.

As mentioned earlier, all incoming and outgoing packets pass through the router R1. We therefore, implement the firewall at R1 in order to monitor and filter all data packets coming in or going out of the university.

The firewall consists of several packet filtering rules. A packet filtering rule specifies an action that should be taken by the firewall for each data packet. The action is either to deny the data packet an entry into the network, or to allow it into the network. Every rule in the firewall is of the following format:

protocol remote address operation local address interface

1. The "protocol" is either TCP, UDP or ICMP. If the value of this field is set to 'all', it implies that the data packet may belong to any of these protocols.
2. The "remote address" is generally either a single IP address - "IP address", or a block of IP addresses - "IP address/mask". In the latter case, the mask specifies the number of bits in the data packet's IP address that should match the bits in the given IP address. e.g. 128.83.139.0/24 is the block of IP addresses in the range: 128.83.139.0 – 128.83.139.255. A data packet with an IP address in this range

would match the value of the ‘remote address’ field. In either case (i.e. a single IP address or a block of IP addresses), a port number can also be specified. In this case, the format of the remote address would be “IP address – port number” and “IP address/mask - port number” respectively.

If the value of this field is set to ‘all’, it implies that the action should be taken independent of the remote IP address.

3. The ‘operation’ is either of the format “X - > log”, which means block packets going to local address and log the event, or “- > log”, which means allow packets going to the local address and log the event. However, the operation may also specify not to log the event, in which case the two rules are “X - > ” and “- >” respectively.
4. The “local address” is the destination address of a machine in the network of the university. The format of the local address is similar to the format of the “remote address”.
5. The “interface” specifies the router interface on which the particular rule would apply. In our example, all data packets pass through interface 1 of the router R1. Hence, the interface value for every rule is set to 1.

In the following example, the comments in the are preceded by a ‘#’. The comments are inserted for better readability and are not part of the firewall. A brief summary also precedes each set of rules, which describes the functionality of the rules in the firewall.

```
# the format of a rule is:

protocol   remote address   operation   local address      interface

include <services.hlfl>
```

==========

**In the following rules, we disallow certain applications by specifying the port number associated with each application**

==========

```
udp|tcp    (any)    X-> log (any sunrpc | any 2049 | any 7)    on 1    # NFS (sunrpc), NFSD (2049), TCP echo (7)
tcp        (any)    X->      (any 6346)                        on 1    # gnutella
udp        (any)    X-> log  (any 161 | any 162)               on 1    # snmp
udp        (any)    X-> log  (any 1993)                        on 1    # cisco snmp
udp        (any)    X-> log  (any 67 | any 68)                 on 1    # bootp/dhcp
udp        (any)    X-> log  (any 49)                          on 1    # tacacs, a protocol that allows a Network
                                                                        # Access Server (NAS) to offload the user
                                                                        # administration to a central server
```

==========

**The following rule disallow packets that have source IP address equal to the IP address of a host inside the network.**

==========

```
all   (128.24.0.0/16)  X-> log (any)   on 1
```

==========

**The following rule allows domain name service (dns) requests to particular machines in the administrative group of machines maintained in the university.**

==========

```
udp   (any)   ->   ((128.24.5.3 | 128.24.6.31) 1949)   on 1
```

==========

**The following rule allows simple mail transfer protocol (smtp) requests to particular machines in the administrative group of machines maintained in the university.**

==========

```
tcp   (any)   -> ((128.24.139.9 | 128.24.139.10 | 128.24.120.2) 25)   on 1
```

==========

**Hosts within a particular remote network have been found to adopt malicious practices in the past. The following rules block all hosts in that remote network from establishing a communication to any host in the university's network.**

==========

```
# The two network addresses blocked by the firewall are: 189.43.0.0/16 and 202.79.0.0/16. In order to ease the
# readability, we define the set of these two network addresses are 'blcknetwork'.

define blcknetwork 189.43.0.0/16 | 202.79.0.0/16

all   (blcknetwork) X->  log (any)   on 1                    # blocks packets from the range of IP addresses
                                                             # defined above
```

==========

**The following rule disallows IP anycast.**

==========

all (255.255.255.255 | 0.0.0.0) X-> log (any) on 1

==========

**The following rule disallows IP multicast**

==========

all (224.0.0.0/8) X-> log (any) on 1

==========

**Some particular hosts have been found to adopt malicious behavior in the past. These rules block data packets from all those hosts. Such hosts are usually blocked when they are found to scan the university machines for security vulnerabilities.**

==========

all (204.69.7.91) X-> log (any) on 1
all (189.13.60.34) X-> log (any) on 1

==========

**Some machines in the research group requested unrestricted access to outside Internet. This requires allowing all packets to enter the network, which have the destination IP address equal to the address of the research machines.**

==========

all (any) -> (128.24.2.0/25) on 1

==========

**The following rules allow Post Office Protocol (POP), POP3 and Internet Mail Access Protocol (IMAP) requests to specific administrative machines in the university.**

==========

tcp (any) -> ((128.24.6.119 | 128.24.8.32) 109-110) on 1 # POP (109), POP3 (110)
tcp (any) -> ((128.24.6.119 | 128.24.8.32) 143) on 1 # IMAP

==========

**Few faculty members of the university pursue collaborative research work with various colleagues at other universities. Those colleagues may require unrestricted access to the machines of the faculty member with whom they work. The following rules grants two such remote machines (possibly in other university campuses) access to the two faculty machines in this university respectively.**

==========

all (128.91.42.2) - > (128.24.10.53) on 1 # 128.91.42.2 is IP address of the remote machine, 128.24.10.53 is
# the teaching machine
all (128.12.145.79) - > (128.24.14.97) on 1 # 128.12.145.79 is IP address of the remote machine, 128.24.14.97 is
# the teaching machine

==========

**Disallowing packets associated with certain applications; POP, POP3 and IMAP have been earlier allowed to only some specific IP addresses (mail servers), and hence are blocked with any other destination IP address.**

==========

```
tcp       (any) X-> log (any 6667)          on 1    # Internet Relay Chat (IRC)
tcp       (any) X-> log (any 6112)          on 1    # /usr/dt/bin/dtspcd
tcp       (any) X-> log (any 109-110)       on 1    # POP, POP3

udp       (any) X-> log (any 1949)          on 1    # DNS
tcp       (any) X-> log (any 25)            on 1    # SMTP
tcp       (any) X-> log (any 1433)          on 1    # Microsoft SQL hole
tcp|udp   (any) X-> log (any 135)           on 1    # Microsoft portmapper
tcp|udp   (any) X-> log (any 137-139)       on 1    # NetBIOS
tcp|udp   (any) X-> log (any 445)           on 1    # NetBIOS
tcp       (any) X-> log (any 143)           on 1    # IMAP
tcp       (any) X-> log (any 540)           on 1    # UUCP
tcp       (any) X-> log (any 515)           on 1    # printer
tcp       (any) X-> log (any 512)           on 1    # exec
tcp|udp   (any) X-> log (any 514)           on 1    # shell (TCP), syslog (UDP)
udp       (any) X-> log (any 69)            on 1    # tftp
```

==========

**Some teaching machines allow remote login sessions. These would allow students to connect to the machines from a remote destination. In the following rule, we allow people to use Virtual Network Computing (VNC) application to connect to the two designated teaching machines.**

==========

```
tcp (any)  -> ((128.24.16.3 | 128.24.17.59) 5900)  on 1       # VNC
```

==========

**The following default rule allows all other data packets that have not been explicitly denied by a previous rule.**

==========

```
all (any)  - > (any)
```

# 3. PROXIES

## 3.1 Definition

A proxy is a system of computer hardware and software which provides security to the applications in the computer network of an organization.

In order to develop a comprehensive understanding of the proxy, we review the structure of a data packet. As discussed earlier in section 1, every data packet consists of three headers apart from the Application Data. These headers are

(i) The Network header
(ii) The Transport header
(iii) The Application header

These three headers correspond to the three well-known layers of the protocol hierarchy in the Internet, which are

(i) The Network layer
(ii) The Transport layer
(iii) The Application layer

The proxy examines the Application header and the Application Data of a data packet. Proxies are generally deployed as part of the Application Monitor Filter or the Application Proxy Filter firewall.

The Application layer consists of several protocols such as the HTTP, the FTP and the Telnet. Each of these protocols requires a unique proxy designed for it. In this paper, we refer to the proxy which has been designed for the HTTP as the HTTP-Proxy, the proxy that has been designed for the Telnet as the Telnet-Proxy, and so on.

**Direct connection between a client and a server**

In order to gauge the usefulness of a proxy, it is instructive to first look at an example in which a client can directly connect to a server. In this example, setting up a connection between the client and the server does not require the services of a proxy.

Fig 5 illustrates this example. In this figure, the client **c.arb.com** belongs to the network **arb.com** and has the IP address ‘c1.c2.c3.c4’. The FTP server **s.rnd.com** belongs to the network **rnd.com** and has the IP address ‘s1.s2.s3.s4’. We assume that the client intends to connect to the FTP server **s.rnd.com**. The FTP service is running on the TCP port 21 on this server.

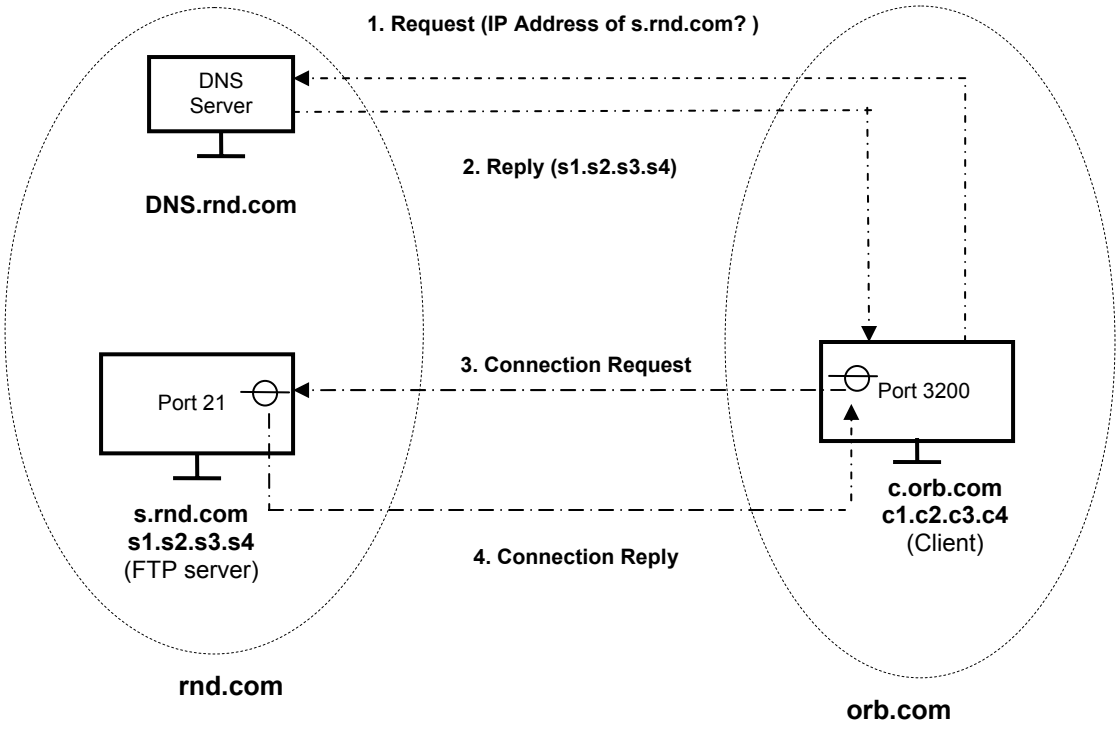


Fig 5: An FTP connection model between a client and a server

In order for the client and the FTP server to exchange request and reply messages, the client needs to establish a TCP connection to the FTP server. For this purpose, the client should first (a) obtain the IP address of the FTP server, and then (b) send a TCP connection request message to the FTP server. We now discuss these two steps in detail.

***(a) Obtaining the IP address of the FTP server***

In order to obtain the IP address of the FTP server, the client needs to resolve the name **s.rnd.com** by generating a DNS query. This query propagates through the root of the DNS tree and eventually reaches the authoritative domain name server for **s.rnd.com**. In the above figure, we refer to this authoritative domain name server as **DNS.rnd.com**.

The authoritative domain name server replies back to the client **c.orb.com** with the IP address of the FTP server **s.rnd.com**, which is 's1.s2.s3.s4'.

***(b) Sending a TCP connection request message to the server***

On receiving the IP address of the FTP server, the client **c.arb.com** now prepares to send a connection request to it. The client randomly selects a TCP port (port number 3200 in

this example) and sends a connection request from this port to the FTP server **s.rnd.com** at its TCP port 21 (the FTP service runs on the TCP port 21).

The FTP server receives the connection request on TCP port 21, and sends back an acknowledgement to the client at TCP port 3200. The client, on receiving this acknowledgement message from the FTP server, sends back a message acknowledging the receipt of the last message. This also completes the *TCP handshaking process* between the client and the FTP server.

After the completion of the TCP handshaking process, the client is said to be **connected** to the FTP server. At this instance, the client and the FTP server can exchange request and reply messages. In particular, the client sends request messages to the FTP server and the FTP server sends back reply messages in lieu of the requests sent by the client.

Every request message sent by the client to the FTP server has the following parameters:

*Source IP address* : = *c1.c2.c3.c4*
*Source port number* : = *3200*
*Destination IP address* : = *s1.s2.s3.s4*
*Destination port number* : = *21*

Every reply message sent by the FTP server to the client has the following parameters:

*Source IP address* : = *s1.s2.s3.s4*
*Source port number* : = *21*
*Destination IP address* : = *c1.c2.c3.c4*
*Destination port number* : = *3200*

We revisit the above connection model, one which additionally contains an FTP proxy, in the section 3.4. In order to understand the differences between these two versions of the FTP connection model, it is instructive to first examine the different types of proxies.

## 3.2 Classification of Proxies

Proxies can be classified as (a) Visible proxies and (b) Transparent proxies [5]. We discuss each of these proxies in detail and also present an overview of their differences.

We say that the hosts, which are protected by a proxy, are members of the set **H**. Every other host that is not protected by the proxy is a member of the set **U**.

Therefore,

**U** = Complement (**H**)

Figure 6 further illustrates this idea. In this figure, a host is either a member of the set **H** or a member of the set **U**. There are four hosts in the set **H**. These are H1, H2, H3 and H4. The set **U** contains every other host except these four hosts. We also denote the proxy as **P**. This proxy protects all hosts in the set **H**.

A host in set **H** can send messages to another host that is either in set **H** or in the set **U**. Likewise, a host in set **U** can send messages to any other host that is either in set **H** or in set **U**. We say that this communication environment is **unrestricted**.

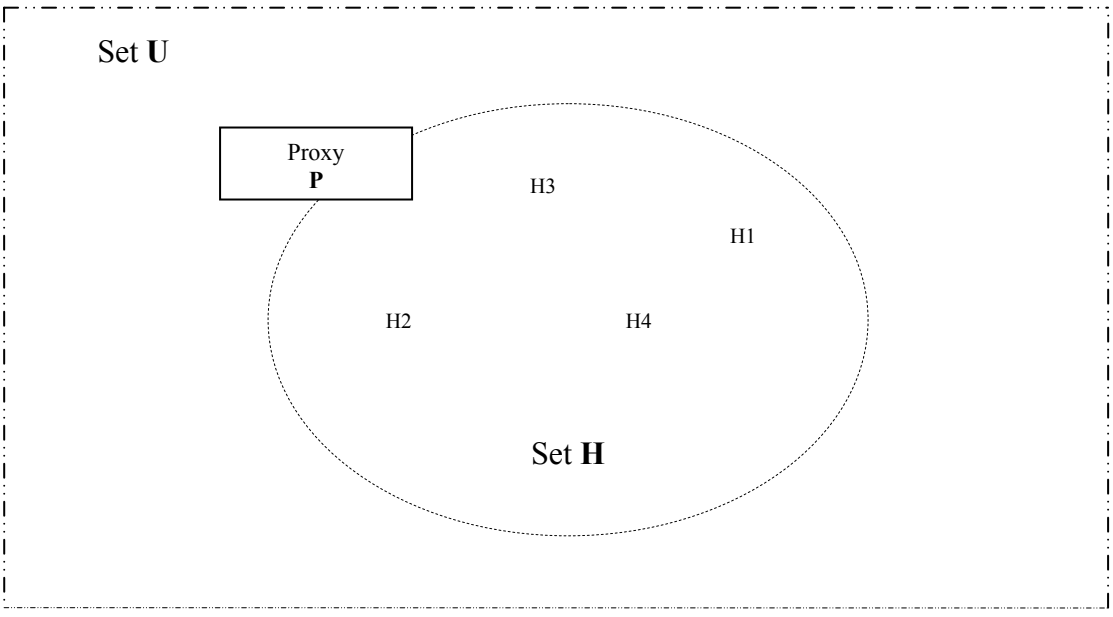


Fig 6: An overview of the distribution of all hosts in the Internet

The proxy **P** examines every data packet that goes in or comes out the set **H**. Therefore, the proxy will examine a data packet in the following two cases:

a. *A host in the set **H** sends a data packet to a host in the set **U***

b. *A host in the set **U** sends a data packet to a host in the set **H***

We discuss each of these cases for both the visible proxy and the transparent proxy.

### *3.2.1 Visible Proxies*

A visible proxy is a system of computer hardware and software that receives a data packet from a host in one set and forwards it to a host in another set. While forwarding it, the visible proxy examines the Application header and the Application Data of the data packet.

We now discuss the two cases in which the visible proxy examines a data packet.

*a. A host in the set **H** sends a data packet to a host in the set **U***

Let us imagine that the host **h1** in the set **H** intends to send a data packet, **D**, to the host **u1** in the set **U** on the latter's TCP port 21. The hosts in the set **H** are protected by the visible proxy **V**. Fig 7 helps us illustrate this case.

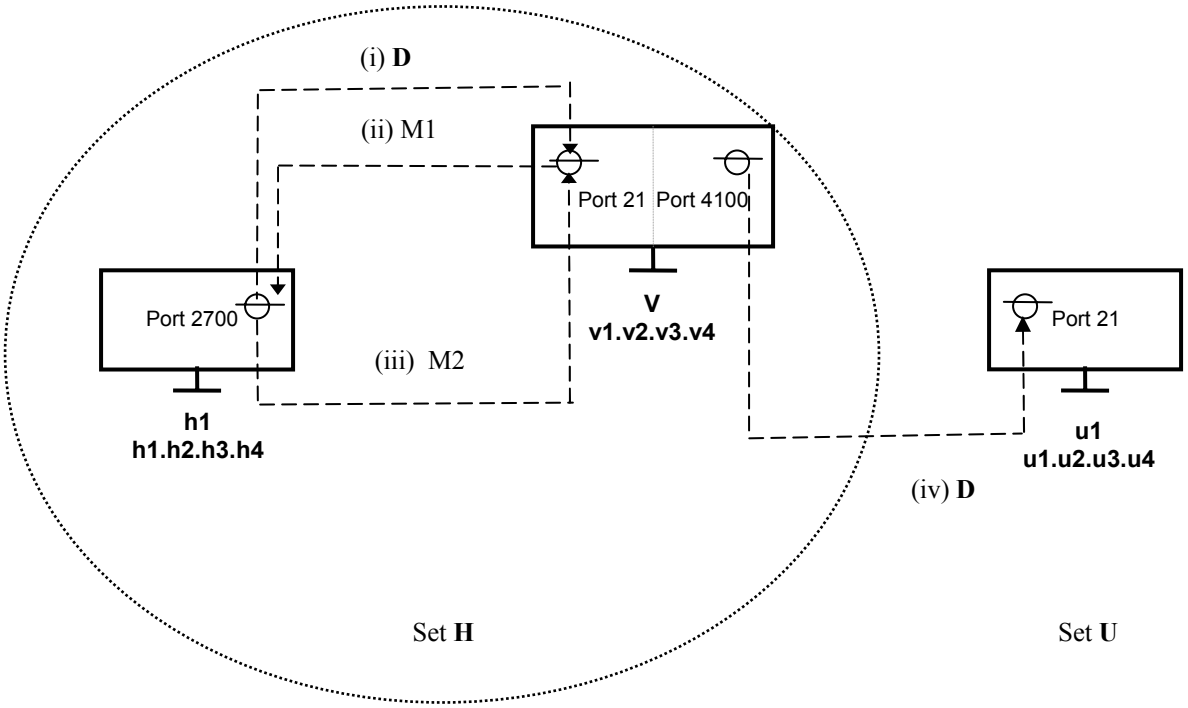


Fig 7: A host sending a message through a visible proxy

The following steps outline the procedure followed in sending this data packet.

1. In order for the host **h1** to send the data packet **D**, it needs to obtain the IP address of the host **u1** in the set **U**. The host **h1** generates DNS queries for this purpose. However, instead of the IP address of the host **u1**, it gets back the IP address of the visible proxy **V** which is *v1.v2.v3.v4*.

2. The host **h1** chooses a random TCP port (port 2700 in this example). It sends the data packet **D** from that port to the visible proxy **V** at the TCP port 21. This data packet has the following parameters:

   *Source IP address* : = *h1.h2.h3.h4*
   *Source port number* : = *2700*
   *Destination IP address* : = *v1.v2.v3.v4*
   *Destination port number* : = *21*

3. The visible proxy **V** sends a message back to the host **h1** asking for the name of the destination host in the set **U** to whom **h1** would like to send the data packet **D**. We denote this message as M1.

4. The host **h1** replies back to the visible proxy with the name of the host in the set **U**, which is **u1**. We denote this message as M2.

5. The visible proxy **V** chooses a random TCP port number (port 4100 in this example). It sends the data packet **D** from that port to the host **u1** at TCP port 21. While forwarding the data packet **D**, the visible proxy **V** examines the Application header and the Application Data of **D**. This data packet has the following parameters:

   *Source IP address* : = *v1.v2.v3.v4*
   *Source port number* : = *4100*
   *Destination IP address* : = *u1.u2.u3.u4*
   *Destination port number* : = *21*

Therefore, the data packet **D** finally reaches from the host **h1** in the set **H** to the host **u1** in the set **U**.

*(b) A host in the set **U** sends a data packet to a host in the set **H***

Let us imagine that the host **u1** in the set **U** intends to send a data packet, **D**, to the host **h1** in set **H** on the latter's TCP port 21. The hosts in the set **H** are protected by the visible proxy **V**. Fig 8 helps us illustrate this case.

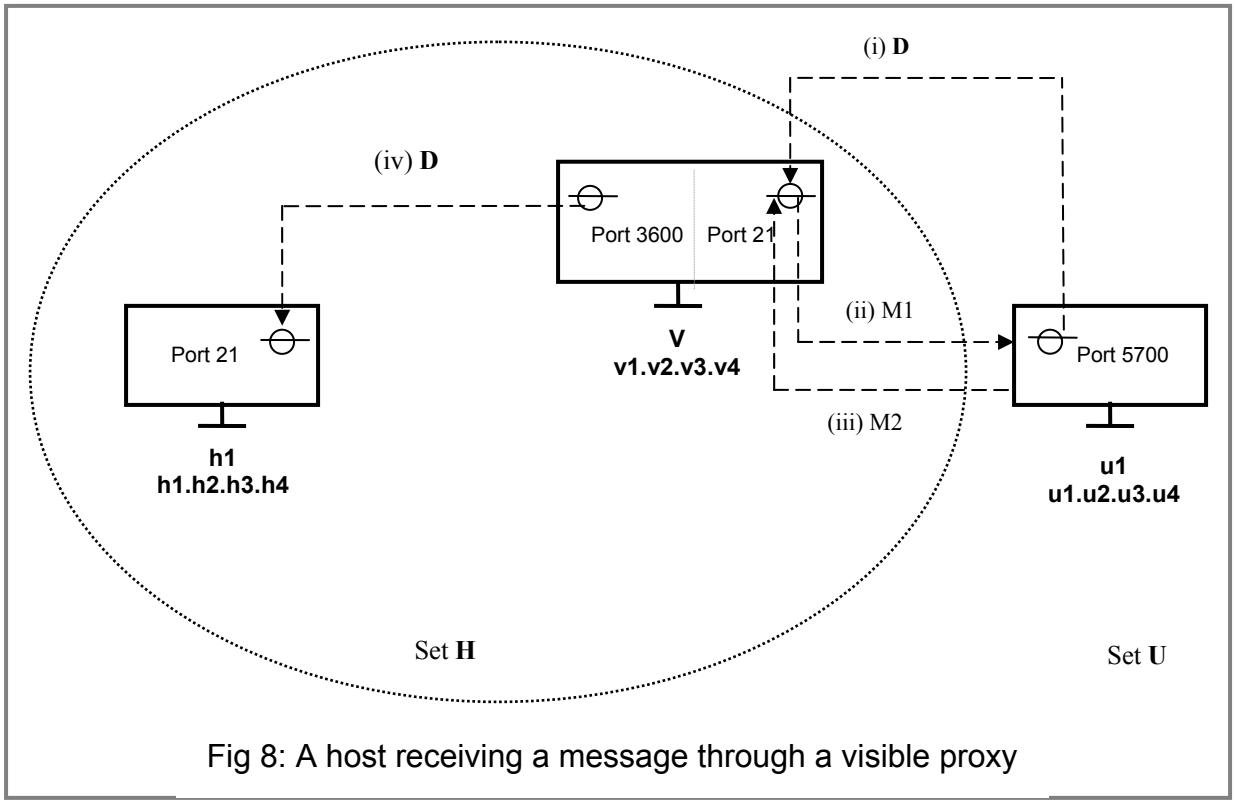


Fig 8: A host receiving a message through a visible proxy

The following describe the steps followed in sending this data packet.

1. In order for the host **u1** to send the data packet **D**, it must first obtain the IP address of the host **h1** in the set **H**. The host **u1** generates DNS queries, but instead of the IP address of the host **h1**, it gets back the IP address of the visible proxy **V** which is *v1.v2.v3.v4.*

2. The host **u1** chooses a random TCP port (port 5700 in this example). It sends the data packet **D** from that port to the visible proxy **V** at TCP port 21. This data packet has the following parameters:

   *Source IP address : = u1.u2.u3.u4*
   *Source port number : = 5700*
   *Destination IP address : = v1.v2.v3.v4*
   *Destination port number : = 21*

3. The visible proxy **V** sends a message back to the host **u1** asking for the name of the destination host in the set **H** to whom **u1** would like to send the data packet **D**. We denote this message as M1.

4. The host **u1** replies back to the visible proxy with the name of the host in the set **H**, which is **h1**. We denote this message as M2.

5. The visible proxy **V** chooses a random TCP port number (port 3600 in this example). It sends the data packet **D** from that port to the host **h1** at TCP port 21. While forwarding the data packet **D**, the visible proxy **V** examines the Application header and the Application Data of **D**. This data packet has the following parameters:

*Source IP address* : = *v1.v2.v3.v4*
*Source port number* : = *3600*
*Destination IP address* : = *h1.h2.h3.h4*
*Destination port number* : = *21*

Therefore, the data packet **D** finally reaches from the host **u1** in the set **U** to the host **h1** in the set **H**.

*Visible Proxies explained with Abstract Protocol (AP) Notation*

It is instructive to understand the visible proxy in terms of the Abstract Protocol (AP) Notation [1]. We assume that two processes, **P** and **Q**, intend to communicate with each other. We also assume that the process **R** is a visible proxy that protects the process **Q**. Figure 9 illustrates our design.

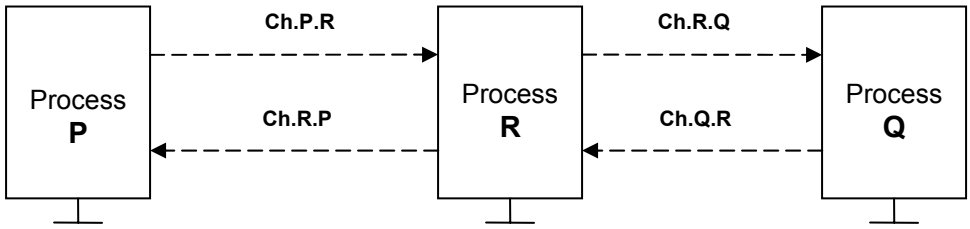


Fig 9: A model of the visible proxy using the AP Notation

We denote the channel from the process **P** to the process **R** as **Ch.P.R**, while the channel from the process **R** to the process **P** as **Ch.R.P**. Similarly, the channel from the process **R**

to the process **Q** is denoted as **Ch.R.Q**, while the channel from the process **Q** to the process **R** is denoted as **Ch.Q.R**.

In designing these processes, we can consider the following properties:

1. The process **R** acts as a visible proxy between the two processes **P** and **Q**. Therefore, every message that is exchanged between the processes **P** and **Q** should pass through the process **R**.

2. As evident from our earlier description of the visible proxies, the process **P** generates a DNS query to obtain the IP address of the destination process **Q**, but instead receives the IP address of the proxy process **R**. This is equivalent to the idea that during compilation of process **P**, every instance of the process **Q** is replaced by an instance of the process **R**.

   During compilation of the process **P**, every instance of **"send** message **to** Q**"** is replaced by **"send** message **to** R**"**. Similarly, every instance of **"recv** message **from** Q**"** is replaced by **"recv** message **from** R**"**.

3. Appropriately, while executing **"send** message **to** R**"**, the process **P** puts the message intended for the process **Q** in the channel **Ch.P.R**. Similarly, while executing **"recv** message **from** R**"**, the process **P** receives every message from the process **Q** from the channel **Ch.R.P**.

4. The process **Q** need not change due to the presence of a visible proxy, i.e. the process **R**. When the process **Q** intends to send a message to the process **R**, it puts that message in the channel **Ch.Q.R**. Similarly, the process **Q** receives every message from the process **R** from the channel **Ch.R.Q**.

### *3.2.2 Transparent Proxies*

A transparent proxy is a system of computer hardware and software that intercepts a data packet, which is being sent by a host in one set, and forwards it to a host in another set. While forwarding it, the transparent proxy examines the Application header and the Application Data of the data packet.

As the transparent proxy intercepts every data packet that traverses between the set **H** and the set **U**, it needs to be deployed at every point of contact between the two sets. Therefore, the transparent proxies function as gateway routers with the ability to examine a data packet at the Application layer of the Internet protocol hierarchy. Every transparent proxy has two network interfaces, each of which has a unique IP address.

We now discuss the two cases in which the transparent proxy examines a data packet.

*(a) A host in the set **H** sends a data packet to a host in the set **U***

Let us imagine that the host **h1** in the set **H** intends to send a data packet, **D**, to the host **u1** in the set **U** on the latter's TCP port 21. The hosts in the set **H** are protected by the transparent proxy **V**. Also, the transparent proxy **V** has two network interfaces, each having a unique IP address. The IP address of the interface **1** is *r1.r2.r3.r4*. The IP address of the interface **2** is *s1.s2.s3.s4*. Fig 10 helps us illustrate this case.

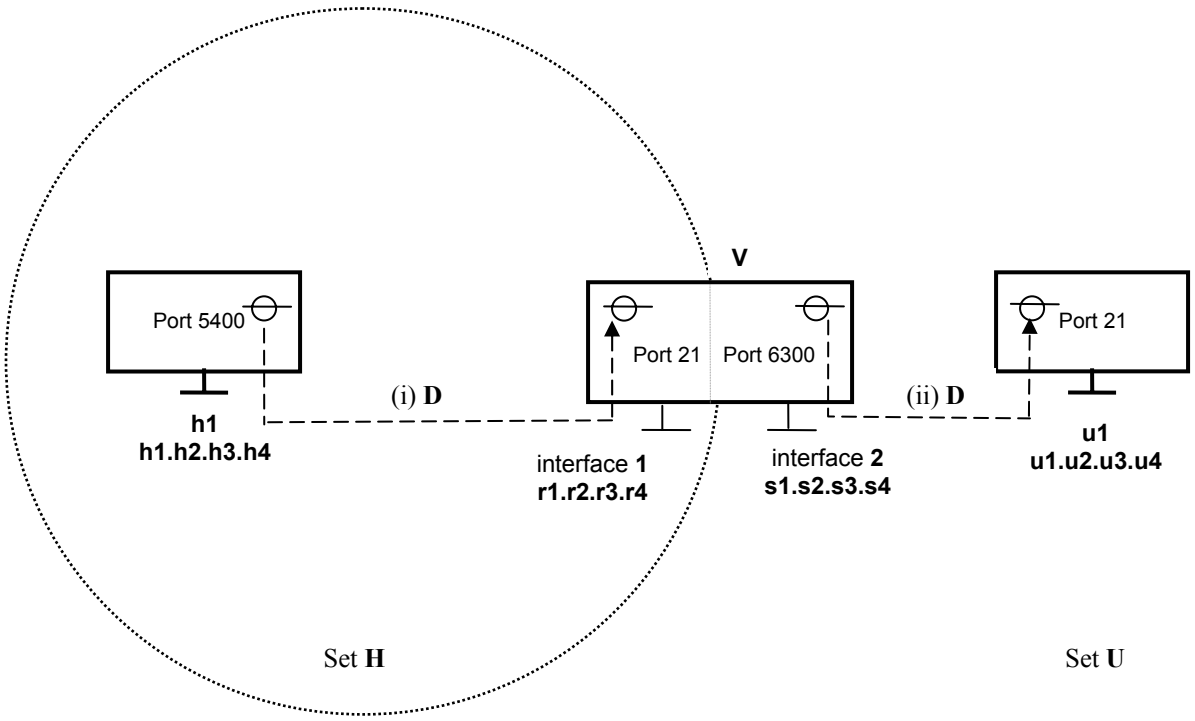


Fig 10: A host sending a message through a transparent proxy

The following describe the steps in sending this data packet:

1. In order for the host **h1** to send the data packet **D**, it needs to obtain the IP address of the host **u1** in the set **U**. The host **h1** generates DNS queries for this purpose and gets back the IP address of the host **u1**, which is *u1.u2.u3.u4.*

2. The host **h1** chooses a random TCP port (port number 5400 in this example). It then sends the data packet **D** from this port to the host **u1** at its TCP port 21. This data packet has the following parameters:

   *Source IP address : = h1.h2.h3.h4*
   *Source port number : = 5400*
   *Destination IP address : = u1.u2.u3.u4*
   *Destination port number : = 21*

3. As the transparent proxy **V** is deployed at every point of contact between the set **H** and the set **U**, the data packet **D** is routed through it. In this example, the data packet **D** reaches the transparent proxy **V** on the interface **1**. The transparent proxy **V** intercepts this data packet, i.e. instead of forwarding or dropping the data packet **D** (which are the only two options according to the RFC specifications), it accepts the data packet on its TCP port 21.

4. The transparent proxy **V** chooses a random TCP port (port number 6300 in this example). It sends the data packet **D** from this TCP port and from its interface **2** to the host **u1** at TCP port 21. While forwarding the data packet **D**, the transparent proxy **V** examines the Application header and the Application Data of **D**. This data packet has the following parameters:

   *Source IP address* : = *s1.s2.s3.s4*
   *Source port number* : = *6300*
   *Destination IP address* : = *u1.u2.u3.u4*
   *Destination port number* : = *21*

Therefore, the data packet **D** finally reaches from the host **h1** in the set **H** to the host **u1** in the set **U**.

*(b) A host in the set **U** sends a data packet to a host in the set **H***

Let us imagine that the host **u1** in the set **U** intends to send a data packet, **D**, to the host **h1** in the set **H** on the latter's TCP port 21. The hosts in the set **H** are protected by the transparent proxy **V**. Also, the transparent proxy **V** has two network interfaces, each having a unique IP address. The IP address of the interface **1** is *r1.r2.r3.r4*. The IP address of the interface **2** is *s1.s2.s3.s4*. Fig 11 helps us illustrate this case.

The following describe the steps in sending this data packet:

1. In order for the host **u1** to send the data packet **D**, it needs to obtain the IP address of the host **h1** in the set **H** The host **u1** generates DNS queries for this purpose and gets back the IP address of the host **h1**, which is *h1.h2.h3.h4.*

2. The host **u1** chooses a random TCP port (port number 6100 in this example). It then sends the data packet **D** from this port to the host **h1** at its TCP port 21. This data packet has the following parameters:

   *Source IP address* : = *u1.u2.u3.u4*
   *Source port number* : = *6100*
   *Destination IP address* : = *h1.h2.h3.h4*
   *Destination port number* : = *21*

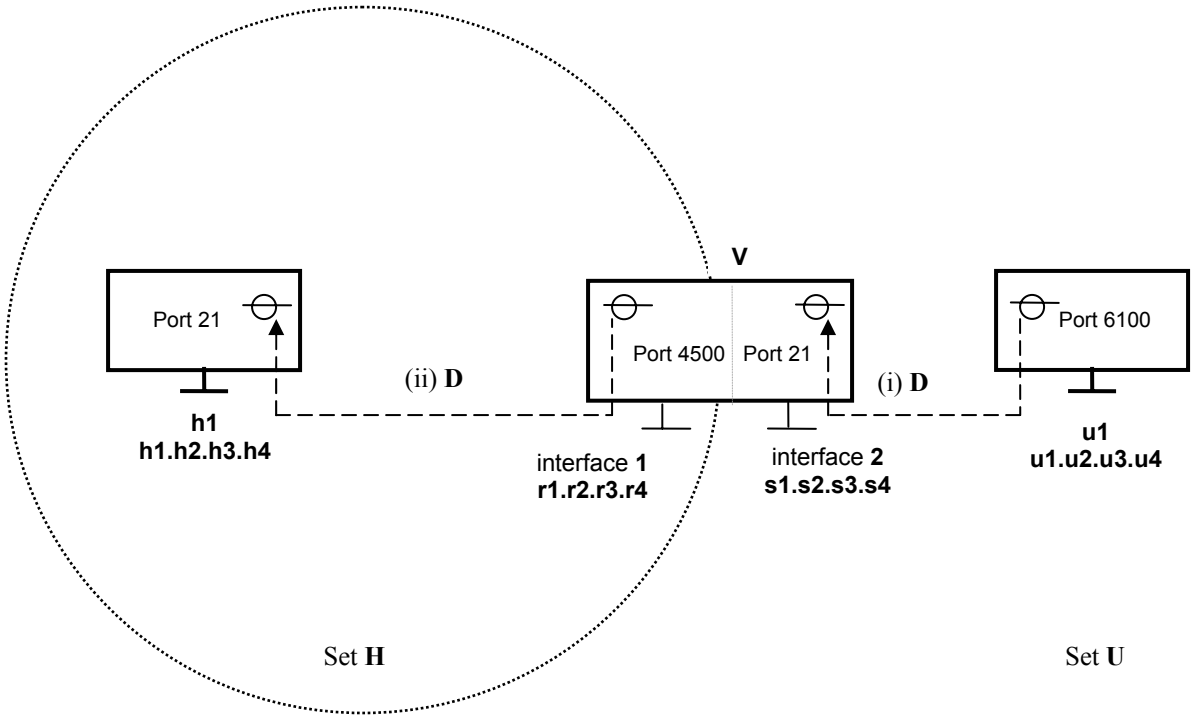


Fig 11: A host receiving a message through a transparent proxy

3. As the transparent proxy **V** is deployed at every point of contact between the set **H** and the set **U**, the data packet **D** is routed through it. In this example, the data packet **D** reaches the transparent proxy **V** on the interface **2**. The transparent proxy **V** intercepts this data packet, i.e. instead of forwarding or dropping the data packet **D** (which are the only two options according to the RFC specifications), it accepts the data packet on its TCP port 21.

4. The transparent proxy **V** chooses a random TCP port (port number 4500 in this example). It sends the data packet **D** from this TCP port and from its interface **1** to the host **h1** at TCP port 21. While forwarding the data packet **D**, the transparent proxy **V** examines the Application header and the Application Data of **D**. This data packet has the following parameters:

   *Source IP address* : = *r1.r2.r3.r4*
   *Source port number* : = *4500*
   *Destination IP address* : = *h1.h2.h3.h4*
   *Destination port number* : = *21*

Therefore, the data packet **D** finally reaches from the host **u1** in the set **U** to the host **h1** in the set **H**.

*Transparent Proxies explained with Abstract Protocol (AP) Notation*

It is instructive to understand the transparent proxy in terms of the Abstract Protocol (AP) Notation [1]. We assume that two processes, **P** and **Q**, intend to communicate with each other. We also assume that the process **R** is a transparent proxy that protects the process **Q**. Further, the process **R** intercepts every message coming from or going to the process **Q**. Figure 10 illustrates our design.

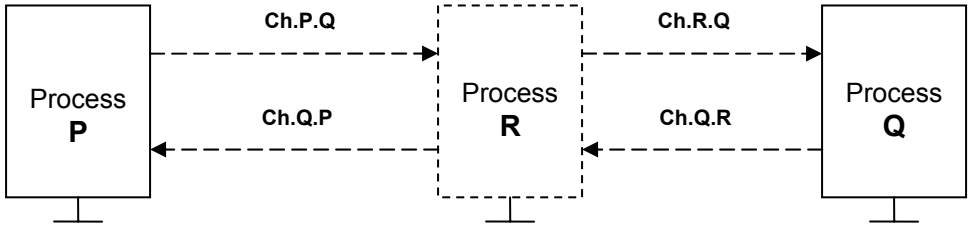


Fig 12: A model of the transparent proxy using the AP Notation

We denote the channel from the process **P** to the process **Q** as **Ch.P.Q**, while the channel from the process **Q** to the process **P** as **Ch.Q.P**. In the above figure, the process **R** intercepts every message traversing on the channels **Ch.P.Q** and **Ch.Q.P**.

The channel from the process **R** to the process **Q** is denoted as **Ch.R.Q**, while the channel from the process **Q** to the process **R** is denoted as **Ch.Q.R**.

In designing these processes, we can consider the following properties:

1. The process **R** acts as a transparent proxy between the two processes **P** and **Q**. Therefore, every message that is exchanged between the processes **P** and **Q** should pass through the process **R**.

2. The process **P** generates a DNS query to obtain the IP address of the destination process **Q**, and receives the actual IP address of the desired process.

3. The process **P** is unaware of the existence of the process **R** between itself and the process **Q**. Appropriately when the process **P** desires to send a message to the process **Q**, it puts the message in the channel **Ch.P.Q**. Similarly, the process **P** receives every message from the process **Q** from the channel **Ch.Q.P**.

However, the messages in the channel **Ch.P.Q** and **Ch.Q.P** are intercepted by the process **R**, which in turn forwards the message to the desired destination.

4. On the other hand, the process **Q** communicates directly with the process **R**. When the process **Q** intends to send a message to the process **R**, it puts that message in the channel **Ch.Q.R**. Similarly, the process **Q** receives every message from the process **R** from the channel **Ch.R.Q**.

*Differences between the Visible and Transparent proxies:*

| Visible Proxies | Transparent Proxies |
|---|---|
| 1. Visible Proxies do not need to function as routers. | 1. Transparent Proxies need to function as routers. |
| 2. Visible proxies can be located either inside or outside the computer network. Therefore, they do not provide for a single point of failure. | 2. Transparent proxies need to be located at the gateway of the computer network. Therefore, they provide for a single point of failure. |
| 3. IP address resolution of the hosts inside the computer network, which is protected by a visible proxy, is relative; i.e. when a host in the outside Internet attempts to resolve the IP of a host inside the computer network, it receives the IP address of the visible proxy instead. | 3. IP address resolution of the hosts inside the computer network, which is protected by a transparent proxy, is simple; i.e. when any host attempts to resolve the IP of a host inside the computer network; it receives the actual IP address of the desired host. |
| 4. It is possible for a user in the outside Internet to discover the actual IP address of the host in the computer network and connect to the same directly i.e. visible proxies can be bypassed. | 4. As the transparent proxies are deployed at the gateway of the computer network, they monitor every data packet going in or coming out of the computer network i.e. transparent proxies can not be bypassed. |
| 5. Visible proxies must create a new data packet for every data packet that it receives on behalf of a host in the computer network. | 5. Transparent proxies may forward the same data packet that it receives on behalf of a host in the computer network i.e. it need not create a new data packet. |

## 3.3 Applications of Proxies

In this section, we discuss various applications of the proxy. These applications can be classified in three broad categories, which are as follows:

**1. Security**

**2. Performance**

**3. Administration**

### 1. Security

A proxy is primarily used to provide security to the computer network of an organization. We refer to such a proxy as the *security proxy*.

The three important advantages of the security proxy are as follows:

*a)* *Application layer filtering*: The security proxy examines and filters the data packets at the Application layer of the Internet protocol hierarchy. It rejects every data packet with malicious Application header or Application Data. This filtering prevents several classes of attacks against the hosts in the computer network of the organization.

*b)* *DOS attack prevention*: The security proxy is often used to prevent *Denial of Service (DOS)* attacks against the hosts in the computer network of an organization. In order to prevent these *DOS* attacks, the security proxy implements the following policies.

(i) The security proxy disallows any user outside the computer network of the organization to directly connect to a host inside the network. Instead, these users are required to connect to the security proxy, which in turn connects to the desired host inside the computer network.

(ii) The security proxy enforces a maximum on the number of connections that can be simultaneously opened on a particular host inside the computer network.

When implemented together, these two policies allow the security proxy to prevent *DOS* attacks launched against the hosts in the computer network.

c) *Authentication*: The security proxy can authenticate users on behalf of the hosts in the computer network of an organization. This feature of the security proxy helps to significantly reduce the load on individual hosts in the computer network.

We next present a model of the security proxy that authenticates users on behalf of a Telnet server in the computer network of an organization. Figure 10 illustrates our model.

In this figure, the computer network **orb.com** consists of one Telnet server with the IP address *S*. This Telnet server is protected by a security proxy with the IP address *V*.
Also in the figure, a client with the IP address *C* exists outside the network **orb.com**. This client desires to initiate a Telnet session with the Telnet server. The following steps outline the procedure for the initiation of this Telnet session:

1. The client chooses a TCP port at random (port 2900 in this example) and sends a connection request to the security proxy at the latter's TCP port 23 (port 23 is the TCP port associated with the Telnet protocol). The parameters of this connection request message are as follows:

   *Source IP address* : = *C*
   *Source port number* : = *2900*
   *Destination IP address* : = *V*
   *Destination port number* : = *23*

2. The security proxy requests the client for a username and a password. We assume that both the username and the password are required by the security proxy in order to authenticate the client. The parameters of this request message are as follows:

   *Source IP address* : = *V*
   *Source port number* : = *23*
   *Destination IP address* : = *C*
   *Destination port number* : = *2900*

3. The client sends the username and the password to the security proxy. On receiving this information, the security proxy authenticates the client. In particular, the security proxy verifies the legitimacy of the client to connect to the Telnet server in the computer network **orb.com**.

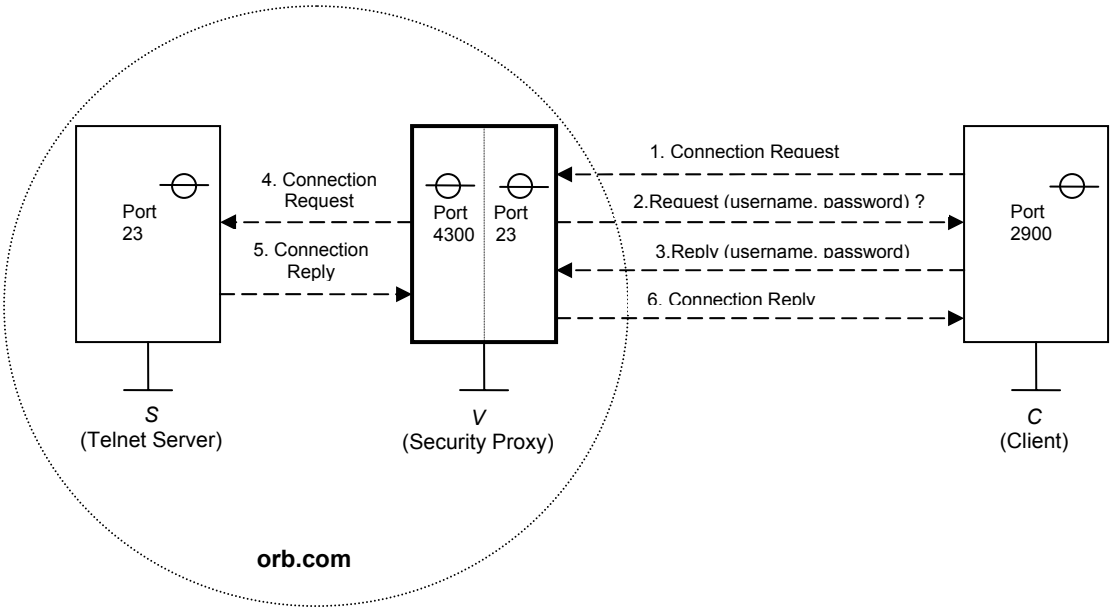


Fig 13: A model of the security proxy used for authentication

4. On successful authentication of the client, the security proxy chooses a TCP port at random (port number 4300 in this example) and forwards the connection request to the Telnet server. The parameters of the connection request message are as follows:

   *Source IP address* : = *V*
   *Source port number* : = *4300*
   *Destination IP address* : = *S*
   *Destination port number* : = *23*

   While forwarding the connection request to the Telnet server, the security proxy saves certain information about the current connection. This enables the security proxy to forward all subsequent replies coming from the Telnet server back to the client. We represent the information, which is saved by the security proxy, as follows.

   In an incoming data packet,
   **if** (Source IP address, Source port number) = (*S*, 23)
   (Destination IP address, Destination port number) = (*V*, 4300)

   **(Requestor IP Address, Requestor Port number)** = (*C*, 2900)
   **(Replier IP Address, Replier Port number)** = (*V*, 23)

This information signifies that if the security proxy receives a data packet on its TCP port 4300 from a host with the IP address *S* and TCP port number 23, then it should forward the data packet to the host with the IP address *C* on the TCP port number 2900. In this data packet, the source IP address should be set to *V* and the source port number to 23.

5. On receiving the connection request, the Telnet server sends back a connection reply to the security proxy. The parameters of this connection reply message are as follows:

   *Source IP address* : = *S*
   *Source port number* : = *23*
   *Destination IP address* : = *V*
   *Destination port number* : = *4300*

6. The security proxy finally forwards the connection reply to the client. The parameters of this connection reply message are as follows:

   *Source IP address* : = *V*
   *Source port number* : = *23*
   *Destination IP address* : = *C*
   *Destination port number* : = *2900*

The security proxy, by authenticating users on behalf of the Telnet server, significantly reduces the load on the latter. Also, by enforcing a maximum on the number of simultaneous connections that are acknowledged, the security proxy may prevent *Denial of Service* attacks against the Telnet server.

**2. Performance**

A proxy is often used by an organization to improve the performance of its computer network. We refer to such a proxy as the *performance proxy*.

The performance proxies are most commonly used to distribute the HTTP request messages amongst a group of HTTP servers. The decision to forward an HTTP request message to a particular HTTP server is usually based on the following factors:

*a)* *Load*: The performance proxies often distribute the HTTP request messages equally amongst a group of HTTP servers. This helps to reduce the load on any one server, thereby improving the overall performance of the network. This feature of the performance proxy is generally referred to as *load-balancing*.

*b)* *Location*: The performance proxy may also decide to forward an HTTP request message to a particular HTTP server based upon the location of the client which

originally sent the request message. This feature often allows customized HTTP content delivered to clients that reside in different locations around the world.

We next present a model of the performance proxy that performs load-balancing for a group of HTTP servers. Figure 11 illustrates our example. In this figure, there are three HTTP servers with the IP addresses *S1, S2* and *S3* respectively. These HTTP servers are identical in every respect. We assume that amongst other similarities, each of these HTTP servers store a copy of the object *OB*.

A performance proxy, which has the IP address *V*, performs load-balancing for these three HTTP servers. Also in the figure, the client has the IP address *C*. In our example, the client wishes to retrieve a copy of the object *OB* from one of the three HTTP servers. The following steps outline the procedure followed in retrieving the copy of this object:

1. In order for the client to retrieve the object *OB*, it needs to obtain the IP address of one of the three HTTP servers. Accordingly the client generates a DNS query and receives the IP address of the performance proxy.

2. The client chooses a TCP port at random (port 5100 in this example). It then sends a request for the object *OB* from this port to the performance proxy at its TCP port 80 (port 80 is the TCP port associated with HTTP). The parameters of this request message are as follows:

   *Source IP address* : = *C*
   *Source port number* : = *5100*
   *Destination IP address* : = *V*
   *Destination port number* : = *80*

3. The performance proxy chooses a TCP port at random (port 4800 in this example) and forwards the request to one of the three HTTP servers that store a copy of the object *OB*. In this example, the performance proxy forwards the request to the HTTP server with the IP address *S2* at the latter's TCP port 80. The parameters of this request message are as follows:

   *Source IP address* : = *V*
   *Source port number* : = *4800*
   *Destination IP address* : = *S2*
   *Destination port number* : = *80*

   While forwarding the request to the HTTP server, the performance proxy saves certain information about the current connection. This enables the performance proxy to forward all subsequent replies coming from the HTTP server back to the client. We represent the information, which is saved by the performance proxy, as follows.

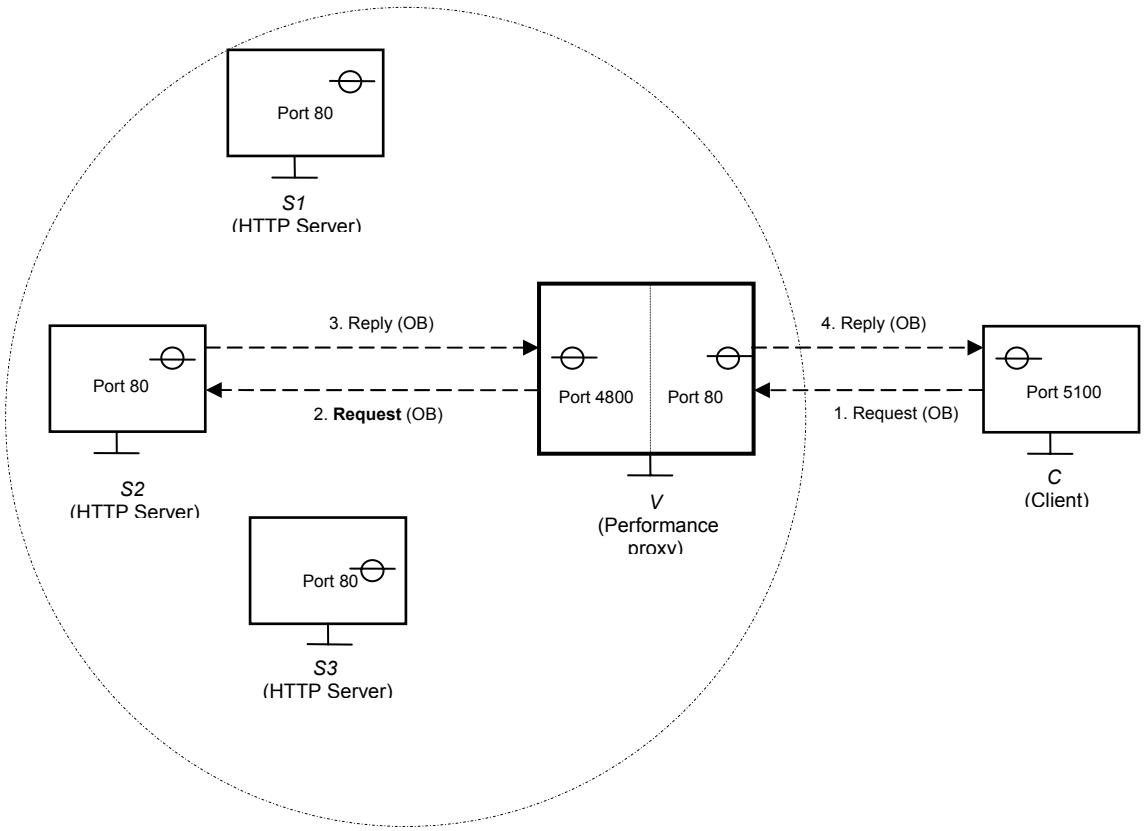


Fig 14: A model of the performance proxy used by a group of HTTP

In an incoming data packet,

**if** (Source IP address, Source port number) = (*S2*, 80)
(Destination IP address, Destination port number) = (*V*, 4800)

**(Requestor IP Address, Requestor Port number)** = (*C*, 5100)
**(Replier IP Address, Replier Port number)** = (*V*, 80)

This information signifies that if the performance proxy receives a data packet on its TCP port 4800 from a host with the IP address *S2* and TCP port number 80, then it should forward the data packet to the host with the IP address *C* on the TCP port number 5100. In this data packet, the source IP address should be set to *V* and the source port number to 80.

4. On receiving the request from the performance proxy, the HTTP server replies back with a copy of the object *OB*. The parameters of this reply message are as follows:

*Source IP address* : = *S2*
*Source port number* : = *80*

*Destination IP address* : = *V*
*Destination port number* : = *4800*

5. The performance proxy finally forwards the object *OB* to the client. The parameters of this reply message are as follows:

   *Source IP address* : = *V*
   *Source port number* : = *80*
   *Destination IP address* : = *C*
   *Destination port number* : = *5100*

The performance proxy attempts to balance the number of requests equally amongst the three HTTP servers. This load-balancing improves the overall performance of the network. It also helps in mitigating *Denial of Service* attacks that may be directed towards a particular HTTP server.

## 3. Administration

A proxy is often used by an organization for the administration of its computer network. We refer to such a proxy as the *administrative proxy*.

The administrative proxies are primarily used for the following purposes.

*a)* *Access Control*: The administrative proxy allows an organization to efficiently share a resource (e.g. a paid web service) amongst the users in its computer network. Such a resource is often configured to only accept requests from the administrative proxy of the organization. Subsequently, every user that wishes to acquire the resource sends its request to the administrative proxy. This enables the administrative proxy to verify the legitimacy of every user that intends to acquire the resource.

*b)* *IP address hiding*: The administrative proxy is often used by an organization to hide the IP addresses of all the hosts in its computer network. Every host in its computer network sends the data packets to the administrative proxy, which in turn forwards it to the desired host that is outside the network. Similarly, the administrative proxy receives the data packets on behalf of every host in the network, and later forwards them to the proper receipient. This feature of the administrative proxy is commonly referred to as *Network Address Translation (NAT)*.

We next present a model of an administrative proxy that performs *IP address hiding*. Figure 12 illustrates this example. In this figure, there are two computer networks, **orb.com** and **rand.com**. Each of these computer networks has an administrative proxy. The administrative proxy for **orb.com** has two network interfaces which have the IP

addresses *C1* and *E* respectively. The administrative proxy for **rand.com** has two network interfaces which have the IP addresses *C2* and *F* respectively.

The computer network **orb.com** has one Telnet server with the IP address *A1*, one HTTP server with the IP address *B1*, and one client with the IP address *h1*. The computer network **rand.com** has one Telnet server with the IP address *A2* and one HTTP server with the IP address *B2*.

As mentioned earlier, the administrative proxy allows an organization to hide the IP addresses of the hosts in its computer network. Both the computer networks in our example hide the IP addresses of their hosts by using a Class-A network number. We assume that both the computer networks use the Class-A network number 10. We also assume the following.

$$A1 = A2$$
$$B1 = B2$$
$$C1 = C2$$

We next outline the steps involved when the client in the computer network **org.com** intends to send a request *R* to the Telnet server in the computer network **rand.com**.

1. The client in the computer network **orb.com** chooses a TCP port at random (port number 5300 in this example). It then sends the request *R* to the administrative proxy of its computer network at the IP address *C1* and port number 23 (port 23 is the TCP port associated with the Telnet protocol). The parameters of this request message are:

   *Source IP address* := *h1*
   *Source port number* := *5300*
   *Destination IP address* := *C1*
   *Destination port number* := *23*

2. The administrative proxy of the computer network **orb.com** chooses a new TCP port at random (port number 2001 in this example). It then forwards the request *R* from this port and the IP address *E* to the administrative proxy of the computer network **rand.com** at the latter's IP address *F* and port number 23. The parameters of this request message are:

   *Source IP address* := *E*
   *Source port number* := *2001*
   *Destination IP address* := *F*
   *Destination port number* := *23*

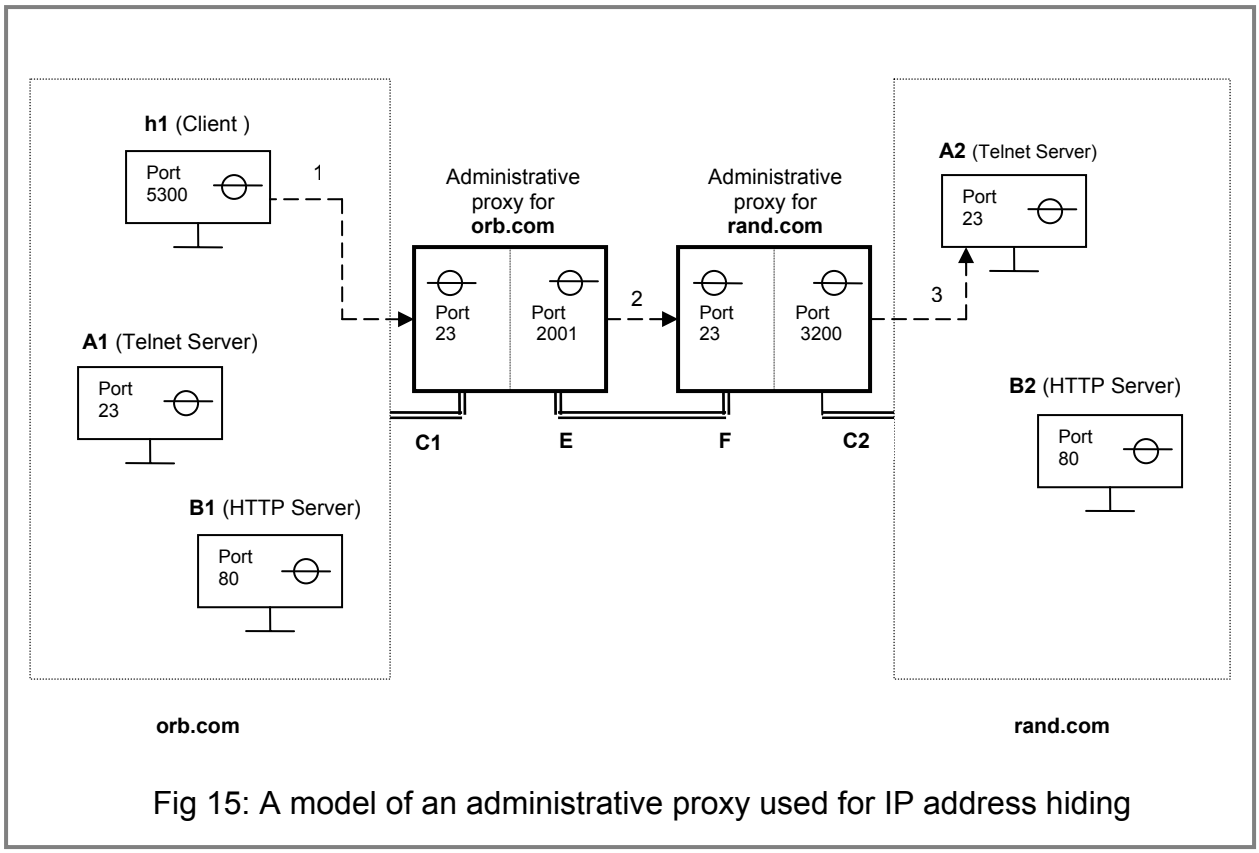


Fig 15: A model of an administrative proxy used for IP address hiding

While forwarding the request *R*, the administrative proxy of the computer network **orb.com** saves certain details about the current connection. This enables it to forward all subsequent replies from the administrative proxy of the computer network **rand.com** back to the client. We represent the information, which is saved by the administrative proxy, as follows.

In an incoming data packet,

**if** (Source IP address, Source port number) = (*F*, 23)
(Destination IP address, Destination port number) = (*E*, 2001)

**(Requestor IP Address, Requestor Port number)** = (*h1*, 5300)
**(Replier IP Address, Replier Port number)** = (*C1*, 23)

This information signifies that if the administrative proxy of the computer network **orb.com** receives a data packet on its interface with the IP address *E* and on the TCP port 2001 from a host with the IP address *F* and TCP port number 23, then it should forward the data packet to the host with the IP address *h1* on the TCP port number 5300. In this data packet, the source IP address should be set to *C1* and the source port number to 23.

3. When the administrative proxy of the computer network **rand.com** receives the request *R*, it chooses a new TCP port at random (port number 3200 in this

example). It then forwards the request *R* from its IP address *F* to the Telnet server in the computer network **rand.com** at the latter's IP address *A2* and port number 23. The parameters of this request message are:

*Source IP address* := *F*
*Source port number* := *3200*
*Destination IP address* := *A2*
*Destination port number* := *23*

While forwarding the request, the administrative proxy of the computer network **rand.com** saves certain details about the current connection. This enables it to forward all subsequent replies from the Telnet server back to the administrative proxy of the computer network **orb.com**. We represent the information, which is saved by the performance proxy, as follows.

In an incoming data packet,
**if** (Source IP address, Source port number) = (*A2*, 23)
(Destination IP address, Destination port number) = (*C2*, 3200)

**(Requestor IP Address, Requestor Port number)** = (*E*, 2100)
**(Replier IP Address, Replier Port number)** = (*F*, 23)

This information signifies that if the administrative proxy of the computer network **rand.com** receives a data packet on its interface with the IP address *C2* and on the TCP port 3200 from a host with the IP address *A2* and TCP port number 23, then it should forward the data packet to the host with the IP address *E* on the TCP port number 2100. In this data packet, the source IP address should be set to *F* and the source port number to 23.

When the Telnet server in the computer network **rand.com** receives the request, it replies back to the administrative proxy of its network, which in turn forwards the reply to the administrative proxy of the computer network **orb.com**. The latter finally forwards the reply back to the client in computer network **orb.com**. Therefore, even through the two Telnet servers in **orb.com** and **rand.com** having identical IP addresses, the client can successfully differentiate between them using the administrative proxies.

## 3.4 A model of the FTP proxy

We next present the model of a proxy which provides security to an FTP server in the computer network of an organization. We denote this proxy as the *FTP proxy*.

Figure 13 illustrates this model. In this figure, the computer network **orb.com** contains an FTP server **s.orb.com** with the IP address *s1.s2.s3.s4*. This computer network also contains an FTP proxy **v.orb.com** that protects the FTP server **s.orb.com**. The IP address of this FTP proxy is *v1.v2.v3.v4*.

Our model also consists of a client that intends to establish an FTP session with the FTP server **s.orb.com**. This client exists outside the computer network **orb.com** and has the IP address *c1.c2.c3.c4*. We outline the steps involved in the establishment of the FTP session between this client and the FTP server.

1. In order for the client to send a connection request to the FTP server, it needs to obtain the IP address of the latter. The client generates a DNS query for this purpose. This DNS query propagates through the root of the DNS tree and ultimately reaches the DNS server of the computer network **orb.com**.

   We denote this DNS server as **DNS.orb.com**. **DNS.orb.com** replies to the client with the IP address of the FTP proxy **v.orb.com**. Therefore, instead of getting the IP address of the FTP server (i.e. *s1.s2.s3.s4*), the client receives the IP address of the FTP proxy (i.e. *v1.v2.v3.v4*).

2. The client chooses a TCP port at random (port 5100 in this example). It then sends a connection request from this port to the FTP proxy **v.orb.com** at its TCP port 21 (port 21 is the TCP port associated with the FTP). The parameters of this connection request message are:

   *Source IP address* : = *c1.c2.c3.c4*
   *Source port number* : = *5100*
   *Destination IP address* : = *v1.v2.v3.v4*
   *Destination port number* : = *21*

3. The FTP proxy may choose to authenticate every client on behalf of the FTP server **s.orb.com**. This authentication process is similar to the authentication performed by the security proxy in section 3.3 (Figure 10 illustrates this case).

   The FTP proxy chooses a TCP port at random (port 4800 in this example). It then forwards the connection request to the FTP server **s.orb.com** at its TCP port 21. The parameters of this connection request message are:

   *Source IP address* : = *v1.v2.v3.v4*
   *Source port number* : = *4800*
   *Destination IP address* : = *s1.s2.s3.s4*
   *Destination port number* : = *21*

   While forwarding the connection request to the FTP server, the FTP proxy saves certain information about the current connection. This enables the FTP proxy to forward all subsequent replies coming from the FTP server **s.orb.com** back to the client. We represent the information, which is saved by the FTP proxy **v.orb.com**, as follows.

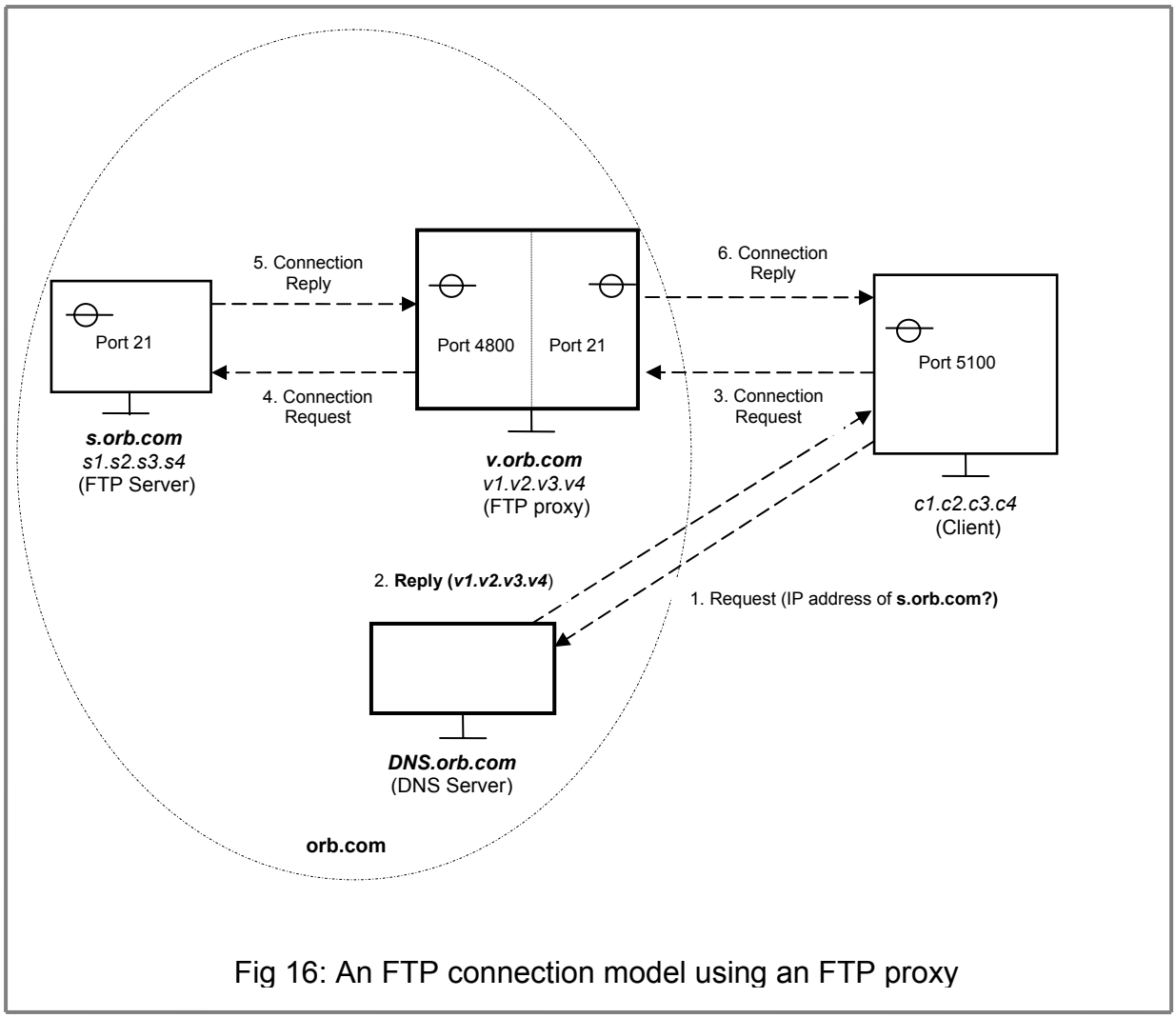


Fig 16: An FTP connection model using an FTP proxy

In an incoming data packet,

**if** (Source IP address, Source port number) = (*s1.s2.s3.s4*, 21)
(Destination IP address, Destination port number) = (*v1.v2.v3.v4*, 4800)

**(Requestor IP Address, Requestor Port number)** = (*c1.c2.c3.c4*, 5100)
**(Replier IP Address, Replier Port number)** = (*v1.v2.v3.v4*, 21)

This information signifies that if the FTP proxy receives a data packet on its TCP port 4800 from a host with the IP address *s1.s2.s3.s4* and TCP port number 21, then it should forward the data packet to the host with the IP address *c1.c2.c3.c4* on the TCP port number 5100. In this data packet, the source IP address should be set to *v1.v2.v3.v4* and the source port number to 21.

4. On receiving the connection request from the FTP proxy, the FTP server **s.orb.com** sends back a connection reply. The parameters of this connection reply message are.

*Source IP address : = s1.s2.s3.s4*
*Source port number : = 21*
*Destination IP address : = v1.v2.v3.v4*
*Destination port number : = 4800*

5. The FTP proxy **v.orb.com** finally forwards the connection reply to the client. The parameters of this connection reply message are.

*Source IP address : = v1.v2.v3.v4*
*Source port number : = 21*
*Destination IP address : = c1.c2.c3.c4*
*Destination port number : = 5100*

After the successful establishment of the FTP session, the client sends FTP request messages to the FTP proxy **v.orb.com** that are forwarded to FTP server **s.orb.com**. Similarly, the FTP server **s.orb.com** sends replies to the FTP proxy **v.orb.com** that are forwarded to the client.

Each of these FTP request and reply messages are examined by the FTP proxy. In particular, the FTP proxy rejects every data packet that contains malicious Application header or Application Data. The FTP proxy can therefore, provide security to the FTP server in **orb.com** from network attacks and intrusion attempts.

# 4. CONCLUSION

In this paper, we have presented a novel classification of the firewalls and the proxies. The firewalls have been classified in five categories, which are *the Static Packet Filter, the Packet Filter, the Connection Filter, the Application Filter* and *the Application Proxy*. Also, we have classified the proxies in two categories, namely *the Visible proxies* and *the Transparent proxies*.

Future research in the area of firewalls and proxies will progress towards designing sophisticated means for detecting and preventing network attacks and intrusion attempts. As such, our classification can be easily extended to encompass newly designed hardware and software components.

***Acknowledgement***

I owe the successful completion of this honors thesis to Dr. Mohamed Gouda. His perseverance and insight guided me in the right direction through out the course of my research. Additionally, I would like to thank Xiang-Yang Alex Liu and Chin-Tser Huang for their valuable suggestions on various topics of this thesis.

***References***

[1] Henry, Paul. "An Examination of Firewall Architectures".

[2] Greenwald, Singhal, Stone, Cheriton. "Designing an Academic Firewall: Policy, Practice, and Experience with SURF".

[3] Frantzen, Kerschbaum, Schultz, Fahmy. "A Framework for Understanding Vulnerabilities in Firewalls Using a Dataflow Model of Firewall Internals".

[4] Gouda, M.G. Elements of Network Protocol Design. John Wiley & Sons.

[5] Chatel, M. "Classical versus Transparent IP Proxies". RFC 1919