15-441 Project 3, Fall 2001

Stateful Functionality in IP Layer

Out: Thursday, November 1, 2001

Due: Tuesday, December 4, 2001

1. Introduction

In Project 2 we asked you to implement the IP layer of the network stack. When implementing the IP layer, you also added support for network address translation (NAT) and stateless firewall. However, the simple firewall you implemented is not robust enough, as we will see later. In this project, you will implement a more robust firewall and more general NAT. More specifically, your task in this project is as follows:

- You will implement a *stateful firewall* that utilizes the states of existing connections derived from their communication history and transport and application layer information. We will see that this kind of firewall provides a higher level of security. (See section 2 for details).
- You will utilize the state information of the connections to build an improved version of the network address translator (NAT) that will use port address translation (PAT) to simultaneously serve multiple connections using the same globally unique IP address. (See section 3 for details).
- Your implementation of the firewall should support FTP clients inside the firewall to communicate with some external server. For example, a client inside the firewall should be able to use the sequence of commands PORT and RETR through the firewall to a server outside the firewall. (See section 4 for details).

You need to hand in the following:

- Source code of the IP layer, with all the functionalities mentioned before.
- Source code of the programs `setnat` and `setfilter`.
- A `Makefile` that builds all the programs with the command `gmake`. It should build the IP layer and link it to the kernel (same as project 2). It should build the execution files `setnat` and `setfilter` too.
- A brief report (see section 5.6)

You may make the following assumptions:

- There is no packet loss between the client and the firewall and between the firewall and the server.
- There is no fragmentation of packets. The packets are large enough so that, for example, in FTP protocol, the command and parameters will be in the same packet.
- The firewall will support only TCP/UDP/ICMP packets. Other packets will go through the firewall without checking. However, the UDP and ICMP packets will go through the policy table only, but the TCP packets will be consulted against both the tables (details will be in section 2.2).

The rest of the handout is organized as follows. Section 2 gives a tutorial on stateful firewall. The section gives hints on how you can implement such a firewall. Section 3 describes how a dynamic NAT works. Section 4 shows how a stateful firewall can

support a FTP back channel (using PORT command). Section 5 gives logistics for the project, e.g., groups, grading policy, etc. We will have the assumptions stated earlier throughout the handout.

All of the source code and header files referred to in this document can be found under the course directory: `/afs/cs.cmu.edu/academic/class/15411-f01/projects/project3/.` In the rest of the document, we refer to this directory as `$PDIR`. Since the support code is the same as for project 2, most of the contents of this directory are just symbolic links to those in project 2 directory.

2 Stateful Firewall

2.1 Overview

A *Stateful Firewall* (SF) is very similar to the stateless firewall (which examines packets in isolation) you implemented in project 2, but offers a slightly expanded functionality. In addition to checking the static rules about accepting/dropping packets based on their source/destination addresses and ports, SFs maintain some state about each connection passing through them. The SF is supposed to have built-in knowledge about TCP/IP's rules for data flow between the two hosts. The extra advantage is that an SF can detect incorrectly sequenced numbered packets, incorrect IP protocol options or inconsistent packets coming from an attacker.

To see how a stateful firewall provides better security over a stateless one, consider the following scenario. Alice (client) wishes to communicate through a firewall to Bob (a web server) using the HTTP protocol. Bob listens to the standard HTTP port 80 for connection from clients. So, Alice sends HTTP request packets to Bob from one of the so-called "High" port (in the range 1024-65535). Alice is afraid of Evil Charlie who can attack her, so she wants to restrict incoming packets to her machine as carefully as possible.

Figure 1 shows the requests from Alice (IP address:10.0.0.1) and corresponding response from Bob (IP address: 192.168.1.1).

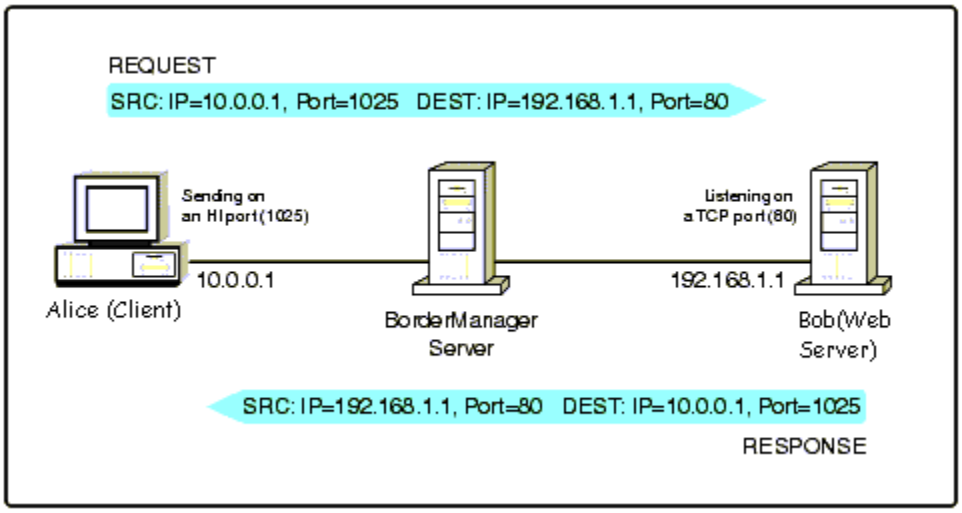

Figure 1: HTTP traffic outbound and resulting response

If Alice uses a stateless firewall (like the one you implemented in project 2), she can use the following rules to restrict access to her machine.

- Allow any outgoing packet to port 80.
- Do not allow any SYN packet in. (*e.g.*, no incoming connection is allowed). Incoming SYN+ACK packet is allowed.
- Allow incoming packets from port 80, and to 1024-65535 (because the browser can bind to any of these ports and so the response from the server can come to that port.)

Alice hopes that since Evil Charlie cannot establish any TCP connection to her machine and that any fraudulent packet from him will be dropped by the firewall. Is this really true? Suppose, Evil Charlie (who does not have any existing connection to Alice) sends a packet with port 80 and some bogus flag (other than SYN) or no flag set. The stateless firewall is unable to catch this fraudulent packet and will allow this packet to get in.

The problem here is that the stateless firewall checks each packet in isolation. So it is not able to determine if an incoming packet is actually the first packet in an externally initiated session or if it is a part of an internal client-initiated session.

A stateful firewall keeps track of the outgoing packets that it allows to pass and allows only corresponding response packets to return. In addition to the static rule checking used by stateless firewalls, it will create a temporary (time-limited) inbound filter (that would allow, in previous example, HTTP packets from 192.168.1.1:80 to 10.0.0.1:1025 to be accepted) for the connection. This temporary filter will only allow packets from the host and port to which the outbound packet was sent. When the firewall receives the packet sent by Evil Charlie, the firewall will check if the packet is part of an existing connection. Since it is not, firewall will drop the packet, thus defying Evil Charlie's malicious effort.

This is also true for applications using other protocols like FTP, Telnet etc. Therefore, it is not sufficient to examine packets in isolation. State information, derived from past communications and other applications, is an essential factor in making the control decision for new communication attempts. Depending upon the communication attempt, both the communication state (derived from past communications) and the application state (derived from other applications) may be critical in the control decision. To ensure the highest level of security, a firewall must be capable of accessing, analyzing and utilizing the following:

- *Transport-derived state:* the state derived from the transport protocol layer of previous communications. In the previous example, Alice's firewall should maintain states of all the connection to detect that Evil Charlie's packet is not part of any of the existing connections.
- *Application-derived State:* the state information derived from applications For example, the outgoing PORT command of an FTP session could be saved so that an incoming FTP data connection can be verified against it.

The firewall you build in this project should utilize all these information.

2.2 Implementation

While section 2.1 gives us a motivation for building a stateful firewall, this section gives a high level overview of building such firewalls. As in project 2, the firewall you build in this project should be implemented at the IP layer so that it can intercept and inspect all inbound and outbound packets on all interfaces. Because it has access to the "raw message", it can inspect all the information in the message, including information relating to all the higher communication layers, as well as the message data itself (the communication- and application derived state and context).

Figure 2 gives a flowchart showing how a stateful firewall can take decision on whether some incoming packet should be dropped or accepted.

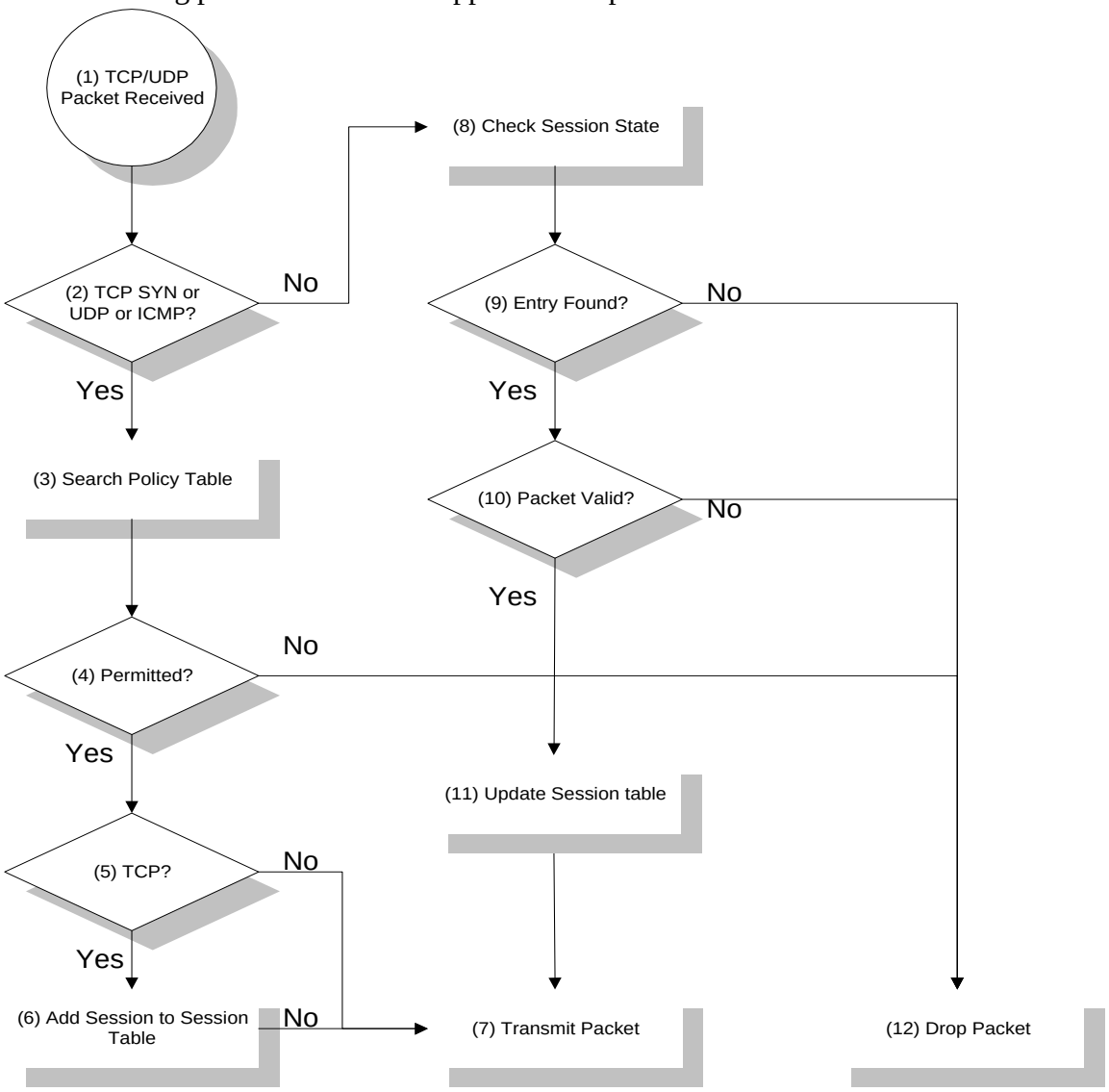

Figure 2: Flowchart showing functionality of a stateful firewall

A stateful firewall maintains two separate tables. The first one, a *policy table*, is the static rule base that indicates whether a request for some connection should be accepted or not.

This decision is made based on criteria such as protocol, source/destination addresses, source/destination ports of the connection request etc. You have already implemented this table and this part of the filtering in project 2. You will use the same `setfilter` program to read the rules from a file and install them in the policy table.

The second table, a *session table*, is dynamic and maintains state of each of the existing connections. e.g., protocol, source/destination addresses and ports, TCP state of the connection (see section 2.3), sequence number of last packet seen on that connection, a timestamp, etc. Some information about the connection entry (e.g., TCP state, sequencing information etc.) in the table may get changed as packets for that connection pass through the firewall. Whenever a TCP connection is established through the firewall, an entry for the connection is entered into the session table. The entry remains there until the termination of the connection.

Let us see how different packets are accepted or rejected by the firewall according to the flowchart in figure 2. Consider the following sequence of packets. (Refer back to figure 2 for step numbers).

1. A SYN packet sent from Alice (10.0.0.1:1025) to Bob (192.168.1.1:80). Since this is a SYN packet (step 2), the firewall checks the policy table (step 3) to see if the packet is permitted to go through. Considering the rules in section 2.1, assume the packet is permitted (step 4). Since it is a TCP packet (step 5), an entry for this connection will be created in the session table (step 6), and the packet will be transmitted (step 7).
2. A SYNACK packet is sent from Bob to Alice. Since this is not a SYN packet, the firewall will consult the session table (step 8). It will find an entry for the connection to which the packet is belongs (step 9). The entry will suggest that an SYNACK packet is valid in this stage of the connection (Step 10). So, the session table will be updated (step 11) and the packet will be forwarded to Alice (step 7).
3. A HTTP request packet is sent from Alice to Bob. Same as previous scenario.
4. Evil Charlie sends a packet to Alice with source port 80, and no flag set. Since this is not a SYN packet, session table will be consulted for the packet (step 8). There will be no connection entry for the packet, and so the packet will be dropped (step 12). (Note, Alice's stateless firewall would let this packet go through).

You need to decide what information you should maintain for each connection in the session table. The goal is that you should be able to decide whether a packet belongs to an existing connection or not, and whether a packet is consistent with the current state of the connection. The latter goal can be achieved by keeping track of current state of each existing connection in the TCP state machine.

You may think of having a timestamp in the session table entry, so that if one end of the connection dies, the entry corresponding to the session entry times out and is removed from the table (garbage collection). However, we have decided that you don't need to implement this feature. Entries will be removed from the session table when the connection terminates. If one end of the connection dies, the connection entry will be left in the session table.

2.3 TCP State Machine

The operation of TCP with regard to connection establishment and connection termination can be specified with a state transition diagram. We show this is figure 3.

There are 11 different states defined for a connection and the rules of TCP dictate the transitions from one state to another, based on their current state and the packet received in that state. For example, if the application performs an active open in the CLOSED state, TCP sends a SYN and the new state is SYN_SENT. If TCP next receives a SYN with an ACK, it sends an AC and the new state is ESTABLISHED. This final state is where most of the data transfer occurs. (UNIX `netstat` command shows the existing connections in the machine with their states.)

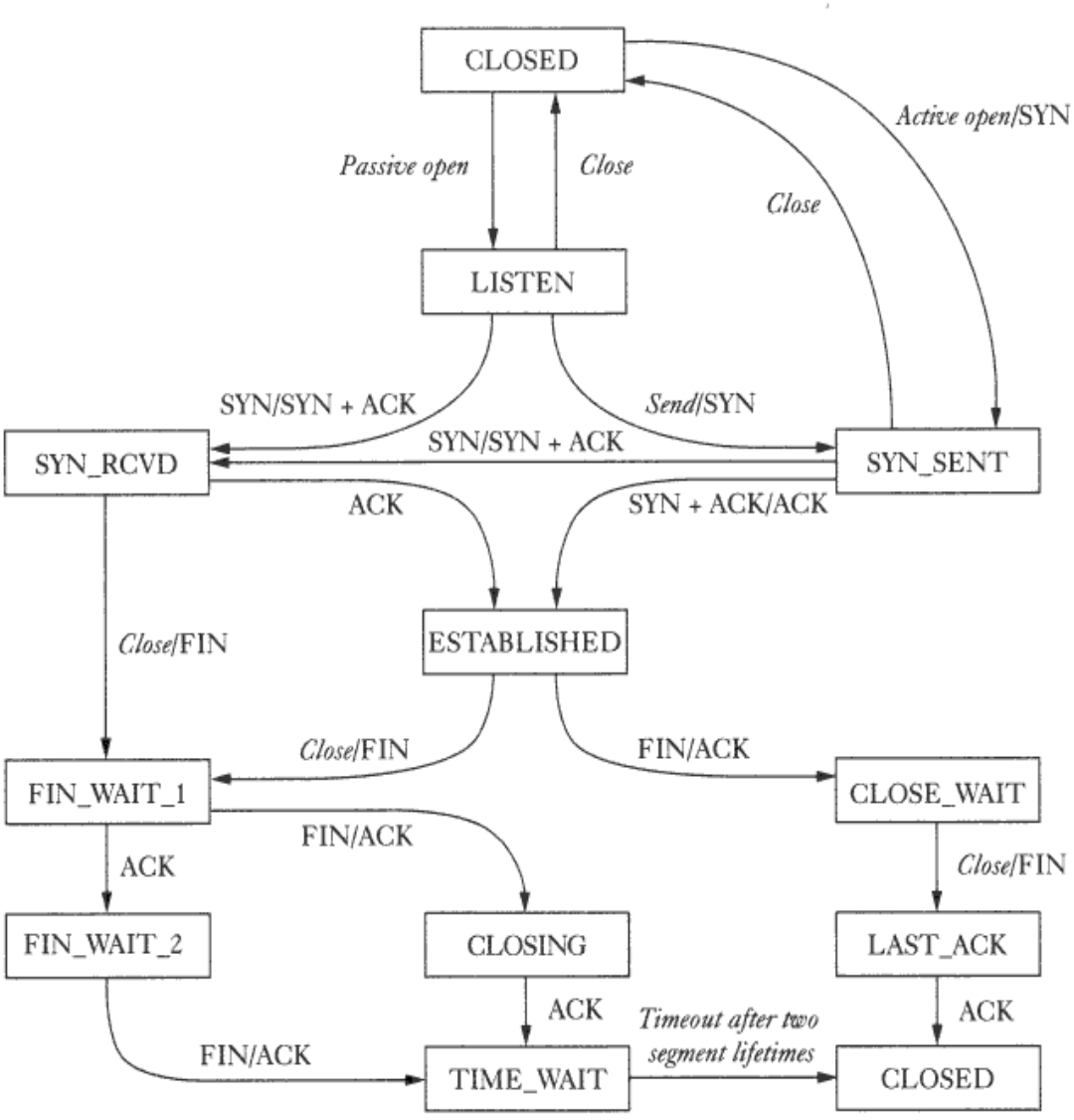

Figure 3: TCP State Machine

The session table entry of your stateful firewall should keep track of the state of each of the connections. Whenever a new packet passes through the firewall, the state of the connection to which the packet belongs to is updated. This allows detection of inconsistent packets. For example, if a connection is in ESTABLISHED state, a SYN packet in that connection will not be allowed to pass through the firewall and be dropped.

2.4 Filter Rules and Their Instantiation

As explained before, the policy table of the stateful firewall implements the same filter rules as stateless firewall. You will use the same syntax and instantiation system for the filter rules that you used in project 2. There is no static filtering rule for the session table, but the entries will be added and removed dynamically.

2.5 Filter and NAT positioning

The assumption about the placement of filtering and NAT functionalities are same as project 2. Figure 4 from project 2 still holds here.

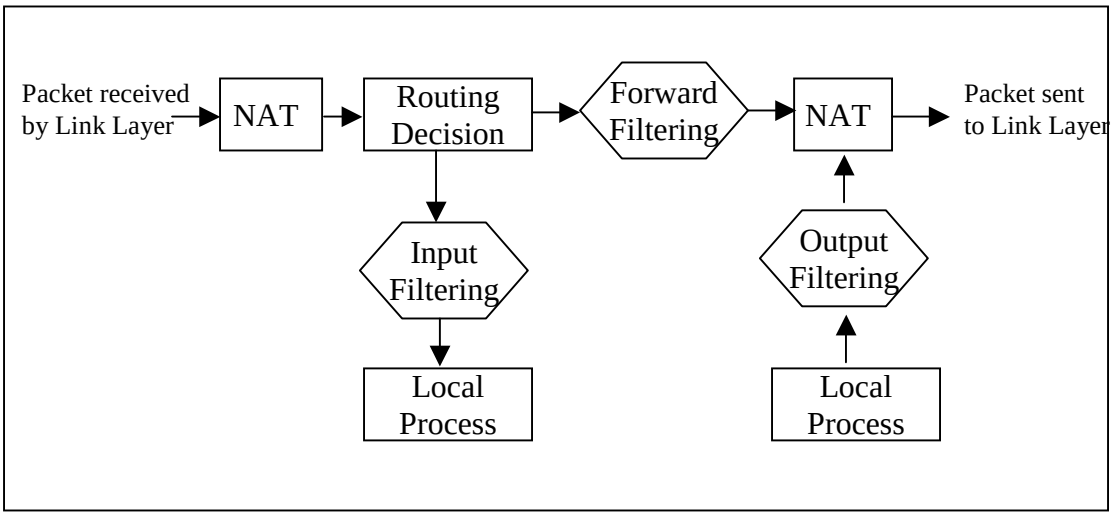

Figure 4: processing of packets in the IP layer

2.6 TCP Layer of the Simulator

The implementation of TCP in the simulator differs in some respects from the requirements listed in RFC 793. However, most differences do not have any influence on the project and you can ignore them. Some of the differences include:

- The TCP version included in the simulator does not generate packets that have the PUSH, RST, or URG flag set. (Nonetheless, your firewall implementation should to be able to filter packets based on these flags).
- Sequence numbers are per packet, not per byte.
- ACKS and data are never sent in one packet.
- ACKS have the sequence number field set to zero.
- Data packets have the acknowledgment number field set to zero.

We will post more information on this topic on the course bboard, so please check it regularly.

3 Dynamic NAT Support

In project 2, you have implemented basic NAT in the IP layer. In this project we will implement a dynamic NAT that use Port Address Translation (PAT) feature. This feature is used to connect an isolated address realm with private unregistered addresses to an external realm with *only one* globally unique registered IP address. The NAT dynamically assigns a port number to distinguish between the connections that use unregistered addresses.

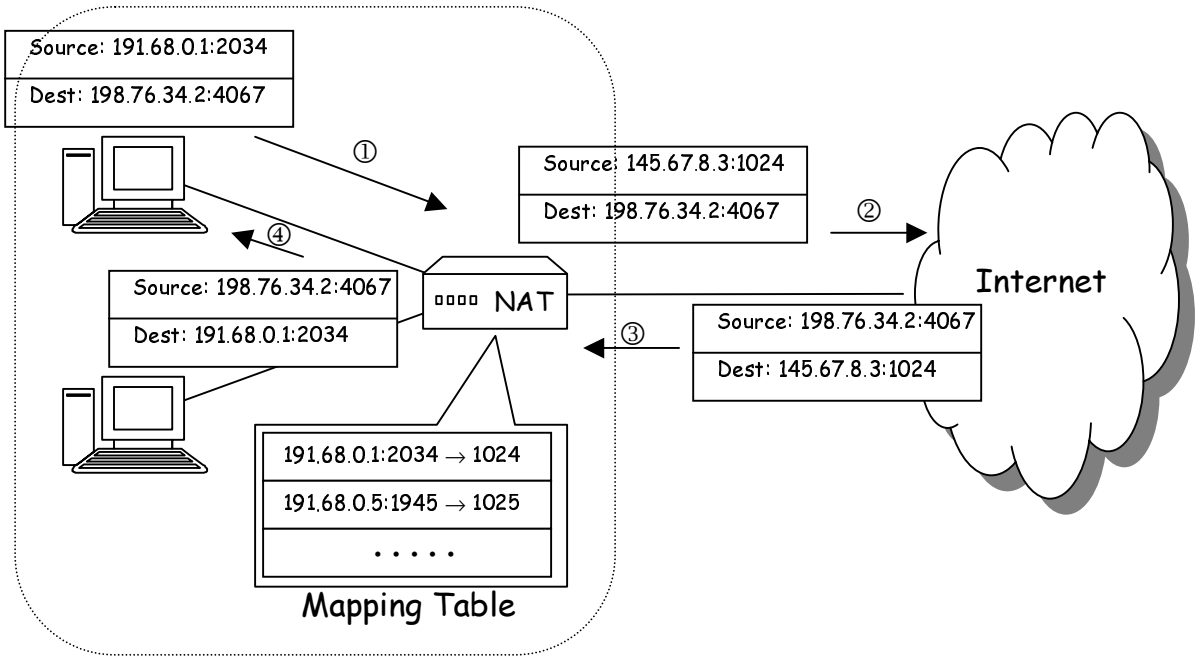

Figure 5: NAT with port address translation

Figure 5 demonstrates a sample scenario to explain how NAT with PAT works. Hosts A and B have private IP addresses 191.68.0.1 and 191.68.0.5. Private IP addresses are unique within the local network, but not globally unique. All the packets going from the private network to the Internet have to go through a NAT box. The NAT box is assigned a globally unique IP address (145.67.8.3) for address translation. Assume host A wants to communicate with a host on the Internet that has globally unique address 198.76.34.2. It generates a packet with source address 191.68.0.1, source port 2034, destination address 198.76.34.2, and destination port 4067. When the NAT box receives this packet, it replaces the private source address with the global address 198.76.34.2 and the source port number with some port number 1024. Similarly, if B wants to communicate with some host on the Internet, the NAT box will use 198.76.34.2 (NAT's IP address) and 1025 (some unused port number) to replace the source address and port number of the connection. The NAT maintains the mapping in a table. When a packet comes from outside world (from the host whom A is sending data) to the NAT with destination address 198.76.34.2 and destination port 1024, the NAT consults its table and find that the packet is for A, replaces the destination address and port with 191.68.0.1:2034 (from the table), and forwards it to A.

You do not need to deal with ICMP packets here. As before, in addition to changing the source address and port number, you may have to also modify some other fields in the TCP/UDP/IP header to make end-to-end communication work. As in project 2, you need to write a user program `setnat` that takes a single public IP address as argument and sets it as the IP address of the NAT. You can use any port numbers of your convenience as replacement of the source port numbers (but they should be unique for each existing connections, after all the NAT will identify different connections using port numbers).

Note that sometimes NAT may need to modify the data portion of the packet to make some applications work in the environment described earlier. For example, a FTP client inside the firewall may issue a PORT command with its private address and port number to a FTP server outside the firewall. NAT must modify the command (replace the IP address of the PORT command) before it forwards the command to the outside server. You should be aware of this because your implementation of firewall should allow FTP clients to communicate with outside FTP server. We will discuss more about FTP support in next section.

4 Back Channel Support in FTP

One challenge of using a firewall is to support FTP application passing through it. There are some situations that a stateless firewall cannot handle at all. One example is to support back channel for a FTP connection. Suppose a client inside the firewall specifies a port number by the PORT command and requests some outside server to send a file to that port. Then the server creates a connection back to the client to send the file. Stateless firewalls may block such incoming connections (thinking this is an attack from outside). Your implementation of the firewall should handle such scenarios and ensure that a FTP client inside the firewall can communicate with some server outside the firewall. Your firewall should detect the FTP PORT command and record the client specified port number for that particular server. Later, when the outside server attempts the FTP data connection to that port, firewall should extract the IP addresses and ports from the connection request packet, examine the list, and verify that the attempt is in response to a valid request.

You do not need to implement the FTP server and client that work with the simulator. You will be provided with a very simple FTP client and server in the directory `$PDIR/ftp/`. They are not full-featured programs, however they can be used to test whether your firewall allows the FTP applications to work through it. `$PDIR/ftp/README` provides description of these utilities.

5 Logistics

5.1 Starting Code

The project will be based on your project-2 code. We encourage you to use your own peoject-2 code (since you know the details of the code), and add additional features to it to implement the project-3 functionalities described before. However, if your IP layer does not already meet the requirements for project 2, you must fix the remaining bugs and implement the remaining features in order to successfully complete this project. You

don't need to have the NAT and DHCP working to complete this project, although the NAT of this project will have similar functional structure as the one in project 2. If you believe your project 2 code is hopelessly broken, please bring it to our attention.

5.2 Groups

Since you will be extending the code you have written for project 2, we assume that all the groups will remain the same. If you need to change your group for whatever reason, you must see Suman Nath (sknath@cs.cmu.edu), one of the TAs for this course, by Tuesday, November 6, and no later.

You are not permitted to share code of any kind with other groups, but you may help each other debug code. Each member of the group is responsible for both sharing the work equally, and for studying the work of their partner to the point they understand it and can explain it unassisted.

5.3 Equipment and tools

For this project (and all subsequent projects in the course) you will use computers from Sun Microsystems running the Solaris operating system. Such machines are available in several clusters, notably the Wean Hall clusters. You may work from any workstation you have access to, be it a Solaris machine or not, by telnetting to unix*XX*.andrew.cmu.edu. The support code for this project uses the Solaris threads package internally, so *you will not be able to compile, test or run your network stacks except on Solaris machines.*

All programs must be written in either C or C++. Note that the TAs do not provide support for problems occurring due to the usage of C++. We strongly recommend you compile your code with gcc and debug it with gdb. Also strongly recommended is the use of the -Wall and -Werror flags for gcc. This will force you to get rid of all possible (well, almost) sources of easily avoidable bugs (so that you can concentrate your debugging efforts on the unavoidable ones).

5.4 Provided Code

All of the provided code and infrastructure is available in $PDIR, which has the following contents:

- README: a file in the project2 directory describing the contents;
- include/: the header files for your interface with the simulator -- **DO NOT** copy or modify these!!;
- lib/: the archive files containing the simulator that you link to;
- template/: a template for the code you must write. It also includes a Makefile, the startkernel.pl script, etc. -- you should copy all these files to your own working directory;
- utils/: a collection of utilities for testing and configuring your network, explained in the README file in this directory.

It is very important that you *do not* copy the libraries that we provide into your own directory. You should link against the library where it resides in the official project2 directory. This is already taken care of in the template Makefile that we provide. We

reserve the right to make changes to the library as needed, and you will not receive the updates if you link against your own copy.

The code in the `$PDIR/template/` directory is merely a suggestion: you are free to modify any of the code there or start from scratch. However, it will make it easier for us to help you if you have used the template and retain the function prototypes that we have suggested. Also, the template was designed with a C implementation in mind. It may be more difficult to follow this template if you intend to use C++ to implement your networking layer.

Also note that your project code will use both the Solaris standard header files and the header files specific to this project. The header files such as <netinet/*.h> or <sys/*.h> are the standard header files on typical UNIX machines. <project2/include/*.h> and "*.h" are header files specific to this project. When including a header file, you should be conscious whether the header file is a standard header file or a project-specific file.

FYI, you don't need to define data structures for the various headers. They are already defined in several standard Solaris header files. Several examples:

The UDP header: struct udphdr in <netinet/udp.h>.
The ICMP header: struct icmp in <netinet/ip_icmp.h>.
The IP header: struct ip in <netinet/ip.h>.

There are some other header files in the /usr/include/netinet/ directory that you may find useful. However, you probably don't need all of them. In fact, you don't need to use these structures at all if you prefer defining your own. We just want to let you know that they exist.

5.5 Communication

We reserve the right to change the support code as the project progresses to fix bugs (not that there will be any :-) and to introduce new features that will help you debug your code. You are responsible for reading the bboards and the project home page to stay up-to-date on these changes. We will assume that all students in the class will read and be aware of any information posted to the bboards and the project home page.

If you have any question regarding this project, please post your question to the class bboard. Please make your questions clear and specific to increase the chance that we can solve your problem with one response. As always, the course staff is available for help during office hours.

5.6 A Brief Report

Each group should create a brief report describing their efforts, in one of the following formats: plain text, postscript, or html. Please use the file name proj3_report.{txt|ps|html}. Your report should describe the following:

- A breakdown of what each group member did (use a table for this).
- At a high-level, describe your implementation of the firewall support and the NAT implementation.

- The simulator uses per packet TCP sequence number, whereas RFC 793 suggests per byte sequence number. Mention in your report what modification you need to make if the simulator uses per-byte sequence number.
- Describe any interesting testing strategies that you have used.
- Describe what works and what does not, and if not, why (use a table for this as well).
- Your thoughts on the project: was anything too difficult? What would improve the project?

5.7 Submission and Policy

You must use a Makefile, and your project should build completely by typing gmake. We will post further guidelines for how to submit your project and report. Remember the following guidelines when handing in the code.

- Do not create directories in your handin directory.
- Do not hand in any configuration files (e.g., topology files files containing filtering rules) or scripts.
- Do not hand in any source code of applications other than source code of required applications (e.g., no source code of testing applications)

Note that we do not expect to grant any extensions for this project. We must be notified *immediately* of any extenuating circumstances you might have, including problems with your partner. A disaster that occurs during the last few days before the deadline may **NOT** qualify you for an extension, since you still would have had the majority of the project duration to work. For example, don't come to us complaining that your partner has been bad for two weeks or you have been sick for two weeks and you need an extension. If you need help with this sort of problem, you must come to us before the damage is irreparable.

5.8 Grading

We will post the grading scheme for the project in the bboard. The grading philosophy for this project is that your grade increases with the number of components that work. *It is better to have a few solidly-working components than many slightly-broken components, and this should influence your implementation plan.*

6 Final Advice

Start Early. Check course bboard regularly.