Implementing a Simple VNS Router and Firewall

VNS developer and provider: Stanford University
Assignment Designer: Andreas Terzis
Presented by Jong Lim
Goal of this project

• Understand one of important concepts on networking layer, forwarding, by developing a software router.

• Understand some firewall concepts/functionalities and implement them by adding them to the router.
Problems developing a software router

• Developing real software router in your machine is extremely difficult and time-consuming.
  – Need to deal with kernel to interact with OS.
  – Debugging will take millions of years.
  – Assigning real HW router to student would also be problematic

• Instead, we develop our software router on top of application layer using a system called VNS.
VNS

- **VNS**: Virtual Network System, developed by Stanford University

- **System Components**
  - VN Server: a user level process (application) running at Stanford (Linux Machine)
  - VN client: the client program we are going to develop. A user level process running on your PC.
How does the system work?

- Problem: VN server is on application layer (layer 7), but a router works on layer 3.
- 3. VN server process intercepts incoming packets (raw packet: Ethernet header + IP/ICMP header + data) and forwards it after encapsulating the raw packet with some additional headers to the VN client via the established TCP connection.
- 4. VN client looks at the forwarded packet and 1) performs forwarding decision, 2) changes Ethernet header and 3) forwards the modified packet back to the VN server.
- 5. Once VN server receives the modified packet, it actually sends it out to a physical interface.
Each interface (LAN card, such as eth0 in the previous figure) has a hardware address, called MAC.

When a datagram is generated and sent through an interface, it needs to be encapsulated with Ethernet header.

When this packet travels through a network, destination and source MAC are changed at each hop you pass, but destination and source IP are not going to be changed.
ARP reply/request

• ARP request:
  – This packet is sent whenever you don’t know destination’s MAC address, but only know its IP address.

• ARP reply:
  – When a node receives a ARP request, if the ARP request asks this node’s MAC address, the node replies with ARP reply packet by specifying its MAC address.
  – When ARP requester receives the ARP reply, it can then send data packets to this node by specifying destination MAC address in the Ethernet header.
Things to do

We are NOT going to develop the VN server. We are developing a VN client !!

• Do not need to do:
  – 1. Intercept incoming packets from OS. For this dirty job, VN server will do that for you
  – 2. Build and maintain routing table. We will use a static routing table

• Do need to do:
  – 1. Make a forwarding decision based on your given routing table.
  – 2. Construct/Modify a variety of headers: Ethernet header, IP header, ARP header, ICMP header etc.
    • Change TTL field and update checksum
  – 3. Build and maintain an ARP cache
  – 4. Buffer packets while performing necessary operations (will be explained in “Ping Example”)
  – 5. Use send and receive functions provided by libraries to communicate with VN server

• Things provided to you
  – Interface MAC addresses (eth0, eth1, and eth2 in previous figure)
  – IP addresses of the virtual router’s interfaces in the previous figure
  – Bootstrap code and libraries to make a connection with VN server, send and receive data to and from VN server.
* What is provided:
  - MAC addresses: eth0, eth1, and eth2
  - IP addresses associated with each interface (eth0, eth1, eth2)
  - Routing table having three entries
    - IP of application server 1
    - IP of application server 2
    - IP of default route: firewall
Ping Example

Suppose User A finished implementing VN client and connected to VN server at “virtual router”.

* Note: User A is running two processes: 1) ping and 2) VN client
User A, VN client

2. Sends ARP request to get eth0’s MAC
Ping Example

3. VN server forwards this ARP request to VN client

User A, VN client
Ping Example

4. VN client inspects the packet and recognizes that it needs to send ARP reply to the firewall

User A, VN client
Ping Example

5. Construct ARP reply packet: Ethernet header + ARP header

User A, VN client
Ping Example

6. Send the constructed ARP reply packet back to VN server

User A, VN client
Ping Example

7. VN server sends the ARP reply (from VN client) to firewall

User A, VN client
8. The firewall then forwards User A’s ping message.
Ping Example

9. VN server forwards this ping packet to VN client

User A, VN client
Ping Example

10. VN client looks up its routing table and 1) if the ping (ICMP request) is destined to a local interface, constructs ICMP echo reply and sends it to VN server
Ping Example

User A, VN client

Eventually, your ping process will show the reply message!
Comments on Ping Example

• Steps 2 ~ 7 (ARP request/reply) are performed only when the firewall does not have MAC of the VN server’s interface, eth0. (or ARP cache times out so that it does not have that entry)

• Thus, once it learns the MAC (caching to ARP cache and it is not timed out), it does not need to repeat steps 2 ~ 7 (as it is cached); it is going to handle ping message only.

• In step 10, if the packet is destined to “application server 1” you need to forward the ping packet through interface “eth1”
  – In this case, decrement TTL by 1, and re-calculate the checksum value (IP header).
Packet Generation/Forwarding

• When generating a packet: (e.g. ping is received to your interface, eth0)
  – 1. Ethernet header (sr_protocol.h, most of header formats are described here)
  – 2. IP/ICMP header
  – You need to specify destination/src MAC and IP addresses correctly (using routing table and ARP cache, APR reply/response)
  – Checksum (if needed, e.g. IP and ICMP) must also be calculated.

• When forwarding a packet: (e.g. ping is destined to application server 1)
  – Refer to your routing table, find destination and associated interface (e.g. eth0)
  – Before sending, decrease the TTL of IP header field by 1, and recalculate the checksum.
  – IP header’s destination and sources address are kept intact, but MAC source and destination addresses must be changed.
  – Just in case you don’t know destination MAC, generate an ARP request, and buffer the data packet. After the destination MAC (via ARP reply) becomes known, then flush out the buffer, which contains data packet.
What to do: Router Part

- The router can successfully route packets between the firewall and the application servers.
- The router correctly handles ARP requests and replies.
- The router correctly handles traceroutes through it and to it.
- The router responds correctly to ICMP echo requests.
- The router handles TCP/UDP packets and to one of its interfaces. In this case, the router should respond with an ICMP port unreachable.
- The router maintains an ARP cache whose entries are invalidated after a timeout period.
- The router queues all packets waiting for outstanding ARP replies. If a host does not respond to 5 ARP requests, the queued packet is dropped and an ICMP host unreachable message is sent back to the source of the queued packet.
- The router does not needlessly drop packets.
What to do: Firewall Part

• The ability to declare an interface as internal or external. For simplicity only a single external interface is supported.
• By default, packets arriving to the external interface and destined to an internal interface are silently dropped (no ICMP packet is generated)
• By default, packets arriving from an internal interface that need to be forwarded through the external interface are allowed to pass through the firewall. Doing so, allows end-hosts within the protected network to access services on the public Internet.
• Detail, http://yuba.stanford.edu/vns/assignments/firewall
Getting Started

• To run your VN client
  – You need to specify ID, authentication key, and topology

• ID, password and authentication key
  – ID will be “group number” you already use (e.g. group13)
  – Password is currently the same as group ID (please change your password in webpage: http://vns-1.stanford.edu/)
  – Authentication key will be shown in the webpage after you login to the system with your ID and password.
  – Topology number: each group will have a pre-assigned topology number associated with ID (think of it as your subnets)

• Download skeleton code
  – The code consists of 1) bootstrap code for user to connect to VN server and 2) libraries to manipulate packet (header structure as well) and develop your own code.
Skeleton Code Test Run (1)

• To download source code package:
  – Go to http://yuba.stanford.edu/vns/assignments/simple-router
  – Click on “Stub Code”

• Untar the package and try ‘make’ in the directory

• Routing table: go to http://vns-1.stanford.edu/

• Login with your group ID and password
Click on 'Details' to see the topology of your Group and Routing Table.

<table>
<thead>
<tr>
<th>Details</th>
<th>Public</th>
<th>19.17.67.240</th>
<th>1768/29</th>
<th>29</th>
<th>1-Router 2-Server</th>
<th>Jonghyun Lim</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology Routing Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19.17.67.240.0/21 (Public)</th>
<th>IP Addressed From</th>
<th>IP Addressed To (Public)</th>
<th>Type</th>
<th>Allowed Source IPS (for traffic)</th>
<th>Allowed Users</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Server</td>
<td></td>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Skeleton Code Test Run (3)

• You also need to know authentication key to connect to the server, which is described in the webpage
  – Click on your ID and then save it into file, which is referred to as ‘key file’ below
• ./sr -t [topology number] -s vns-2.stanford.edu -u [ID] -a [key file]
• Your topology number is at the end of slides
• The output would look like as follows
Skeleton Code Test Run (4)

Routing Table: “sr_rt” data structure

Interface: “sr_if” data structure

The two data structures are successfully generated!!
Where to develop your code?

• sr_router.c and sr_router.h
  – Implement the functionalities described before in “sr_router.c”
  – Whenever a packet is received (from VN server), the function “sr_handle_packet()” will be called, within which you are going to process incoming packets
  – You can also make separate modules
  – Declare the function interfaces and constants you make in “sr_router.h”
Programming Language

• C programming only
Data Structures to use for constructing a packet

• sr_protocol.h:
  1. Ethernet, IP, ARP headers to use are defined
  2. Constants are also defined
     - e.g. Ethernet header has ‘type’ field to indicate which upper layer protocol (such as IP or ARP) is used
     - For this, ETHERTYPE_IP constant is defined in the file
     - The number for each type can be found in the reference
Three Important Data Structures to Use (1)

• **1. sr_instance**: (defined in sr_router.h) contains all the metadata we need to use for the entire course of operations:
  1. Socket descriptor
  2. User name, hostname, authentication key for verification
  3. Topology ID
  4. Interface list (next slide)
  5. Routing tables (next slide)
* This data structure is used every time you send/receive a packet

• e.g. when sending data to VN server, the sending interface sr_send_packet has the declaration
  – int sr_send_packet(*struct sr_instance*, unsigned short, char*)
  – also used for receiving data
Three Important Data Structures to Use (2)

• 2. Interface, `sr_if`: (defined in `sr_if.h`), describes the hardware interface (Ethernet interface)
  – After connecting, the server will send the client the hardware information for that host. The stub code uses this information to create a linked list of interfaces and attaches the list to `sr_interface`.
  – automatically built when it successfully connects to VN server

• 3. Routing Table, `sr_rt`: (defined in `sr_rt.h`)
  – The routing table contents are structured in memory following the data structure of `sr_rt`.
  – `sr_rt` is attached to the `sr_interface`.
  – Automatically built when the client starts and reads routing table from a file (by default, `rtable`).
Two Important Functions

• For receiving, `sr_handlepacket( ... )`: each time a packet is received, this function is called.
  – Interface is written in `sr_router.c`, but the function is pretty much empty.
  – You need to implement your own code inside this function.

• For sending, `sr_send_packet(...)`: this function will send an arbitrary packet of length, `len`, to the network out of the interface specified in the ‘iface’ parameter.
  –_iface: eth0/eth1 (e.g.)
  – “sr_vns_comm.c” module provides implementation and interface to users.
  – You are going to use this function to send data
Due Date & Deliverable

• Due Date: 29 April 2010, 11:59pm
• Source file (including sr_stub codes)
• No binary executables
• README file: group number, member name, CS344/444
  – Describe, if any, which bullet points are missing in slide 21 ~ 22
Information

• Our assignment description:
  – http://yuba.stanford.edu/vns/assignments/firewall
• General description:
  – http://yuba.stanford.edu/vns/
• Ethernet Header Type field:
  – http://en.wikipedia.org/wiki/EtherType
• IP Header protocol number field
  – http://www.iana.org/assignments/protocol-numbers
• Internet checksum: for IP header and ICMP
  – Peterson and Davie, Computer Networks, 3rd edition 91 pages (describing algorithm and actual code as well)
• FAQ:
  – http://yuba.stanford.edu/vns/assignments/simple-router/faq
### Group number and corresponding assigned topology number

<table>
<thead>
<tr>
<th>Group number</th>
<th>Topology number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group number</th>
<th>Topology number</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>44</td>
</tr>
<tr>
<td>14</td>
<td>45</td>
</tr>
<tr>
<td>15</td>
<td>46</td>
</tr>
<tr>
<td>17</td>
<td>47</td>
</tr>
<tr>
<td>20</td>
<td>48</td>
</tr>
<tr>
<td>23</td>
<td>49</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
</tr>
</tbody>
</table>