CSC/ECE573 Internet Protocol Project #2

Relay server

Due date : 11:45 PM Mar 19

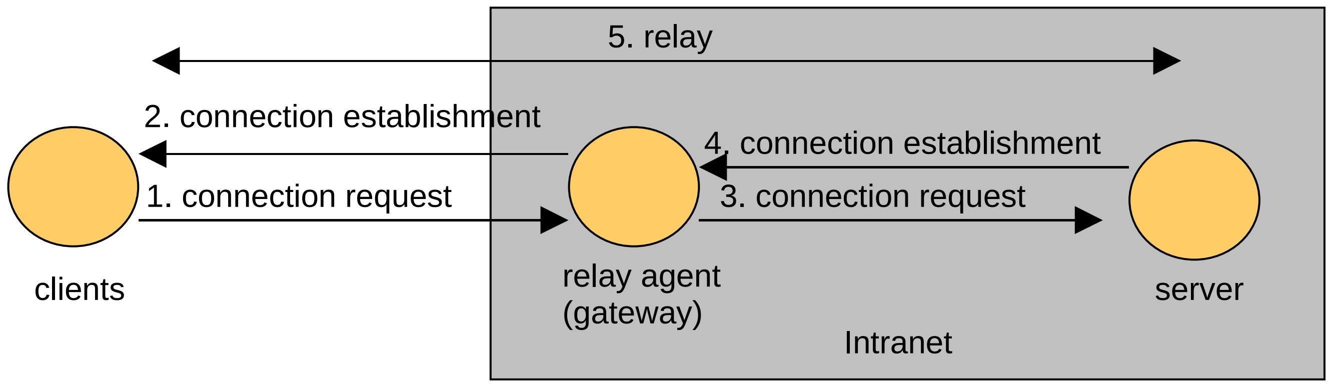
This assignment is an individual assignment. This project implements simplified TCP, UDP delay server operating as a kernel module (or user-mode daemon for TCP relay server) as a preparation of final project. In this project, performance measurement of data packets relay will not be made, instead your module must guarantee "functionality" issues.

Implementing TCP relay server

In this assignment, we build a simplified tcp relay server that relay packets between a server and clients to enhance the understanding of socket programming in kernel environment.

The basic operation of the TCP relay server is :

1. Listens for a TCP connection request from a client
2. Establishes a "client connection" with that client and makes a connection request to a server to which the connection is bound.
3. Forward the packets from the client to the server and forward the packets from the server to the client.
4. Closes the both peer connections in case of close request from one of the connections (graceful disconnection)



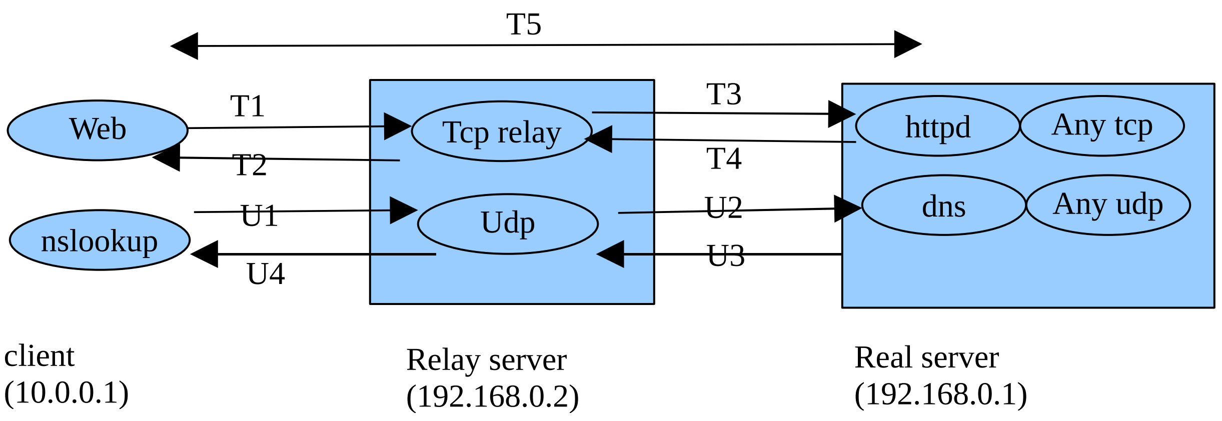
The transferred packets , both client to server and server to client, will flow through your module. TCP relay server must handle multiple , simultaneous connections with a minimum delay. Sender's packets delivered to the application layer are transferred to receiver through the established channel.

Implementing UDP relay server (NAT on netfilter)

Your relay module will also relay UDP packets between clients and servers. To do this, register your translate functions in NF_IP_PRE_ROUTING , NF_IP_POST_ROUTING hooking points. Hooking functions probe the UDP packets which flows through the hooking points , and change the client (source) address to the relay server address. And then, the address pair should be managed in your module to find the requester when the reply from real server arrives in relay server. As UDP uses NAT , the packets are directly forwarded to the real server. (The UDP packet does not reach to the application layer)

Implementation(detailed).

You will use C programming language on a linux OS (Kernel ver >= 2.4). Your web-browser, nslookup in client machine connects to the TCP,UDP relay server with a standard service port. Your clients should act in a same way when they connect to the httpd,DNS in a real server. To make virtual network environment, each relay virtual machine and real virtual machine should be installed as guest-OS in Vmware.



Packet type	(Source address) , (Destination address)
T1. TCP (C->krelay) (DNAT) (krelay ->tcp relay)	(10.0.0.1:x) , (192.168.0.2:80) dest port: (192.168.0.2:80) ==> (192.168.0.2:A) (10.0.0.1:x),(192.168.0.1:A)
T2. (tcp relay -> krelay) (SNAT) Tcp (krelay->C)	(192.168.0.2:A) , (10.0.0.1:x) source port: (192.168.0.2:A) ==> (192.168.0.2:80) (192.168.0.2:80) , (10.0.0.1:x)
T3. (tcp relay -> krelay) (SNAT) Tcp (krelay->S)	(192.168.0.2:y),(192.168.0.1:80) source ip : (192.168.0.2:y) ==> (10.0.0.1:y) (10.0.0.1:y), (192.168.0.1:80)
T4. Tcp(S->krelay) (DNAT) (krelay -> tcp relay)	(192.168.0.1.:80),(10.0.0.1:y) dest ip: (10.0.0.1:y) ==> (192.168.0.2:y) (192.168.0.1:80), (192.168.0.2:y)

T5. TCP(C->krelay) (DNAT) (krelay ->tcp relay) (tcp relay-> krelay) (SNAT) TCP(krelay->S) TCP(S->krelay) (DNAT) (krelay ->tcp relay) (tcp_relay->krelay) (SNAT) TCP(krelay ->C)	(10.0.0.1:x) , (192.168.0.2:80) dest port:(192.168.0.2:80) ==> (192.168.0.2:A) (10.0.0.1:x),(192.168.0.2:A) (192.168.0.2:y) , (192.168.0.1:80) source ip: (192.168.0.2:y) ==> (10.0.0.1:y) (10.0.0.1:y), (192.168.0.1:80) (192.168.0.1:80), (10.0.0.1:y) dest ip: (10.0.0.1:y) ==> (192.168.0.2:y) (192.168.0.1:80),(192.168.0.2:y) (192.168.0.2:A),(10.0.0.1:x) source port: (192.168.0.2:A) ==> (192.168.0.2:80) (192.168.0.2:80),(10.0.0.1:x)
Close same flow T1,T2,T3,T4	Close both(to client, to server) peer connections if the tcp_relay server gets close packet.

<tcp packet flow>

*x : ephemeral client port
*y : ephemeral relay port (We can not use x)
*A : Gateway port of relay server

Packet type	(Source address) , (Destination address)
U1. UDP (C->krelay)	(10.0.0.1:x) , (192.168.0.2:53)
U2. (DNAT) UDP (krelay->S)	Dest ip: (192.168.0.2:53) ==> (192.168.0.1:53) (10.0.0.1:x), (192.168.0.1:53)
U3. UDP (S->krelay)	(192.168.0.1:53),(10.0.0.1:x)
U4. (SNAT) UDP(krelay->C)	Source ip: (192.168.0.1:53) ==> (192.168.0.2:53) (192.168.0.2.:53),(10.0.0.1:x)

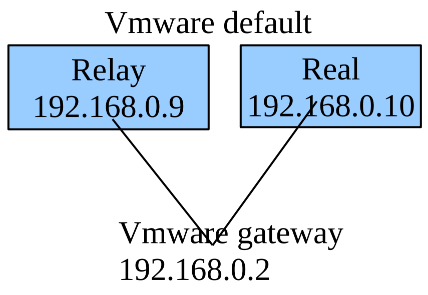
<udp packet flow>

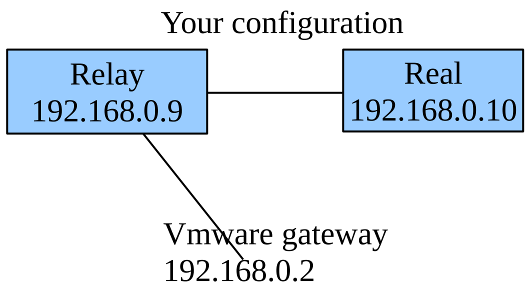
1. krelay (kernel module) : Kernel has 3 hooking point "NF_IP_PRE_ROUTING" , "NF_IP_PORT_ROUTING", and "NF_IP_FORWARD". We use "NF_IP_PRE_ROUTING" for pre_routing phase and "NF_IP_PORT_ROUTING" for post_routing phase. Krelay will register 2 function to these hooking points. These 2 functions will do SNAT , DNAT role. You should consider various cases: UDP , TCP connection, TCP flow , TCP close, other packets, etc. krelay also has built-in data-structure(simple linked-list or array) to contain the NAT table.
2. tcp_relay(user daemon or kernel module) : It will relay tcp packets between client and server as well as virtual connection and close. To help the development, the old tcp_relay(user level , C) source is provided. Here are 2 development options:
   1) Modifying the tcp_relay server (user mode)
      You can modify the tcp_relay server to work in your environment.
   2) Adopting the tcp_relay server on krelay (kernel mode)
      You can also make krelay module do the tcp_relay role. It is the best choice for the final project. This strategy is recommended to the students who have enough skill for the kernel programming.
3. Communication between tcp_relay and krelay (user mode tcp_relay)
   1) NAT table should be accessed from user mode process(tcp_relay) as well as from kernel module. To do this, "getsockopt handler" should be registered to reply to "getsockopt()" call

from user mode process.
2) packets flow via socket.

Test.

You should ensure the packet from real server go through relay server. To do this, on real server set relay server as a default gateway and check the unnecessary static routing entry (you may delete some configuration if needed). Enable ip_forward on relay server.
Vmware network topology differs from the one in Lab#1 in that it adopts additional address as a gateway. We already treated this subject in Lab#1.





Use any TCP, UDP client, servers to test the functionality. Your client should get reply from relay server for the request to well-known server port.
Deactivate all the test server daemons in your relay server for the complete test. Packet capture program (wire-shark) will help you to identify the packet status.

Configuration

1. TCP Gateway port: /proc/krelay/gport   default : 16000
2. Real Server Address : /proc/krelay/real_ip   default : 192.168.0.10 (or any other)
3. Start/stop : /sys/module/krelay/parameters/start   default: 0
   - Your module initially stopped. When the event “start : 0 -> 1” , it registers DNAT, SNAT to hooking points and start its relay role. When the event “start: 1-> 0”, it unregisters its hooking functions.
4. tcp_relay:
   It refers to the /proc/krelay/gport, real_ip to get the configuration. /proc files can be read in user mode process as they are normal files.

- tcp_relay do not have start/stop function.

Notes

– Before implementing in kernel level , thorough understanding of socket programming is required to the students who do not have enough experience. To do this, prototype development in user level will greatly enhance your understanding of what you do.

– tcp_relay must handle multiple connections.
– Test will be done after configuration change.

- For your development convenience, make sure to implement “SOCKET REUSE “.
- Do not consider “FTP constraints in NAT”. We will not test FTP.
- Analyze tcp_relay source(tcp_booster.c) especially client_socket(), redirect() , main() to enhance your understand.
- You are permitted to use “Ksocket” module and reference sources, but must not use other classmates’ source.
- use cscope to traverse&find the API info in kernel source directory.

Stability & service daemon requirements.

As Relay server provides network service in kernel module , it must fulfill several essential requirements.

1. Allocated resources(Memory, socket , ..) should be released when they are not used any more.
2. Your OS must be completely free from the malfunction of your module.
3. Your service module should maintain the consistency.

Submission

1. Due date : Mar 19th 23:45
2. Report (1~2 page)
   1) Spec – package ingredients, manual
   2) Test&development Environment
   3) Bug
      - Some problems which your program has.
   4) Self – evaluation and deficiency
      - Do not evaluate your program better than it deserves. For the poor evaluation points are being deducted as this represents that you did not test your program enough.
      - Penalty entries are assigned(e.g. system reliabilty). To let the tester skip your deficiency is a good strategy to evade the penalty.

3. Sources (submit individual files . Do not zip or tar)
   1) krelay – krelay.c (and other files if any)
   2) tcp_relay
   In case of kernel version, sources should be included in krelay package. Otherwise, user version sources. (do not submit Makefile)

Simple preparation List.

1. Compose virtual network environment with Vmware or UML. You need at least two guest-OS running on two Vmware instances, each for relay server and real server machine.
2. Module service environment: Service start/stop , thread termination , /proc configuration , so forth
3. Kernel thread and thread synchronization or (multiplexing – select , poll) (for the kernel tcp_relay)
4. Understanding of socket channel multiplexing.

References.

1. Project A references
   http://courses.ncsu.edu/csc573/lec/001/prog1.html

2. CSC573_2008Spring_Project_QA.ppt references
   http://courses.ncsu.edu/csc573/lec/001/wrap/CSC573_2008Spring_Project_QA.ppt

3. Using Netfilter hooks (kernel 2.4)
   http://www.topsight.net/article.php?story=2003050621055083

4. tcp_relay server source
   http://courses.ncsu.edu/csc573/lec/001/wrap/tcp_booster-0.1.0.tar.gz

5. cscope tutorial
   http://cscope.sourceforge.net/cscope_vim_tutorial.html