CS60, Lab 4: Emulating a TCP server connection

Sergey Bratus, Spring 2017

Date due: Solutions to this lab will be due on Thursday May 11 at 9pm *(extended)*.

The Task: In this lab, you will use Linux raw sockets to emulate the server side of a TCP connection. Your program will receive raw TCP/IP packets on a given port, and respond with the TCP stream whose payload is taken from a file that contains a recorded response to an HTTP GET request.

External TCP clients such as *netcat*, *wget*, or a web browser should be able to connect to the specified port on your VM and get the stream correctly, even in presence of random packet loss (which you will induce with a Netfilter command).

What to implement: You should implement the following features of TCP for the receiving (server) side of a socket:

- The relevant states and transitions of the TCP State Machine (see, e.g., `http://tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm`);
- TCP retransmission timers for every sent packet (see, e.g., `http://www.tcpipguide.com/free/t_TCPSegmentRetransmissionTimersandtheRetransmission-3.htm`);
- TCP sliding window (see, e.g., `http://www.tcpipguide.com/free/t_TCPSlidingWindowDataTransferandAcknowledgementMech-2.htm`, `http://www.tcpipguide.com/free/t_TCPSlidingWindowDataTransferandAcknowledgementMech-5.htm`).

Your emulator will start in the `LISTEN` state, and will implement (at least) the Responder Sequence parts of the diagram (as linked above). In this lab, don't worry about the Simultaneous Open or Simultaneous Close.

Besides these, you will need to implement the TCP 3-way handshake (cf. `http://www.tcpipguide.com/free/t_TCPConnectionEstablishmentSequenceNumberSynchroniz-2.htm`) and TCP connection termination (cf. `http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm`). Without these, even a lossless TCP connection would not work.

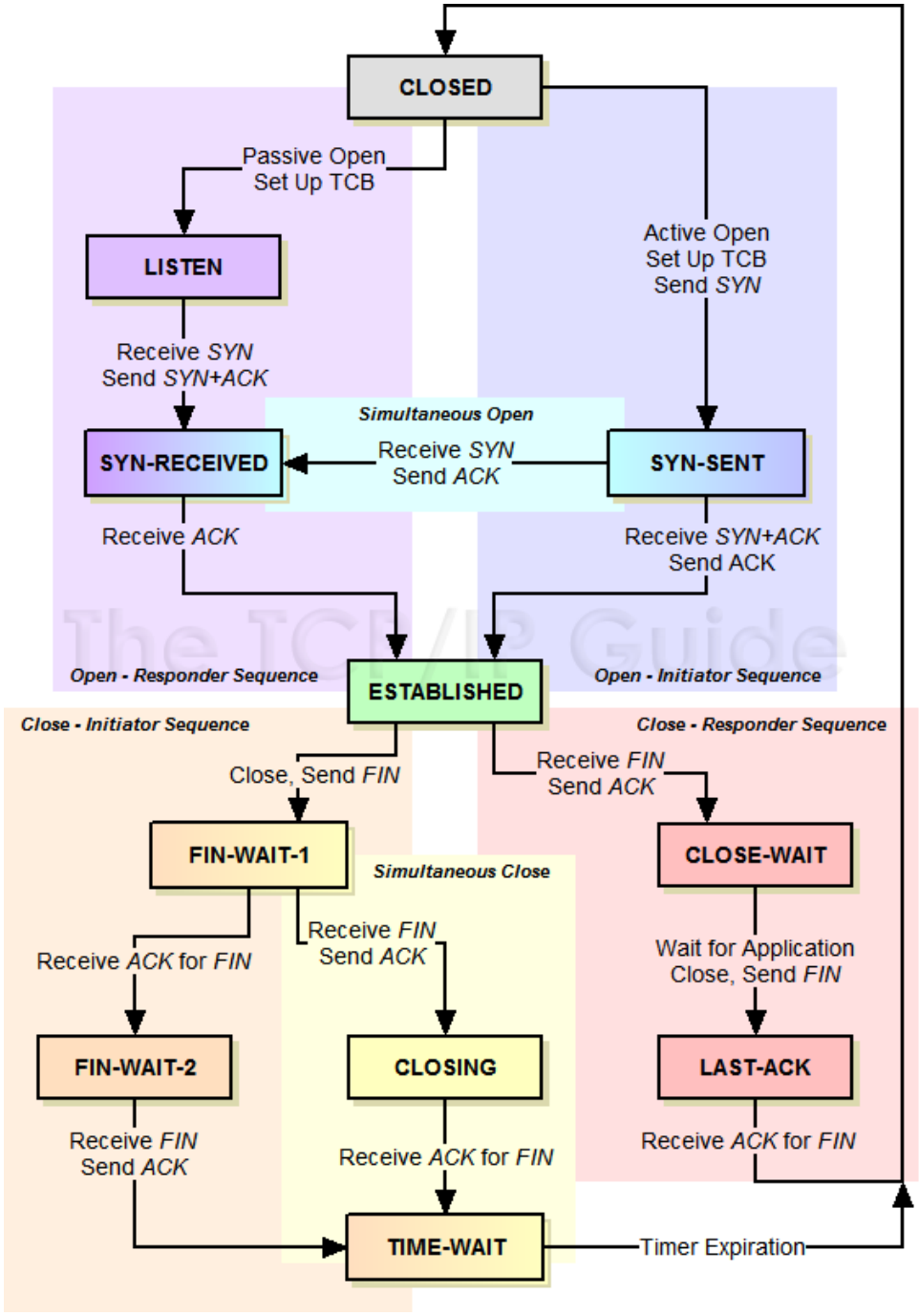
Some code from a naïve "Hello world" emulation in `http://www.cs.dartmouth.edu/~sergey/cs60/lab4/tcp-raw-responder.c` may help, but be sure to read the derogatory comments in that code, and shun what they derogate!

Simplifications: You need not worry about capturing the input your emulated socket receives (although you'll want to ACK it, to avoid unnecessary retransmissions). You may assume that any incoming connection gets the exact same response. You can use the file *http-jpg-response.txt*, which is a recorded response of an Apache web server containing an image of approximately 100Kb.[1]

Blocking RSTs: Your will need to block your VM kernel's responses to the TCP packets that, so far as it knows, do not belong to any connection it has established. Your VM kernel will consider them bogus, and will respond to each with a RST packet, which will break down your external client's connection. To block these RSTs (from just one port), use

```
iptables -A OUTPUT -p tcp --sport 80 --tcp-flags RST RST -j DROP
```

[1] If you run `nc -l 8080 < http-jpg-response.txt` on *tahoe*, and then point your browser at `http://tahoe.cs.dartmouth.edu:8080`, you should get this image displayed in your browser (provided that no one else is using port 8080). Even better: `wget -O img1.jpg http://tahoe.cs.dartmouth.edu:8080` will save this image to your disk; the correct MD5 sum for it is `e91108f934ad5ce5f2d2e245d02a9621`. Vary the host and port as needed.



The TCP Finite State Machine (FSM), Fig. 210 from `http://tcpipguide.com`

where the chosen port is port 80. See notes from Lecture 9 for the meaning of this command.

Simulating Packet Loss: The easiest way to test your code is to run your client on your host system and point it to your VM at *192.168.56.100*; your connection would go via your host's virtual interface *vboxnet0*. This interface has no packet loss, being emulated.

You can, however, induce packet loss on both incoming and outgoing packets with IPtables. Emulating loss of *incoming* packets is easy:

```
# for randomly dropping 10% of incoming packets:
iptables -A INPUT -m statistic --mode random --probability 0.1 -j DROP
```

When you are done testing, delete this rule with

```
iptables -D INPUT  -m statistic --mode random --probability 0.1 -j DROP
```

Emulating loss of *outgoing* packets is a bit more complex. The problem is that when an outgoing packet is dropped by the IPtables firewall on the same machine, the application that sent it gets notified, and `sendto()` returns the `EPERM` error code ("Operation not permitted"). This is annoying, because `DROP` implies just that, silent dropping of packets; this expectation is not fulfilled.

There are two ways around this obstacle. First, you can use the `tc` command, which controls queuing of outgoing packets. One of the queuing algorithms emulates packet loss:

```
tc qdisc add dev eth0 root netem loss 1%
```

To change the amount of packet loss,

```
tc qdisc change dev eth0 root netem loss 5%
```

To disable this algorithm,

```
tc qdisc change del eth0 root netem
```

There is another way, but it requires cooperation from your MacOS PF (and will not work on Windows). The trick is to mark the outgoing packets you want to drop, and then drop them on the MacOS side based on that mark. The mark I tested is setting the Terms-of-Service field of the IP header to `0xA0`, and then dropping all such packets. To mark:

```
# Mark 10% of outgoing TCP packets on port 80 with TOS 0xA0
iptables -t mangle -A OUTPUT -p tcp --sport 80 -m statistic --mode random --probability 0.1
   -j TOS --set-tos 0xa0
```

To clear this rule, use `iptables -t mangle -F`.

Then on the MacOS side, create this simple rule file:

```
# cat drop-tos-0xA0.rules
nat on en0 from 192.168.56.0/24 to any -> (en0)
block in on vboxnet0 all tos 0xa0
```

and load it with `pfctl -f drop-tos-0xA0.rules -e`.[2]

Choosing your Data Structures: You will find that packaging various pieces of related data with your packet makes code much easier to write. For example, the total length of the packet, the start of the IP header in the packet, the start of the TCP header in the packet, the byte sequence number of the packet's payload, the ACK status, and the retransmission timer clearly belong with the packet, and the packets clearly belong in a list of some sort.

The following writeups on the data structures used by Linux for its packets may be useful to you:

[2]Note that some TOS values have reserved meanings that will interfere with your normal connections; the value `0xA0` is not "random". See `https://www.tucny.com/Home/dscp-tos` and `https://en.wikipedia.org/wiki/Type_of_service` for more.

- The SKB structure: `http://vger.kernel.org/~davem/skb.html`
- Handling of a received packet w.r.t. SKBs: `http://vger.kernel.org/~davem/skb_redundancy.html`
- Control blocks (TCBs) for connections: `http://vger.kernel.org/~davem/tcp_skbcb.html`

Also helpful, although not directly relevant to the current tasks:

- Handling of SKB lists: `http://vger.kernel.org/~davem/skb_list.html`
- Handling of payload data: `http://vger.kernel.org/~davem/skb_data.html`
- TCP output engine: `http://vger.kernel.org/~davem/tcp_output.html`

Platform: Your solutions must compile and run in the provided Linux virtual machine (`cs60base` or `cs60mini`. You will *not* be able to develop or run them on CS Unix systems (because you don't have root privileges there). In theory, you could make it work on your MacOS, but the code developed on Darwin/MacOS would need to be substantially changed because the raw socket interfaces are different between the OSes, and behave differently. Ask me for hints on how to make it work on MacOS if you are interested.[3]

Submission: Your solutions must be submitted by checking them into the CS department LabGit system at `https://gitlab.cs.dartmouth.edu/`. In your `cs60` project, create a directory called `lab3` and work there. Don't forget to give access to your project to your section's grader and TAs (they must be able to see your code to grade it!)

Submit: Your `tcp-responder.c`.

Your Virtual Machine: Your VM is the same that you configured for Lab 3, according to the instructions in `http://www.cs.dartmouth.edu/~sergey/cs60/lab3/vm-config/` and *vm-networking.pdf*.

Terms and conditions: You are allowed to use any external materials, printed or electronic. You are allowed to discuss problems and technical/C tricks, but the code you submit must be your own: you are not allowed to copy solutions from other students. Abide by the Honor Code; if in doubt, ask.

Late submissions: Throughout the course, you will get *two* free extensions for a late submission, each of 48 hours after a deadline. Once you've used these up, you will lose 10% points of credit for each late assignment, for each late day.

[3]Essentially, you will need to get raw packets via *libpcap* rather than raw sockets, and then block the kernel's own RST responses with PF, e.g., `block out inet proto tcp all flags R/R`.