Lab 4: Tunneling, Symmetric Client/Servers, and Monitoring

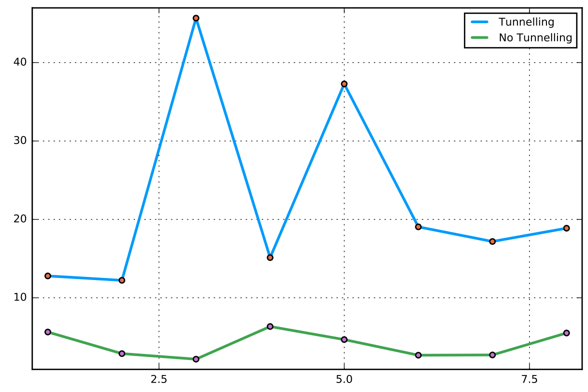
Duc Viet Le
CS536

October 29, 2016

Problem 1.
Testing our code:

- Actual server runs at: `sslab01`
- `Tunneld` runs at: `borg01`
- `Mytunel` and `client` runs at: `Hicks Library`

`myping/mypingd`:
I sent 8 queries using tunneling and not using tunneling. Below is the performance:



	tunnel	no tunnel
1	12.786 ms	5.631 ms
2	12.225 ms	2.875 ms
3	45.676 ms	2.167 ms
4	15.123 ms	6.334 ms
5	37.285 ms	4.665 ms
6	19.046 ms	2.674 ms
7	17.179 ms	2.705 ms
8	18.874 ms	5.505 ms

Discussion: using tunnel increase the ping number which is understandable because instead of directly transmit our UDP packets, we now need to transmit it through another intermediate server (i.e `tunneld`) which will increase time.

`traffic_rcv/traffic_snd`:
I sent 5 queries using tunneling and not using tunneling. Below is the performance:

	tunnel			no tunnel		
	Time	BPS	PPS	Time	BPS	PPS
1	0.109 s	7766275.5	920	0.116 s	7270135.5	861
2	0.108 s	7750229.5	922	0.108 s	7767991.5	920
3	0.108 s	7784829	922	0.109 s	7675029.5	909
4	0.108 s	7781886	927	0.107 s	7855029.5	929
5	0.109 s	7731062	916	0.119 s	7061520.5	836

Discussion: There are not many difference between using tunneling and not using tunneling because the throughtputs from `hicks libary` to `borg` and `sslab01` are similar. There will be no bottleneck node. Also, with tunneling, the result seems to be more stable at receiver, and I think the reason is that connection between `borg` machines and `sslab01` is more stable compared to connection between `hick` machine and `sslab` machines

Problem 2.

Problem 3. Print and inspect all the fields of headers/trailers of the first three Ethernet frames, the headers of IP and UDP packets contained therein, and the first 10 bytes of the UDP payloads. The UDP header is very simple, containing 16-bit length and checksum fields in addition to source and destination port numbers.
Captured package:
- First Package

```
IP (tos 0x0, ttl 64, id 52721, offset 0, flags [none], proto UDP (17), length 4028)
192.168.1.2.35810 > 192.168.1.1.10000: UDP, length 4000
0x0000:  3a56 8f3e 2af8 babc 4274 5230 0800 4500  :V.>*...BtR0..E.
0x0010:  0fbc cdf1 0000 4011 19ec c0a8 0102 c0a8  ......@.........
0x0020:  0101 8be2 2710 0fa8 930d 4c4c 4c4c 4c4c  ....'.....LLLLLL
0x0030:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
0x0040:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
```

- Second Package

```
IP (tos 0x0, ttl 64, id 52722, offset 0, flags [none], proto UDP (17), length 4028)
192.168.1.2.35810 > 192.168.1.1.10000: UDP, length 4000
0x0000:  3a56 8f3e 2af8 babc 4274 5230 0800 4500  :V.>*...BtR0..E.
0x0010:  0fbc cdf2 0000 4011 19eb c0a8 0102 c0a8  ......@.........
0x0020:  0101 8be2 2710 0fa8 930d 4c4c 4c4c 4c4c  ....'.....LLLLLL
0x0030:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
0x0040:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
```

- Third package

```
IP (tos 0x0, ttl 64, id 52723, offset 0, flags [none], proto UDP (17), length 4028)
192.168.1.2.35810 > 192.168.1.1.10000: UDP, length 4000
0x0000:  3a56 8f3e 2af8 babc 4274 5230 0800 4500  :V.>*...BtR0..E.
0x0010:  0fbc cdf3 0000 4011 19ea c0a8 0102 c0a8  ......@.........
0x0020:  0101 8be2 2710 0fa8 930d 4c4c 4c4c 4c4c  ....'.....LLLLLL
0x0030:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
0x0040:  4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c 4c4c  LLLLLLLLLLLLLLLL
```

1. What is the default value of the TTL field observed?
   Ans: The default TTL is 64

2. Use the /bin/ping app to gauge the TTL values from www.purdue.edu, www.cisco.com, and another web site of your choice.
   Ans: Using ping app, we get:
   `www.purdue.edu`: TTL = 250

   ```
   64 bytes from www.purdue.edu (128.210.7.200): icmp_seq=1 ttl=250 time=0.995 ms
   64 bytes from www.purdue.edu (128.210.7.200): icmp_seq=2 ttl=250 time=0.887 ms
   ```

   `www.cisco.com`: TTL = 55

   ```
   a23-79-213-27.deploy.static.akamaitechnologies.com (23.79.213.27) ... ttl=55 ...
   a23-79-213-27.deploy.static.akamaitechnologies.com (23.79.213.27) ... ttl=55 ...
   ```

   `www.stackoverflow.com`: TTL = 57

   ```
   64 bytes from 151.101.129.69: icmp_seq=1 ttl=57 time=7.49 ms
   64 bytes from 151.101.129.69: icmp_seq=2 ttl=57 time=7.10 ms
   ```

3. Do the values equal to the TTL value you observed during sniffing? Check if TTL values can vary across operating systems and protocols.
   Ans. No. TTL during sniffing is 64 while other varies. Also, TTL values vary across OS and protocol. For example, in Windows 7, for IMCP/TCP/UDP, ttl is 128. In Linux/UNIX, TTL is 64 (may vary on different verions of Linux)

4. How might an attacker exploit TTL information from ping or other sources?
   Ans. with knowledge of TTL information, adversary can send lots of packages in a way that those packages will expire at the switch/router. Thus, the switch is forced to generate large amount of ICMP exceed messages. This may caused heavy load on switch, and may cause Deny of Service on the target.

5. Is the TOS field being used in the sniffed IP packets?
   Ans. No. In the sniffed IP package, the TOS field is after Version Number (i.e 4) and IFL (i.e 5), it's `00` which means it was not used.

6. How about the fragmentation fields?.
   Ans. No. Because in our sniffed package, combination of flag field and fragmentation field is `0000`. Therefore, framentation was not used.