Security and Project 3

Edan, Jason, Mark, Monty
Symmetric (Secret Key) Encryption

- Alice and Bob have the same secret key, $K_{AB}$
  - Anyone with the secret key can encrypt/decrypt

Alice

$$\text{Encrypt}$$

Plaintext

I ♥ networks

Secret key $K_{AB}$

Ciphertext

Hi there

Plaintext

I ♥ networks

Decipher

Secret key $K_{AB}$

Bob

Introduction to Computer Networks
Public Key (Asymmetric) Encryption

- Alice and Bob have public/private key pairs \((K_B, K_B^{-1})\)
  - Public keys are well-known, private keys are secret

Alice

Bob's public key

Encrypt

K_B

Plaintext

Hi there

Ciphertext

Decrypt

K_B^{-1}

Plaintext

I ♥ networ

Bob

Bob's private key

ks

ks
Public Key Encryption (2)

- Alice encrypts w/ Bob’s pubkey $K_B$; anyone can send
- Bob decrypts w/ his private key $K_B^{-1}$; only he can
Man-in-the-Middle Attacks

How can we trust a public key?
Certificates

• A certificate binds pubkey to identity, e.g., domain
  • Distributes public keys when signed by a party you trust
  • Commonly in a format called X.509

I hereby certify that the public key
19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A
belongs to
Robert John Smith
12345 University Avenue
Berkeley, CA 94702
Birthday: July 4, 1958
Email: bob@superduper.net.com

Signed by CA
PKI (Public Key Infrastructure)

- Adds hierarchy to certificates to let parties issue
- Issuing parties are called CAs (Certificate Authorities)

I certified the ABC website!
PKI (2)

• Need public key of PKI root and trust in servers on path to verify a public key of website ABC
  • Browser has Root’s public key
  • \{RA1’s key is X\} signed Root
  • \{CA1’s key is Y\} signed RA1
  • \{ABC’s key Z\} signed CA1
PKI (3)

• Browser/OS has public keys of the trusted roots of PKI
  • >100 root certificates!
  • That’s a problem …
  • Inspect your web browser

Certificate for wikipedia.org issued by DigiCert
PKI (4)

• Real-world complication:
  • Public keys may be compromised
  • Certificates must then be revoked

• PKI includes a CRL (Certificate Revocation List)
  • Browsers use to weed out bad keys
<table>
<thead>
<tr>
<th><a href="http://www.google.com">www.google.com</a></th>
<th>GTS CA 1C3</th>
<th>GTS Root R1</th>
<th>GlobalSign Root CA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject Name</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>BE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>GlobalSign nv-sa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational Unit</td>
<td>Root CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common Name</td>
<td>GlobalSign Root CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Issuer Name</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>BE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>GlobalSign nv-sa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational Unit</td>
<td>Root CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common Name</td>
<td>GlobalSign Root CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Validity</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not Before</td>
<td>Tue, 01 Sep 1998 12:00:00 GMT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not After</td>
<td>Fri, 28 Jan 2028 12:00:00 GMT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bufferbloat

“Bufferbloat is a cause of high latency in packet-switched networks caused by excess buffering of packets” – Wikipedia
Bufferbloat – Cause

- Host doesn’t know the bandwidth of the bottleneck link.
- TCP relies solely on packet losses to guide how fast to send.
  - It keeps sending faster and faster until a packet drops.
- With a queue, this can fill up the queue pretty quickly.
Bufferbloat – Problem

- Suppose h1 knows to send at 1.5 Mb/s, what’s the RTT when the queue is full?
  - …when it’s not full?
- TCP at the end of the day will operate at the bottleneck bandwidth, but is it necessary to fill up the queue?
A motivational example…

Not all “speedtests” capture bufferbloat… took a long time for the networking community to realize it was a problem!

- A regular “ping” test, used to measure RTT in practice, won’t fill the buffers!

Let’s as a class try it out:

- [https://www.waveform.com/tools/bufferbloat](https://www.waveform.com/tools/bufferbloat)
  - Loaded latency vs. unloaded latency
  - How big is the difference?
Real-world Initiatives

Active Queue Management (AQM)

- Goal is to use better queue management techniques
- Leverage ECN to give fast feedback without causing loss
  - Unfortunately hard to deploy ECN CC “fairly” with existing CC algos (Reno, Cubic)
  - It works so much more responsively (aka better) it tends to takeover throughput from legacy TCP!

L4S “Low-latency, low-loss, scalable throughput” initiative at IETF

- One solution is to mandate a split at bottlenecks, two queues with independent behavior
- Required in latest cable modem standards
One AQM Technique: FQ_CODEL

- Initiatives to add *flow-independent* queues to bottleneck routers...
  - like L4S to an extreme...
  - Each flow gets its own queue, and it's the router's job to make them all fair!
  - Attempts to estimate bottleneck and not queue any more than necessary to fill the pipe
    - Similar big idea to BBR
- Available in all modern Linux distros (kernel > 3.16)
  - Default in some
- Default in OpenWRT
  - Used as the basis for some commercial routers too (SpaceX Starlink is a prominent example)
- Relatively resource intensive though, so not feasible on "core" routers yet

💡 FQ == “Fair Queue”
CoDel == “Controlled Delay”
Bufferbloat aware transport: BBR

- Different type of solution than AQM
  - Operates only on end hosts
- Developed at Google in 2016 for YouTube traffic.
- Uses a model instead of loss to formulate how fast to send
  - Probe RTT and latency and predict the bottleneck bandwidth.
Project 3 – Goal

- Simulate bufferbloat problem.
- See the worse performance when queue size is larger
- See the difference between TCP Reno and TCP BBR.
Experiment Setup

- Long-lived TCP flow from h1 to h2
  - Simulate background traffic
- Back-to-back ping from h1 to h2
  - Measure RTT
- Spawn a webserver on h1 and periodically fetch a page
  - Simulate more important load
  - Measure time
- Plot time series of RTT and number of queued packets.

- Run the experiment with
  - Q=20 and Q=100
  - Reno and BBR
  - 4 experiments total
In groups of 3-4,...

- In your own words, what is bufferbloat problem?
- For each of the 4 experiments (Q=20 or 100; and with Reno or BBR),
  - How do the webpage fetch time compare?
- How would plot between queue size and time look like for TCP Reno?
Detour – Hypothesize

In groups of 3-4,…

● In your own words, what is bufferbloat problem?
● For each of the 4 experiments (Q=20 or 100; and with Reno or BBR),
  ○ How do the webpage fetch time compare?
● How would plot between queue utilization and time look like for TCP Reno?

<table>
<thead>
<tr>
<th></th>
<th>Q=20</th>
<th>Q=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reno</td>
<td>&lt;:=&gt;</td>
<td>&lt;:=&gt;</td>
</tr>
<tr>
<td>BBR</td>
<td>&lt;;=;&gt;</td>
<td>&lt;;=;&gt;</td>
</tr>
</tbody>
</table>
Setup

- Use Mininet VM (same as Project 2)
- Get the starter code and install dependencies

```bash
cd ~
unzip project3.zip
sudo apt-get update
sudo apt install python3-pip
sudo python3 -m pip install mininet matplotlib
```
Starter Code

- run.sh
  - Run the entire experiment
    - Run bufferbloat.py on q=20 and q=100
    - Generate latency and queue length graphs

- bufferbloat.py
  - Complete the TODOs
    - Setup the mininet topology and the experiment
    - Write shell commands to do the measurements
Long-lived TCP Flow

- Starter code sets up iperf server on h2

- Goal: start iperf client on h1, connect to h2
  - Should be “long-lasting”, i.e. for time specified by --time parameter

- How do I connect to a certain IP or make the connection long-lasting?
  - man pages are your friend!
  - type `man iperf` in a Linux terminal
Ping Train

● Goal: Start “ping train” between h1 and h2
  ○ Pings should occur at 10 per second interval
  ○ Should run for entire experiment

● How do I specify the ping interval and how long the ping train runs?
  ○ man pages are your friend!
  ○ type `man ping` in a Linux terminal

● Write the RTTs recorded from `ping` to `{args.dir}/ping.txt
  ○ See starter code comments for more detail
Download Webpage with curl

- Starter code spawns webserver on h1

- Goal: Use `curl` to measure fetch time to download webpage from h1
  - Starter code has hint on formatting curl command
  - Make sure `curl` doesn't output an error
    - Errors report very small latency

- No need to plot fetch times; just need to report average fetch time for each experiment.
Plotting

● Starter code contains scripts for plotting, `plot_queue.py`, `plot_ping.py`  
  ○ Expects queue occupancy in `$dir/q.txt`, ping latency in `$dir/ping.txt`  
  ○ Plots are useful for debugging!

● Part 3, run same experiments with TCP BBR instead of TCP Reno  
  ○ How do you expect the graph outputs to differ?