Internet Architecture

A review

Lecture 3

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23
Software/hardware organization at hosts

Communication functions broken up and “stacked”

Each layer depends on the one below it.

Each layer supports the one above it.

The interfaces between layers are well-defined and standardized.

Application: useful user-level functions

Transport: provide guarantees to apps

Network: best-effort global pkt delivery

Link: best-effort local pkt delivery
Software/hardware organization at hosts

Application: useful user-level functions
Transport: provide guarantees to apps
Network: best-effort global pkt delivery
Link: best-effort local pkt delivery

Demultiplexing
Reliability
Congestion control
(3) How much data to keep in flight?

- Avoid overwhelming network resources: Congestion control
- Internet: every endpoint makes its own decisions!
  - Distributed algorithm: no central authority
  - Goal 1: efficiency (use available capacity)
  - Goal 2: fairness (distribute capacity equitably)
Finding the right congestion window

• There is an unknown bottleneck link rate that the sender must match

• If sender sends more than the bottleneck link rate:
  • packet loss, delays, etc.

• If sender sends less than the bottleneck link rate:
  • all packets get through; successful ACKs

• Congestion window (cwnd): amount of data in flight
Quickly finding a rate: TCP slow start

- Initially \( cwnd = 1 \) MSS
  - MSS is “maximum segment size”

- Upon receiving an ACK of each MSS, increase the \( cwnd \) by 1 MSS

- Effectively, double \( cwnd \) every RTT

- Initial rate is slow but ramps up \textbf{exponentially fast}

- On loss (RTO), restart from \( cwnd := 1 \) MSS
Behavior of slow start

Congestion Window

1 MSS

Packet drops/
RTO

Slow start

Time
Slow start has problems

• Congestion window **increases too rapidly**
  • Example: suppose the “right” window size $cwnd$ is 17
  • $cwnd$ would go from 16 to 32 and then dropping down to 1
  • Result: massive packet drops

• Congestion window **decreases too rapidly**
  • Suppose the right $cwnd$ is 31, and there is a loss when $cwnd$ is 32
  • Slow start will resume all the way back from $cwnd$ 1
  • Result: unnecessarily low speed of sending data

• Instead, perform finer adjustments of $cwnd$: **congestion avoidance**
TCP New Reno: Additive Increase

- Remember the recent past to find a good estimate of link rate
- The last good cwnd without packet drop is a good indicator
  - TCP New Reno calls this the slow start threshold (ssthresh)

- Increase cwnd by 1 MSS every RTT after cwnd hits ssthresh
  - Effect: increase window additively per RTT
TCP New Reno: Additive increase

- Start with ssthresh = 64K bytes (TCP default)

- Do slow start until ssthresh

- Once the threshold is passed, do additive increase
  - Add one MSS to cwnd for each cwnd worth data ACK’ed
  - For each MSS ACK’ed, cwnd = cwnd + (MSS * MSS) / cwnd

- Upon a TCP timeout (RTO),
  - Set cwnd = 1 MSS
  - Set ssthresh = max(2 * MSS, 0.5 * cwnd)
  - i.e., the next linear increase will start at half the current cwnd
Behavior of Additive Increase

Say $\text{MSS} = 1 \text{ KByte}$
Default $\text{ssthresh} = 64\text{KB} = 64 \text{ MSS}$

Packet drops/ $\text{RTO}$

Loss occurs at $\text{cwnd} = 54\text{K}$

Set $\text{ssthresh}$ to $27\text{ MSS}$

Loss occurs at $\text{cwnd} = 40\text{K}$

Set $\text{ssthresh}$ to $20\text{ MSS}$

Additive increase

AI is slow.
Persistent connections
Large window sizes
Different laws to evolve congestion window
Routing
Two key network-layer functions

• **Forwarding**: move packets from router’s input to appropriate router output

• **Routing**: determine route taken by packets from source to destination
  - routing algorithms

• The network layer solves the routing problem.

Analogy: taking a road trip

- **Forwarding**: process of getting through single exit
- **Routing**: process of planning trip from source to destination

The network layer runs everywhere
Control/Data Planes

Data plane = Forwarding
• local, per-router function
• determines how datagram arriving on router input port is forwarded to router output port

Control plane = Routing
• network-wide logic
• determines how datagram is routed along end-to-end path from source to destination endpoint
• two control-plane approaches:
  • Distributed routing algorithm running on each router
  • Centralized routing algorithm running on a (logically) centralized machine
Application architecture

Web servers
Components of an Internet Service

- Endpoints
- Routers
- Modularized applications
- Servers
- Interconnect: Routers
- Data Center
- Storage
Web server

Often the first app point where a user request lands

Parse HTTP request

```
GET / HTTP/1.1
Host: example.com
```

(many other headers!)

Find a file, run a script, ...

Send response header

```
HTTP/1.1 200 OK
Content-Type: text/html
```

Read file, send() data
Overloaded with functionality

Often the first app point where a user request lands

Find a file, run a script, …

Scripting: Python/PHP/nodejs, fastCGI
Reverse proxy
Caching
Compression
Media streaming
Access control
Image filtering

bind(IPaddr_B, port_B)
listen()
accept()
recv()/send()/..
How does one design a web server?

• Process connections one at a time?

Many other requests waiting in the meanwhile

Powerful server doing nothing most of the time
How does one design a web server?

- Process other requests while waiting for one to finish
Parallelism

- Process other requests while waiting for one to finish
- A first design: multiprocessing/threading

Great to avoid blocking (disk I/O, fastCGI, …)

Overhead grows with # connections

More longer lived
Concurrency

• Process other requests while waiting for one to finish
• A better design: event driven

Lightweight
Can block if any of the requests block

State of the art designs combine parallelism (multiprocess/thread) with concurrency (event-driven)

epoll, select, kqueue, etc.

bind(IPaddr_B, port_B)
listen()
accept()
recv() / send() / ..