Last week on Communication Networks

Congestion control aims at solving three problems:

1. Bandwidth estimation: How to adjust the bandwidth of a single flow to the bottleneck bandwidth? It could be 1 Mbps or 1 Gbps...
2. Bandwidth adaptation: How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
3. Fairness: How to share bandwidth “fairly” among flows, without overloading the network.

The sender adapts its sending rate based on two windows:

- Receiving Window (RWND): How many bytes can be sent without overflowing the receiver buffer? Based on the receiver input.
- Congestion Window (CWND): How many bytes can be sent without overflowing the routers? Based on network conditions.
- Sender Window: minimum(CWND, RWND)

The 2 key mechanisms of Congestion Control:

- Detecting congestion
- Reacting to congestion
The 2 key mechanisms of Congestion Control

- Detecting
- Reacting to

Detecting losses can be done using ACKs or timeouts, the two signal differ in their degree of severity

- Duplicated ACKs: mild congestion signal
  - Packets are still making it
- Timeout: severe congestion signal
  - Multiple consequent losses

Initially, you want to quickly get a first-order estimate of the available bandwidth

- Intuition: Start slow but rapidly increase until a packet drop occurs
- Increase policy:
  - \( cwnd = 1 \)
  - \( cwnd += 1 \)
  - Initially: \( cwnd = 1 \)
  - Upon receipt of an ACK

Then, you want to "oscillate" around the estimate ensuring fairness along the way

- Increase behavior: gentle/gentle
- Decrease behavior: aggressive/gentle

Congestion control makes TCP throughput look like a "sawtooth"

The DNS system is a distributed database which enables to resolve a name into an IP address

- Name
  - DNS
  - IP address

- www.ethz.ch
  - 129.132.19.216

- google.ch
  - 172.217.16.131
To scale, DNS adopt three intertwined hierarchies:

- **naming structure**: hierarchy of addresses
  - [https://www.ee.ethz.ch/de/departement/](https://www.ee.ethz.ch/de/departement/)
- **management**: hierarchy of authority over names
- **infrastructure**: hierarchy of DNS servers

---

**This week on Communication Networks**

---

The Web as we know it was founded in ~1990, by Tim Berners-Lee, physicist at CERN

- **His goal**: provide distributed access to data
- **The World Wide Web (WWW)**: a distributed database of "pages" linked together via the Hypertext Transport Protocol (HTTP)

---

The Web was and still is so successful as it enables everyone to self-publish content

- **Self-publishing on the Web** is easy, independent & free and accessible, to everyone
- **People weren't looking for technical perfection**
- **little interest in collaborative or idealistic endeavor**
- **People essentially want to make their mark** and find something neat…

---

The WWW is made of three key components:

- **Infrastructure**
- **Content**
- **Implementation**

---

- **Clients/Browser**
- **Servers**
- **Proxies**
- **Objects**
  - Files, pictures, videos, …
  - organized in
  - Web sites
  - a collection of objects
- **URL**: name content
- **HTTP**: transport content
We’ll focus on its implementation.

A Uniform Resource Locator (URL) refers to an Internet resource.

protocol://hostname:port/directory_path/resource

default to protocol’s standard
HTTP:80, HTTPS:443

identify the resource on the destination

protocol://hostname:port/directory_path/resource

DNS Name
IP address
HTTP is a rather simple synchronous request/reply protocol

HTTP is layered over a bidirectional byte stream typically TCP, but QUIC is ramping up

HTTP is text-based (ASCII) human readable, easy to reason about

HTTP is stateless it maintains no info about past client requests

HTTP clients make request to the server

<table>
<thead>
<tr>
<th>method</th>
<th>GET</th>
<th>return resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEAD</td>
<td>return headers only</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>send data to server (forms)</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>relative to server (e.g., /index.html)</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>1.0, 1.1, 2.0</td>
<td></td>
</tr>
</tbody>
</table>

Requests headers are of variable lengths, but still, human readable

<table>
<thead>
<tr>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization info</td>
</tr>
<tr>
<td>Acceptable document types/encoding</td>
</tr>
<tr>
<td>From (user email)</td>
</tr>
<tr>
<td>Host (identify the server to which the request is sent)</td>
</tr>
<tr>
<td>If-Modified-Since</td>
</tr>
<tr>
<td>Referrer (cause of the request)</td>
</tr>
<tr>
<td>User Agent (client software)</td>
</tr>
</tbody>
</table>
Recall that multiple DNS names can map to the same IP address:

<table>
<thead>
<tr>
<th>name</th>
<th>DNS</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.ethz.ch">www.ethz.ch</a></td>
<td>129.132.19.216</td>
<td></td>
</tr>
<tr>
<td>vanbever.eu</td>
<td>82.130.102.71</td>
<td></td>
</tr>
<tr>
<td>route-aggregation.net</td>
<td>82.130.102.71</td>
<td></td>
</tr>
<tr>
<td>comm-net.ethz.ch</td>
<td>82.130.102.71</td>
<td></td>
</tr>
</tbody>
</table>

The "Host" header indicates the server (82.130.102.71) the desired domain name (this is known as virtual hosting):

```
GET / HTTP/1.1
Host: comm-net.ethz.ch
```

HTTP servers answers to clients’ requests:

```
HTTP/1.1 200 OK
Date: Fri, 01 May 2020 08:36:56 GMT
Server: Apache/2.4.18 (Ubuntu)

<title>Communication Networks 2020</title>
```

Virtual hosting enables one IP address to host multiple websites:

```
GET / HTTP/1.1
Host: vanbever.eu
```

HTTP servers answers to clients’ requests:

```
HTTP/1.1 200 OK
Date: Fri, 01 May 2020 08:44:26 GMT
Server: Apache/2.4.18 (Ubuntu)

<title>Laurent Vanbever</title>
```
3 digit response code | reason phrase
--- | ---
1XX | informational
2XX | success
3XX | redirection
4XX | client error
5XX | server error

<table>
<thead>
<tr>
<th>version</th>
<th>status</th>
<th>phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200</td>
<td>OK</td>
</tr>
</tbody>
</table>
| 301    | Moved Permanently
| 303    | Moved Temporarily
| 404    | Not Found
| 505    | Not Supported

### Like request headers, response headers are of variable lengths and human-readable

- **Uses**: Location (for redirection), Allow (list of methods supported), Content encoding (e.g., gzip), Content-Length, Content-Type, Expires (caching), Last-Modified (caching)

### HTTP is a stateless protocol, meaning each request is treated independently

**Advantages**
- Server-side scalability
  - Some applications require state!
  - (Shopping cart, user profiles, tracking)

**Disadvantages**
- Failure handling is trivial

How can you maintain state in a stateless protocol?

HTTP makes the client maintain the state. This is what the so-called cookies are for!

- Client stores small state on behalf of the server
- Client sends state in all future requests to the server
- Can provide authentication

Performance goals vary depending on who you ask

<table>
<thead>
<tr>
<th>User</th>
<th>Network operators</th>
<th>Content provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wish</td>
<td>Fast downloads, high availability</td>
<td>No overload</td>
</tr>
<tr>
<td>Solution</td>
<td>Improve HTTP to compensate for TCP weakspots</td>
<td>Caching and Replication</td>
</tr>
</tbody>
</table>
Improve HTTP to compensate for TCP weak spots

User
- fast downloads
- high availability

solution
- Improve HTTP to compensate for TCP weak spots

Client
- SYN
- SYN/ACK
- ACK + HTTP GET

Server
- Establish connection
- Request response
- Close connection

Relying on TCP forces a HTTP client to open a connection before exchanging anything

Most Web pages have multiple objects, naive HTTP opens one TCP connection for each…

Fetching n objects requires ~2n RTTs

TCP establishment
HTTP request/response

One solution to that problem is to use multiple TCP connections in parallel

User
- Happy!

Content provider
- Happy!

Network operator
- Not Happy!
- Why?

Another solution is to use persistent connections across multiple requests (the default in HTTP/1.1)

- Avoid overhead of connection set-up and teardown
  clients or servers can tear down the connection

- Allow TCP to learn more accurate RTT estimate
  and with it, more precise timeout value

- Allow TCP congestion window to increase
  and therefore to leverage higher bandwidth

Yet another solution is to pipeline requests & replies asynchronously, on one connection

- batch requests and responses to reduce the number of packets
- multiple requests can be packed into one TCP segment

Considering the time to retrieve n small objects, pipelining wins

<table>
<thead>
<tr>
<th># RTTS</th>
<th>one-at-a-time</th>
<th>~2n</th>
</tr>
</thead>
<tbody>
<tr>
<td>M concurrent</td>
<td>~2n/M</td>
<td></td>
</tr>
<tr>
<td>persistent</td>
<td>~n</td>
<td></td>
</tr>
<tr>
<td>pipelined</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Considering the time to retrieve n big objects, there is no clear winner as bandwidth matters more

<table>
<thead>
<tr>
<th># RTTS</th>
<th>~n * avg. file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth</td>
<td></td>
</tr>
</tbody>
</table>
The average webpage size nowadays is 2.3 MB as much as the original DOOM game...

Top web sites have decreased in size though because they care about TCP performance

Caching leverages the fact that highly popular content largely overlaps

Yet, a significant portion of the HTTP objects are “uncachable”

To limit staleness of cached objects, HTTP enables a client to validate cached objects

Caching can be (and is) performed at different locations

Many clients request the same information
This increases servers and network’s load, while clients experience unnecessary delays.

Reverse proxies cache documents close to servers, decreasing their load. This is typically done by content provider.

Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies. This is typically done by ISPs or enterprises.

The idea behind replication is to duplicate popular content all around the globe.

The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica.

Akamai is one of the largest CDNs in the world, boasting servers in more than 20,000 locations.

Akamai uses a combination of:
- **pull caching**: direct result of clients requests
- **push replication**: when expecting high access rate
  - together with some dynamic processing
    - dynamic Web pages, transcoding,...