Congestion control aims at solving three problems:

1. **bandwidth estimation**: How to adjust the bandwidth of a single flow to the bottleneck bandwidth? (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.

Congestion control differs from flow control both are provided by TCP though:

- **Flow control**: prevents one fast sender from overloading a slow receiver
- **Congestion control**: prevents a set of senders from overloading the network

The sender adapts its sending rate based on these two windows:

- **Receiving Window (RWND)**: How many bytes can be sent without overloading the receiver buffer? based on the receiver input
- **Congestion Window (CWND)**: How many bytes can be sent without overloading the routers? based on network conditions
- **Sender Window**: minimum(CWND, RWND)
The 2 key mechanisms of Congestion Control

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

TCP approach is to **gently increase** when not congested and to **rapidly decrease** when congested

- **question**: What increase/decrease function should we use?
  - it depends on the problem we are solving...

Congestion control aims at solving three problems

- **#1 bandwidth estimation**: How to adjust the bandwidth of a single flow to the bottleneck bandwidth?
  - 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

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 initially
  - **upon receipt of an ACK** cwnd += 1
#2 bandwidth adaptation
How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?

<table>
<thead>
<tr>
<th></th>
<th>increase behavior</th>
<th>decrease behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIAD</td>
<td>gentle</td>
<td>gentle</td>
</tr>
<tr>
<td>AIMD</td>
<td>gentle</td>
<td>aggressive</td>
</tr>
<tr>
<td>MIAD</td>
<td>aggressive</td>
<td>gentle</td>
</tr>
<tr>
<td>MIMD</td>
<td>aggressive</td>
<td>aggressive</td>
</tr>
</tbody>
</table>

AIMD converge to fairness and efficiency, it then fluctuates around the optimum (in a stable way)

#3 fairness
How to share bandwidth “fairly” among flows, without overloading the network

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

<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></td>
<td>129.132.19.216</td>
</tr>
</tbody>
</table>

To scale, DNS adopt three intertwined hierarchies

- Naming structure: addresses are hierarchical
  - www.ee.ethz.ch
- Management: hierarchy of authority over names
- Infrastructure: hierarchy of DNS servers
A name, e.g., ee.ethz.ch, represents a leaf-to-root path in the hierarchy

The DNS system is hierarchically administered

13 root servers (managed professionally) serve as root (*)

The bottom (and bulk) of the hierarchy is managed by Internet Service Provider or locally

Every server knows the address of the root servers (*) required for bootstrapping the systems

(*) see: http://www.root-servers.org/

(*) see: https://www.internic.net/domain/named.root
Each server knows the address of all children.

**This week on**
Communication Networks

### DNS

| google.ch → 172.217.16.131 |
| [http://www.google.ch](http://www.google.ch) |

(input)

### Web

| google.ch → 172.217.16.131 |
| [starting from slide 47/90](http://www.google.ch) |

(output)

---

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

<table>
<thead>
<tr>
<th>Infrastructure</th>
<th>Content</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clients/Browser</td>
<td>Objects</td>
<td>URL: name content</td>
</tr>
<tr>
<td>Servers</td>
<td>Files, pictures, videos, …</td>
<td>HTTP: transport content</td>
</tr>
<tr>
<td>Proxies</td>
<td>organized in</td>
<td></td>
</tr>
<tr>
<td>Web sites</td>
<td>a collection of objects</td>
<td></td>
</tr>
</tbody>
</table>
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
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

HTTP request

- method
- URL
- version

- header field name: value
- header field name: value
- header field name: value

body

method GET return resource
HEAD return headers only
POST send data to server (forms)

URL relative to server (e.g., /index.html)
version 1.0, 1.1, 2.0

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

Uses
- Authorization info
- Acceptable document types/encoding
- From (user email)
- Host (identify the server to which the request is sent)
- If-Modified-Since
- Referrer (cause of the request)
- User-Agent (client software)
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)

<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>

Virtual hosting enables one IP address to host multiple websites

HTTP servers answers to clients’ requests
## Communication Networks

### Status

<table>
<thead>
<tr>
<th>Status</th>
<th>3 digit response code</th>
<th>reason phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1XX</td>
<td>informational</td>
<td></td>
</tr>
<tr>
<td>2XX</td>
<td>success</td>
<td>200 OK</td>
</tr>
<tr>
<td>3XX</td>
<td>redirection</td>
<td>301 Moved Permanently</td>
</tr>
<tr>
<td></td>
<td></td>
<td>302 Found</td>
</tr>
<tr>
<td></td>
<td></td>
<td>303 Moved Temporarily</td>
</tr>
<tr>
<td>4XX</td>
<td>client error</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>404 Not Found</td>
</tr>
<tr>
<td>5XX</td>
<td>server error</td>
<td>500 Internal Server Error</td>
</tr>
</tbody>
</table>

### 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

#### disadvantages
- some applications need state! (shopping cart, user profiles, tracking)
- 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 X
- client sends state in all future requests to X
- can provide authentication

### Performance goals vary depending on who you ask

- **User**
  - fast downloads
  - high availability
- **Network operators**
  - no overload
- **Content provider**
  - happy users
  - cost-effective infrastructure

- **wish**
  - fast downloads
  - high availability
- **solution**
  - Improve HTTP to compensate for TCP weakspots
  - Caching and Replication

### Performance Protocol

- Protocol: http://
- Performance: Speed 80

### telnet google.ch 80

**request**

GET / HTTP/1.1
Host: www.google.ch

**answer**

HTTP/1.1 200 OK
Date: Sun, 01 May 2016 14:10:30 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Server: gws

Set-Cookie: NID=79=qluqjRTq_BCGHSTFfEyj1G7VitSnCQ1vysTJ26O8s3xy9xywmg2Dashxq1b8fLxjohN5c7jmCA5TFnBY7aW5hjRmQmV1x7T8iCD0xJiCL0mvyYBkqj8X4Nw9AOzE. expires=Mon, 31-Oct-2016 14:10:30 GMT, path=/, domain=.google.ch; HttpOnly
Most Web pages have multiple objects, naive HTTP opens one TCP connection for each…

Fetching \( n \) objects requires \( \approx 2n \) RTTs

TCP establishment
HTTP request/response

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

Considering the time to retrieve \( n \) small objects, pipelining wins

\[
\begin{align*}
\text{# RTTs} & \\
\text{one-at-a-time} & \approx 2n \\
\text{M concurrent} & \approx 2n/M \\
\text{persistent} & \approx n+1 \\
\text{pipelined} & 2 \\
\end{align*}
\]

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

\[
\begin{align*}
\text{# RTTs} & \\
\approx n \times \text{avg. file size} \\
\text{bandwidth} &
\end{align*}
\]
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 and is performed at different locations

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

Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies.

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

Spreads load on server
E.g., across multiple data-centers
Places content closer to clients
Only way to beat the “speed-of-light”
Helps speeding up uncachable content
Still have to pull it, but from closer

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, …
“Akamaizing” content is easily done by modifying content to reference the Akamai’s domains

Akamai creates domain names for each client
a128.g.akamai.net for cnn.com

Client modifies its URL to refer to Akamai’s domain
http://www.cnn.com/image-of-the-day.gif becomes
http://a128.g.akamai.net/image-of-the-day.gif

Requests are now sent to the CDN infrastructure