TSIN02: Internetworking

Lecture 9: Streaming video

Guest Instructor: Niklas Carlsson
Email: niklas.carlsson@liu.se
Web: http://www.ida.liu.se/~nikca/

These notes have been derived and adapted based on various online resources, including slides from Anirban Mahanti, Carey Williamson, and “Computer Networking: A Top Down Approach”, by Jim Kurose and Keith Ross, Addison-Wesley.
Today’s service/company landscape include ...
Today’s service/company landscape include ...
Today’s service/company landscape include ...

Equipment manufacturers (also sell services and help Operate networks)
Today’s service/company landscape include ...

Network operators

Equipment manufacturers (also sell services and help Operate networks)
Today’s service/company landscape include ...

Enterprise solutions and network service (e.g., data center solutions and cloud providers)
Today’s service/company landscape include ... 

Content delivery networks

Enterprise solutions and network service (e.g., data center solutions and cloud providers)
Today’s service/company landscape include ... 

End user services (e.g., web-based social networks, search, communication, and streaming)
Some common applications today ...

- World Wide Web (WWW)
- Remote login (telnet, rlogin, ssh)
- File transfer
- Peer-to-peer file sharing
- Cloud computing/services
- Instant messaging (chat, text messaging, etc.)
- Live and video-on-demand streaming
- Internet phone (Voice-Over-IP)
- Distributed games
... and tomorrow

The 2020 vision

- Everything that can be connected will be connected
  - 50B devices (perhaps more like 500B ...)

- IoT and smart cities
  - Machine-to-machine

- High-definition 3D streaming to heterogeneous clients
Requirements and quality of service

- **Quality of Service (QoS)**
  - Real-time requirements (e.g., latency, jitter)
  - Loss/stall requirements (e.g., drop rates, late packets)
  - Bandwidth requirements (e.g., throughput)
  - Service availability

- **Quality of Experience (QoE)**
  - Measure of the users quality of experience
  - Multimedia: Most negatively effected by stalls
Requirements and quality of service

Quality of Service (QoS)
- Real-time requirements (e.g., latency, jitter)
- Loss/stall requirements (e.g., drop rates, late packets)
- Bandwidth requirements (e.g., throughput)
- Service availability

Quality of Experience (QoE)
- Measure of the users quality of experience
- Multimedia: Most negatively effected by stalls
Scalable Content Delivery

- Use of Internet for content delivery is massive ... and becoming more so (e.g., majority of all IP traffic is video streaming content)

- Variety of approaches: HTTP-based Adaptive Streaming (HAS), broadcast/multicast, batching, replication/caching (e.g. CDNs), P2P, peer-assisted, ...

- In these slides, we only provide a few high-level examples
Service models

- Client-server (one-to-one)
- Peer-to-peer (machines can act as both client and server)
- Multicast/broadcast (one-to-many and many-to-many)
  - Application layer, IP-based, and down at the MAC-layer
- ISP-based caching, CDNs, cloud, and other third-party solutions
**Client-server architecture**

Client/server model has well-defined roles.

**Server:**
- always-on host
- permanent IP address
- server farms for scaling

**Clients:**
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
Pure P2P architecture

No fixed clients or servers: Each host can act as both client and server at any time

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
Additional Multimedia Support

Multicast/Broadcast

Source-duplication versus in-network duplication. (a) source duplication, (b) in-network duplication

Also, application-layer multicast …
Evolved Multimedia Broadcast/Multicast Service (eMBMS) in LTE-advanced

- Separation of control plane and data plane

Evolved Multimedia Broadcast/Multicast Service (eMBMS) in LTE-advanced

- MBMSFN and use of services areas

Content distribution networks (CDNs)

**Content replication**
- Replicate content at hundreds of servers throughout Internet (often in edge/access network)
- Content “close” to user reduce impairments (loss, delay) of sending content over long paths
Content distribution networks (CDNs)

Content replication

- CDN (e.g., Akamai, Limewire) customer is the content provider (e.g., CNN)
- Other companies build their own CDN (e.g., Google)
- CDN replicates customers’ content in CDN servers.
- When provider updates content, CDN updates servers

Diagram:
- Origin server in North America
- CDN distribution node
- CDN server in S. America
- CDN server in Europe
- CDN server in Asia
**CDN: “simple” content access scenario**

Bob (client) requests video http://video.netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V

2. resolve http://video.netcinema.com/6Y7B23V via Bob’s local DNS
3. netcinema’s DNS returns URL http://KingCDN.com/NetC6y&B23V
4&5. Resolve http://KingCDN.com/NetC6y&B23 via KingCDN’s authoritative DNS, which returns IP address of KingCDN server with video
6. request video from KINGCDN server, streamed via HTTP
Multimedia Networking Applications

Classes of MM applications:
Multimedia Networking Applications

Classes of MM applications:
1) Streaming stored audio and video
2) Streaming live audio and video
3) Real-time interactive audio and video
Consider first ... Streaming Stored Multimedia

application-level streaming techniques for making the best out of best effort service:
- client-side buffering
- use of UDP versus TCP
- multiple encodings of multimedia

Media Player
- jitter removal
- decompression
- error concealment
- graphical user interface w/ controls for interactivity
Internet multimedia: simplest approach

- audio or video stored in file
- files transferred as HTTP object
  - received in entirety at client
  - then passed to player

Audio, video is downloaded, not streamed:
- long delays until playout, since no pipelining!
Progressive Download

- browser retrieves **metafile** using HTTP GET
- browser launches player, passing metafile to it
- media player contacts server directly
- server *downloads* audio/video to player
Streaming from a Streaming Server

- This architecture allows for non-HTTP protocol between server and media player.
- Can also use UDP instead of TCP.
Streaming Multimedia: client rate(s)

**Q:** how to handle different client receive rate capabilities?

- 28.8 Kbps dialup
- 100 Mbps Ethernet
**Streaming Multimedia: client rate(s)**

**Q:** how to handle different client receive rate capabilities?
- 28.8 Kbps dialup
- 100 Mbps Ethernet

**A1:** server stores, transmits multiple copies of video, encoded at different rates

**A2:** layered and/or dynamically rate-based encoding
Streaming Multimedia: UDP or TCP?

**UDP**
- server sends at rate appropriate for client (oblivious to network congestion!)
  - often send rate = encoding rate = constant rate
  - then, fill rate = constant rate - packet loss
- short playout delay (2-5 seconds) to compensate for network delay jitter
- error recover: time permitting

**TCP**
- send at maximum possible rate under TCP
- fill rate fluctuates due to TCP congestion control
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls
Fairness of UDP Streams (1/2)

- R1-R2 is the bottleneck link
- Streaming uses UDP at the transport layer; requested media encoded at 1 Mbps
- What fraction of the bottleneck is available to FTP?

Credit: MSc thesis work by Sean Boyden (2006)
Fairness of RealVideo Streams (2/2)
A protocol family for streaming

- RTSP
- RTP
- RTCP
RTSP: out-of-band control

RTSP messages sent out-of-band:

- RTSP control messages use different port numbers than media stream: out-of-band.
  - port 554
- media stream is considered “in-band”.

RTSP Example

Scenario:

- metafile communicated to web browser
- browser launches player
- player sets up an RTSP control connection, data connection to streaming server
Metafile Example

<title>Twister</title>
<session>
  <group language=en lipsync>
    <switch>
      <track type=audio
        e="PCMU/8000/1"
        src = "rtsp://audio.example.com/twister/audio.en/lofi">
      <track type=audio
        e="DVI4/16000/2" pt="90 DVI4/8000/1"
        src="rtsp://audio.example.com/twister/audio.en/hifi">
        </switch>
      <track type="video/jpeg"
        src="rtsp://video.example.com/twister/video">
    </group>
  </session>
RTSP Operation

HTTP GET
presentation desc.

setup
play
pause
teardown

Web browser
Web server
media player
media server
client
server
Real-Time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RFC 3550
- RTP runs in end systems
- RTP packets encapsulated in UDP segments
RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:
- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping
Payload Type (7 bits): Indicates type of encoding currently being used. If sender changes encoding in middle of conference, sender informs receiver via payload type field.

- Payload type 0: PCM mu-law, 64 kbps
- Payload type 3, GSM, 13 kbps
- Payload type 7, LPC, 2.4 kbps
- Payload type 26, Motion JPEG
- Payload type 31, H.261
- Payload type 33, MPEG2 video

Sequence Number (16 bits): Increments by one for each RTP packet sent, and may be used to detect packet loss and to restore packet sequence.
RTP Header (2)

- **Timestamp field (32 bytes long):** sampling instant of first byte in this RTP data packet
  - for audio, timestamp clock typically increments by one for each sampling period (for example, each 125 usecs for 8 KHz sampling clock)
  - if application generates chunks of 160 encoded samples, then timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.

- **SSRC field (32 bits long):** identifies source of t RTP stream. Each stream in RTP session should have distinct SSRC.
Real-time Control Protocol (RTCP)

Receiver report packets:
- fraction of packets lost, last sequence number, average interarrival jitter

Sender report packets:
- SSRC of RTP stream, current time, number of packets sent, number of bytes sent
- RTCP attempts to limit its traffic to 5% of session bandwidth

feedback can be used to control performance
Synchronization of Streams

- RTCP can synchronize different media streams within a RTP session.
- Consider videoconferencing app for which each sender generates one RTP stream for video, one for audio.
- Timestamps in RTP packets tied to the video, audio sampling clocks
  - *not* tied to wall-clock time
- Each RTCP sender-report packet contains (for most recently generated packet in associated RTP stream):
  - Timestamp of RTP packet
  - Wall-clock time for when packet was created.
- Receivers use association to synchronize playout of audio, video.
RTCP Bandwidth Scaling

- RTCP attempts to limit its traffic to 5% of session bandwidth.

**Example**
- Suppose one sender, sending video at 2 Mbps. Then RTCP attempts to limit its traffic to 100 Kbps.
- RTCP gives 75% of rate to receivers; remaining 25% to sender

- 75 kbps is equally shared among receivers:
  - with R receivers, each receiver gets to send RTCP traffic at 75/R kbps.
- sender gets to send RTCP traffic at 25 kbps.
- participant determines RTCP packet transmission period by calculating avg RTCP packet size (across entire session) and dividing by allocated rate
HTTP-based streaming

- Allows easy caching, NAT/firewall traversal, etc.
- Use of TCP provides natural bandwidth adaptation
- Split into fragments, download sequentially
- Some support for interactive VoD
HTTP-based adaptive streaming (HAS)

- Multiple encodings of each fragment (defined in manifest file)
- Clients adapt quality encoding based on (buffer and network) conditions
Chunk-based streaming

- Chunks begin with keyframe so independent of other chunks
- Playing chunks in sequence gives seamless video
- Hybrid of streaming and progressive download:
  - Stream-like: sequence of small chunks requested as needed
  - Progressive download-like: HTTP transfer mechanism, stateless servers
**Example**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Quality</td>
<td>Quality</td>
</tr>
<tr>
<td>1,4</td>
<td>1,4</td>
</tr>
<tr>
<td>1,3</td>
<td>2,4</td>
</tr>
<tr>
<td>1,2</td>
<td>3,3</td>
</tr>
<tr>
<td>1,1</td>
<td>3,2</td>
</tr>
<tr>
<td>2,2</td>
<td>2,3</td>
</tr>
<tr>
<td>2,1</td>
<td>2,2</td>
</tr>
<tr>
<td>3,1</td>
<td>3,3</td>
</tr>
<tr>
<td>4,1</td>
<td>4,2</td>
</tr>
<tr>
<td>5,1</td>
<td>5,2</td>
</tr>
<tr>
<td>6,1</td>
<td>6,2</td>
</tr>
<tr>
<td>7,1</td>
<td>7,2</td>
</tr>
</tbody>
</table>
Example
Example

```
Client
<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>2,4</td>
</tr>
<tr>
<td>1,3</td>
<td>3,4</td>
</tr>
<tr>
<td>1,2</td>
<td>3,3</td>
</tr>
<tr>
<td>1,1</td>
<td>2,2</td>
</tr>
</tbody>
</table>

Server
<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>3,4</td>
</tr>
<tr>
<td>1,3</td>
<td>4,4</td>
</tr>
<tr>
<td>1,2</td>
<td>5,4</td>
</tr>
<tr>
<td>1,1</td>
<td>6,4</td>
</tr>
</tbody>
</table>
```

@1300 Kbit/s
@850 Kbit/s
@500 Kbit/s
@250 Kbit/s
## Example

**Client**

<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>1,4</td>
<td>3,4</td>
<td>1,3</td>
<td>1,2</td>
<td>1,1</td>
</tr>
<tr>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>1,1</td>
<td>2,2</td>
<td>2,1</td>
</tr>
<tr>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>2,2</td>
<td>3,2</td>
<td>3,1</td>
</tr>
<tr>
<td>1,2</td>
<td>1,2</td>
<td>2,2</td>
<td>1,1</td>
<td>1,2</td>
<td>1,1</td>
</tr>
<tr>
<td>1,1</td>
<td>1,1</td>
<td>1,1</td>
<td>1,1</td>
<td>2,2</td>
<td>2,2</td>
</tr>
</tbody>
</table>

**Server**

<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>1,4</td>
<td>3,4</td>
<td>1,3</td>
<td>1,2</td>
<td>1,1</td>
</tr>
<tr>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>1,1</td>
<td>2,2</td>
<td>2,1</td>
</tr>
<tr>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>2,2</td>
<td>3,2</td>
<td>3,1</td>
</tr>
<tr>
<td>1,2</td>
<td>1,2</td>
<td>2,2</td>
<td>1,1</td>
<td>1,2</td>
<td>1,1</td>
</tr>
<tr>
<td>1,1</td>
<td>1,1</td>
<td>1,1</td>
<td>1,1</td>
<td>2,2</td>
<td>2,2</td>
</tr>
</tbody>
</table>
Example
**Example**

<table>
<thead>
<tr>
<th>Quality</th>
<th>1,1</th>
<th>2,1</th>
<th>3,1</th>
<th>4,1</th>
<th>5,1</th>
<th>6,1</th>
<th>7,1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>5,4</td>
<td>6,4</td>
<td>7,4</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>6,3</td>
<td>7,3</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
<td>6,2</td>
<td>7,2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
<td>6,1</td>
<td>7,1</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality</th>
<th>1,1</th>
<th>2,1</th>
<th>3,1</th>
<th>4,1</th>
<th>5,1</th>
<th>6,1</th>
<th>7,1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>5,4</td>
<td>6,4</td>
<td>7,4</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>6,3</td>
<td>7,3</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
<td>6,2</td>
<td>7,2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
<td>6,1</td>
<td>7,1</td>
<td>...</td>
</tr>
</tbody>
</table>

- Client: Laptop
- Server: Server

@1300 Kbit/s
@850 Kbit/s
@500 Kbit/s
@250 Kbit/s

**Time**
Example
### Example

- **Client**
  - Quality: 1,4, 2,4, 3,4, 4,4, 5,4, 6,4, 7,4
  - Time: 1, 2, 3, 4, 5, 6, 7

- **Server**
  - Quality: 1,4, 2,4, 3,4, 4,4, 5,4, 6,4, 7,4
  - Time: 1, 2, 3, 4, 5, 6, 7

- **Bandwidth**
  - @1300 Kbit/s
  - @850 Kbit/s
  - @500 Kbit/s
  - @250 Kbit/s
HTTP-based Adaptive Streaming (HAS)

- Other terms for similar concepts: Adaptive Streaming, Smooth Streaming, HTTP Chunking
- Actually a series of small progressive downloads of chunks (or range requests)
- No standard protocol ...
HTTP-based Adaptive Streaming (HAS)

- Other terms for similar concepts: Adaptive Streaming, Smooth Streaming, HTTP Chunking
- Actually a series of small progressive downloads of chunks (or range requests)
- No standard protocol ...
  - Apple HLS: HTTP Live Streaming
  - Microsoft IIS Smooth Streaming: part of Silverlight
  - Adobe: Flash Dynamic Streaming
  - DASH: Dynamic Adaptive Streaming over HTTP
## Example players

<table>
<thead>
<tr>
<th>Player</th>
<th>Container</th>
<th>Type</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Smooth Streaming</td>
<td>Silverlight</td>
<td>Chunk</td>
<td>X</td>
</tr>
<tr>
<td>Netflix player</td>
<td>Silverlight</td>
<td>Range</td>
<td>X</td>
</tr>
<tr>
<td>Apple HLS</td>
<td>QuickTime</td>
<td>Chunk</td>
<td>X</td>
</tr>
<tr>
<td>Adobe HDS</td>
<td>Flash</td>
<td>Chunk</td>
<td>✓</td>
</tr>
</tbody>
</table>
BitTorrent-like systems

- File split into many smaller pieces
- Pieces are downloaded from both seeds and downloaders
- Distribution paths are dynamically determined
  - Based on data availability

Arrivals

Downloader
Downloader
Downloader
Downloader

Torrent
(x downloaders; y seeds)

Download time
Seed residence time

Departures

Downloader
Downloader
Seed
Seed
Download using BitTorrent

Background: Incentive mechanism

- Establish connections to large set of peers
  - At each time, only upload to a small (changing) set of peers
- Rate-based tit-for-tat policy
  - Downloaders give upload preference to the downloaders that provide the highest download rates

![Diagram showing optimistic unchoke strategy]

- Highest download rates
- Pick top four
- Pick one at random

Optimistic unchoke
Download using BitTorrent

Background: Piece selection

- **Rarest first piece selection policy**
  - Achieves high piece diversity

- **Request pieces that**
  - the uploader has;
  - the downloader is interested (wants); and
  - is the rarest among this set of pieces
BitTorrent Model (random)

Arrival rate = $\lambda$

Torrent
(with $x$ downloaders and $y$ seeds)

Departure rate = $\mu y$

Peers
(sorted by age)
BitTorrent Model (chaining)

Arrival rate = $\lambda$

Torrent
(with x downloaders and y seeds)

Departure rate = $\mu \cdot y$

Peers
(sorted by age)
Peer-assisted VoD streaming

- Can BitTorrent-like protocols provide scalable on-demand streaming?

- How sensitive is the performance to the application configuration parameters?
  - Piece selection policy (rarest vs. in-order tradeoff)
  - Peer selection policy
  - Upload/download bandwidth

- What is the user-perceived performance?
  - Start-up delay
  - Probability of disrupted playback
Live Streaming using BT-like systems

- Live streaming (e.g., CoolStreaming)
  - All peers at roughly the same play/download position
    - High bandwidth peers can easily contribute more ...
  - (relatively) Small buffer window
    - Within which pieces are exchanged
More slides ....
**Example: HAS and proxy**

![Diagram showing a laptop connected to the internet, with a proxy in between]

**Clients’ want**
- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

---

Example: HAS and proxy

Clients' want

- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

HAS is increasingly responsible for larger traffic volumes ...

... proxies to reduce traffic??

Example: HAS and proxy

Clients’ want
- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Network providers’ want
- High QoE of customers/clients

Example: HAS and proxy

Clients’ want
- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Network providers’ want
- High QoE of customers/clients
- Low bandwidth usage
- High hit rate

Example: HAS and proxy

Clients’ want
- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Network providers’ want
- High QoE of customers/clients
- Low bandwidth usage
- High hit rate

Example: HAS and proxy

Proxy example ...
Example: HAS and proxy
Example: HAS and proxy
Example: HAS and proxy
Example: HAS and proxy
Example: HAS and proxy

Client

<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>5,3</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
</tr>
</tbody>
</table>

Proxy

<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
<th>Quality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>5,3</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
</tr>
</tbody>
</table>
Example: HAS and proxy

Client 1

Proxy before

Proxy after
Example: HAS and proxy

<table>
<thead>
<tr>
<th>Quality</th>
<th>Time</th>
<th>Client 1</th>
<th>Proxy before</th>
<th>Proxy after</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
</tr>
<tr>
<td>1,3</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
</tr>
<tr>
<td>1,1</td>
<td>2,2</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
</tr>
</tbody>
</table>

Quality:
- 1,4
- 1,3
- 1,2
- 1,1

Time:
- 1,1
- 2,2
- 3,1
- 4,1

Proxy before:
- 1,4
- 1,3
- 1,2
- 1,1

Proxy after:
- 1,4
- 1,3
- 1,2
- 1,1
Example: HAS and proxy
Example: HAS and proxy

Proxy before

Proxy after

Client 2
**Example: HAS and proxy**

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Quality</th>
<th>1,1</th>
<th>1,2</th>
<th>1,3</th>
<th>1,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
<td>6,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
<td>6,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>6,3</td>
</tr>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>5,4</td>
<td>6,4</td>
</tr>
</tbody>
</table>
```

**Proxy before**

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Quality</th>
<th>1,1</th>
<th>1,2</th>
<th>1,3</th>
<th>1,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
<td>6,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
<td>6,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>6,3</td>
</tr>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>5,4</td>
<td>6,4</td>
</tr>
</tbody>
</table>
```

**Proxy after**

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Quality</th>
<th>1,1</th>
<th>1,2</th>
<th>1,3</th>
<th>1,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td>5,1</td>
<td>6,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td>5,2</td>
<td>6,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td>5,3</td>
<td>6,3</td>
</tr>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td>5,4</td>
<td>6,4</td>
</tr>
</tbody>
</table>
```
Example: HAS and proxy