Challenges with Layering: a Video Conferencing Case Study

COS 316: Principles of Computer System Design
Lecture 12

Amit Levy & Ravi Netravali
Modularity Through Layering

• Systems on systems on systems though layering

• Each layer hides complexity with abstraction

• Pro: simplifies design of each layer (and overall system)
Internet Protocol Layers

Application
- Application Messages
  - HTTP, SMTP, FTP, Skype, etc.

Transport
- Reliable streams
- Datagrams
  - TCP, UDP

Network
- Best-effort *global* packet delivery
  - IP

Link
- Best-effort *local frame* delivery
  - Ethernet, WiFi, etc.

Physical
- Local bit delivery
  - Coaxial cable, fiber optic cable, etc.
Physical communication

• Communication goes down to the **physical network**

• Then from **network** peer to peer

• Then up to the **relevant application**
Cons of Layering?

• Focus on generality rather than top-to-bottom optimality

• Lack of visibility across layers

• Potential lack of coordination across layers
Salsify:
Low-Latency Network Video Through Tighter Integration Between a Video Codec and a Transport Protocol

Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu⁺, Riad S. Wahby, Keith Winstein

Stanford University, ⁺Saratoga High School

https://snr.stanford.edu/salsify
Video Systems: Streaming

- **Use TCP**
  - Want reliability, not terribly latency-sensitive

- **Video encoded beforehand**

- **Uses playback buffer**
  - (downloads ahead of time)
  - Can hide errors in network prediction

---

**Internet**

**Video Server**

- Rate 1
  - Chunk 1
  - Chunk 2
  - ...
  - Chunk n

- Rate 2
  - Chunk 1
  - Chunk 2
  - ...
  - Chunk n

- Rate 3
  - Chunk 1
  - Chunk 2
  - ...
  - Chunk n

**Playback Buffer**
Video Systems: Conferencing

- Use **UDP**
  - Prioritize low latency over reliability
- Video **encoded on the fly**
- **No playback buffer**
  - Network prediction errors can be costly!
Today's systems combine two (*loosely-coupled*) components.
Two distinct modules, two separate control loops

Existing conferencing systems already break layering we discussed! → expose network forecasts
Current systems do not react fast enough to network variations, end up congesting the network, causing stalls and glitches.
The problem: codec and transport are too decoupled

• The codec can only respond to changes in target bit rate over coarse time intervals.
  • Individual frames may cause packet loss/queueing.

• The transport has little control over what codec produces.

⇒ The resulting system is slow to react to network variations.
Enter *Salsify*

- Salsify is a new architecture for real-time Internet video.
  - Salsify tightly integrates a *video-aware transport protocol*, with a *functional video codec*, allowing it to *respond quickly to changing network conditions*.

- Salsify achieves *4.6× lower p95-delay* and *2.1 dB SSIM higher visual quality* on average when compared with FaceTime, Hangouts, Skype, and WebRTC.
Salsify explores a more tightly-integrated design
Salsify’s architecture:
Video-aware transport protocol
There’s no notion of bit rate, only the next frame size!

Transport uses packet inter-arrival time, reported by the receiver.
Transport tells us how big the next frame should be, but...

It’s challenging for any codec to choose the appropriate quality settings upfront to meet a target size—they tend to over-/undershoot the target.
How to get an accurate frame out of an inaccurate codec

• **Trial and error:** Encode with different quality settings, pick the one that fits.
  
  • *Not possible with existing codecs.*
After encoding a frame, the encoder goes through a state transition that is impossible to undo.
Functional video codec to the rescue

\[ \text{encode}(\text{state}, \hat{a}) \rightarrow \text{state}', \text{ frame} \]

Salsify’s functional video codec exposes the state that can be saved/restored.
Order two, pick the one that fits!

• Salsify’s functional video codec can explore different execution paths without committing to them.
• For each frame, codec presents the transport with three options:
  ▶ A slightly-higher-quality version,
  ▼ A slightly-lower-quality version,
  ✗ Discarding the frame.
Codec → Transport

“Here’s two versions of the current frame.”
“I picked #2. Base the next frame on its state.”
Codec → Transport
“Here’s two versions of the latest frame.”
Transport → Codec

“I picked #1. Base next frame on its state.”
Codec → Transport
“Here’s two versions of the latest frame.”
Transport → Codec

“Can’t send frames right now. Discard them.”

target frame size 5 KB
Codec → Transport
“Fine. Here’s 2 versions of the latest frame.”
Transport → Codec
“I picked #1. Base next frame on its state.”
Evaluation results: AT&T LTE Trace

- WebRTC (VP9-SVC)
- FaceTime
- Hangouts
- Skype
- Salsify
Evaluation results: T-Mobile UMTS Trace

![Graph showing video quality (SSIM dB) vs. video delay (95th percentile ms) for different video call applications: WebRTC (VP9-SVC), Skype, FaceTime, Hangouts, and Salsify. The graph indicates that WebRTC (VP9-SVC) has the highest video quality with a low video delay, making it the best choice among the options.]
Evaluation results: **Emulated Wi-Fi (no variations, only loss)**
Network Variations

Salsify

WebRTC (Chrome 65)
Salsify, a new heart from old parts

• Individual component of Salsify are not exactly new:
  • The transport protocol is inspired by “packet pair” and “Sprout-EWMA”.
  • The video format, VP8, was finalized in 2008.
  • The functional video codec was described at NSDI’17.

• Salsify is a **new architecture** for real-time video that integrates these components in a way that responds quickly to network variations.
Improvements to video codecs may have reached the point of diminishing returns, but changes to the architecture of video systems can still yield significant benefits.
Takeaways

• Salsify is a new architecture for real-time Internet video.

• Salsify tightly integrates a video-aware transport protocol, with a functional video codec, allowing it to respond quickly to changing network conditions.

• Salsify achieves \(4.6 \times\) lower p95-delay and 2.1 dB SSIM higher visual quality on average when compared with FaceTime, Hangouts, Skype, and WebRTC.

• The code is open-source, and the paper and raw data are open-access: 
  https://snr.stanford.edu/salsify
Breaking Layers!

• **Expose more information to application**
  • WebRTC (and other transport protocols like Sprout) share forecasts of what they think network can support based on ACKs, etc.

• **Coordinate the control loops**
  • Transport and application layers each have their own designs and flaws; jointly operating them helps mask those issues
  • 2 big techniques: (1) late-binding decisions and (2) test alternatives