In-Order Recording with Unordered TCP Delivery

Michael F. Nowlan
working with Bryan Ford (advisor)
and Avi Silberschatz

Official Graduate Student Talk
Department of Computer Science
Yale University

December 6\textsuperscript{th} 2011
Roadmap

- Motivation: Voice-over-IP (VoIP)
- Tng Project and uTCP
- Stream Recording (srec)
- Preliminary Results
- Applications, Discussion and Conclusion
The Goal

- Acme Inc. and Biotech Inc. are considering a merger.
- They use Voice-over-IP to discuss terms of the deal.
- They want the lowest latency possible, for natural dialogue.
  → Skip voice frames dropped by the network.
- They want a perfect, complete audio transcript of the call.
  → Transcript should include dropped frames.
Consider the VoIP Application

- Voice-over-IP (VoIP) is a real-time application
  - Heavy on I/O – speech input, audio output
- Latency is the gap in time from speaking to playback at the receiver.
- A longer time gap implies poor performance.
- If the application just records everything from the speakers, the skipped voice frames will NOT be in the transcript.
VoIP and Audio

- Voice is sampled $N$ times per second
  Eg. 8KHz = 8000 samples/sec.

- The application doesn't send each sample individually
  It sends a packet containing a bunch of compressed samples.
  Eg. 20ms packet

- So, in our example, there are:
  $\frac{1000}{20} = 50$ packets/sec
  with each packet containing:
  $\frac{8000}{50} = 160$ samples (compressed)

- Calculation above is *per channel*
  (Stereo audio has 2 channels, Mono audio has 1 channel)
VoIP and Audio cont'd

- At the receiver, the codec can interpolate, or “guess”, the output sound of P<sub>2</sub> based on the output of P<sub>1</sub>.
  - A single dropped packet causes minor degradation.

- For a burst of dropped packets, audio quality degrades quickly.
  - The codec is guessing based on previous “guesses”.

Yale University Department of Computer Science
Protocols for VoIP

Some protocols are well-suited for VoIP:

- **UDP**
  The application must perform its own reliability.
- **SIP, RTP and RTCP**
  Standardized protocols for VoIP and Video, usually run atop UDP.
- **SCTP**
  Offers unordered delivery.

However, connectivity for anything but TCP is uncertain with increasing use of middleboxes, Performance-enhancing Proxies (PePs) and firewalls, which often alter or block non-TCP connections.

Skype, for example, often uses TCP (Baset '06).
Transport Choice

Which transport protocol would you use?
- TCP – delivers every voice frame reliably, but delays all frames within 1 RTT of a dropped frame.
- UDP – best effort (usually in-order), but may drop frames (and may not even get through the Internet).
Transport Comparison

Let's compare audio from VoIP calls with TCP to UDP:

- 30ms one-way delay
- 3Mbps bandwidth
- 4 competing streams (simulate contention)
- No artificial loss
- 100ms jitter buffer (1.67x RTT)
Transport Choice

Which transport protocol would you use?

- TCP – delivers every voice frame reliably, but delays all frames within 1 RTT of a dropped frame.
- UDP – best effort (usually in-order), but may drop frames (and may not even get through the Internet).

Neither of these transports will meet the requirements for our scenario:

- Sacrifice responsiveness and quality to use TCP
  - dropped 339/750 packets due to delays
- Sacrifice the perfect transcript to use UDP
  - dropped 23/750 due to network loss
Back to the Merger

Ideally, we want something like an “unordered” TCP, or uTCP.
Roadmap

- Motivation: Voice-over-IP (VoIP)
- Tng Project and uTCP
- Stream Recording (srec)
- Preliminary Results
- Applications, Discussion and Conclusion
Tng Project

- Support the roll-out of new transports by *adapting* TCP and TLS.
- Split the Transport Layer into multiple layers to simplify layer semantics.
- Break the choke-hold of HTTP-over-TCP.
TCP Minion

Ferrying other transports, such as UDP, with the TCP wire format requires an out-of-order delivery mode for TCP.
Unordered TCP (uTCP)

(a) Delivery in standard TCP

1. In-Order Arrival
   - Application receive buffer
   - TCP Stack
   - (delivered)
     - CumAck = 101

2. Out-of-Order Arrival
   - Application receive buffer
   - TCP Stack
   - (delivered)
     - CumAck = 201

3. Gap-Filling Arrival
   - Application receive buffer
   - TCP Stack
   - (delayed data delivered)

(b) Delivery in uTCP

1. In-Order Arrival
   - Application fragment buffer
     - (delivered)
       - CumAck = 101

2. Out-of-Order Arrival
   - Application fragment buffer (with hole)
     - (delivered)
       - CumAck = 201

3. Gap-Filling Arrival
   - Application fragment buffer (hole filled)
     - (delivered)
       - CumAck = 201

Yale University Department of Computer Science
uTCP vs. TCP

Bytes Received By App (KB)

Time (ms)
Datagrams in a TCP Stream

- Application data writes are bookended with some “marker”.
- The network may re-segment the data en route.
- The receiver looks for a contiguous block of data between two markers.
- In our work, we use the COBS (Cheshire '07) encoding scheme to enforce datagram boundaries, but any encoding could be used.
COBS-over-uTCP

Normal Case

At app sender
m3 m2 m1
App messages to send

At TCP-minion sender
m3' m2' m1'
Encoded app msgs, with markers at beginning and end of each msg

On the wire

TCP segments
m3' m2' 2 m2' 1 m1'
Minion-encoded msgs framed by TCP and sent as TCP segments. Message m2 gets split between the two TCP segments.

At TCP-minion receiver
m3' m2' m1'
Minion-encoded msgs extracted from received TCP segments

At app receiver
m3 m2 m1
Decoding minion-encoded msgs yields app msgs

Lost Segments

At app sender
m3 m2 m1
App messages to send

At TCP-minion sender
m3' m2' m1'
Encoded app msgs, with markers at beginning and end of each msg

On the wire

TCP segments
m3' m2' 2 m2' 1 m1'
Minion-encoded msgs framed by TCP and sent as TCP segments. Segment 1 is dropped in the network

At TCP-minion receiver
m3' m2' 2 m2' 1
m3' is received completely (both beginning and end markers are received), but m2' is partially received and no part of m1' is received.

At app receiver
m3
Decoding fully-received minion-encoded msgs yields m3 which is delivered to app

Yale University Department of Computer Science
Roadmap

- Motivation: Voice-over-IP (VoIP)
- Tng Project and uTCP
- Stream Recording (srec)
- Preliminary Results
- Applications, Discussion and Conclusion
Stream Recording (srec)

Applications use the single srec abstraction for ordered, unordered, datagram or stream communication.

If received data increases the cumulative ACK, write to record.
Srec API

void srec_init(void);

// Toggles:
//   datagram
//   out-of-order delivery
// what information should be recorded: data, meta-data, or both.
void srec_set_mode(int sock_fd, int mode_mask);

// Sets the file to be used for replaying data to application.
void srec_set_replay(char *filename);

// Reads data into 'buf' using current mode.
int srec_recv(int sock_fd, unsigned char *buf, int max);

// Sends data from 'buf' using the current mode.
int srec_send(int sock_fd, unsigned char *buf, int len);

// Closes the storage file (if there is one).
void srec_finish();
# Code Comparisons

<table>
<thead>
<tr>
<th>Transport</th>
<th>TCP</th>
<th>uTCP</th>
<th>DCCP</th>
<th>SCTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Code</td>
<td>12,982</td>
<td>+565</td>
<td>6,338</td>
<td>19,312</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App-Level</th>
<th>SREC</th>
<th>RTP/RTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Code</td>
<td>646 + ~100</td>
<td>7,155</td>
</tr>
</tbody>
</table>

NOTE: RTP and RTCP provide some functionality not included in srec, such as timed sending and sequence numbering. Our prototype VoIP application implements these features in roughly 100 lines of code.
Roadmap

• Motivation: Voice-over-IP (VoIP)
• Tng Project and uTCP
• Stream Recording (srec)
• Preliminary Results
• Applications, Discussion and Conclusion
CPU Overhead (uTCP and SREC)

We measure CPU overhead, for user time and kernel time, using GNU/Linux's "getrusage" system call.

We compare CPU usage between transports, and with and without recording for a 1 minute VoIP call.
CPU Overhead (uTCP and SREC)

5 minute VoIP call, uTCP uses uCOBS, and srec uses uTCP with recording.

Values represent total CPU processing time as a comparison to TCP. The darker portion of each bar is the percentage of total time spent in the kernel.
VoIP Performance Metrics

VoIP applications perform best with:

- low-latency message delivery
  - User experiences faster response time
- few bursty losses (more than 1 message dropped)
  - User experiences fewer “blackouts”

Our VoIP experiments send a 1 minute voice file under the following network conditions:

- 30ms one-way delay
- 3Mbps bandwidth
- Artificial loss, or “competing stream” loss (as noted)
- Varying jitter buffer sizes (as noted)
Application Message Latency

Artificial Loss - 2%

End-to-End Application Message Latency
(2% loss)
Application Message Latency

Competing Streams - 4

End-to-End Application Message Latency
(4 competing streams)

Fraction of Messages (CDF)

Delay (ms)

uTCP
TCP
UDP
Bursty Losses

Audio codecs handle a single dropped packet much better than multiple dropped packets (recall, guessing based on guesses).

Due to TCP's in-order requirement, a single dropped packet causes all packets within 1 RTT to be delayed at the receiver.

Here, we show CDFs of the *length* of losses in number of packets.
Bursty Losses

Competing Streams - 4

![Graph showing Bursty Losses with fraction of bursts on the y-axis and loss burst length on the x-axis. The graph includes lines for uTCP, TCP, and UDP.]
PESQ Scores

Perceptual Evaluation of Speech Quality (PESQ) is an industry-recognized algorithm for objectively measuring the user-perceived quality of an audio file. It is tailored specifically for voice audio, and is an international standard for evaluating VoIP applications. We use the open source, reference implementation of the PESQ software provided by the International Telecommunication Union (ITU) to measure the quality of audio streams across transport protocols.

In VoIP applications, the *jitter buffer*, helps maintain continuous playback while individual frames take longer to propagate through the network. The jitter buffer is an additional delay, beyond the one-way propagation delay, enforced at the receiver for playing out an audio frame. We test different jitter buffers in our experiments.
PESQ by Transport

Competing streams doubled each minute.
Jitter buffer = 150ms (2.5x RTT)

VoIP Audio Quality w/ competing streams
60ms RTT, 150ms jitter buffer
PESQ by Transport

Competing streams doubled each minute.
Jitter buffer = 200ms (3.3x RTT)
PESQ for the Record

Srec produces an identical file to the transmitted audio (both pass through the codec).
Roadmap

- Motivation: Voice-over-IP (VoIP)
- Tng Project and uTCP
- Stream Recording (srec)
- Preliminary Results
- Applications, Discussion and Conclusion
Application: HFT

Perhaps an ideal potential application for srec is in High-Frequency Trading (HFT). HFT refers to executing BUY and SELL orders on stocks and equities at the millisecond granularity. Many HFT traders track technical indicators, such as the asset's current price or bid/ask spread, to identify the ideal moment to trigger these orders.

Characteristics of srec that benefit an HFT trader:

- Received prices are more likely to reflect actual current pricing on the exchange.
- All received prices are stored to be used for historical pricing later.
- Old prices (ie. TCP sequence number less than last seen) can be discarded by the application, preventing orders based on stale information.
- Going live with real-time data, or back-testing a strategy is a single programming abstraction: the srec socket.
Application: Call Center

One scenario we imagine involves a corporation call center that connects customer telephone calls to the call center by routing calls over the Internet using VoIP.

In these scenarios, many companies already record a transcript of the call for “quality and training purposes”. To use srec in this case, we envision a “middlebox” version of the library that automatically records VoIP traffic from within the network, without deploying the library to all end hosts.
Application: Military Raid

Imagine a powerful and precise tactical military force of 79 soldiers (and 1 dog) rappels from helicopters onto a fortified enemy compound with the intent of taking out the evil leader. Each soldier's helmet is outfitted with a night-vision camera and microphone for transmitting full audio and video of the raid back to headquarters.

A group of political leaders gathers at headquarters to watch the raid in real-time. Meanwhile, a much larger group of analysts waits to comb through the footage of the raid for further information about the compound and its inhabitants.

Using the srec library and uTCP, the footage arrives without bursty losses, as would be the case with TCP, but the analysts have the complete and perfect footage for analysis without any lost frames, as would be the case with UDP.
Discussion (ie. Feedback)

Are there ways to extend the srec library?

In particular, is there any way to make it more technically challenging, or is it just a simple hack that opens up a wide application domain?
Conclusion

Much of what we do on the Internet is done in real-time. We participate in the information exchange actively providing data or passively receiving information. Either way, we do so with an expectation about its timing. This inclines us to receive the data as quickly as possible. Furthermore, we rarely delete anything. Storage is cheap; we have a desire to save everything.

Given the above, would you rather save the potentially flawed real-time data or a perfect copy of the data?
Related Work and References

Our Work

- Dedis Group
  http://dedis.cs.yale.edu/
- Tng Project (including publications, drafts and reports)
  http://dedis.cs.yale.edu/2009/tng/

References (mentioned in slides)

Thank You

Questions?
Extra Slides
Internet Architecture Design

- The Internet was built for growth:
  - Narrow Waist
    - Internet Protocol (IP) common language
  - Transport Layer
    - Network supports all transports equally
  - End-to-End arguments
    - Minimal edge requirements spurs innovation
The Internet was built for growth:

- **Narrow Waist**
  - Internet Protocol (IP) common language
- **Transport Layer**
  - Network supports all transports equally
- **End-to-End arguments**
  - Minimal edge requirements spurs innovation

Increasingly, however, successfully traversing network paths requires using TCP (and even HTTP in some cases) on top of IP.
Shifting of the “Narrow Waist”

- Connectivity
  Middleboxes, NATs and Firewalls have become so restrictive as to make HTTP-with-TCP the only reliable connectivity path.

- Performance
  Hardware TCP offload engines and Performance Enhancing Proxies (PEPs) improve specifically TCP without benefiting other transports.

- Familiarity
  TCP has great cultural inertia; every network programmer knows how to use it. No other transport has such widespread recognizability.
Shifting of the “Narrow Waist”

- **Connectivity**
  Middleboxes, NATs and Firewalls have become so restrictive as to make HTTP-with-TCP the only reliable connectivity path.

- **Performance**
  Hardware TCP offload engines and Performance Enhancing Proxies (PEPs) improve specifically TCP without benefiting other transports.

- **Familiarity**
  TCP has great cultural inertia; every network programmer knows how to use it. No other transport has such widespread recognizability.

*The result? TCP tunnels everywhere...*
TCP Tunnels

- Media streaming/conferencing applications (eg. Skype)
- New transport services (eg. Google's SPDY and W3C's WebSocket API)
- Virtual Private Networks (VPNs) (eg. Microsoft's DirectAccess)
Throughput Comparison

![Bar chart showing normalized throughput comparison with different loss rates. The x-axis represents the loss rate (%) and the y-axis represents the normalized throughput (x TCP). The chart compares TCP, COBS, uCOBS, TLS, and uTLS. The throughput for each protocol is consistent across different loss rates.]
Application Message Latency

Competing Streams - 8

End-to-End Application Message Latency
(8 competing streams)
Bursty Losses

Competing Streams - 4