1. Introduction
   - why QoS?
   - what are the problems?

2. Basic operations
   - jitter buffers (at hosts)
   - task scheduling (at hosts)
   - packet shaping (at hosts)
   - packet dropping (at routers)
   - packet scheduling (at routers)

3. Types of service
   - Integrated Services (IntServ) and Resource Reservation Protocol (RSVP)
   - Differentiated Services (DiffServ)
4. Application-level feedback and control
   — Real-time Protocol (RTP), Real-time Control Protocol (RTCP)
   — Real-time Streaming Protocol (RTSP)

5. Application signaling and device control
   — Session Announcement Protocol (SAP)
   — Session Description Protocol (SDP)
   — Session Initiation Protocol (SIP)
   — Media Gateway Control Protocol (MGCP)

6. Routing
   — Multi-protocol Label Switching (MPLS)
   — multicasting
Transporting Multimedia Over IP

- **TCP** overhead + delay is **unacceptable** for much multimedia
  - preferable: use **UDP** for transport

- **Problem**: **UDP** is connectionless, unreliable

- **RTP**: the Real-Time Transport Protocol
  - IETF RFC 1889

- **RTP = application-level framing**
  - application controls **recovery**, in-order delivery, playback timing
  - **synchronization** of sender and receiver
  - negotiation between sender and receiver **allows application to adapt** to changing network or receiver conditions
1. **Timing reconstruction**
   - e.g., control of playout (jitter) buffer

2. **Synchronization** of different media types
   - e.g., audio + video

3. **Packet sequencing**

4. **Loss detection**

5. **Content identification**

6. **QoS feedback and rate adaptation**
   - e.g., reduce frame rate if loss rate is too high
RTP Non-Functions

• It does not...
  • guarantee reliable delivery of packets
  • guarantee QoS
Translators and Mixers

- Participants in a multimedia session may use different media formats or compression standards
  - a *translator* (also called a *media gateway*) converts from one media format to another

- Audio from multiple senders can be mixed into a single audio stream
  - a *mixer* combines media streams
## RTP Header

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Version field</td>
</tr>
<tr>
<td>1</td>
<td>Padding Flag <em>(1 = packet has been “padded”, last byte of packet specifies how many padding bytes there are)</em></td>
</tr>
<tr>
<td>1</td>
<td>Extension Flag <em>(1 = extension header follows RTP header)</em></td>
</tr>
<tr>
<td>4</td>
<td>Number of Sources mixed (max of 15)</td>
</tr>
<tr>
<td>1</td>
<td>Application Marker Bit <em>(indicates frame start, or beginning of “talkspurt”)</em></td>
</tr>
<tr>
<td>7</td>
<td>Payload Type <em>(audio or video encoding method)</em></td>
</tr>
</tbody>
</table>
### Header (cont’d)

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td><strong>Sequence Number</strong> (max of 65535); gaps → packet loss</td>
</tr>
<tr>
<td>32</td>
<td><strong>Timestamp</strong> (format specified by the application)</td>
</tr>
</tbody>
</table>
| 32     | **Synchronization Source Identifier (SSRC)**  
original sender of message, or mixer if used  
“identifiers” are randomly generated; conflicts can be detected and resolved |
| N*32   | **Contributor Source Identifiers (CSRC)**  
N = Number of Sources – 1 |
RTP Timestamps

- Initial value is random number
- Resolution is payload-dependent, specified as part of the A/V profile standard
  - resolution must be at least = sampling rate of the media type (e.g., 8KHz for compressed voice)
- **Timestamp** in an RTP packet is time of the first sample in the packet
- Several consecutive RTP packets may have equal **Timestamps** if they are (logically) generated at once
  - e.g., belong to the same video frame
- Consecutive RTP packets may contain **Timestamps** that are not monotonic if the data is not transmitted in the order it was sampled
  - e.g., MPEG interpolated video frames
RTP Ports and Profiles

- A port-pair for RTP and RTCP
  - e.g., 5004 for RTP, 5005 for RTCP
  - each media stream has own RTP connection

- RFC 1990 specifies A/V Profiles
  - default Payload Types for common audio and video compression standards
  - mapping of media encoding to payload format
  - default packet rate
Summary: RTP

- Adds a new header to each packet
  - minimum of 12 bytes long
  - most important info: payload type, sequence number, and timestamp
RTCP – the RTP Control Protocol

- Closely tied to RTP; used for negotiation between senders and receivers
- Reports the quality of the connection between sender and receivers
  - the frequency of reporting between senders and receivers is carefully controlled to prevent excessive overhead
- RTCP message types
  1. Sender Report – current time and amount of data sent so far
  2. Receiver Report – feedback about what has been received so far
  3. Source Description – useful information about the source
  4. Bye – source is disconnecting
  5. Application-specific

Copyright 2001 Douglas S. Reeves  (http://reeves.csc.ncsu.edu)
### RTCP Sender Report Message

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+1</td>
<td>Version+Padding Flag</td>
</tr>
<tr>
<td>5</td>
<td>Number of Receiver Blocks in packet</td>
</tr>
<tr>
<td>8</td>
<td>Packet Type (=200 for sender report)</td>
</tr>
<tr>
<td>16</td>
<td>Packet Length</td>
</tr>
<tr>
<td>32</td>
<td>Sender SRC ID</td>
</tr>
<tr>
<td>64</td>
<td>NTP Timestamp (standardized representation of actual time of day)</td>
</tr>
<tr>
<td>32</td>
<td>RTP Timestamp (application specific)</td>
</tr>
<tr>
<td>32</td>
<td>Number of Packets Sent (so far)</td>
</tr>
<tr>
<td>32</td>
<td>Number of Bytes Sent (so far)</td>
</tr>
<tr>
<td>M<em>24</em>8</td>
<td>Receiver Report Blocks, M = # of remote sources</td>
</tr>
</tbody>
</table>
Receiver Report Blocks

- There is one Receiver Report Block for each receiver which is reporting to this sender

- Receiver Report Block confirms feedback from receiver, and also distributes it to other receivers
### Sender Report (cont’d) → Receiver Report Blocks

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>SSRC of source</td>
</tr>
<tr>
<td>8</td>
<td>Fraction of Blocks Lost Since Last Report (relative to 255)</td>
</tr>
<tr>
<td>24</td>
<td>Cumulative Number of Packets Lost</td>
</tr>
</tbody>
</table>
| 32     | Highest Packet Sequence Number Received  
RTP sequence numbers wrap around after $2^{16} - 1$  
this is “extended sequence number” |
| 32     | Interarrival Jitter Estimate |
| 32     | Time of Last Report From This Receiver  
NTP format, just the middle 4 bytes |
| 32     | Time Since Last Report From This Receiver  
in units of 1/65536 seconds |
**RTCP Receiver Report Message**

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+1</td>
<td>Version+Padding Indicator</td>
</tr>
<tr>
<td>5</td>
<td>Number of Source Report Blocks in packet</td>
</tr>
<tr>
<td>8</td>
<td>Packet Type (=201 for sender report)</td>
</tr>
<tr>
<td>16</td>
<td>Packet Length</td>
</tr>
<tr>
<td>32</td>
<td>SSRC</td>
</tr>
<tr>
<td>M*24</td>
<td>Source Report Blocks, M = # of remote senders</td>
</tr>
</tbody>
</table>

- Source report blocks have same format and info as receiver report blocks
- There is one Source Report Block for each sender about which the receiver is reporting

Copyright 2001 Douglas S. Reeves  (http://reeves.csc.ncsu.edu)
### RTCP Source Description Message

<table>
<thead>
<tr>
<th># Bits</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+1</td>
<td>Version+Padding Indicator</td>
</tr>
<tr>
<td>5</td>
<td>Number of Source Descriptors in packet</td>
</tr>
<tr>
<td>8</td>
<td>Packet Type (=202 for source description pkt)</td>
</tr>
<tr>
<td>16</td>
<td>Packet Length</td>
</tr>
<tr>
<td>M*x</td>
<td>Source Descriptors, M = # of remote sources length (x) is application dependent</td>
</tr>
</tbody>
</table>
Source Descriptors

• Each **Source Descriptor** contains
  — Source ID (32 bits)
  — sequence of TLV (tag, length, value)-encoded fields

• Source description fields
  — each TLV must be aligned on 32-bit boundary
  — T and L are 8 bits each
  — some fields have been standardized
    • examples: name, email, phone number, location, ...
Additional RTCP Messages…

- **Bye** message
  - SSRCs of sources
  - optionally, “reason for leaving” (LV-encoded)

- **Application-Specific** message
  - allows RTCP extensions easily

- **Message combining**
  - can put multiple messages into one “compound” packet to save overhead
RTCP Traffic Volume

- RTCP traffic designed to be no more than 5% of the media traffic (RTP)
  - 1.25% allocated to senders, and 3.75% allocated to receivers
  - randomized response; as number of receivers increases, frequency of response per receiver decreases

- Minimum packet transmission frequency is 5 seconds

- (Can also explicitly indicate the bandwidth for RTCP)
  - Using b=RS:xxxx and b=RR:yyyy extension parameters in SDP
    - RS = bandwidth for sender report
    - RR = bandwidth for receiver report (in bits/sec)
RTP For Multicasting

- RTP designed for multi-party applications
- Feedback from each receiver is multicast to all participants
  - allows easy monitoring of quality by an external entity
  - makes controlling the total amount of feedback to sender from all receivers simpler
Computing Round-Trip Delay


Sender n  

SR(n)  

A = 0xb710:8000 (46864.500 s)

ntp_sec = 0xb44db705
ntp_frac = 0x20000000
(3024992016.125 s)

dlsr = 0x0005:4000 (5.250 s)
lsr = 0xb705:2000 (46853.125 s)

Receiver  

RR(n)  

DLSR (5.25 s)

A  0xb710:8000 (46864.500 s)
DLSR  -0x0005:4000 (5.250 s)
LSR  -0xb705:2000 (46853.125 s)
delay  0x 6:2000 (6.125 s)
RTP Jitter Calculations

• Calculating jitter (statistical variance of the RTP data interarrival times) to be inserted in the interarrival jitter field of Receiver Reports
  – s points to state for the source, r points to state for the receiver, rr is receiver report
  – jitter field of the reception report is integer, jitter estimate is floating point

• Inputs
  – r->ts, the timestamp from the incoming packet
  – arrival, the current time in the same units

• Outputs
  – s->transit holds the relative transit time for the previous packet
  – s->jitter holds the estimated jitter
RTP Jitter Calculations (cont’d)

- As each data packet arrives, jitter is estimated:

```c
int transit = arrival - r->ts;
int d = | transit - s->transit |;
s->transit = transit;
s->jitter += (1./16.) * ((double)d - s->jitter);
rr->jitter = (u_int32) s->jitter;
```
Jitter Calculation Example

Initially, s->transit = 0, s->jitter = 0

arrival = 1000, r->ts = 700
transit = 300
d = 300 – 0 = 300
s->transit = 300
s->jitter = 0 + 1/16*(300-0) = 18.75
rr->jitter = 18

arrival=1800,r->ts=1700
transit = 100
d = 200
s->transit = 100
s->jitter = 18.75 + 1/16*(200 – 18.75) = 30.01
rr->jitter = 30
RTP Types for DTMF Digits, Tones, and Events

• Problem: audio compression may distort DTMF digits and other signals beyond safe recognition
  — voice message systems, flashhook, etc.

• Rather than transmit as compressed audio, recognize at the sending point and transmit as a well-defined event
  — Must convey duration and volume as part of the payload
Header Compression: Motivation

• Every voice packet has overhead...
  — IP = 20 bytes
  — UDP = 8 bytes
  — RTP = 12 bytes
  — Total = 40 bytes

• With 20 bytes of payload, 66% of the packet is overhead!
  — this is particularly a problem with slow links (e.g., access network)

• Why not put more payload per packet?
  — more payload bytes = more packetization delay

• Goal: reduce packet overhead
  — transparent to application
  — transparent to IP, UDP, RTP
  — solution: compress at the link layer
Header Compression: Principles

- Many of the fields in the headers do not change
  - same in packet i, i+1, i+2, ...
  - transmit only the first time!

- Many of the fields change by a constant increment between packets
  - transmit the first time, and indicate increment value to use

- Many fields values change by only small amount
  - use differential coding on these fields

- Actual (transmitted) header: index #, and indicator to “use the normal prediction”
**HC: Identifying Each RTP Session**

- Identifying a “session context”
  - IP source and destination addresses
  - UDP source and destination ports
  - RTP session source ID (SSRC)
- Hash the session context to a unique session index #
- Substitute for (IP+UDP+RTP) header…
  - session index #
  - indicator to use “the normal prediction”
### HC: IP Header

<table>
<thead>
<tr>
<th>Field</th>
<th>No Change</th>
<th>Fixed Increment</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header Length</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOS</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Length</td>
<td>X</td>
<td></td>
<td>Redundant, let link layer handle</td>
</tr>
<tr>
<td>Identification Number</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header Checksum</td>
<td></td>
<td></td>
<td>Redundant, let link layer handle</td>
</tr>
<tr>
<td>Source IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**HC: UDP Header**

<table>
<thead>
<tr>
<th>Field</th>
<th>No Change</th>
<th>Fixed Increment</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port Number</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Port Number</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP Length</td>
<td></td>
<td></td>
<td>Redundant, let link layer handle</td>
</tr>
<tr>
<td>UDP Checksum</td>
<td></td>
<td></td>
<td>Optional, but if present cannot be predicted or compressed</td>
</tr>
</tbody>
</table>
HC: RTP Header

- No change: version, padding flag, extension flag, number of sources, payload type, SSRC, contributor source identifiers

<table>
<thead>
<tr>
<th>Field</th>
<th>Fixed Increment</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market Bit</td>
<td></td>
<td>Needed</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
HC: Increment Encoding

- Default increments are stored in the session context table

- Updated when an explicit value is sent
  — encoding of these values may be negotiated previously
**HC: Sending Uncompressed Headers**

- Used to automatically "refresh" the synchronization of sender and receiver
  - or, explicitly requested by receiver when synchronization is lost

- Contains complete, normal header
  - must add session ID # and sequence number
  - use existing fields!

- Keep a "negative cache" of sessions that won't compress well
  - don't keep trying to compress; just give up!
HC: Results

- Amount of compression (typical): down to 2-4 bytes!
- Success rate of prediction (typical): 95-98%!
- How trigger removal from session context table?
Sources of Info

• Books
  — Thomas, “IPNg and the TCP/IP Protocols”, 1996 (chapter 11)
  — Douskalis, “IP Telephony”, 2000 (chapter 2)
  — Davidson, “Voice over IP Fundamentals”, 2000 (chapters 8 and 9)

• Web
  — RTP: A Transport Protocol for Real-Time Applications
  — Compressing TCP/IP Headers for Low-Speed Serial Links