

Streaming Audio and Video

CSC 790

WAKE FOREST
UNIVERSITY

Department of Computer Science

Fall 2009

Audio and Video from a Web Server

A client, using a web browser, requests audio/video from a web server

- Client requests audio/video from server (`http` request)
- Web server acknowledges request using `http`
- The web server can deliver audio/video in two fashions
 - Over `http` protocol using a web browser
 - Over non-`http` using a *helper-application*
- As you may recall about `http` ...
 - It is an application layer protocol
 - Uses **TCP** as its transport (layer 4) protocol

Delivery Using http

The following would occur when delivering audio/video using http

1. Browser (client) establishes TCP connection to server and requests audio/video file in http message
2. Web server sends audio/video file in http response message
 - Content line of message indicates the audio/video encoding
 - Browser reads message and starts associated media player
 - Sends file to media player (helper application)
3. Media player plays file

What are the drawbacks to this approach?

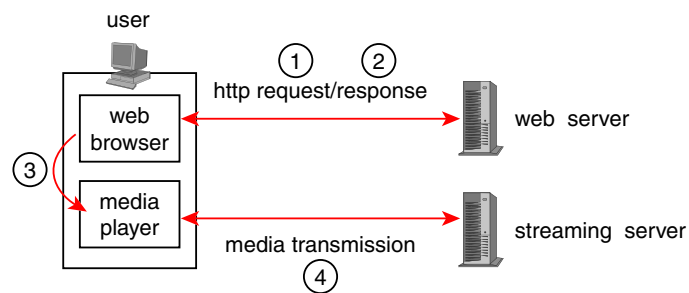
- Problems with the *simple* (http based) approach
 - Must receive entire file before playing
 - Player must interact with browser
- Is there any advantage to this approach?*
- We prefer a method of delivery that...
 - Allows **streaming**
 - Does not use the web browser for sending the media
 - * Want a direct socket between server and player
 - Avoids TCP and http for transmitting audio/video

Media Servers and Helper Applications

- To avoid TCP and http for sending audio/video
 - Use a **streaming server**
 - Streaming server will use UDP for transmission
 - It cannot interact with web browser, only helper application

To access audio/video using media server and help application

1. Browser requests audio/video file in http message
2. Web server responds with http message, specifying the media type and server location
3. Browser reads response and starts media player (helper application) and send message to media player
4. Media player starts connection with media server (directly)



Are the servers two independent machines?

Is the firewall an issue?

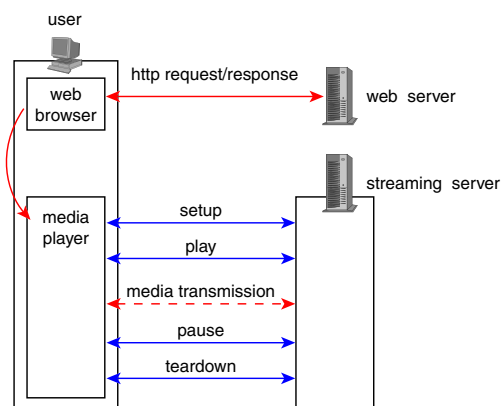
How is the media stream controlled?

Real-Time Streaming Protocol

- Real-Time Streaming Protocol (RTSP) does **not** define
 - Compression methods for audio/video
 - How media is encapsulated in packets (RTP defines this)
 - How media is transported (TCP/UDP)
 - Buffering at the client
- RTSP sets-up, controls (start, pause, stop), and tears-down a media stream (session control [RFC 2326])
- RTSP is an **out-of-band** protocol
 - A separate socket is used for RTSP (port 544)
 - RTSP can use TCP or UDP
 - Media transmitted using another socket, called **in-band**

Another out-of-band example?

RTSP Control



- To start the streaming media
 1. Web browser request presentation description file, has several references to media files
 2. Web server provides HTTP response

```

<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>

```

- Media controlled with a series of RTSP messages
 1. Setup the connection
 2. Play/pause media (start/stop stream)
 3. Teardown connection

RTSP Session Between Client and Server

| | |
|--------|--|
| Client | SETUP rtsp://audio.example.com/twister/audio RTSP/1.0 Transport: rtp/udp; compression; port=3056; mode=PLAY |
| Server | RTSP/1.0 200 1 OK Session 4231 |
| Client | PLAY rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 Session: 4231 Range: npt=0- |
| Client | PAUSE rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 Session: 4231 Range: npt=37 |
| Client | TEARDOWN rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 Session: 4231 |
| Server | 200 3 OK |

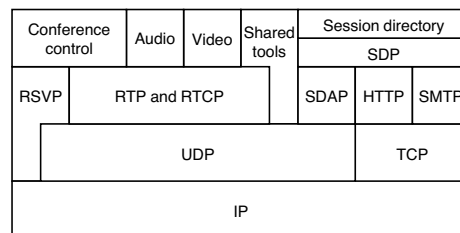
Streaming Audio and Video

We know how to *control* a media stream, but how is it actually sent?

- We have already discussed the importance of
 - Buffering at the receiver
 - The need for sequence numbers and timestamps
- In addition, we know UDP is the preferred transport layer for streaming audio/video; however,
 - UDP does not provide sequence number and timestamps
- Another protocol is needed to provide this functionality

Real-time Transport Protocol

- Real-time Transport Protocol (RTP) runs on top of UDP

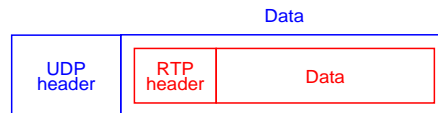


- Considered an application layer protocol [RFC 3550]
 - Implemented at the client and server
 - Networking devices (routers) are **unaware** of RTP *So what?*
- RTP provides sequence numbers and timestamps
 - Does **not** provide guarantees *What does this mean?*

RTP Operation

Consider transmitting voice using RTP

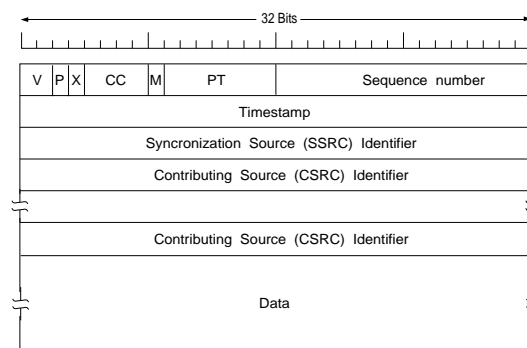
- PCM samples created from voice
- Application collects 20 msec of samples (160 bytes)
- Application creates a RTP packet
- RTP packet sent using UDP socket



RTP packet → UDP packet → IP packet → layer 2 frame

- We know what the layers 2-4 headers provide (*class says, yes!*)
- What does the RTP header provide?

RTP Header



There are 4 main (important) fields in the RTP header

- Payload type (PT) - Identifies the type of data (0 = PCM)
- Sequence number - Incremented once for each RTP packet
- Timestamp - 32 bits, reflects the sampling instance of the first byte in the data field

- Synchronization Source Identifier (SSRC) - Identifies the source of the RTP stream

More about the timestamp field

- Derived from a *sampling* clock of the sender
 - Clock increments by one for every sampling period
 - Consider PCM with sampling rate of 8000 Hz, the clock would increment by one every 125 μ sec
- Unlike the sequence number, the clock continues to increase at a constant rate even if the source is inactive

Why have sequence numbers and timestamp? Could we just use a timestamp instead?

RTP Header Continued

- Version (V) - 2 bits indicating RTP version
- Padding (P) - Single bit indicating the RTP payload is padded
- eXtension (X) - Single bit indicating presence of extension header
- Contributing Sources (CC) - 4 bits, number of contributing sources
 - Stream may be composite (mix) of other streams
 - Field indicates the number of streams mixed together
 - Contributors listed as SSRC's
- Marker (M) - Single bit set to mark something

What might be useful to mark in a media stream?

- Contributing Source Identifier (CSRC) - Identifies a contributing source to the RTP stream

RTP Control Protocol

- Real-Time Control Protocol (RTCP) provides stream feedback
 1. Feedback on application and network performance
 2. Correlate and synchronize different stream from same sender
 3. Convey identity of sender
- Application and network performance
 - Periodically, receiver/sender send *QoS reports*
 - How many packets lost, what is the jitter, ...
 - Given this information, can adjust transmission

How do you adjust transmission?

Why can't we adjust the network?

- Correlating streams from same sender
 - Sender may use one stream for audio, another for video

Why might this be done?

 - Different streams may have different clocks, must synchronize
- Convey identity of sender
 - May need to display actual ID of sender
 - Consider IP telephony

Periodically sender and receiver generate RTCP *reports*

- Report the information described above
 - QoS, synchronization, or identity
- Must be certain not to use too much bandwidth...

RTCP Scaling

- RTCP has a potential scaling problem
 - Consider one sender and a large number of receivers
 - Receiver send RTCP reports, overwhelming the sender

Amount of media sent is less than the RTCP?
- RTCP limits its traffic to 5% of the session bandwidth
 - Senders *share* 25% of this amount, receivers *share* the 75%

$$r_s = \frac{n_s}{0.25 \times 0.05 \times s} \times p$$

$$r_r = \frac{n_r}{0.75 \times 0.05 \times s} \times p$$

- Where r_s is the sender rate, r_r is the receiver rate, n_s is the number of senders, n_r is the number of receivers, s is the session bandwidth, and p is the average RTCP packet size

Call and Session Control

RTP and RTCP provide a range of functionality *except*

- Call or session control

Suppose you wanted to create a video conference at a certain time and make it available to a group of people

- Furthermore you decide...
 - Encode stream using MPEG-2
 - Use the multicast IP address 244.1.1.1
 - Send using RTP over port 4834

*Participants **must** know this information, but how?*

IETF Session Control

IETF Multiparty Multimedia Session Control Group protocols

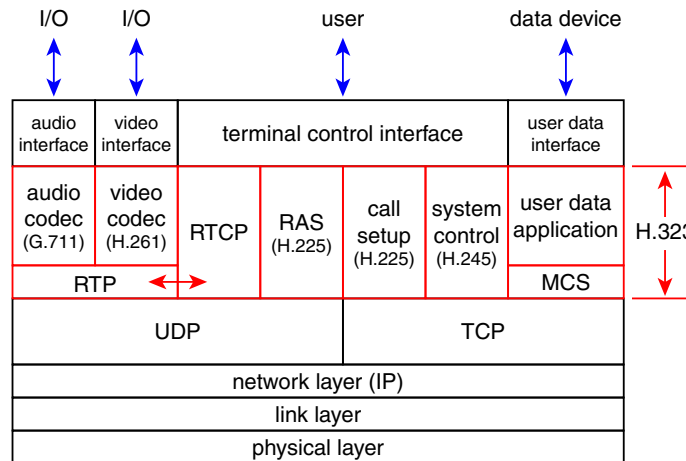
1. Session Description Protocol (SDP)
2. Session Announcement Protocol (SAP)
3. Session Initiation Protocol (SIP)
4. Simple Conference Control Protocol (SCCP)

- Example usage
 - Video conference would use SDP and SAP
 - IP telephony would use SDP and SIP

Similarly the ITU has developed a similar set of standards for multimedia transmission called H.323

H.323

H.323 is an ITU standard for real-time audio and video conferencing



H.323 is actually a collection of protocols

- Call control, session control, and media transmission

Example H.323 protocols include

- G.711 for audio compression
- H.261 for video compression
- RTP for sending audio/video data
- H.245 an out-of-band control protocol for controlling media (analogous to RTSP)
- H.225 for establishing and terminating calls
 - Also called Registration/Admission/Status (RAS) protocol

Why do many control protocols use TCP?