Latency and Loss Requirements

Fundamental characteristics of multimedia applications:

Typically **delay sensitive**

- live audio < 150 msec end-to-end delay is not perceptible
- 150-400 msec end-to-end delay is tolerable
- > 400 msec end-to-end delay makes audio unintelligible
- streaming audio: can wait 5 secs or more before starts of playback
- low variability of packet delays (jitter) within the same packet stream

But **loss tolerant**: infrequent losses cause minor glitches

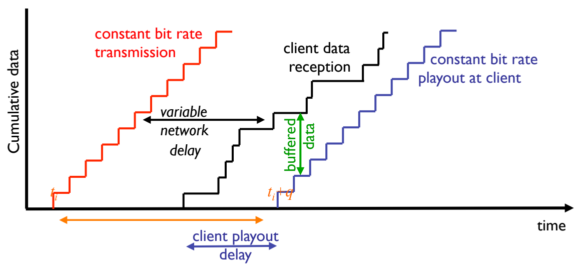
- 1% to 10% or even 20% loss is tolerable

Antithesis of data, which is loss intolerant but delay tolerant

1

Receiver-side Buffering

Jitter removal: receiver-side buffering and delayed playback



If q is large enough, by the time sampled must be played back, it would have arrived at the receiver

Playback point: transmitted time (t_i) + e2e delay (τ_i) + buffer time (b_i)

2

Dealing with Loss

Two types of loss:

1. network loss: network dropped packet
2. playback loss: packet arrives too late for play out at receiver
   - latency: delays due to processing, queueing in network; end-system (sender, receiver) delays
   - typical maximum tolerable one-way latency: 50-400 ms

Loss tolerance:

- depending on data encoding and application, losses may be concealed from user, e.g., by simply replaying the last sample
- packet loss rates between 1% and 10% may be tolerated

3

Loss Recovery

Due to the delay intolerance of multimedia application, ARQ may not be a feasible option

Requesting retransmission from the source is also not feasible for multicast and broadcast scenario

Error correcting code or forward error correction (FEC) allows for the detection and correction of errors by sending additional information

4

Introduction to Coding Theory

Main problems of information and coding theory:

1. How can we tell when data (transmitted or stored) has been corrupted?
2. How to recover the original data?

Example alternatives:

- do nothing: loss may not be discernable, e.g., concealed by interpolation
- send each bit 100 times, majority value accepted as original value at receiver
- parity bit: append one single parity bit at the end of message

Fundamental concept and trade-off:

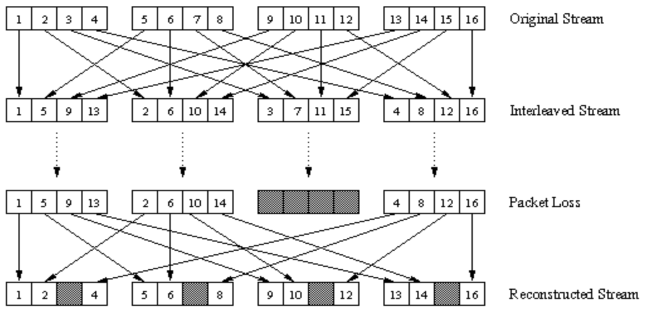
- by sending additional, redundant information, we can detect, and perhaps correct, transmission errors
- the more redundancy, the more effective in error detection/correction, but the less efficient in bandwidth

5

Do "Nothing": Interleaving

Idea: disperse the effects of packet loss

- samples are broken up into smaller units
- for example, 40 ms of sound is about 1 phoneme
- packet contains small units from different samples
- if packet is lost, still have most of every sample
- has no redundancy overhead, but adds to playout delay

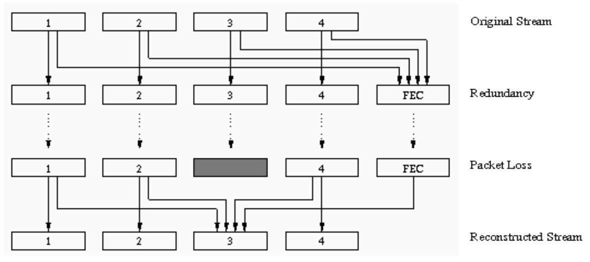


Perkins et al.

6

FEC: Simple XOR Parity Packet

XOR operation across n packets
Transmit 1 parity packet for every n data packets
If 1 in n packet is lost, can fully recover



Perkins et al.

Playout delay must accommodate receipt of all $n+1$ packets
Tradeoff: larger n

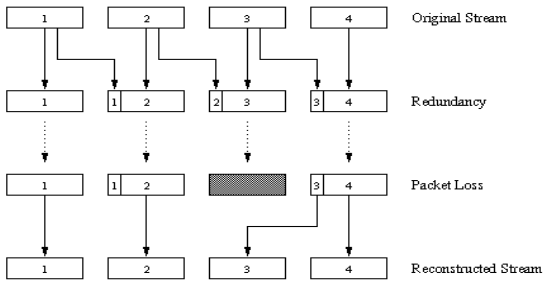
- less bandwidth "wastage," but
- longer playout delay and
- higher probability that 2 or more packets can be lost

7

FEC: Piggy-back Lower-quality Stream

Send lower resolution stream as the redundant information

- for example, nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps



Perkins et al.

- whenever there is non-consecutive loss, the receiver can conceal the loss
- can also append (n-1)st and (n-2)nd low-bit rate samples

8

FEC: Reed-Solomon Code

Based on polynomial codes over finite fields

Used in CD, DAT, DVD, Blu-ray, Compact Flash, MPEG-2 TS, DSL, RAID, WiMax, DVB, ATSC, Zattoo, the Voyager, Mars Pathfinder, Galileo, Mars Rover, etc.

Good for correcting burst errors

With dedicated hardware can achieve throughput over 600 Mbps

For more info, take EECS 554 Intro to Digital Communication and Coding . . . .

Summary: Internet Multimedia Bag of Tricks

Use UDP to avoid TCP congestion control (delays) for time-sensitive traffic

Receiver-side adaptive playout delay: to compensate for delay

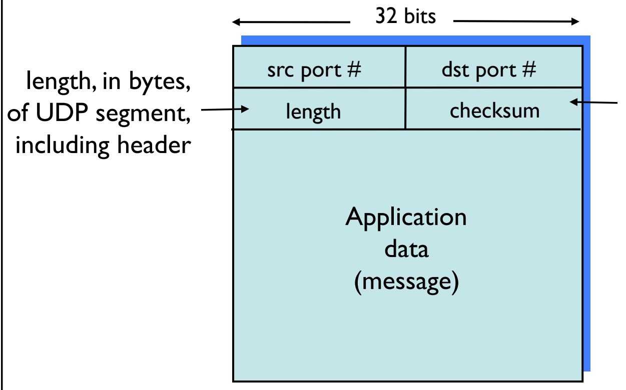
Sender-side matches stream bandwidth to available client-to-server path bandwidth

- chose among pre-encoded stream rates
- dynamic server encoding rate
- scalable/layered coding

Error recovery (on top of UDP)

- retransmissions, time permitting
- conceal errors: repeat or interpolate nearby data or interleaving
- error correction: FEC

UDP [RFC 768]


32 bits
src port #
dst port #
length
checksum
Application data (message)
length, in bytes, of UDP segment, including header

Checksum:
- checksum computed including pseudo IP header (containing src and dst IP addresses, protocol, segment length)
- computed with all 0's in the checksum field
- 1's complement of result stored in checksum field (a.k.a. Internet checksum)

⇒ when checksum is computed at receiver, result is 0

Internet Checksum Example

1's complement arithmetic: when adding numbers, a carryout from the most significant bit needs to be added to the result

Example: add two 16-bit integers

```
                            1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
                            1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
                          ---------------------------------
wraparound                1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
                          ---------------------------------
1's complement sum          1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
1's complement of sum       0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
(Internet Checksum)
```

Incremental update of checksum, e.g., updating ttl: ~ `(~checksum + ~ttl + ttl')`

Types of Sockets

Different types of sockets implement different service models

- Stream vs. datagram

Stream socket (TCP)

- connection-oriented
- reliable, in order delivery
- at-most-once delivery, no duplicates
- used by e.g., ssh, http

Datagram socket (UDP)

- connectionless (just data-transfer)
- "best-effort" delivery, possibly lower variance in delay
- used by e.g., IP telephony, streaming audio, streaming video, Internet gaming, etc.

13

Connectionless Demultiplexing

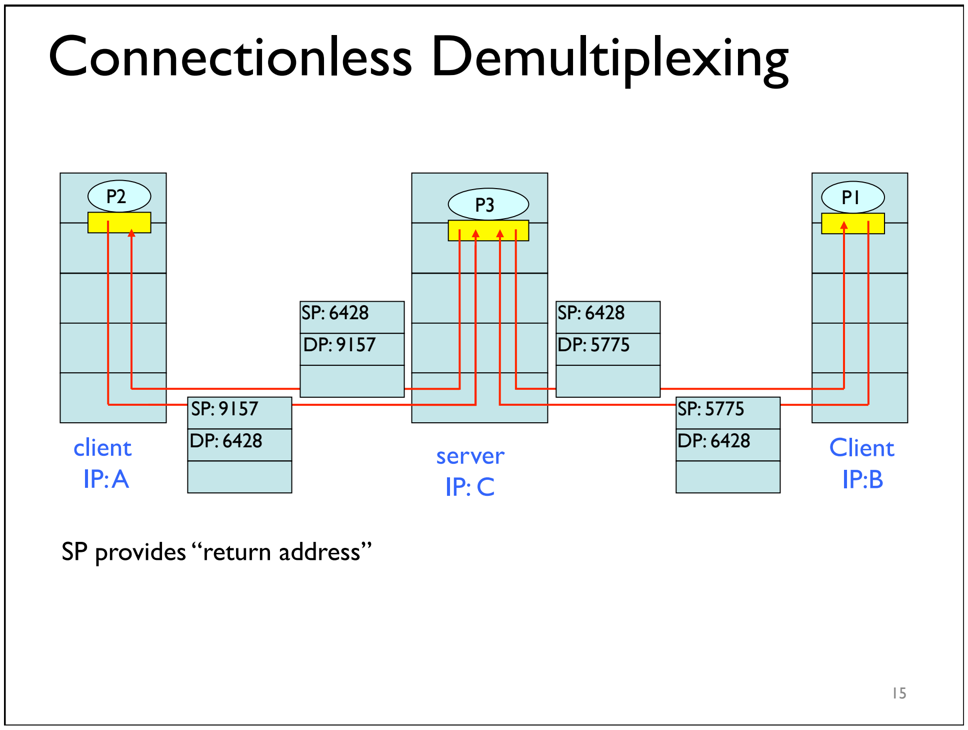
UDP socket identified by two-tuple:
<dest IP address, dest port number>
(contrast with TCP's four-tuple!)

When host receives UDP segment:

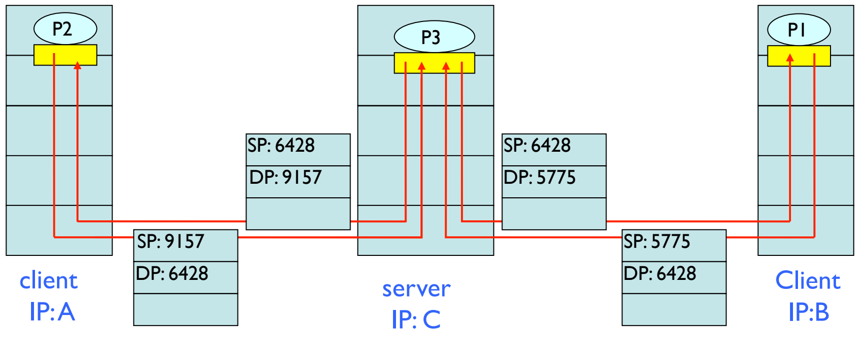
- checks destination port number in segment
- directs UDP segment to socket with that port number

⇒ IP datagrams with different source IP addresses and/or source port numbers directed to the same socket

14

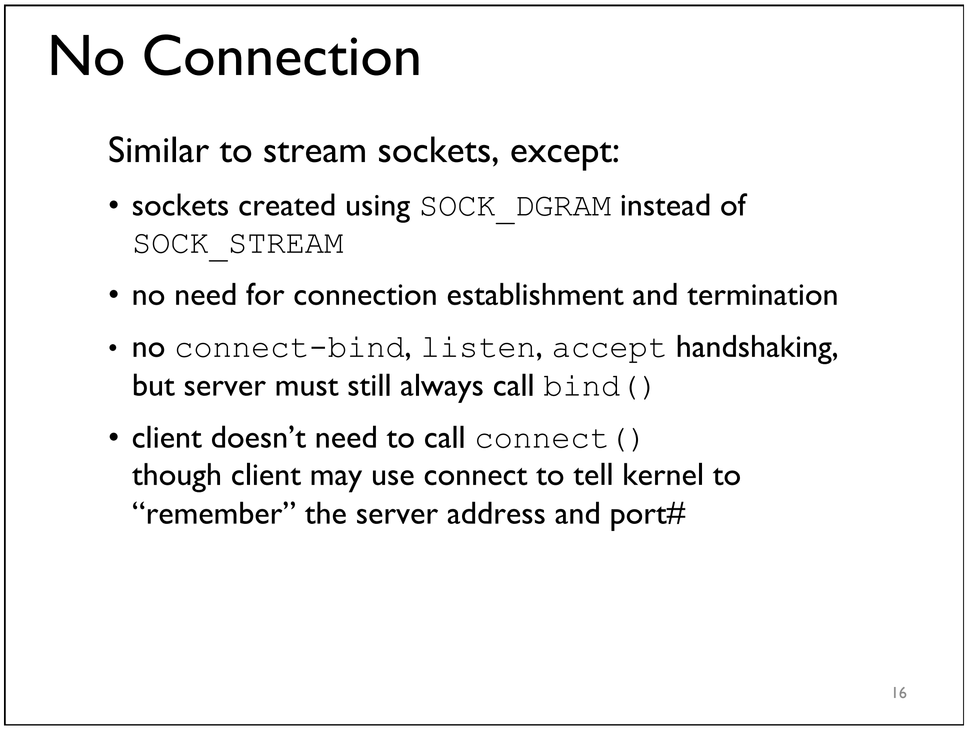


Connectionless Demultiplexing



SP provides "return address"

15



No Connection

Similar to stream sockets, except:

- sockets created using `SOCK_DGRAM` instead of `SOCK_STREAM`
- no need for connection establishment and termination
- no `connect-bind`, `listen`, `accept` handshaking, but server must still always call `bind()`
- client doesn't need to call `connect()` though client may use connect to tell kernel to "remember" the server address and port#

16

No Connection

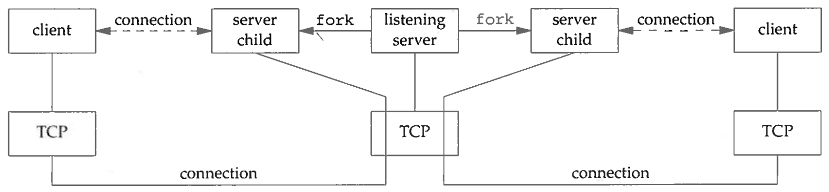


Figure 8.5 Summary of TCP client/server with two clients.

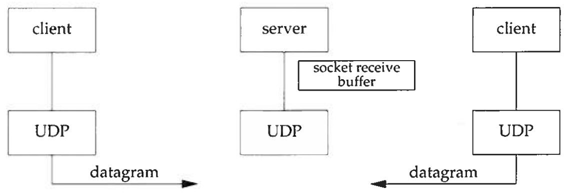


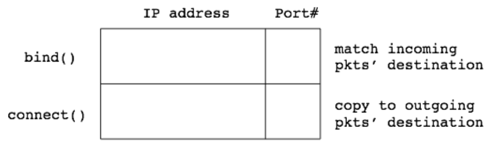
Figure 8.6 Summary of UDP client/server with two clients.

Stevens

17

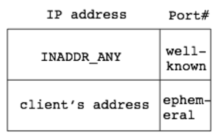
Socket Addresses

Somewhere in the socket structure:



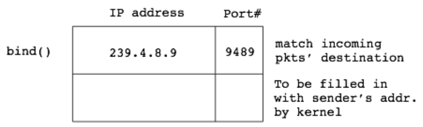
	IP address	Port#	
bind()			match incoming pkts' destination
connect()			copy to outgoing pkts' destination

TCP Server:



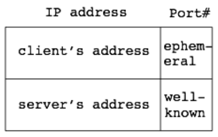
IP address	Port#
INADDR_ANY	well-known
client's address	ephem-eral

UDP Server:



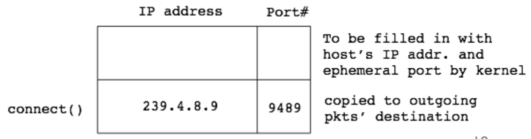
	IP address	Port#	
bind()	239.4.8.9	9489	match incoming pkts' destination
			To be filled in with sender's addr. by kernel

TCP Client:



IP address	Port#
client's address	ephem-eral
server's address	well-known

UDP Client:



	IP address	Port#	
			To be filled in with host's IP addr. and ephemeral port by kernel
connect()	239.4.8.9	9489	copied to outgoing pkts' destination

18

Data Transmission

No "connection" between client and server

- client explicitly attaches IP address and port# of destination to each packet, by using `sendto()` instead of `send()`
- if server uses `recvfrom()` it can extract IP address and port# of sender from received packet
  - if these are not needed, `recv()` may be used instead
- data sent in packets, not byte-stream oriented
- UDP packets have boundary, not a byte-stream as in TCP, so `recv()` retrieves one message at a time, i.e., no need to call `recv()` in a loop

UDP: transmitted data may be received out of order, or not received at all

19

Data Transmission

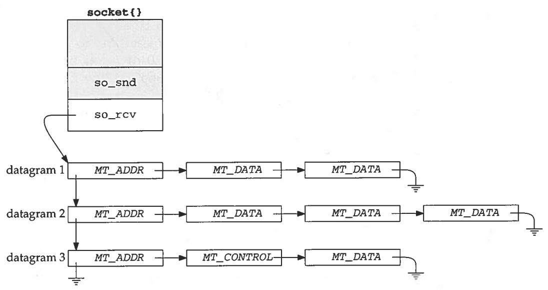


Figure 16.35 UDP receive buffer consisting of three datagrams.

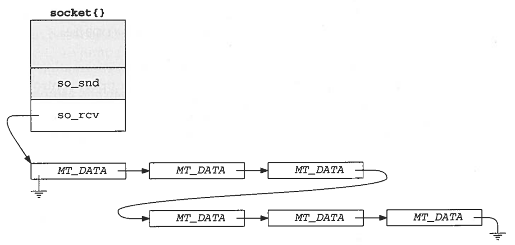


Figure 16.36 so_rcv buffer for TCP.

Stevens

20

Real-Time Protocol (RTP)

Despite the name, RTP does not provide real-time service, i.e., no guarantees on delivery time

RTP simply specifies a packet structure for packets carrying audio and video data [RFC 1889]

RTP runs in the end systems

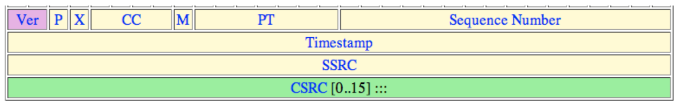
RTP packets are encapsulated in UDP segments

Why bother? Interoperability: if two Internet phone applications run RTP, they may be able to work together

Example applications using RTP: QuickTime, Apple's Darwin Streaming Server, VLC server and client, Zattoo

21

RTP Header



Version: as usual

Marker: start of talk spurt

Payload **T**ype (7 bits): type of encoding used
If sender changes encoding in middle of conference, sender informs the receiver through this payload type field

- 0: PCM mu-law, 64 kbps
- 3: GSM, 13 kbps
- 26: Motion JPEG
- 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

22

RTP Header

Timestamp (32 bytes long): reflects the sampling instant of the first byte in the RTP data packet

- Timestamp clock continues to increase at constant rate when source is inactive

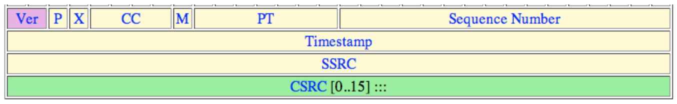
Example: a packet of 800 bytes is sent once every 100 msecs, with silence suppression:

time (in ms)	packet (seq#, timestamp) (timestamp in bytes/pkt * time(ms)/pkt gap))
0	(0,0)
100	(1,800)
200	(2,1600)
300-500	silence suppression
600	(3,4800)
700	(4,5600)
…	…

RTP marker bit would also be set on pkts (0,0) and (3, 4800)

23

RTP Header



SSRC field (32 bits): identifies the source of the RTP stream, either the data source or a (audio) mixer

CSRC (32 bits): identifies up to 16 contributing source in case of SSRC being a (audio) mixer

24

RTP Companion Protocols

RTCP (Real-Time Control Protocol): used by participants in a multimedia session to send periodic performance statistics

RTSP (Real-Time Streaming Protocol): support user interactivity in streaming stored/archival multimedia

25

Real-Time Control Protocol (RTCP)

Works in conjunction with RTP

Each participant in RTP session periodically transmits RTCP control packets to all other participants

RTCP control packet contains sender and/or receiver statistics useful to application, including

- number of packets and bytes sent,
- number of packets lost,
- interarrival jitter, etc.

Feedback can be used to control performance, e.g., sender may modify its transmissions based on feedback

To limit control traffic, each participant reduces its RTCP traffic as the number of participants increases

26

Synchronization of Streams

RTCP can be used to synchronize different media streams within an RTP session

Consider a session with separate audio and video streams

Timestamps in RTP packets are tied to the sampling clocks, not the wall-clock time

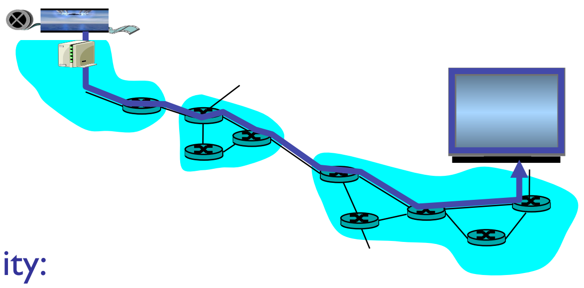
Each RTCP sender-report packet contains (for the most recently generated packet in the associated RTP stream):

- timestamp of the RTP packet
- wall-clock time for when packet was created

Receivers can use this association to synchronize the playout of audio and video

27

Streaming Stored Multimedia: Interactivity



VCR-like functionality:
client can pause, rewind, FF, push slider bar

User tolerance to interactive delay:

- 10 sec initial play out delay OK
- 1-2 sec until command effect OK

28

Interactive Control of Streaming Media RTSP [RFC 2326]

Client-server, application layer protocol

For user to control playback: rewind, fast forward, pause, resume, repositioning, etc.

RTSP messages are sent out-of-band:

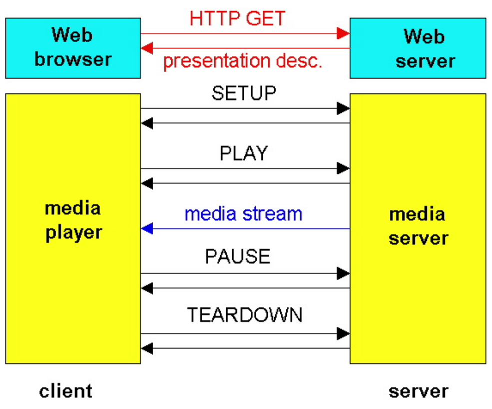
- RTSP control messages use a different port than the RTP media stream
- the RTP media stream is considered "in-band"

29

RTSP Operation Example

Scenario:

1. presentation description metafile communicated to web browser
2. browser launches media player
3. player sets up an RTSP control connection and an RTP data connection to streaming server



30

Presentation Description 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>
```

31

RTSP Exchange Example

```
C: SETUP rtsp://audio.example.com/twister/audio RTSP/1.0
   Transport: rtp/udp; compression; port=3056; mode=PLAY

S: RTSP/1.0 200 1 OK
    Session 4231

C: PLAY rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Session: 4231
    Range: npt=0-

C: PAUSE rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Session: 4231
    Range: npt=37

C: TEARDOWN rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Session: 4231

S: 200 3 OK
```

32