ECE 466/566
Advanced Computer Networks

Thinh Nguyen
Email: thinhq@eecs.orst.edu
Electrical Engineering and Computer Science
Oregon State University

TA: Monchai Lertsutthiwong
Email: lertsumo@engr.orst.edu
Office Hours: 3-5 PM.
Network Programming in C

Socket programming

**Goal:** learn how to build client/server application that communicate using sockets

**Socket API**
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

- a *host-local, application-created, OS-controlled* interface (a “door”) into which application process can both send and receive messages to/from another application process
Socket-programming using TCP

**Socket**: a door between application process and end-end-transport protocol (UCP or TCP)

**TCP service**: reliable transfer of *bytes* from one process to another

socket

controlled by application developer

controlled by operating system

TCP with buffers, variables

host or server

process

host or server

TCP with buffers, variables

controlled by application developer

controlled by operating system

internet
Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client’s contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

When contacted by client, server TCP creates new socket for server process to communicate with client

- allows server to talk with multiple clients
- source port numbers used to distinguish clients (more in Chap 3)

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server
Socket programming with TCP

Example client-server app:
1) client reads line from standard input (inFromUser stream), sends to server via socket (outToServer stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (inFromServer stream)
**Client/server socket interaction: TCP**

**Server (running on hostid)**

- create socket, port=x, for incoming request:
  - wait for incoming connection request
  - read request from Server socket
  - write reply to Server socket
  - close

**Client**

- create socket, connect to hostid, port=x
  - send request using Client socket
  - read reply from Server socket
  - close

---

**TCP connection setup**
Socket programming with UDP

UDP: no "connection" between client and server
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server
Client/server socket interaction: UDP

Server (running on hostid)

- create socket, port=x, for incoming request:
  - read request from serverSocket
  - write reply to serverSocket specifying client host address, port number

Client

- create socket, port=x,
- Create, address (hostid, port=x, send datagram request using clientSocket
- read reply from clientSocket
- close clientSocket
Example: (UDP)

Client process

Output: sends packet (TCP sent "byte stream")

Input: receives packet (TCP received "byte stream")

client UDP socket

7: Multimedia Networking  7-10
**Network Programming in C (Streaming Server)**

/*
** server.c -- a stream socket server demo
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

define MYPORT 3490    // the port users will be connecting to

define BACKLOG 10    // how many pending connections queue will hold

void sigchld_handler(int s)
{
    while(waitpid(-1, NULL, WNOHANG) > 0);
}

7: Multimedia Networking 7-11
Network Programming in C (Streaming Server)

```c
int main(void)
{
    int sockfd, new_fd; // listen on sock_fd, new connection on new_fd
    struct sockaddr_in my_addr; // my address information
    struct sockaddr_in their_addr; // connector's address information
    socklen_t sin_size;
    struct sigaction sa;
    int yes=1;

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    my_addr.sin_family = AF_INET; // host byte order
    my_addr.sin_port = htons(MYPORT); // short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
    memset(&my_addr.sin_zero, '\0', 8); // zero the rest of the struct

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }
}
```
if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
    perror("sigaction");
    exit(1);
}

while(1) {  // main accept() loop
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1) {
        perror("accept");
        continue;
    }
    printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));
    if (!fork()) { // this is the child process
        close(sockfd); // child doesn't need the listener
        if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
            perror("send");
        close(new_fd);
        exit(0);
    }
    close(new_fd); // parent doesn't need this
}
return 0;
Network Programming in C (Client)

/*
** client.c -- a stream socket client demo
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3490 // the port client will be connecting to
#define MAXDATASIZE 100 // max number of bytes we can get at once
int main(int argc, char *argv[]) 
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr; // connector's address information

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) { // get the host info
        herror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET; // host byte order
    their_addr.sin_port = htons(PORT); // short, network byte order
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // zero the rest of the struct
Network Programming in C (Client)

```c
if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
    perror("connect");
    exit(1);
}
if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
    perror("recv");
    exit(1);
}
buf[numbytes] = '\0';
printf("Received: %s",buf);
close(sockfd);
return 0;
}
```
Network Programming in C (Datagram listener)

/*
** listener.c -- a datagram sockets "server" demo
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MYPORT 4950    // the port users will be connecting to
#define MAXBUFLEN 100

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MYPORT 4950    // the port users will be connecting to
#define MAXBUFLEN 100
int main(void)
{
    int sockfd;
    struct sockaddr_in my_addr;   // my address information
    struct sockaddr_in their_addr; // connector's address information
    socklen_t addr_len;
    int numbytes;
    char buf[MAXBUFLEN];

    if ((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;    // host byte order
    my_addr.sin_port = htons(MYPORT); // short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
    memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

    if (bind(sockfd, (struct sockaddr *)&my_addr,
             sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }
}
Network Programming in C (Datagram listener)

addr_len = sizeof(struct sockaddr);
    if ((numbytes=recvfrom(sockfd, buf, MAXBUFLEN-1, 0,
        (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }

    printf("got packet from \%s\n",inet_ntoa(their_addr.sin_addr));
    printf("packet is \%d bytes long\n",numbytes);
    buf[numbytes] = '\0';
    printf("packet contains \"\%s\"\n",buf);

    close(sockfd);

    return 0;
}
/*
** talker.c -- a datagram "client" demo
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SERVERPORT 4950    // the port users will be connecting to

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SERVERPORT 4950    // the port users will be connecting to
int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in their_addr; // connector's address information
    struct hostent *he;
    int numbytes;

    if (argc != 3) {
        fprintf(stderr,"usage: talker hostname message\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  // get the host info
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
their_addr.sin_family = AF_INET; // host byte order
their_addr.sin_port = htons(SERVERPORT); // short, network byte order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(their_addr.sin_zero), '\0', 8); // zero the rest of the struct

if ((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0,
    (struct sockaddr *)&their_addr, sizeof(struct sockaddr))) == -1) {
        perror("sendto");
        exit(1);
    }

printf("sent %d bytes to %s\n", numbytes, inet_ntoa(their_addr.sin_addr));
close(sockfd);
return 0;
**Multimedia, Quality of Service: What is it?**

**Multimedia applications:**
network audio and video ("continuous media")

**QoS**

network provides application with *level of performance needed for application to function.*
Chapter 7: Goals

Principles
- Classify multimedia applications
- Identify the network services the apps need
- Making the best of best effort service
- Mechanisms for providing QoS

Protocols and Architectures
- Specific protocols for best-effort
- Architectures for QoS
Chapter 7 outline

- 7.1 Multimedia Networking Applications
- 7.2 Streaming stored audio and video
- 7.3 Real-time Multimedia: Internet Phone study
- 7.4 Protocols for Real-Time Interactive Applications
  - RTP, RTCP, SIP
- 7.5 Distributing Multimedia: content distribution networks
- 7.6 Beyond Best Effort
- 7.7 Scheduling and Policing Mechanisms
- 7.8 Integrated Services and Differentiated Services
- 7.9 RSVP
MM Networking Applications

Classes of MM applications:
1) Streaming stored audio and video
2) Streaming live audio and video
3) Real-time interactive audio and video

Jitter is the variability of packet delays within the same packet stream

Fundamental characteristics:
- Typically delay sensitive
  - end-to-end delay
  - delay jitter
- But loss tolerant: infrequent losses cause minor glitches
- Antithesis of data, which are loss intolerant but delay tolerant.
Streaming Stored Multimedia

Streaming:
- media stored at source
- transmitted to client
- streaming: client playout begins \textit{before} all data has arrived
  - timing constraint for still-to-be transmitted data: in time for playout
Streaming Stored Multimedia: What is it?

1. Video recorded
2. Video sent
3. Video received, played out at client

Streaming: at this time, client playing out early part of video, while server still sending later part of video.
Streaming Stored Multimedia: Interactivity

- **VCR-like functionality**: client can pause, rewind, FF, push slider bar
  - 10 sec initial delay OK
  - 1-2 sec until command effect OK
  - RTSP often used (more later)

- **timing constraint for still-to-be transmitted data**: in time for playout
Streaming Live Multimedia

Examples:
- Internet radio talk show
- Live sporting event

Streaming
- playback buffer
- playback can lag tens of seconds after transmission
- still have timing constraint

Interactivity
- fast forward impossible
- rewind, pause possible!
Interactive, Real-Time Multimedia

- **applications**: IP telephony, video conference, distributed interactive worlds

- **end-end delay requirements**:
  - audio: < 150 msec good, < 400 msec OK
    - includes application-level (packetization) and network delays
    - higher delays noticeable, impair interactivity

- **session initialization**
  - how does callee advertise its IP address, port number, encoding algorithms?
Multimedia Over Today’s Internet

TCP/UDP/IP: “best-effort service”

- no guarantees on delay, loss

Today’s Internet multimedia applications use application-level techniques to mitigate (as best possible) effects of delay, loss.

But you said multimedia apps require QoS and level of performance to be effective!
How should the Internet evolve to better support multimedia?

Integrated services philosophy:
- Fundamental changes in Internet so that apps can reserve end-to-end bandwidth
- Requires new, complex software in hosts & routers

Laissez-faire
- no major changes
- more bandwidth when needed
- content distribution, application-layer multicast
  - application layer

Differentiated services philosophy:
- Fewer changes to Internet infrastructure, yet provide 1st and 2nd class service.

What’s your opinion?
Multimedia Introduction
Analog to Digital Conversion

- Loss from A/D conversion
  - Aliasing (worse than quantization loss) due to sampling
  - Loss due to quantization
Analog to Digital Conversion

- Loss from A/D conversion.

- For perfect re-construction, sampling rate (Nyquist's frequency) needs to be twice the maximum frequency of the signal.

- However, in practice, loss still occurs due to quantization.

- Finer quantization leads to less error at the expense of increased number of bits to represent signals.
Audio Compression

- Analog signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec

- Each sample quantized, i.e., rounded
  - e.g., $2^8 = 256$ possible quantized values

- Each quantized value represented by bits
  - 8 bits for 256 values

- Example: 8,000 samples/sec, 256 quantized values --> 64,000 bps

- Receiver converts it back to analog signal:
  - some quality reduction

Example rates
- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 - 13 kbps
PCM (Pulse Code Modulation)

- The 2 step process of sampling and quantization is known as **Pulse Code Modulation**.
- Used in speech and CD recording.

Much less compression than MP3
DPCM (Differential Pulse Code Modulation)

**Coder**

**Decoder**

Prediction error

\[ e = s - \hat{s} \]

Reconstruction

\[ s' = e' + \hat{s} \]

Reconstruction error = quantization error

\[ s' - s = e' - e = q \]
DPCM

Simple example:

Code the following value sequence:
- 1.4  1.75  2.05  2.5  2.4
- Quantization step: 0.2
- Predictor: current value = previous quantized value + quantized error.

- Error = 1.75 - 1.4 = 0.35 => 0.4
- Prediction value = 1.4 + 0.4 = 1.8

- Error = 2.05 - 1.8 = => 0.25 => 0.2
- Prediction value = 1.8 + .2 = 2.0

- Error = 2.5 - 2.05 = 0.45 => 0.4
- Prediction value = 2.0 + .4 = 2.4

Send 0.4, 0.2, 0.4, …
Video Compression

- Video is sequence of images displayed at constant rate
  - e.g. 24 images/sec
- Digital image is array of pixels
- Each pixel represented by bits
- Redundancy
  - spatial
  - temporal

Examples:
- MPEG 1 (CD-ROM) 1.5 Mbps
- MPEG2 (DVD) 3-6 Mbps
- MPEG4 (often used in Internet, < 1 Mbps)

Research:
- Layered (scalable) video
  - adapt layers to available bandwidth
Video Coding Standards

- Mobile videophone
- Videophone over PSTN
- ISDN videophone
- Video CD
- Digital TV
- HDTV

Bitrates:
- Very low bitrate: 8 kbit/s
- Low bitrate: 16 kbit/s
- Medium bitrate: 64 kbit/s
- High bitrate: 384 kbit/s
- Very high bitrate: 1.5 Mbit/s
- Medium high bitrate: 5 Mbit/s
- High speed: 20 Mbit/s

Video Coding Standards:
- MPEG-4
- H.263
- H.261
- MPEG-1
- MPEG-2
MPEG Frame Types

- Intra-coded I-frame
- Predictively coded P-frames
- Bi-directionally predictively coded B-frames

Group of frames (GOF)
Motion estimation for different frames

Available from earlier frame (X)

Available from later frame (Z)
Motion Compensation
Motion Compensation

- Typically one motion vector per macroblock (4 transform blocks)
- Motion estimation is a time consuming process
  - Hierarchical motion estimation
  - Maximum length of motion vectors
  - Clever search strategies
- Motion vector accuracy:
  - Integer, half or quarter pixel
  - Bilinear interpolation
Motion Compensation

Motion Vectors
- static background is a very special case, we should consider the displacement of the block.
- Motion vector is used to inform decoder exactly where in the previous image to get the data.
- Motion vector would be zero for a static background.
Motion Compensation

Block Matching--how to find the matching block?

- **Matching criteria:**
  - In practice we couldn’t expect to find the exactly identical matching block, instead we look for close match.
  - Most motion estimation schemes look for minimum mean square error (MMSE) between block.

\[
MSE = \frac{1}{N} \sum_{n=1}^{N} (I_n(x, y) - I'_n(x, y))^2
\]

- **Matching block size:**
  - How large the matching block will affect coding efficiency
  - block size MPEG used: 16×16
A Simplified MPEG encoder
Examples video frames resulted from transmission errors
Chapter 7 outline

- 7.1 Multimedia Networking Applications
- 7.2 Streaming stored audio and video
- 7.3 Real-time Multimedia: Internet Phone study
- 7.4 Protocols for Real-Time Interactive Applications
  - RTP, RTCP, SIP
- 7.5 Distributing Multimedia: content distribution networks
- 7.6 Beyond Best Effort
- 7.7 Scheduling and Policing Mechanisms
- 7.8 Integrated Services and Differentiated Services
- 7.9 RSVP
Streaming Stored Multimedia

Application-level streaming techniques for making the best out of best effort service:
- client side buffering
- use of UDP versus TCP
- multiple encodings of multimedia

Media Player
- jitter removal
- decompression
- error concealment
- graphical user interface w/ controls for interactivity
Internet multimedia: simplest approach

- audio or video stored in file
- files transferred as HTTP object
  - received in entirety at client
  - then passed to player

audio, video not streamed:
- no, “pipelining,” long delays until playout!
Internet multimedia: streaming approach

- browser GETs metafile
- browser launches player, passing metafile
- player contacts server
- server streams audio/video to player
Streaming from a streaming server

- This architecture allows for non-HTTP protocol between server and media player.
- Can also use UDP instead of TCP.
Streaming Multimedia: Client Buffering

- Client-side buffering, playout delay compensate for network-added delay, delay jitter
Streaming Multimedia: Client Buffering

- Client-side buffering, playout delay compensate for network-added delay, delay jitter

Diagram:
- From network to client buffer
- Variable fill rate, \( x(t) \)
- Constant drain rate, \( d \)
- Buffered video
- To decompression and playout
Streaming Multimedia: UDP or TCP?

**UDP**
- server sends at rate appropriate for client (oblivious to network congestion!)
  - often send rate = encoding rate = constant rate
  - then, fill rate = constant rate - packet loss
- short playout delay (2-5 seconds) to compensate for network delay jitter
- error recover: time permitting

**TCP**
- send at maximum possible rate under TCP
- fill rate fluctuates due to TCP congestion control
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls
Streaming Multimedia: client rate(s)

Q: how to handle different client receive rate capabilities?
   - 28.8 Kbps dialup
   - 100Mbps Ethernet

A: server stores, transmits multiple copies of video, encoded at different rates
User Control of Streaming Media: RTSP

HTTP
- Does not target multimedia content
- No commands for fast forward, etc.

RTSP: RFC 2326
- Client-server application layer protocol.
- For user to control display: rewind, fast forward, pause, resume, repositioning, etc...

What it doesn’t do:
- does not define how audio/video is encapsulated for streaming over network
- does not restrict how streamed media is transported; it can be transported over UDP or TCP
- does not specify how the media player buffers audio/video
RTSP: out of band control

FTP uses an “out-of-band” control channel:
- A file is transferred over one TCP connection.
- Control information (directory changes, file deletion, file renaming, etc.) is sent over a separate TCP connection.
- The “out-of-band” and “in-band” channels use different port numbers.

RTSP messages are also sent out-of-band:
- RTSP control messages use different port numbers than the media stream: out-of-band.
  - Port 554
- The media stream is considered “in-band”.
RTSP Example

Scenario:
- metafile communicated to web browser
- browser launches player
- player sets up an RTSP control connection, data connection to streaming server
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>
RTSP Operation
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
Chapter 7 outline

- 7.1 Multimedia Networking Applications
- 7.2 Streaming stored audio and video
- 7.3 Real-time Multimedia: Internet Phone case study
- 7.4 Protocols for Real-Time Interactive Applications
  - RTP, RTCP, SIP
- 7.5 Distributing Multimedia: content distribution networks
- 7.6 Beyond Best Effort
- 7.7 Scheduling and Policing Mechanisms
- 7.8 Integrated Services and Differentiated Services
- 7.9 RSVP
Real-time interactive applications

- PC-2-PC phone
  - instant messaging services are providing this
- PC-2-phone
  - Skype
  - Net2phone
- videoconference with Webcams

Going to now look at a PC-2-PC Internet phone example in detail
Interactive Multimedia: Internet Phone

Introduce Internet Phone by way of an example

- speaker’s audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
- pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes data
- application-layer header added to each chunk.
- Chunk+header encapsulated into UDP segment.
- application sends UDP segment into socket every 20 msec during talkspurt.
Internet Phone: Packet Loss and Delay

- **network loss**: IP datagram lost due to network congestion (router buffer overflow)

- **delay loss**: IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400 ms

- **loss tolerance**: depending on voice encoding, losses concealed, packet loss rates between 1% and 10% can be tolerated.
Delay Jitter

Consider the end-to-end delays of two consecutive packets: difference can be more or less than 20 msec
Internet Phone: Fixed Playout Delay

- Receiver attempts to playout each chunk exactly $q$ msecs after chunk was generated.
  - chunk has time stamp $t$: play out chunk at $t+q$.
  - chunk arrives after $t+q$: data arrives too late for playout, data “lost”
- Tradeoff for $q$:
  - large $q$: less packet loss
  - small $q$: better interactive experience
Fixed Playout Delay

- Sender generates packets every 20 msec during talk spurt.
- First packet received at time $r$
- First playout schedule: begins at $p$
- Second playout schedule: begins at $p'$

\[ p' - r \]

\[ p - r \]
Adaptive Playout Delay, I

- **Goal:** minimize playout delay, keeping late loss rate low
- **Approach:** adaptive playout delay adjustment:
  - Estimate network delay, adjust playout delay at beginning of each talk spurt.
  - Silent periods compressed and elongated.
  - Chunks still played out every 20 msec during talk spurt.

\[
\hat{t}_i = \text{timestamp of the } i\text{th packet} \\
\hat{r}_i = \text{the time packet } i \text{ is received by receiver} \\
\hat{p}_i = \text{the time packet } i \text{ is played at receiver} \\
\hat{r}_i - \hat{t}_i = \text{network delay for } i\text{th packet} \\
\hat{d}_i = \text{estimate of average network delay after receiving } i\text{th packet}
\]

**Dynamic estimate of average delay at receiver:**

\[
\hat{d}_i = (1 - u)\hat{d}_{i-1} + u(\hat{r}_i - \hat{t}_i)
\]

where \( u \) is a fixed constant (e.g., \( u = .01 \)).
Adaptive playout delay II

Also useful to estimate the average deviation of the delay, $v_i$:

$$v_i = (1 - u)v_{i-1} + u | r_i - t_i - d_i |$$

The estimates $d_i$ and $v_i$ are calculated for every received packet, although they are only used at the beginning of a talk spurt.

For first packet in talk spurt, playout time is:

$$p_i = t_i + d_i + Kv_i$$

where $K$ is a positive constant.

Remaining packets in talkspurt are played out periodically.
Adaptive Playout, III

Q: How does receiver determine whether packet is first in a talkspurt?

- If no loss, receiver looks at successive timestamps.
  - difference of successive stamps > 20 msec $\rightarrow$ talk spurt begins.

- With loss possible, receiver must look at both time stamps and sequence numbers.
  - difference of successive stamps > 20 msec and sequence numbers without gaps $\rightarrow$ talk spurt begins.
Recovery from packet loss (1)

**Forward error correction (FEC): simple scheme**

- For every group of \( n \) chunks, create a redundant chunk by exclusive OR-ing the \( n \) original chunks.
- Send out \( n+1 \) chunks, increasing the bandwidth by factor \( 1/n \).
- Can reconstruct the original \( n \) chunks if there is at most one lost chunk from the \( n+1 \) chunks.

- Playout delay needs to be fixed to the time to receive all \( n+1 \) packets.
- Tradeoff:
  - Increase \( n \), less bandwidth waste
  - Increase \( n \), longer playout delay
  - Increase \( n \), higher probability that 2 or more chunks will be lost
Recovery from packet loss (2)

2nd FEC scheme
• “piggyback lower quality stream”
• send lower resolution audio stream as the redundant information
• for example, nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps.

• 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 chunk
Recovery from packet loss (3)

Interleaving
- chunks are broken up into smaller units
- for example, 4 5 msec units per chunk
- Packet contains small units from different chunks

- if packet is lost, still have most of every chunk
- has no redundancy overhead
- but adds to playout delay
Summary: Internet Multimedia: bag of tricks

- **use UDP** to avoid TCP congestion control (delays) for time-sensitive traffic
- **client-side adaptive playout delay**: to compensate for delay
- **server side** matches stream bandwidth to available client-to-server path bandwidth
  - chose among pre-encoded stream rates
  - dynamic server encoding rate
- **error recovery** (on top of UDP)
  - FEC, interleaving
  - retransmissions, time permitting
  - conceal errors: repeat nearby data
Chapter 7 outline

- 7.1 Multimedia Networking Applications
- 7.2 Streaming stored audio and video
- 7.3 Real-time Multimedia: Internet Phone study
- 7.4 Protocols for Real-Time Interactive Applications
  - RTP, RTCP, SIP
- 7.5 Distributing Multimedia: content distribution networks
- 7.6 Beyond Best Effort
- 7.7 Scheduling and Policing Mechanisms
- 7.8 Integrated Services and Differentiated Services
- 7.9 RSVP
Real-Time Protocol (RTP)

- RTP specifies a packet structure for packets carrying audio and video data
- RFC 1889.
- RTP packet provides
  - payload type identification
  - packet sequence numbering
  - timestamping
- RTP runs in the end systems.
- RTP packets are encapsulated in UDP segments
- Interoperability: If two Internet phone applications run RTP, then they may be able to work together
RTP runs on top of UDP

RTP libraries provide a transport-layer interface that extend UDP:

- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping
RTP Example

- Consider sending 64 kbps PCM-encoded voice over RTP.
- Application collects the encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk.
- The audio chunk along with the RTP header form the RTP packet, which is encapsulated into a UDP segment.
- RTP header indicates type of audio encoding in each packet:
  - sender can change encoding during a conference.
- RTP header also contains sequence numbers and timestamps.
**RTP and QoS**

- RTP does **not** provide any mechanism to ensure timely delivery of data or provide other quality of service guarantees.
- RTP encapsulation is only seen at the end systems: it is not seen by intermediate routers.
  - Routers providing best-effort service do not make any special effort to ensure that RTP packets arrive at the destination in a timely matter.
RTP Header

Payload Type (7 bits): Indicates type of encoding currently being used. If sender changes encoding in middle of conference, sender informs the receiver through this payload type field.

- Payload type 0: PCM mu-law, 64 kbps
- Payload type 3, GSM, 13 kbps
- Payload type 7, LPC, 2.4 kbps
- Payload type 26, Motion JPEG
- Payload type 31, H.261
- Payload type 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.
RTP Header (2)

- **Timestamp field (32 bytes long)**. Reflects the sampling instant of the first byte in the RTP data packet.
  - For audio, timestamp clock typically increments by one for each sampling period (for example, each 125 usecs for a 8 KHz sampling clock)
  - If application generates chunks of 160 encoded samples, then timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.

- **SSRC field (32 bits long)**. Identifies the source of the RTP stream. Each stream in a RTP session should have a distinct SSRC.
Real-Time Control Protocol (RTCP)

- Works in conjunction with RTP.
- Each participant in RTP session periodically transmits RTCP control packets to all other participants.
- Each RTCP packet contains sender and/or receiver reports
  - report statistics useful to application
- Statistics include number of packets sent, number of packets lost, interarrival jitter, etc.
- Feedback can be used to control performance
  - Sender may modify its transmissions based on feedback
- For an RTP session there is typically a single multicast address; all RTP and RTCP packets belonging to the session use the multicast address.

- RTP and RTCP packets are distinguished from each other through the use of distinct port numbers.

- To limit traffic, each participant reduces his RTCP traffic as the number of conference participants increases.
RTCP Packets

Receiver report packets:
- fraction of packets lost, last sequence number, average interarrival jitter.

Sender report packets:
- SSRC of the RTP stream, the current time, the number of packets sent, and the number of bytes sent.

Source description packets:
- e-mail address of sender, sender's name, SSRC of associated RTP stream.
- Provide mapping between the SSRC and the user/host name.
Synchronization of Streams

- RTCP can synchronize different media streams within a RTP session.
- Consider videoconferencing app for which each sender generates one RTP stream for video and one for audio.
- Timestamps in RTP packets tied to the video and audio sampling clocks
  - not tied to 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.
RTCP Bandwidth Scaling

- RTCP attempts to limit its traffic to 5% of the session bandwidth.

**Example**

- Suppose one sender, sending video at a rate of 2 Mbps. Then RTCP attempts to limit its traffic to 100 Kbps.
- RTCP gives 75% of this rate to the receivers; remaining 25% to the sender.

- The 75 kbps is equally shared among receivers:
  - With R receivers, each receiver gets to send RTCP traffic at 75/R kbps.
- Sender gets to send RTCP traffic at 25 kbps.
- Participant determines RTCP packet transmission period by calculating avg RTCP packet size (across the entire session) and dividing by allocated rate.