ECE 466/566
Advanced Computer Networks

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

application viewpoint

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

Client process

Output: sends packet (TCP sent “byte stream”)

Input: receives packet (TCP received “byte stream”)

keyboard     monitor

inFromUser

client UDP socket

receivePacket

UDP packet

to network

sendPacket

input stream

UDP packet

from network
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
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 <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) { // socket error!
        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
}
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
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
Network Programming in C (Datagram talker)

```c
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);
    }
}
```
Network Programming in C (Datagram talker)

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

**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.

[Jitter is the variability of packet delays within the same packet stream]
Streaming Stored Multimedia

Streaming:
- media stored at source
- transmitted to client
- streaming: client playout begins *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

But you said multimedia apps requires QoS and level of performance to be effective!

Today’s Internet multimedia applications use application-level techniques to mitigate (as best possible) effects of delay, loss
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?
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 => .4
  - Prediction value = 1.4 + 0.4 = 1.8

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

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

7: Multimedia Networking  7-51
**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**

```xml
<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

HTTP GET
presentation desc.

Web browser

Web server

media player

media server

SETUP

PLAY

media stream

PAUSE

TEARDOWN

client

server
**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
  - Dialpad
  - 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.
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 $p'$.
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.

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

where \(u\) is a fixed constant (e.g., \(u = .01\)).

\(t_i = \) timestamp of the \(i\)th packet
\(r_i = \) the time packet \(i\) is received by receiver
\(p_i = \) the time packet \(i\) is played at receiver
\(r_i - t_i = \) network delay for \(i\)th packet
\(d_i = \) estimate of average network delay after receiving \(i\)th packet
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 --> 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 --> 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
RTCP - Continued

- 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.
SIP

- Session Initiation Protocol
- Comes from IETF

SIP long-term vision

- All telephone calls and video conference calls take place over the Internet
- People are identified by names or e-mail addresses, rather than by phone numbers.
- You can reach the callee, no matter where the callee roams, no matter what IP device the callee is currently using.
SIP Services

- Setting up a call
  - Provides mechanisms for caller to let callee know she wants to establish a call
  - Provides mechanisms so that caller and callee can agree on media type and encoding.
  - Provides mechanisms to end call.

- Determine current IP address of callee.
  - Maps mnemonic identifier to current IP address

- Call management
  - Add new media streams during call
  - Change encoding during call
  - Invite others
  - Transfer and hold calls
Setting up a call to a known IP address

- Alice’s SIP invite message indicates her port number & IP address. Indicates encoding that Alice prefers to receive (PCM ulaw)

- Bob’s 200 OK message indicates his port number, IP address & preferred encoding (GSM)

- SIP messages can be sent over TCP or UDP; here sent over RTP/UDP.

- Default SIP port number is 5060.
Setting up a call (more)

- Codec negotiation:
  - Suppose Bob doesn’t have PCM ulaw encoder.
  - Bob will instead reply with 606 Not Acceptable Reply and list encoders he can use.
  - Alice can then send a new INVITE message, advertising an appropriate encoder.

- Rejecting the call
  - Bob can reject with replies “busy,” “gone,” “payment required,” “forbidden”.

- Media can be sent over RTP or some other protocol.
Example of SIP message

INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

```
c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

Notes:
- HTTP message syntax
- sdp = session description protocol
- Call-ID is unique for every call.

- Here we don't know Bob's IP address. Intermediate SIP servers will be necessary.
- Alice sends and receives SIP messages using the SIP default port number 506.
- Alice specifies in Via: header that SIP client sends and receives SIP messages over UDP.
Name translation and user location

- Caller wants to call callee, but only has callee’s name or e-mail address.
- Need to get IP address of callee’s current host:
  - user moves around
  - DHCP protocol
  - user has different IP devices (PC, PDA, car device)
- Result can be based on:
  - time of day (work, home)
  - caller (don’t want boss to call you at home)
  - status of callee (calls sent to voicemail when callee is already talking to someone)

Service provided by SIP servers:
- SIP registrar server
- SIP proxy server
SIP Registrar

- When Bob starts SIP client, client sends SIP REGISTER message to Bob’s registrar server
  (similar function needed by Instant Messaging)

Register Message:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```
SIP Proxy

- Alice sends invite message to her proxy server
  - contains address sip:bob@domain.com
- Proxy responsible for routing SIP messages to callee
  - possibly through multiple proxies.
- Callee sends response back through the same set of proxies.
- Proxy returns SIP response message to Alice
  - contains Bob’s IP address

- Note: proxy is analogous to local DNS server
**Example**

Caller jim@umass.edu with places a call to keith@upenn.edu

(1) Jim sends INVITE message to umass SIP proxy. (2) Proxy forwards request to upenn registrar server. (3) upenn server returns redirect response, indicating that it should try keith@eurecom.fr

(4) umass proxy sends INVITE to eurecom registrar. (5) eurecom registrar forwards INVITE to 197.87.54.21, which is running keith’s SIP client. (6-8) SIP response sent back (9) media sent directly between clients.

**Note:** also a SIP ack message, which is not shown.
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
Content distribution networks (CDNs)

Content replication

- Challenging to stream large files (e.g., video) from single origin server in real time
- Solution: replicate content at hundreds of servers throughout Internet
  - Content downloaded to CDN servers ahead of time
  - Placing content “close” to user avoids impairments (loss, delay) of sending content over long paths
  - CDN server typically in edge/access network
Content distribution networks (CDNs)

**Content replication**

- CDN (e.g., Akamai) customer is the content provider (e.g., CNN)
- CDN replicates customers’ content in CDN servers. When provider updates content, CDN updates servers.
**CDN example**

1. **Origin server (www.foo.com)**
   - distributes HTML

2. **CDNs authoritative DNS server**
   - DNS query for www.cdn.com

3. **Nearby CDN server**

**CDN company (cdn.com)**
- distributes gif files
- uses its authoritative DNS server to route redirect requests
More about CDNs

**routing requests**
- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses “map” to determine best CDN server
- CDN nodes create application-layer overlay network
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
Improving QoS in IP Networks

Thus far: “making the best of best effort”
Future: next generation Internet with QoS guarantees
  - RSVP: signaling for resource reservations
  - Differentiated Services: differential guarantees
  - Integrated Services: firm guarantees

simple model for sharing and congestion studies:
Principles for QoS Guarantees

- Example: 1Mbps, IP phone, FTP share 1.5 Mbps link.
  - bursts of FTP can congest router, cause audio loss
  - want to give priority to audio over FTP

Principle 1
packet marking needed for router to distinguish between different classes; and new router policy to treat packets accordingly
Principles for QoS Guarantees (more)

- what if applications misbehave (audio sends higher than declared rate)
  - policing: force source adherence to bandwidth allocations
- marking and policing at network edge:
  - similar to ATM UNI (User Network Interface)

Principle 2

provide protection (isolation) for one class from others
Principles for QOS Guarantees (more)

- Allocating fixed (non-sharable) bandwidth to flow: inefficient use of bandwidth if flows doesn’t use its allocation

Principle 3

While providing isolation, it is desirable to use resources as efficiently as possible
Principles for QOS Guarantees (more)

- *Basic fact of life:* can not support traffic demands beyond link capacity

**Principle 4**

*Call Admission:* flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs
Summary of QoS Principles

Let's next look at mechanisms for achieving this ....
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
Scheduling And Policing Mechanisms

- **scheduling**: choose next packet to send on link
- **FIFO (first in first out) scheduling**: send in order of arrival to queue
  - real-world example?
  - **discard policy**: if packet arrives to full queue: who to discard?
    - Tail drop: drop arriving packet
    - priority: drop/remove on priority basis
    - random: drop/remove randomly

![Diagram of queue and link](image-url)
Router Support For Congestion Management

- Traditional Internet
  - Congestion control mechanisms at end-systems, mainly implemented in TCP
  - Routers play little role

- Router mechanisms affecting congestion management
  - Scheduling
  - Buffer management

- Traditional routers
  - FIFO
  - Tail drop
Drawbacks of FIFO with Tail-drop

- Buffer lock out by misbehaving flows
- Synchronizing effect for multiple TCP flows
- Burst or multiple consecutive packet drops
  - Bad for TCP fast recovery
FIFO Router with Two TCP Sessions

![Graph showing packet sequence number and number of packets over time for Flow 1 and Flow 2.]

![Graph showing queue size over time.]

7: Multimedia Networking 7-107
RED (Random Early Detection)

- FIFO scheduling
- Buffer management:
  - Probabilistically discard packets
  - Probability is computed as a function of **average** queue length (why average?)

![Discard Probability Graph]

- Discard Probability
- Queue Length
- min_th, max_th, queue_len, Average Queue Length
- Probability is computed as a function of average queue length.
RED (cont’d)

- min_th - minimum threshold
- max_th - maximum threshold
- avg_len - average queue length
  \[ \text{avg}\_\text{len} = (1-w)\times\text{avg}\_\text{len} + w\times\text{sample}\_\text{len} \]
If (avg_len < min_th) → enqueue packet
If (avg_len > max_th) → drop packet
If (avg_len >= min_th and avg_len < max_th) → discard packet with probability $P$
RED (cont’d)

- $P = \text{max}_P \times (\text{avg} \_\text{len} - \text{min} \_\text{th}) / (\text{max} \_\text{th} - \text{min} \_\text{th})$

- Improvements to spread the drops
  
  $P' = P / (1 - \text{count} \times P)$, where
  
  - count - how many packets were consecutively enqueued since last drop
RED Advantages

- Absorb burst better
- Avoids synchronization
- Signal end systems earlier
RED Router with Two TCP Sessions
Scheduling Policies: more

Priority scheduling: transmit highest priority queued packet

- multiple classes, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc..
  - Real world example?
Scheduling Policies: still more

round robin scheduling:
- multiple classes
- cyclically scan class queues, serving one from each class (if available)
- real world example?
Scheduling Policies: bit-by-bit round robins

round robin scheduling:
- What if packets are not the same size?
- Bit-by-bit round robin

\[ P_i = \text{length}, \]
\[ A_i = \text{arrival time} \]
\[ S_i = \text{begin transmit time} \]
\[ F_i = \text{finish transmit time} \]

\[ F_i = S_i + P_i = \max (F_{i-1}, A_i) + P_i \]

Transmit packet with lowest \( F_i \)
Fair Rate Computation

- Denote
  - \( C \) - link capacity
  - \( N \) - number of flows
  - \( r_i \) - arrival rate

- Max-min fair rate computation:
  1. compute \( C/N \)
  2. if there are flows \( i \) such that \( r_i \leq C/N \), update \( C \) and \( N \)
     \[ C = C - \sum_{i \text{ s.t. } r_i \leq C} r_i \]
  3. if no, \( f = C/N \), terminate
  4. go to 1

- A flow can receive at most the fair rate, i.e., \( \min(f, r_i) \)
Example of Fair Rate Computation

- $C = 10; r_1 = 8, r_2 = 6, r_3 = 2; N = 3$
- $C/3 = 3.33 \rightarrow C = C - r_3 = 8; N = 2$
- $C/2 = 4; f = 4$

\[\begin{align*}
8 &\quad 6 &\quad 2 \\
&\quad 10 \quad 4 &\quad 4 &\quad 4 &\quad 2 \\
\end{align*}\]

\[\begin{align*}
&f = 4: \\
&\text{min}(8, 4) = 4 \\
&\text{min}(6, 4) = 4 \\
&\text{min}(2, 4) = 2
\end{align*}\]
Scheduling Policies: Weighted Fair Queuing

Weighted Fair Queuing:
- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?

\[ R_i = B \frac{w_i}{\sum w_i} \]
Max-Min Fairness

- Associate a weight $w_i$ with each flow $i$
- If link congested, compute $f$ such that

$$\sum_{i} \min(r_i, f \times w_i) = C$$

$$(w_1 = 3) \ 8$$
$$(w_2 = 1) \ 6$$
$$(w_3 = 1) \ 2$$

$f = 2$:
- $\min(8, 2 \times 3) = 6$
- $\min(6, 2 \times 1) = 2$
- $\min(2, 2 \times 1) = 2$
Generalized Process Sharing (GPS)

- The methodology:
  - Assume we can send infinitesimal packets
    - single bit
  - Perform round robin.
    - At the bit level
- Idealized policy to split bandwidth
- GPS is not implementable
- Used mainly to evaluate and compare real approaches.
- Has weights that give relative frequencies.
GPS: Example 1

GPS example 1

Packets of size 10, 20 & 30 arrive at time 0
GPS: Example 2

GPS example 2

Packets: time 0 size 15
         time 5 size 20
         time 15 size 10
GPS: Example 3

GPS example 3

Packets: time 0 size 15
         time 5 size 20
         time 15 size 10
         time 18 size 15

7: Multimedia Networking 7-124
Implementing Fair Queuing (Approximate GPS)

- Define a fluid flow system: a system in which flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

- Advantages
  - Each flow will receive exactly its fair rate
Example

Flow 1 (arrival traffic)

Flow 2 (arrival traffic)

Service in fluid flow system

Packet system

7: Multimedia Networking 7-126
System Virtual Time: $V(t)$

- Measure service, instead of time
- $V(t)$ slope - rate at which every active flow receives service
  - $C$ - link capacity
  - $N(t)$ - number of active flows in fluid system at time $t$

\[
\frac{\partial V(t)}{\partial t} = \frac{C}{N(t)}
\]
Generalized Processor Sharing

- A work conserving GPS is defined as

\[
\frac{W_i(t, t + dt)}{w_i} = \frac{W(t, t + dt)}{\sum_{j \in B(t)} W_j} \quad \forall i \in B(t)
\]

- where
  - \(w_i\) - weight of flow \(i\)
  - \(W_i(t_1, t_2)\) - total service received by flow \(i\) during \([t_1, t_2)\)
  - \(W(t_1, t_2)\) - total service allocated to all flows during \([t_1, t_2)\)
  - \(B(t)\) - number of flows backlogged
System Virtual Time: $V(t)$

- Virtual time = Work ($W$)

$W_i(t, t+dt) = w_i \times \frac{W(t, t+dt)}{\sum_{j \in B(t)} w_j}$ \quad \forall i \in B(t)$

$\frac{\partial W_i}{\partial t} = \sum_{j \in B(t)} w_j \times \frac{\partial W}{\partial t}$ \quad \forall i \in B(t)$

$W_i(t_1, t_2) = w_i \times \int_{t=t_1}^{t_2} \left( \frac{1}{\sum_{j \in B(t)} w_j} \times \frac{\partial W}{\partial t} \right) dt$ \quad \forall i \in B(t)$

$\frac{\partial V_{GPS}}{\partial t} = \sum_{j \in B(t)} w_j \times \frac{\partial W}{\partial t}$
Service Allocation in GPS

The service received by flow $i$ during an interval $[t_1, t_2)$, while it is backlogged is

$$W_i(t_1, t_2) = w_i \times \int_{t_1}^{t_2} \frac{\partial V_{GPS}}{\partial t} \, dt \quad \forall i \in B(t)$$

$$W_i(t_1, t_2) = w_i \times (V_{GPS}(t_2) - V_{GPS}(t_1)) \quad \forall i \in B(t)$$
Virtual Time Implementation of Weighted Fair Queueing

\[ V(0) = 0 \quad \quad \quad \quad V(t_j + \tau) = V(t_j) + \frac{\tau}{\sum_{i \in B_j} w_i} \]

\[ S_j^k = \max(F_j^{k-1}, V(a_j^k)) \]

\[ F_j^k = S_j^k + \frac{L_j^k}{W_j} \]

- \( a_j^k \) - arrival time of packet \( k \) of flow \( j \)
- \( S_j^k \) - virtual starting time of packet \( k \) of flow \( j \)
- \( F_j^k \) - virtual finishing time of packet \( k \) of flow \( j \)
- \( L_j^k \) - length of packet \( k \) of flow \( j \)
- \( B_j \) - backlog flow (flow with packets in queue)
- Packets are sent in the increasing order of \( F_j^k \)
System Virtual Time

\[ V_{GPS}(t) \]

- 0
- 4
- 8
- 12
- 16

- 2*C
- C
- 2*C

1/2
1/8
1/8
1/8
1/8
Virtual Start and Finish Times

- Utilize the time the packets would start $S^k_i$ and finish $F^k_i$ in a fluid system

$$F^k_i = S^k_i + \frac{L^k_i}{W_i}$$
GPS vs WFQ (equal length)

Queue 1
@ \( t=0 \)
Queue 2
@ \( t=0 \)

GPS: both packets served at rate 1/2
Both packets complete service at \( t=2 \)

Packet from queue 1 being served
Packet from queue 2 waiting
Packet from queue 2 being served

Packet-by-packet system (WFQ):
queue 1 served first at rate 1; then queue 2 served at rate 1.

7: Multimedia Networking 7-134
GPS vs WFQ (different length)

Queue 1 @ \(t=0\)
Queue 2 @ \(t=0\)

GPS: both packets served at rate \(1/2\)
Packet from queue 2 served at rate 1
Packet from queue 2 waiting
Packet from queue 1 being served at rate 1

7: Multimedia Networking 7-135
GPS vs WFQ

Queue 1 @ $t = 0$

Queue 2 @ $t = 0$

Weight:
Queue 1 = 1
Queue 2 = 3

GPS: packet from queue 1 served at rate $1/4$;
Packet from queue 1 waiting
Packet from queue 2 served at rate $3/4$

WFQ: queue 2 served first at rate 1;
then queue 1 served at rate 1.

Packet from queue 2 being served
Packet from queue 1 being served

7: Multimedia Networking 7-136
Policing Mechanisms
Policing Mechanisms

**Goal:** limit traffic to not exceed declared parameters

Three common-used criteria:

- *(Long term)* **Average Rate:** how many pkts can be sent per unit time (in the long run)
  - crucial question: what is the interval length: 100 packets per sec or 6000 packets per min have same average!
- **Peak Rate:** e.g., 6000 pkts per min. (ppm) avg.; 1500 ppm peak rate
- *(Max.)* **Burst Size:** max. number of pkts sent consecutively (with no intervening idle)
Policing Mechanisms

1. Leaky Bucket Algorithm

2. Token Bucket Algorithm
The Leaky Bucket Algorithm

- The Leaky Bucket Algorithm used to control rate in a network.

- Implemented as a single-server queue with constant service time.

- If the bucket (buffer) overflows then packets are discarded.
The Leaky Bucket Algorithm

(a) A leaky bucket with water. (b) a leaky bucket with packets.
The Leaky Bucket Algorithm

- The leaky bucket enforces a constant output rate regardless of the burstiness of the input. Does nothing when input is idle.

- The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.

- When packets are the same size (as in ATM cells), the one packet per tick is okay. For variable length packets though, it is better to allow a fixed number of bytes per tick.
Token Bucket

**Token Bucket:** limit input to specified Burst Size and Average Rate.

- bucket can hold b tokens
- tokens generated at rate $r$ tokens/sec unless bucket full
- over interval of length $t$: number of packets admitted less than or equal to $(r \times t + b)$. 
Token Bucket

- Characterized by three parameters ($b$, $r$, $R$)
  - $b$ - token depth
  - $r$ - average arrival rate
  - $R$ - maximum arrival rate (e.g., $R$ link capacity)

- A bit is transmitted only when there is an available token
  - When a bit is transmitted exactly one token is consumed

![Diagram of Token Bucket]

- $r$ tokens per second
- $b$ tokens
- $b \times R/(R-r)$
- Slope $R$
- Slope $r$
- $\leq R$ bps
- Regulator

- Graph showing bits over time with slopes $R$ and $r$.
Characterizing a Source by Token Bucket

- Arrival curve - maximum amount of bits transmitted by time $t$
- Use token bucket to bound the arrival curve
Example

- Arrival curve - maximum amount of bits transmitted by time \( t \)
- Use token bucket to bound the arrival curve

![Graph showing the arrival curve and token bucket concept](image-url)
Policing Mechanisms

- token bucket, WFQ combine to provide guaranteed upper bound on delay, i.e., \textit{QoS guarantee}!
Leaky Bucket vs. Token Bucket

- With TB, a packet can only be transmitted if there are enough tokens to cover its length in bytes.
- LB sends packets at an average rate. TB allows for large bursts to be sent faster by speeding up the output.
- TB allows saving up tokens (permissions) to send large bursts. LB does not allow saving.