

# Telematics Course: Computer Networks and the Internet

(Winter Semester 2002/03)

Prof. Dr. Dieter Hogrefe  
Dipl.-Inform. Michael Ebner  
Dr. Xiaoming Fu

## An Overview of This Lecture

Having discussed:

- Introduction to Computer Networks and the Internet
- Network layer
- Transport layer
- Quality of Service
- Mobile Internet

To be discussed:

- **Network Management**
- **Higher Layers (http, ftp, telnet)**
- **Introduction to Multimedia Networking**
- **Computer and Network Security**

Credits: James Kurose, Keith Ross, Computer Networking(2nd Ed.), Addison-Wesley

## Network Management

Handout 6, Computer Networks and the  
Internet, Telematics Course

## Outline

- **What is network management**
  - motivation
  - major components
- Internet network management framework
  - MIB: management information base
  - SMI: data definition language
  - SNMP: protocol for network management
  - security and administration
- presentation services: ASN.1

## What is network management?

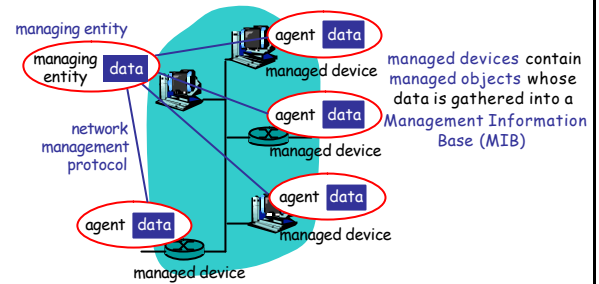
- **autonomous systems (aka "network")**: 100s or 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
  - jet airplane
  - nuclear power plant
  - others?



"Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

## Infrastructure for network management

definitions:



## Network Management standards

### OSI CMIP

- Common Management Information Protocol
- designed 1980's: *the* unifying net management standard
- too slowly standardized

### SNMP: Simple Network Management Protocol

- Internet roots (SGMP)
- started simple
- deployed, adopted rapidly
- growth: size, complexity
- currently: SNMP V3
- *de facto* network management standard

## Outline

- What is network management?
- **Internet-standard management framework**
  - Structure of Management Information: SMI
  - Management Information Base: MIB
  - SNMP Protocol Operations and Transport Mappings
  - Security and Administration
- ASN.1

## SNMP overview: 4 key parts

- **Management information base (MIB):**
  - distributed information store of network management data
- **Structure of Management Information (SMI):**
  - data definition language for MIB objects
- **SNMP protocol**
  - convey manager<->managed object info, commands
- **security, administration capabilities**
  - major addition in SNMPv3

## SMI: data definition language

Purpose: syntax, semantics of management data well-defined, unambiguous

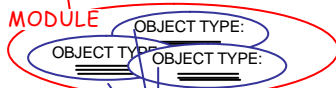
- base data types:
  - straightforward, boring
- OBJECT-TYPE
  - data type, status, semantics of managed object
- MODULE-IDENTITY
  - groups related objects into MIB module

### Basic Data Types

INTEGER  
Integer32  
Unsigned32  
OCTET STRING  
OBJECT IDENTIFIED  
IPAddress  
Counter32  
Counter64  
Gauge32  
Time Ticks  
Opaque

## SNMP MIB

MIB module specified via SMI  
**MODULE-IDENTITY**  
(100 standardized MIBs, more vendor-specific)



objects specified via SMI  
**OBJECT-TYPE** construct

## SMI: Object, module examples

### OBJECT-TYPE:

**ipInDelivers**  
ipInDelivers OBJECT TYPE  
SYNTAX Counter32  
MAX-ACCESS read-only  
STATUS current  
DESCRIPTION  
"The total number of input datagrams successfully delivered to IP user-protocols (including ICMP)"  
 ::= { ip 9 }

### MODULE-IDENTITY:

**ipMIB**  
ipMIB MODULE-IDENTITY  
LAST-UPDATED "941101000Z"  
ORGANIZATION "IETF SNMPv2 Working Group"  
CONTACT-INFO  
" Keith McCloghrie  
....."  
DESCRIPTION  
"The MIB module for managing IP and ICMP implementations, but excluding their management of IP routes."  
REVISION "019331000Z"  
.....  
 ::= { mib-2 48 }

## MIB example: UDP module

Object ID	Name	Type	Comments
1.3.6.1.2.1.7.1	UDPInDatagrams	Counter32	total # datagrams delivered at this node
1.3.6.1.2.1.7.2	UDPNoPorts	Counter32	# undeliverable datagrams no app at port!
1.3.6.1.2.1.7.3	UDInErrors	Counter32	# undeliverable datagrams all other reasons
1.3.6.1.2.1.7.4	UDPOutDatagrams	Counter32	# datagrams sent
1.3.6.1.2.1.7.5	udpTable	SEQUENCE	one entry for each port in use by app, gives port # and IP address

## SNMP Naming

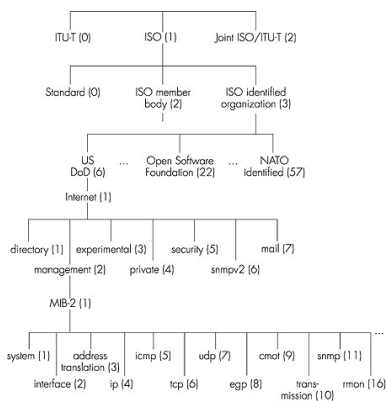
**question:** how to name every possible standard object (protocol, data, more..) in every possible network standard??

**answer:** ISO Object Identifier tree:

- hierarchical naming of all objects
- each branchpoint has name, number



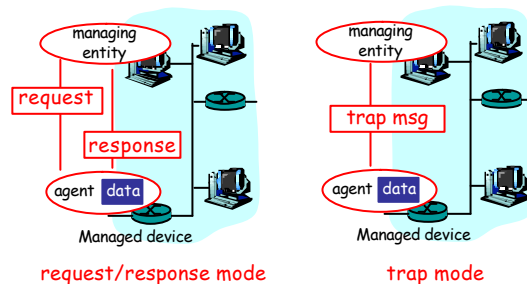
## OSI Object Identifier Tree



Check out [www.alvestrand.no/harald/objectid/top.html](http://www.alvestrand.no/harald/objectid/top.html)

## SNMP protocol

Two ways to convey MIB info, commands:



## SNMP protocol: message types

Message type	Function
GetRequest GetNextRequest GetBulkRequest	Mgr-to-agent: "get me data" (instance,next in list, block)
InformRequest	Mgr-to-Mgr: here's MIB value
SetRequest	Mgr-to-agent: set MIB value
Response	Agent-to-mgr: value, response to Request
Trap	Agent-to-mgr: inform manager of exceptional event

## SNMP protocol: message formats



## SNMP security and administration

- **encryption:** DES-encrypt SNMP message
- **authentication:** compute, send MIC(m,k): compute hash (MIC) over message (m), secret shared key (k)
- **protection against playback:** use nonce
- **view-based access control**
  - SNMP entity maintains database of access rights, policies for various users
  - database itself accessible as managed object!

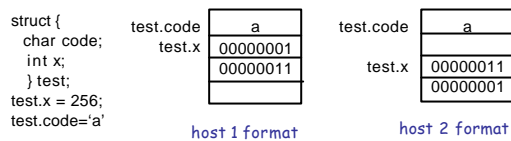
## Outline

- What is network management?
- Internet-standard management framework
  - Structure of Management Information: SMI
  - Management Information Base: MIB
  - SNMP Protocol Operations and Transport Mappings
  - Security and Administration
- **The presentation problem: ASN.1**

## The presentation problem

**Q:** does perfect memory-to-memory copy solve “the communication problem”?

**A:** not always!



**problem:** different data format, storage conventions

## Presentation problem: potential solutions

1. Sender learns receiver's format. Sender translates into receiver's format. Sender sends.
  - real-world analogy?
  - pros and cons?
2. Sender sends. Receiver learns sender's format. Receiver translate into receiver-local format
  - real-world-analogy
  - pros and cons?
3. Sender translates host-independent format. Sends. Receiver translates to receiver-local format.
  - real-world analogy?
  - pros and cons?

## Solving the presentation problem

1. Translate local-host format to host-independent format
2. Transmit data in host-independent format
3. Translate host-independent format to remote-host format

grandma

## ASN.1: Abstract Syntax Notation 1

- **ISO standard X.680**
  - used extensively in Internet
  - like eating vegetables, knowing this “good for you”!
- **defined data types**, object constructors
  - like SMI
- **BER: Basic Encoding Rules**
  - specify how ASN.1-defined data objects to be transmitted
  - each transmitted object has Type, Length, Value (TLV) encoding

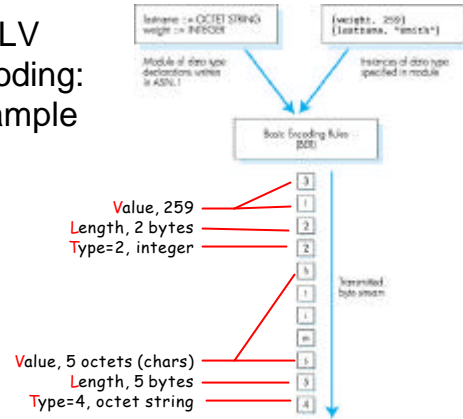
## TLV Encoding

**Idea:** transmitted data is self-identifying

- **T**: data type, one of ASN.1-defined types
- **L**: length of data in bytes
- **V**: value of data, encoded according to ASN.1 standard

Tag Value	Type
1	Boolean
2	Integer
3	Bitstring
4	Octet string
5	Null
6	Object Identifier
9	Real

## TLV encoding: example



## Network Management: summary

- network management
  - extremely important: 80% of network "cost"
  - ASN.1 for data description
  - SNMP protocol as a tool for conveying information
- Network management: more art than science
  - what to measure/monitor
  - how to respond to failures?
  - alarm correlation/filtering?

## Application Layer

Handout 7, Computer Networks and the Internet, Telematics Course

## Outline

- **Principles of app layer protocols**
  - clients and servers
  - app requirements
- **Web and HTTP**
- **FTP**
- **Electronic Mail**
  - SMTP, POP3, IMAP

## Network applications: some jargon

- Process:** program running within a host.
- within same host, two processes communicate using **interprocess communication** (defined by OS).
  - processes running in different hosts communicate with an **application-layer protocol**
- user agent:** interfaces with user "above" and network "below".
- implements user interface & application-level protocol
    - Web: browser
    - E-mail: mail reader
    - streaming audio/video: media player

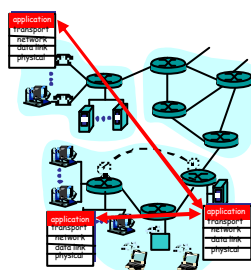
## Applications and application-layer protocols

### Application: communicating, distributed processes

- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

### Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



## App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
  - Syntax of message types: what fields in messages & how fields are delineated
  - Semantics of the fields, ie, meaning of information in fields
  - Rules for when and how processes send & respond to messages
- Public-domain protocols:**
- defined in RFCs
  - allows for interoperability
  - eg, HTTP, SMTP
- Proprietary protocols:**
- eg, KaZaA

## Client-server paradigm

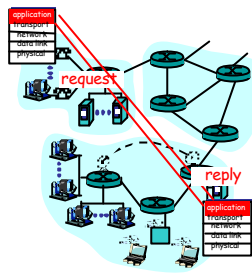
Typical network app has two pieces: *client* and *server*

### Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- Web: client implemented in browser; e-mail: in mail reader

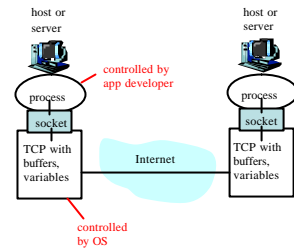
### Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



## Processes communicating across network

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process assumes transport infrastructure on other side of door which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

## Addressing processes:

- For a process to receive messages, it must have an identifier
- Every host has a unique 32-bit IP address
- **Q:** does the IP address of the host on which the process runs suffice for identifying the process?
- **Answer:** No, many processes can be running on same host
- Identifier includes both the IP address and **port numbers** associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- **More on this later**

## What transport service does an app need?

### Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

### Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

### Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

## Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

## Internet transport protocols services

### TCP service:

- *connection-oriented*: setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control*: sender won't overwhelm receiver
- *congestion control*: throttle sender when network overloaded
- *does not providing*: timing, minimum bandwidth guarantees

### UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

## Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

## Outline

- Principles of app layer protocols
  - clients and servers
  - app requirements
- **Web and HTTP**
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP

## Web and HTTP

### First some jargon

- **Web page** consists of **objects**
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of **base HTML-file** which includes several referenced objects
- Each object is addressable by a **URL**
- Example URL:

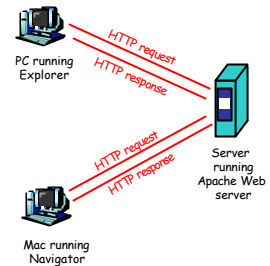
www.someschool.edu/someDept/pic.gif

host name                      path name

## HTTP overview

### HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client*: browser that requests, receives, "displays" Web objects
  - *server*: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



## HTTP overview (continued)

### Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

### HTTP is "stateless"

- server maintains no information about past client requests

*aside*  
Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

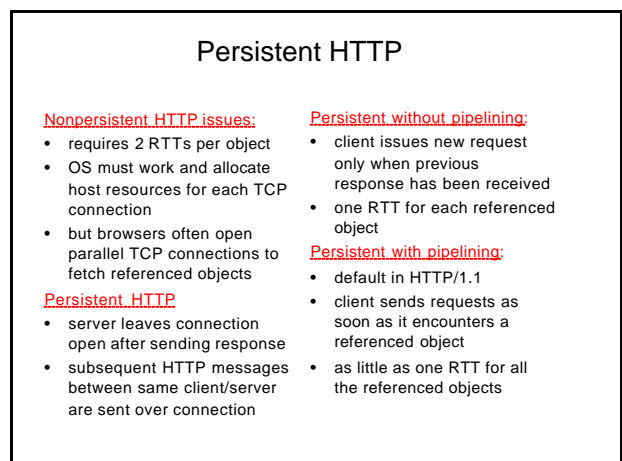
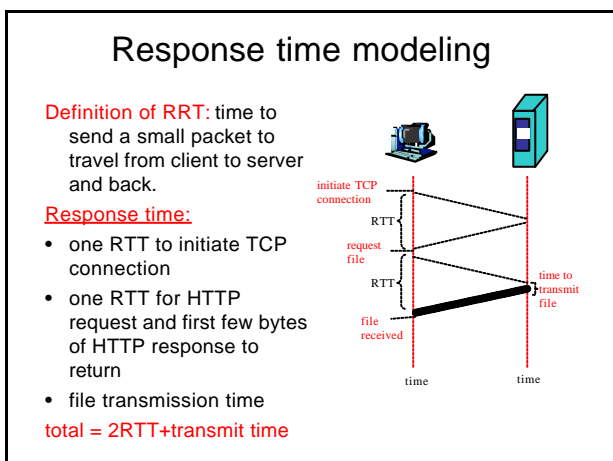
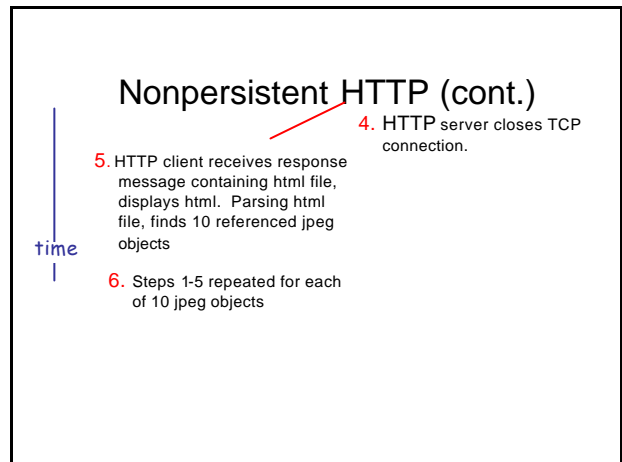
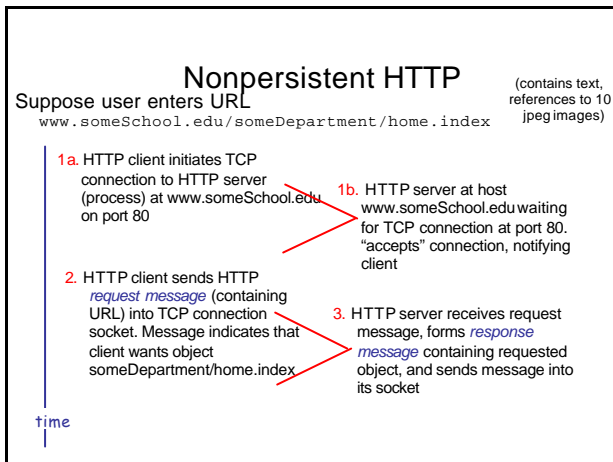
## HTTP connections

### Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

### Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode



## HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:**
  - ASCII (human-readable format)

request line  
(GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
```

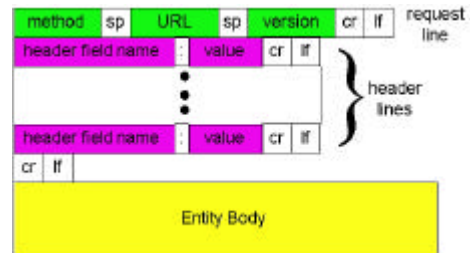
header lines

```
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

Carriage return  
line feed  
indicates end  
of message

(extra carriage return, line feed)

## HTTP request message: general format



## Uploading form input

### Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

### URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

[www.somesite.com/animalsearch?monkeys&banana](http://www.somesite.com/animalsearch?monkeys&banana)

## Method types

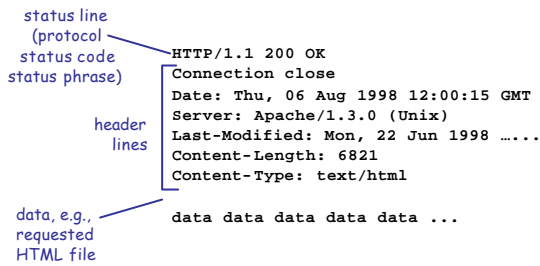
### HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

### HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

## HTTP response message



## HTTP response status codes

In first line in server->client response message.

A few sample codes:

### 200 OK

- request succeeded, requested object later in this message

### 301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

### 400 Bad Request

- request message not understood by server

### 404 Not Found

- requested document not found on this server

### 505 HTTP Version Not Supported

## Trying out HTTP (client side) for yourself

### 1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

### 2. Type in a GET HTTP request:

```
GET /-ross/index.html HTTP/1.0
```

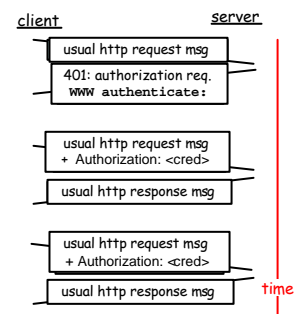
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

### 3. Look at response message sent by HTTP server!

## User-server interaction: authorization

**Authorization** : control access to server content

- authorization credentials: typically name, password
- **stateless**: client must present authorization in *each* request
  - **authorization**: header line in each request
  - if no **authorization**: header, server refuses access, sends **WWW authenticate:** header line in response



## Cookies: keeping "state"

Many major Web sites use cookies

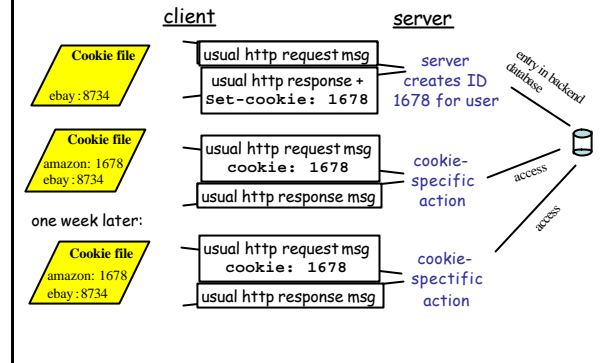
### Four components:

- 1) cookie header line in the HTTP response message
- 2) cookie header line in HTTP request message
- 3) cookie file kept on user's host and managed by user's browser
- 4) back end database at Web site

### Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

## Cookies: keeping "state" (cont.)



## Cookies (continued)

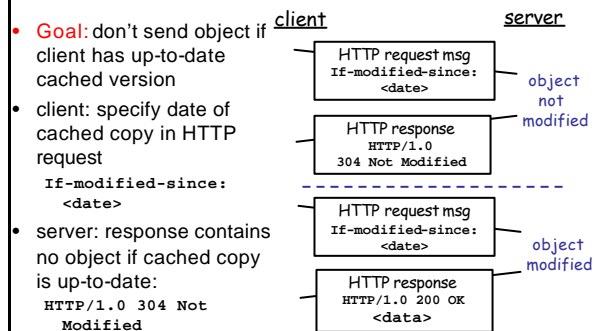
### What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

### Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

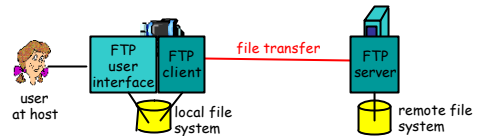
## Conditional GET: client-side caching



## Outline

- Principles of app layer protocols
  - clients and servers
  - app requirements
- Web and HTTP
- **FTP**
- Electronic Mail
  - SMTP, POP3, IMAP

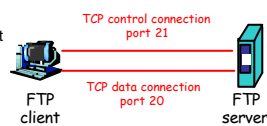
## FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ftp: RFC 959
- ftp server: port 21

## FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.



- Server opens a second TCP data connection to transfer another file.
- Control connection: "out of band"
- FTP server maintains "state": current directory, earlier authentication

## FTP commands, responses

### Sample commands:

- sent as ASCII text over control channel
- **USER** *username*
- **PASS** *password*
- **LIST** return list of file in current directory
- **RETR** *filename* retrieves (gets) file
- **STOR** *filename* stores (puts) file onto remote host

### Sample return codes

- status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

## Outline

- Principles of app layer protocols
  - clients and servers
  - app requirements
- Web and HTTP
- FTP
- **Electronic Mail**
  - SMTP, POP3, IMAP

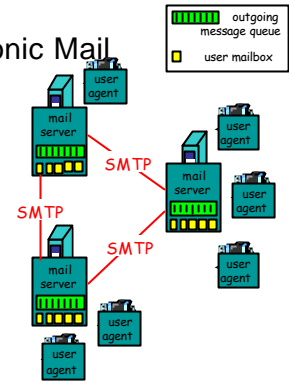
## Electronic Mail

### Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

### User Agent

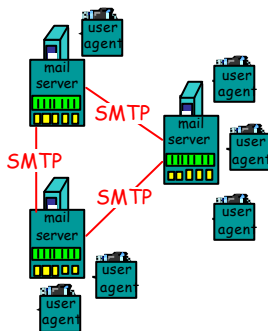
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



## Electronic Mail: mail servers

### Mail Servers

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

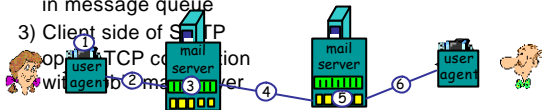


## Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - **commands**: ASCII text
  - **response**: status code and phrase
- messages must be in 7-bit ASCII

## Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) Client side of SMTP opens TCP connection with mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



## Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:   How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

## Try SMTP interaction for yourself:

- telnet servername 25
  - see 220 reply from server
  - enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands
- above lets you send email without using email client (reader)

## SMTP: final words

- SMTP uses persistent connections
  - SMTP requires message (header & body) to be in 7-bit ASCII
  - SMTP server uses CRLF.CRLF to determine end of message
- Comparison with HTTP:**
- HTTP: pull
  - SMTP: push
  - both have ASCII command/response interaction, status codes
  - HTTP: each object encapsulated in its own response msg
  - SMTP: multiple objects sent in multipart msg

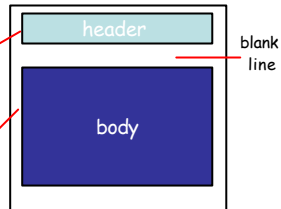
## Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

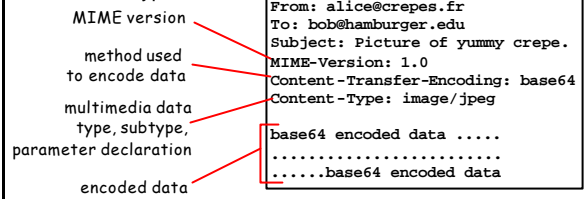
- header lines, e.g.,
  - To:
  - From:
  - Subject:

*different from SMTP commands*
- body
  - the "message", ASCII characters only



## Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type



## MIME types

Content-Type: type/subtype; parameters

### Text

- example subtypes: plain, html

### Image

- example subtypes: jpeg, gif

### Audio

- example subtypes: basic (8-bit mu-law encoded), 32kadpcm (32 kbps coding)

### Video

- example subtypes: mpeg, quicktime

### Application

- other data that must be processed by reader before "viewable"
- example subtypes: msword, octet-stream

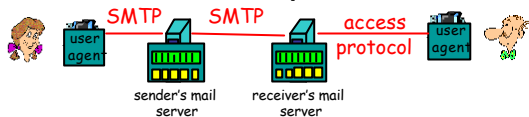
## Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart
```

```
--StartOfNextPart
Dear Bob, Please find a picture of a crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.....base64 encoded data
--StartOfNextPart
Do you want the recipe?
```

Red brackets on the right side of the text indicate the structure of the multipart message, grouping the text part and the image part.

## Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: Hotmail , Yahoo! Mail, etc.

## POP3 protocol

### authorization phase

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - +OK
  - -ERR

### transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```

S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
    
```

## POP3 (more) and IMAP

### More about POP3

- Previous example uses "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

### IMAP

- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

## Introduction to Multimedia Networking

Handout 8, Computer Networks and the Internet, Telematics Course

## Outline

- **Multimedia Networking Applications**
- Streaming stored audio and video
  - RTSP
- Real-time Multimedia: Internet Phone Case Study
- Protocols for Real-Time Interactive Applications
  - RTP, RTCP
  - SIP
- Beyond Best-effort: QoS

## MM Networking Applications: audio & video

(“continuous media”)

### 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:

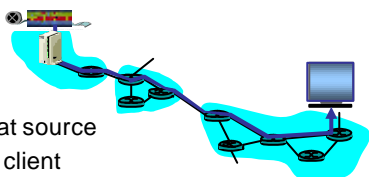
- Typically **delay sensitive**
  - end-to-end delay
  - delay jitter
- But **loss tolerant**: infrequent losses cause minor glitches
- Antithesis of data, which are loss intolerant but delay tolerant.

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

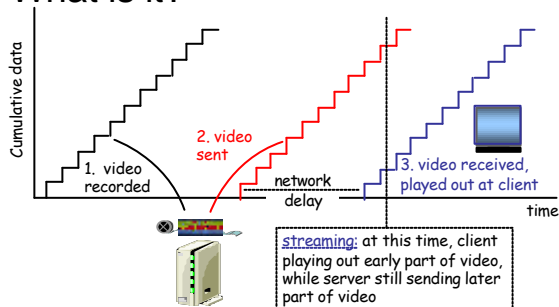
## Streaming Stored Multimedia

### Streaming:

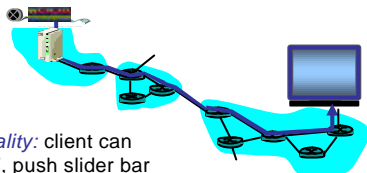
- media stored at source
- transmitted to client
- streaming: client playout begins *before* all data has arrived
- timing constraint for still-to-be transmitted data: in time for playout



## Streaming Stored Multimedia: What is it?



## 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

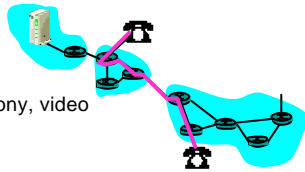
### Streaming

- 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?

## A few words about audio compression

- Analog signal sampled at constant rate
    - telephone: 8,000 samples/sec
    - CD music: 44,100 samples/sec
  - Each sample quantized, ie, rounded
    - eg,  $2^8=256$  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

## A few words about 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
    - spacial
    - 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

## Outline

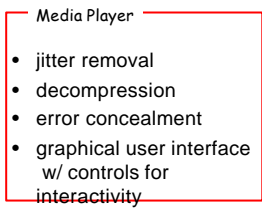
- Multimedia Networking Applications
- **Streaming stored audio and video**
  - RTSP
- Real-time Multimedia: Internet Phone Case Study
- Protocols for Real-Time Interactive Applications
  - RTP, RTCP
  - SIP
- Beyond Best-effort: QoS

## Streaming Stored Multimedia

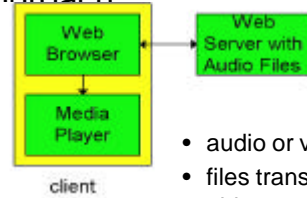
Application-level

streaming techniques for making the best out of best effort service:

- client side buffering
- use of UDP versus TCP
- multiple encodings of multimedia



## Internet multimedia: simplest approach

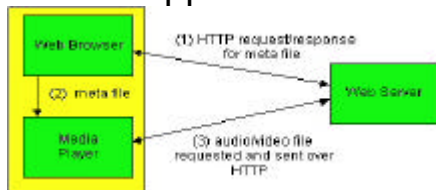


- audio or video stored in file
- files transferred as HTTP object
  - received in entirety at client
  - then passed to player

audio, video not streamed:

- no, "pipelining," long delays until playback!

## Internet multimedia: streaming approach



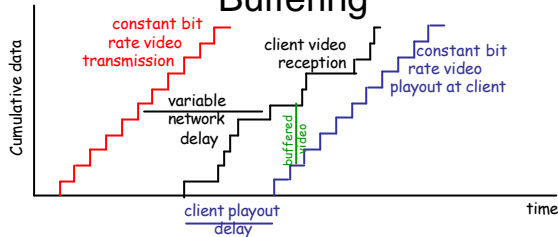
- browser GETs **metafile**
- browser launches player, passing metafile
- player contacts server
- server **streams** audio/video to player

## Streaming from a streaming server



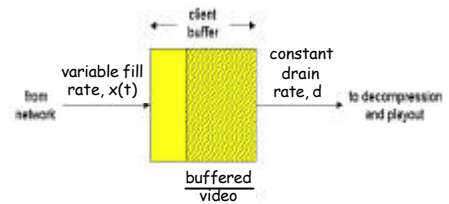
- This architecture allows for non-HTTP protocol between server and media player
- Can also use UDP instead of TCP.

## 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?

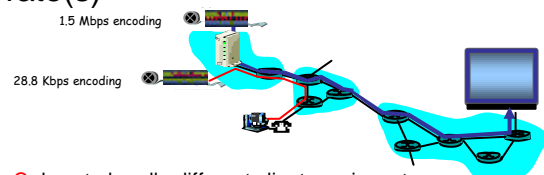
### UDP

- 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".

## RTSP Example

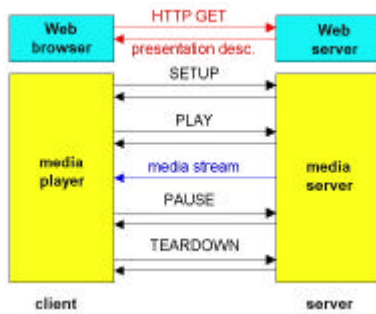
### Scenario:

- 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

## Outline

- Multimedia Networking Applications
- Streaming stored audio and video
  - RTSP
- Real-time Multimedia: Internet Phone Case Study
- Protocols for Real-Time Interactive Applications
  - RTP, RTCP
  - SIP
- Beyond Best-effort: QoS

## 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

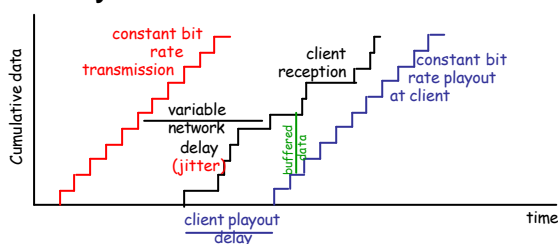
### Introduce Internet Phone by way of an example

- 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.

## Delay Jitter



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

## Recovery from packet loss (1)

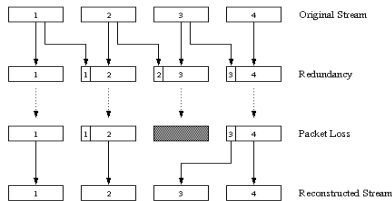
### forward error correction (FEC): simple scheme

- 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)

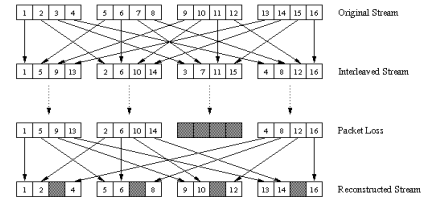
### 2nd FEC scheme

- "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

## Outline

- Multimedia Networking Applications
- Streaming stored audio and video
  - RTSP
- Real-time Multimedia: Internet Phone Case Study
- Protocols for Real-Time Interactive Applications
  - RTP, RTCP
  - SIP
- Beyond Best-effort: QoS

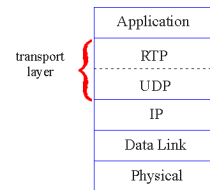
## Real-Time Protocol (RTP)

- 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.

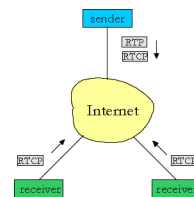
## RTP Header (2)

- **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.
  - 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.
- 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

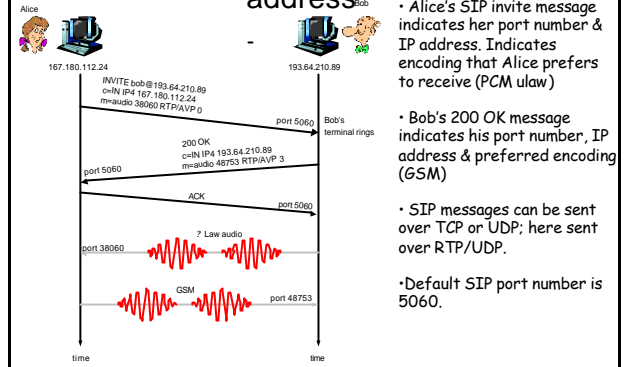
## SIP

- 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



## 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 5060.

• Alice specifies in Via: header that SIP client sends and receives SIP messages over UDP

## Name translation and user locataion

- 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 send's 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

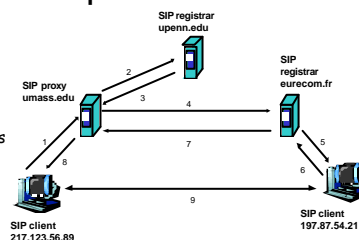
## Example

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.



## Comparison with H.323

- H.323 is another signaling protocol for real-time, interactive
- H.323 is a complete, vertically integrated suite of protocols for multimedia conferencing: signaling, registration, admission control, transport and codecs.
- SIP is a single component. Works with RTP, but does not mandate it. Can be combined with other protocols and services.
- H.323 comes from the ITU (telephony).
- SIP comes from IETF: Borrows much of its concepts from HTTP. SIP has a Web flavor, whereas H.323 has a telephony flavor.
- SIP uses the KISS principle: Keep it simple stupid.

## Outline

- Multimedia Networking Applications
- Streaming stored audio and video
  - RTSP
- Real-time Multimedia: Internet Phone Case Study
- Protocols for Real-Time Interactive Applications
  - RTP, RTCP
  - SIP
- **Beyond Best-effort: QoS**

## Summary of QoS Principles

- Four components of QoS:
  - Classification
  - Policing and shaping
  - Scheduling
  - Admission Control
- Next Generation Internet: QoS-enabled
  - RSVP
  - IntServ
  - DiffServ

## Summary

Most importantly: learned about protocols

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated
- control vs. data msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"
- security: authentication