Application Layer Concepts and Design Issues

Dr. Rahul Banerjee
Computer Science and Information Systems Group
Birla Institute of Technology and Science
Pilani - 333 031, INDIA
E-mail: rahul@bits-pilani.ac.in
Home Page: http://discovery.bits-pilani.ac.in/rahul/

Interaction Goals:

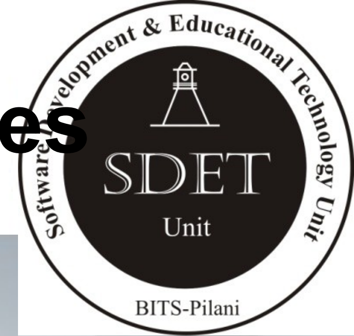
- Understanding of the **Application Layer**
- A look at **various Design Issues**
- A comparative study of various **Application Layer Protocols**
- Introduction to **Network Security**
- **Current Industry Practices and Evolving Trends**
- **Research Directions**

Application Layer: What is it?

- Application Layer is a layer of the Network Architecture that is primarily concerned with getting TPDU from the lower layer (usually Transport Layer) and delivering it to the Application and vice-versa, with or without explicit presentation and session management support as per situation.
- HTTP, DHCP, DNS, SNMP, FTP are examples of some of the Application Layer protocols.
- Web-services, Video-on-Demand over the network, Video/Voice-conferencing over the network etc. are examples of Applications that reside atop the protocols belonging to this layer.

Copyright: Rahul Banerjee BITS, Pilani (India)

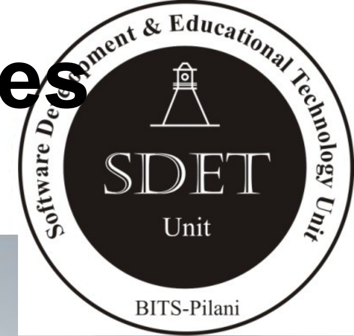
Application Layer Responsibilities (1 of 2)



- It primarily deals with:
  - Accepting messages from the Application Layer through the APIs
  - Processing these messages and generating APDUs
  - Deciding transport connection requirements (for further transmitting this DU after encapsulating it within an APDU)
  - Passing this packet through the SAP to the lower layer (TL)

Copyright: Rahul Banerjee BITS, Pilani (India)

Application Layer Responsibilities (2 of 2)



- It also deals with ...
  - Accepting APDU from the lower layer through the SAP
  - Processing the APDU
  - Removing the encapsulation and passing the messages to the respective destination application.
  - Provide diagnostic support for network monitoring, configuration, management and trouble-shooting at the Application Layer or lower layer.

Copyright: Rahul Banerjee BITS, Pilani (India)

QoS Considerations in the AL

- Per-Flow or Aggregate Flow QoS suitability
- Priority of Service
- Maximum Acceptable Delays
- Minimum Acceptable Throughput
- MTBF
- Maximum Acceptable AL-initiated Abnormal Termination of Service
- Security Specifications

Copyright: Rahul Banerjee BITS, Pilani (India)

The Hourglass View of the Protocols

DNS DHCP SNMP Telnet SMTP IMAP POP NNTP FTP TFTP HTTP

RTP RTCP RTSP

SSL/TLS

SCTP | TCP | UDP

IPv4 / IPv6 ICMP / IGMP

IEEE 802.2 LLC Sub-layer

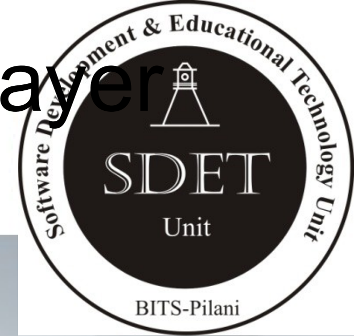
IEEE 802.x or equivalent MAC Sub-layer

Ethernet WiFi WiMAx Bluetooth

Physical Layer

Copyright: Rahul Banerjee BITS, Pilani (India)

Network Applications vs. Application-layer Protocols

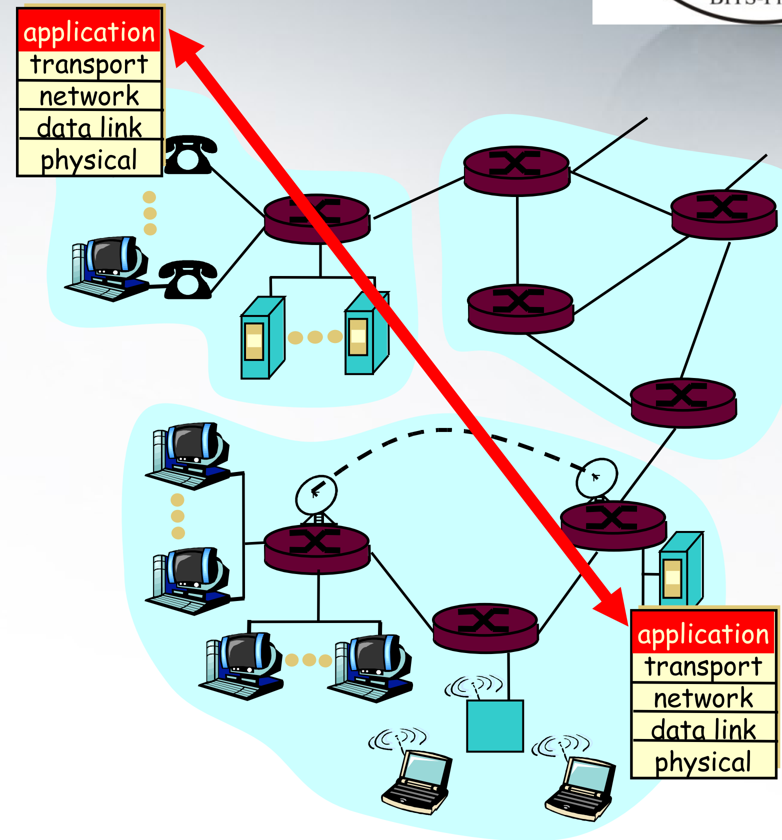


Network application: communicating, distributed processes

- a process is a program that is running within a host
  - a user agent is a process serving as an interface to the user
    - web: browser
    - streaming audio/video: media player
- processes running in different hosts communicate by an application-layer protocol
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- implementing services by using the service provided by lower layers



Copyright: Rahul Banerjee BITS, Pilani (India)

Application-Layer Protocols: How to Use the Transport Service?

API: application programming interface

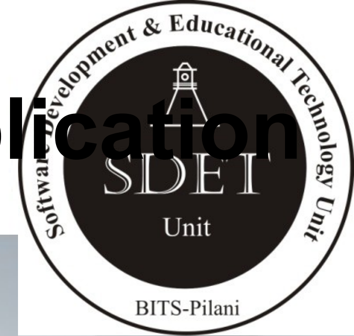
- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

How does a process "identify" the other process with which it wants to communicate?

- IP address of host running the other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

Copyright: Rahul Banerjee BITS, Pilani (India)

What Transport Service Does an Application Need?



Data loss

- some apps can tolerate some packet losses
- other apps require 100% reliable data transfer

Bandwidth

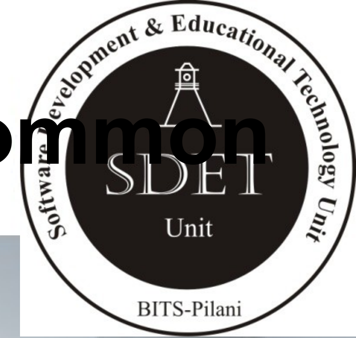
- some apps require minimum amount of bandwidth to be "effective"
- other apps make use of whatever bandwidth they get

Timing

- some apps require low delay to be "effective"

Copyright: Rahul Banerjee BITS, Pilani (India)

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+	yes, 100’s msec
financial apps	no loss	elastic	yes and no

Internet Apps: Their Protocols and Transport Protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

Client-Server Paradigm

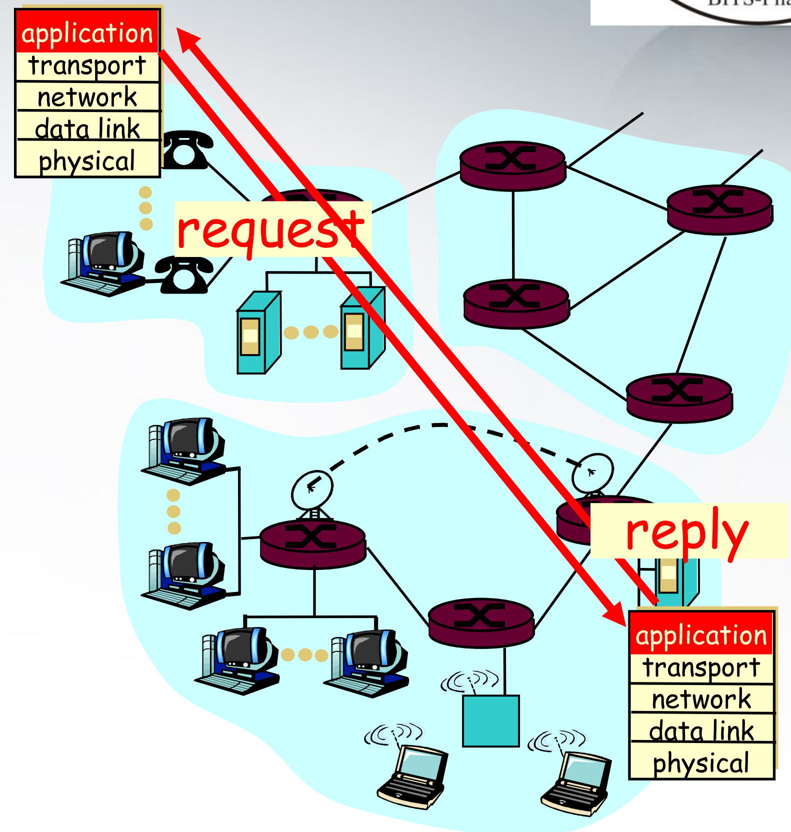
Typical network app has two pieces: *client* and *server*

Client:

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

Server:

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



Copyright: Rahul Banerjee BITS, Pilani (India)

Software Development & Educational Technology Unit
SDET Unit
BITS-Pilani

The Web: Some Terms

- Web page:
  - consists of "objects"
  - addressed by a URL
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects.
- URL has two components: host name, port number and path name:

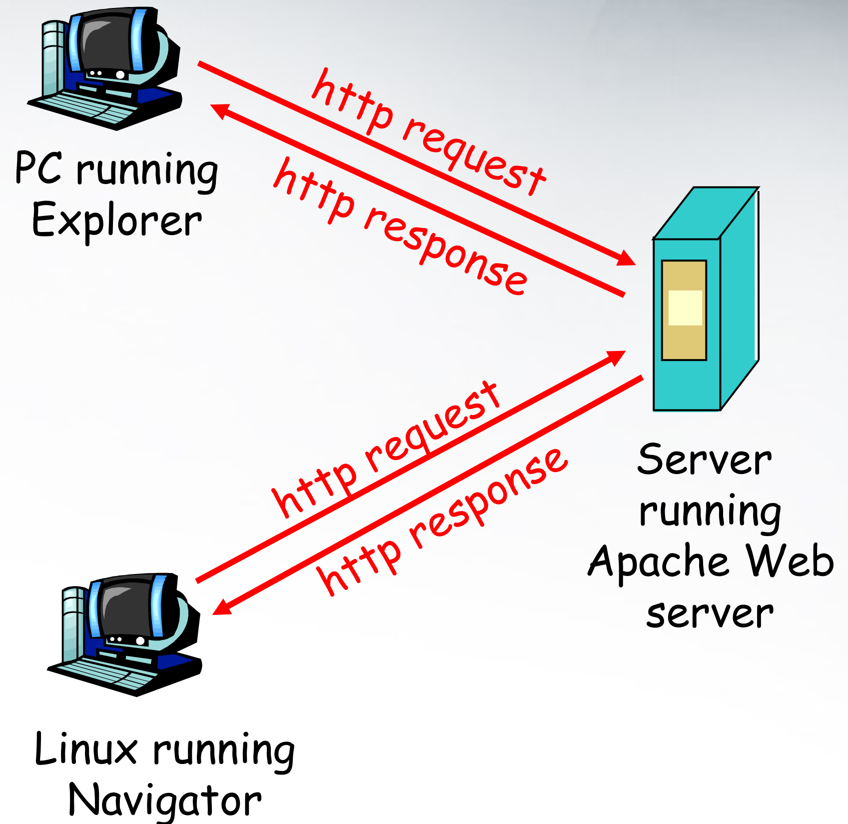
- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Navigator
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server

Copyright: Rahul Banerjee BITS, Pilani (India)

The Web: the HTTP Protocol

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
- http1.0: RFC 1945
- http1.1: RFC 2068



Copyright: Rahul Banerjee BITS, Pilani (India)

The HTTP Protocol: Message Flow

HTTP: TCP transport service:

- 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

Copyright: Rahul Banerjee BITS, Pilani (India)

HTTP Example

Suppose user enters URL
www.bits-pilani.ac.in

(contains text, references to images too)

1a. http client initiates TCP connection to http server (process) at www.bits-pilani.ac.in Port 80 is default for http server.

1b. http server at host www.bits-pilani.ac.in waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)

time

Copyright: Rahul Banerjee, BITS, Pilani (India)

HTTP Example (cont.)

4. http server closes TCP connection.

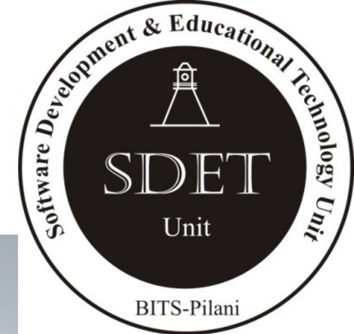
5. http client receives response message containing html file, parses html file, finds embedded image

time

6. Steps 1-5 repeated for each of the embedded images

Copyright: Rahul Banerjee BITS, Pilani (India)

Non-Persistent and Persistent Connections



Non-persistent

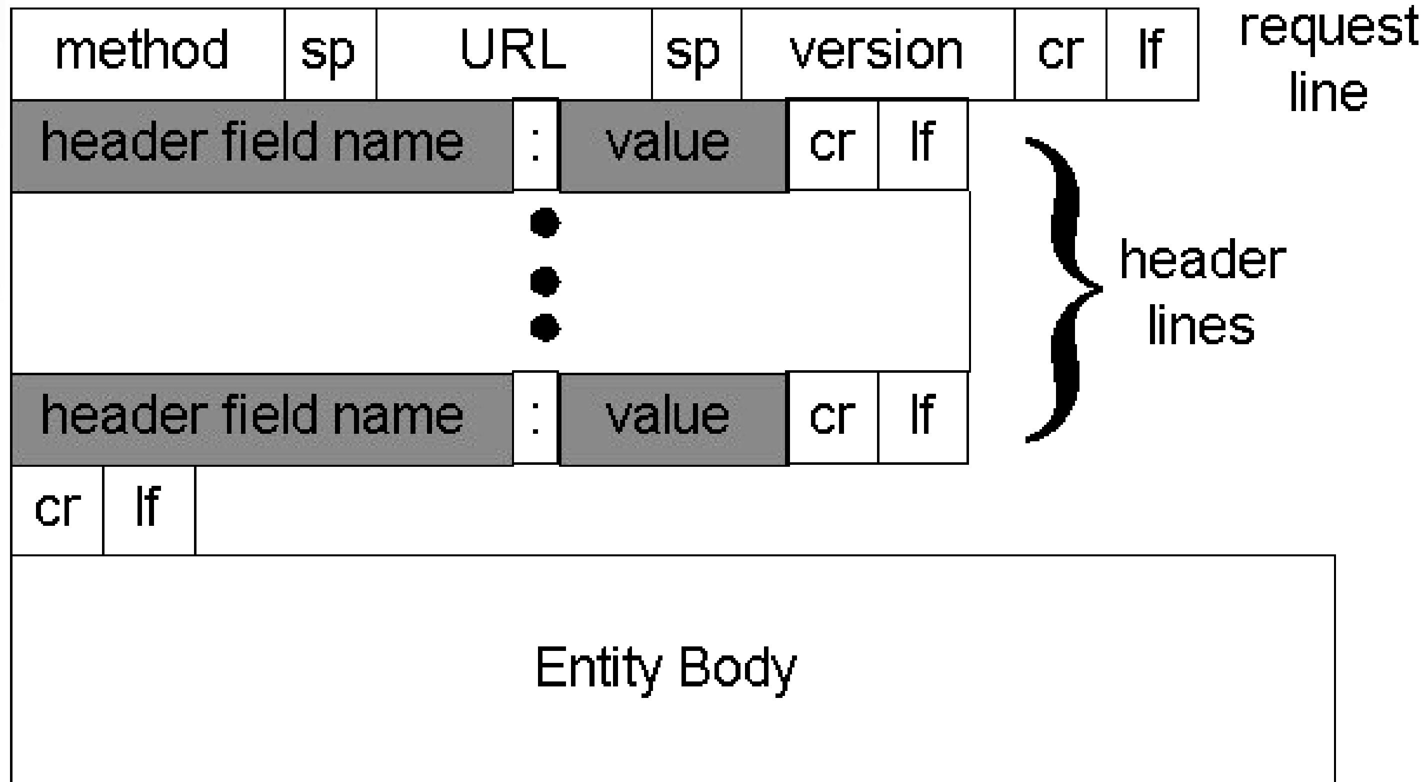
- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer is independent

Persistent

- default for HTTP/1.1
- on same TCP connection: server parses request, responds, parses new request, …
- client sends requests for all referenced objects as soon as it receives base HTML.
- fewer RTTs

But most 1.0 browsers use parallel TCP connections.

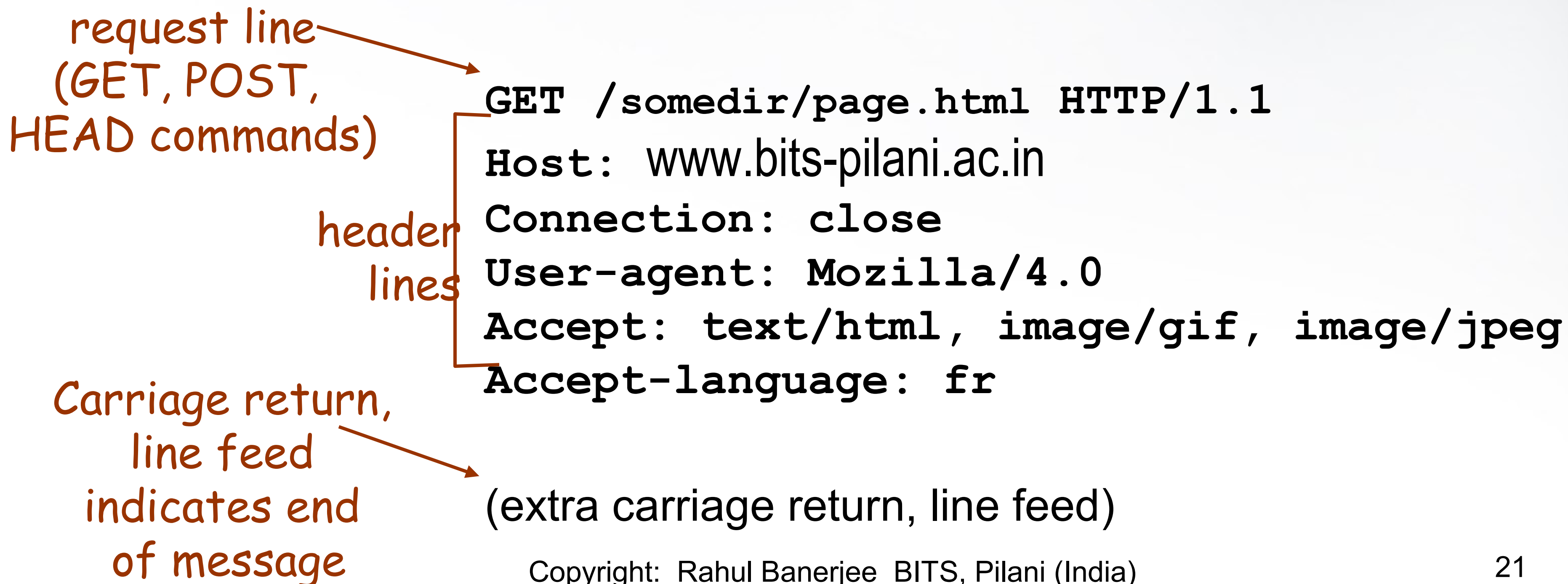
HTTP Request Message Format



Copyright: Rahul Banerjee BITS, Pilani (India)

The HTTP Protocol: Message Format

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



Copyright: Rahul Banerjee BITS, Pilani (India)

HTTP Message Format: Response

status line (protocol status code status phrase)

header lines

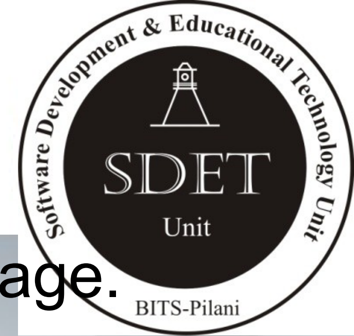
data, e.g., requested html file

```
HTTP/1.1 200 OK
Date: Wed, 29 Apr 2004 12:00:15 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 26 Apr 2004...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

Copyright: Rahul Banerjee BITS, Pilani (India)

HTTP Response Status Codes



In the first line of the 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

Copyright: Rahul Banerjee BITS, Pilani (India)

Trying out HTTP (client side) for yourself

1. Telnet to the BITS Web server:

telnet www.bits-pilani.ac.in **80**

Opens TCP connection to port 80 (default http server port) at www.bits-pilani.ac.in. Anything typed in sent to port 80 at www.bits-pilani.ac.in

2. Type in a GET http request:

```
GET /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!

Copyright: Rahul Banerjee BITS, Pilani (India)

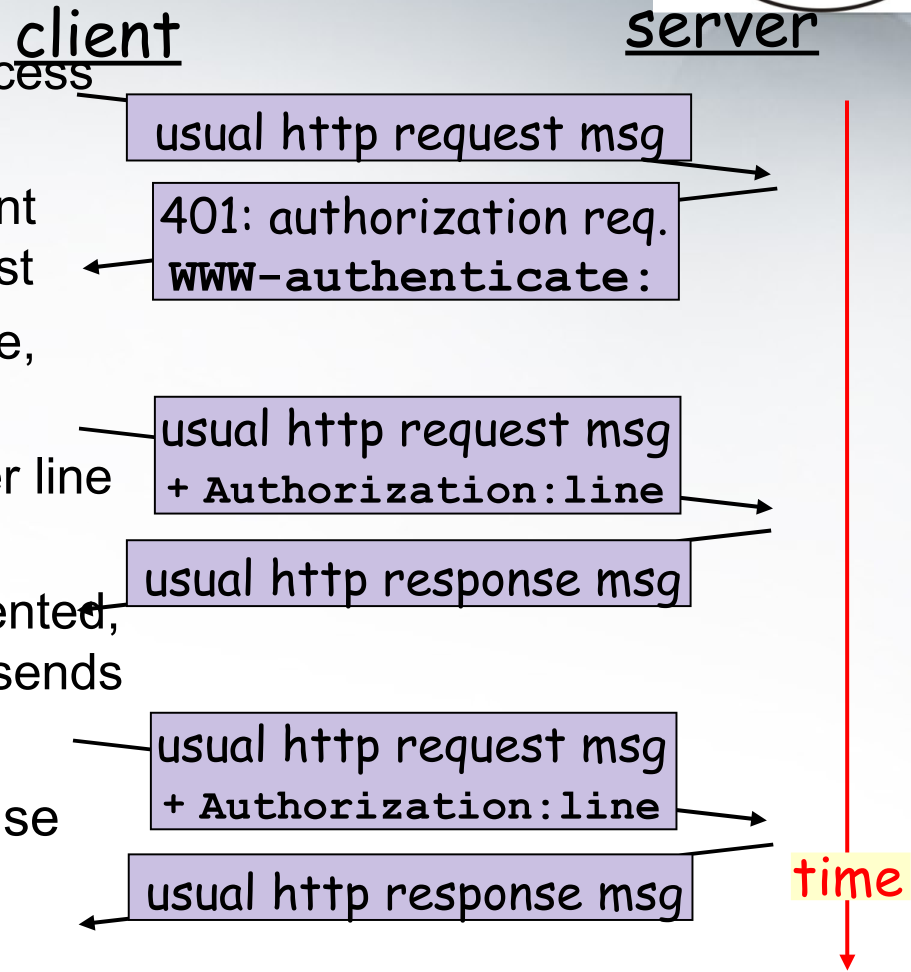
User-Server Interaction: Authentication

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - `Authorization:` header line in request
  - if no authorization presented, server refuses access, sends

    `WWW-authenticate:`

    header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

User-server Interaction: Cookies

Software Development & Educational Technology Unit
SDET
Unit
BITS-Pilani

- server sends "cookie" to client in response msg

  `Set-cookie: 1678453`
- client presents cookie in later requests

  `Cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices

client

server

usual http request msg

usual http response +
Set-cookie: #

usual http request msg
Cookie: #

cookie-specific action

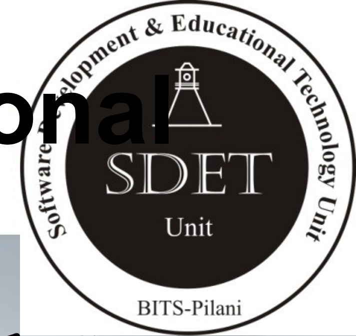
usual http response msg

usual http request msg
Cookie: #

cookie-specific action

usual http response msg

User-Server Interaction: Conditional GET



- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:

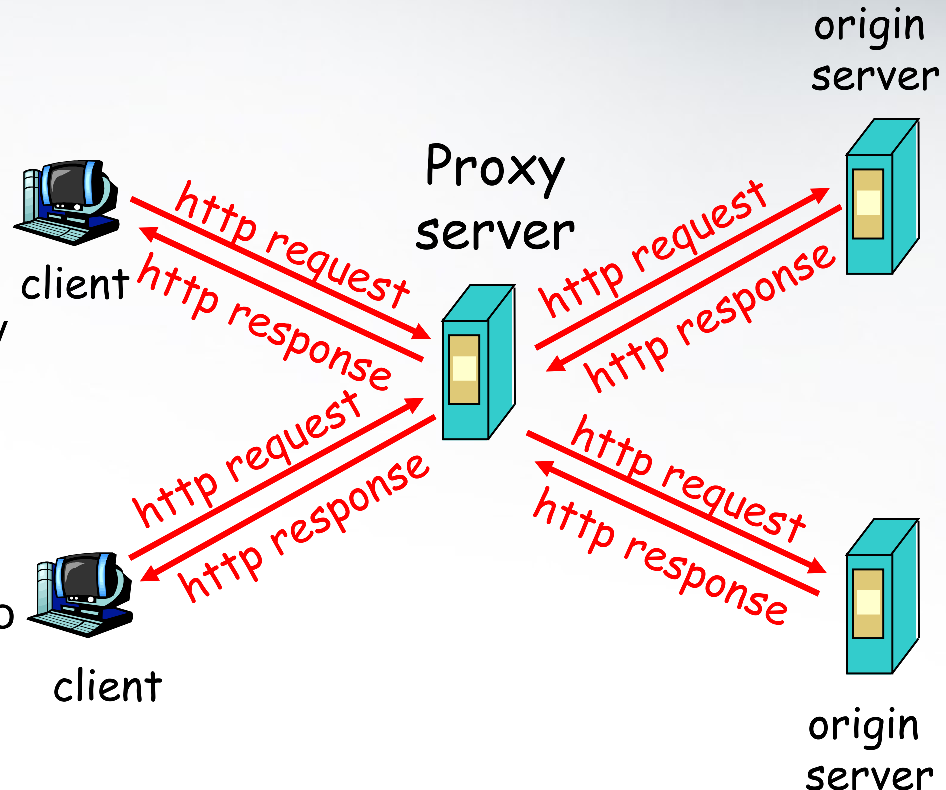
  `HTTP/1.0 304 Not Modified`

Copyright: Rahul Banerjee BITS, Pilani (India)

Web Caches (HTTP Proxy)

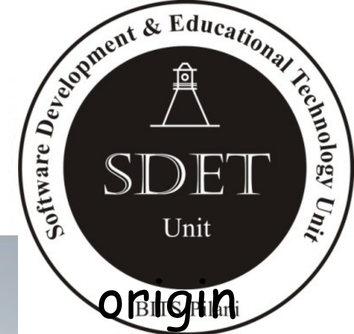
Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



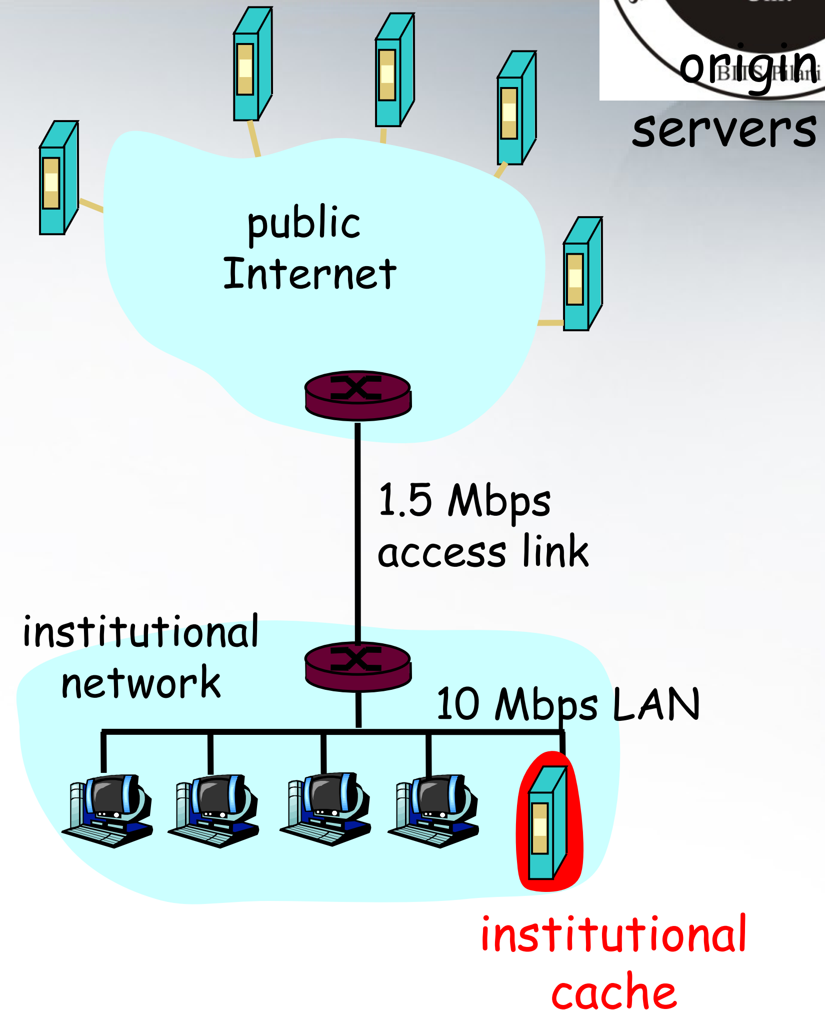
Copyright: Rahul Banerjee BITS, Pilani (India)

Why Web Caching?



Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck



Copyright: Rahul Banerjee BITS, Pilani (India)

Cache Sharing: Internet Cache Protocol (ICP)

. . . Rest of Internet . . .

Bottleneck

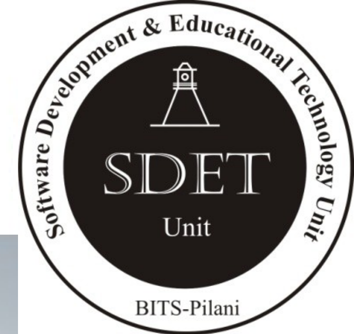
Regional Network

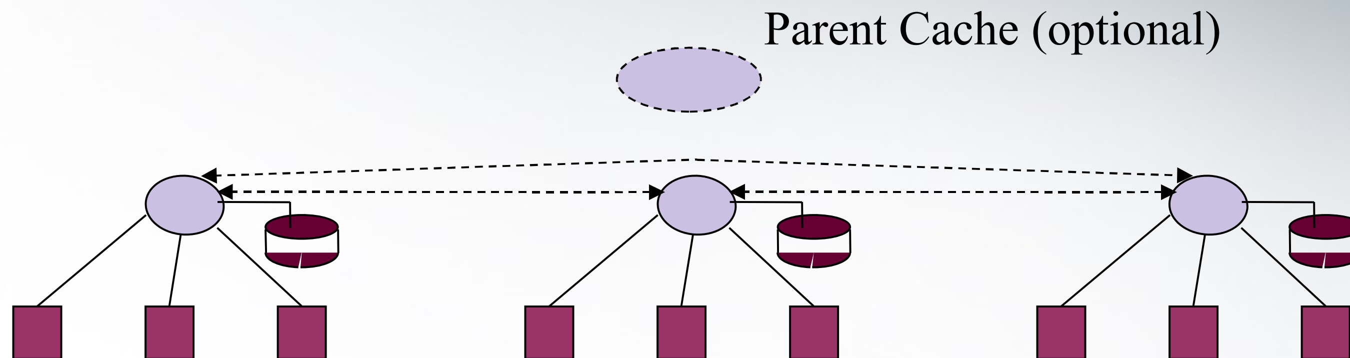
Proxy Caches

Users

Copyright: Rahul Banerjee BITS, Pilani (India)

Cache Sharing via ICP

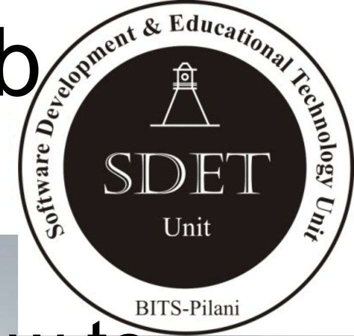




- When one proxy has a cache miss, send queries to all siblings (and parents): “do you have the URL?”
- Whoever responds first with “Yes”, send a request to fetch the file
- If no “Yes” response within certain time limit, send request to Web server

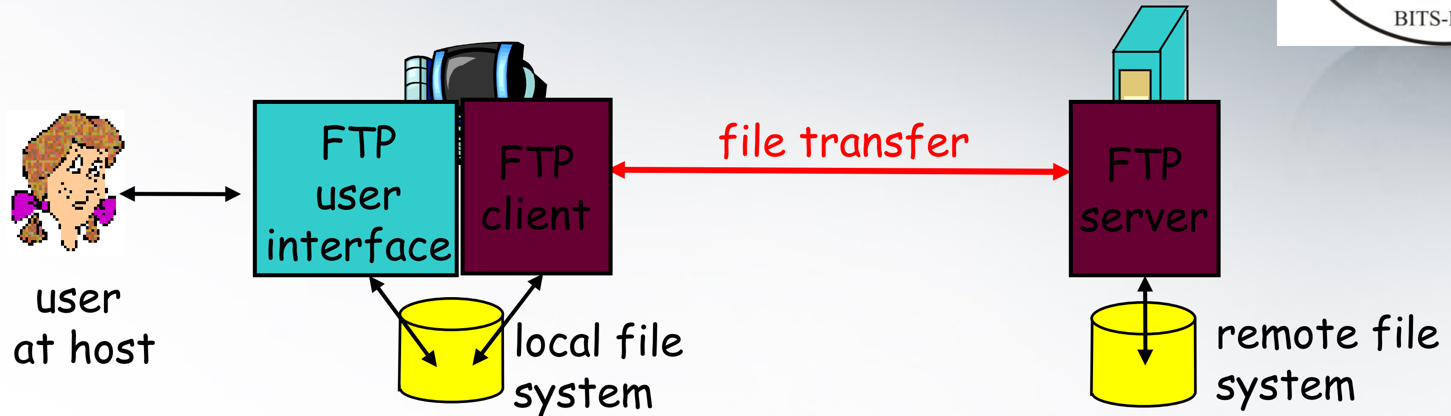
Copyright: Rahul Banerjee BITS, Pilani (India)

Economics: Consistency of Web Caching



- The major issue of web caching is how to maintain consistency
- Two ways
  – Pull
    - Web caches periodically pull the web server to see if a document is modified
  – Push
    - Whenever a server gives a copy of a web page to a web cache, they sign a lease with an expiration time; if the web page is modified before the lease, the server notifies the cache

FTP: the File Transfer Protocol



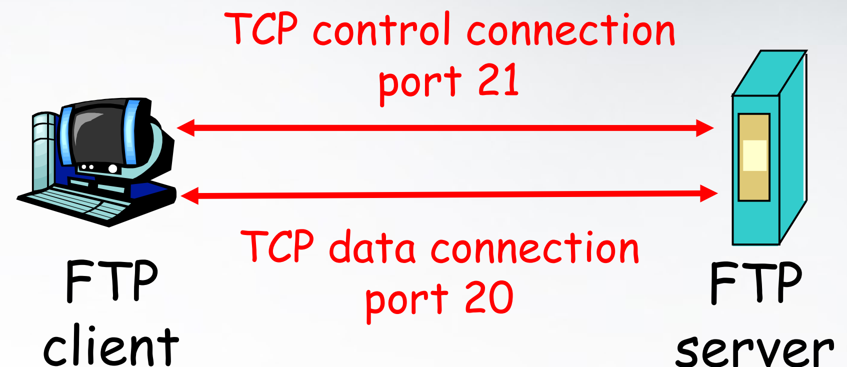
- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

Copyright: Rahul Banerjee BITS, Pilani (India)

FTP: Separate Control, Data Connections

- ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- two parallel TCP connections opened:
  - control: exchange commands, responses between client, server.

    "out of band control"
  - data: file data to/from server
- ftp server maintains "state": current directory, earlier authentication



FTP Commands, Responses

Software Development & Educational Technology Unit SDET Unit BITS-Pilani

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**

Copyright: Rahul Banerjee BITS, Pilani (India)

What is Email?

- **A mail that is sent electronically (often across the Internet).**
- **Quickly delivered in seconds to minutes (based on bandwidth, size, priority, policy etc.).**

	Telephone	E-mail	Post
Speed	High	Moderate	Low
Synchronized	Yes	No	No
Formality	Varies	Moderate	Varies
Conferencing	Small Group	Any to all	One-way only
Security	Moderate	Low	High

Pros & Cons

Advantages

- Convenience, Fast speed, Inexpensive, Printable, Reliable, Global, Generality (not limited to text, but graphics, programs, even sounds)

Disadvantages

- Misdirection, Interception, Forgery, overload, Funk (Spamming), No response (from the receiver).

The growth of Email Users & Messages

E-mail Users and Messages

500,000,000
400,000,000
300,000,000
200,000,000
100,000,000

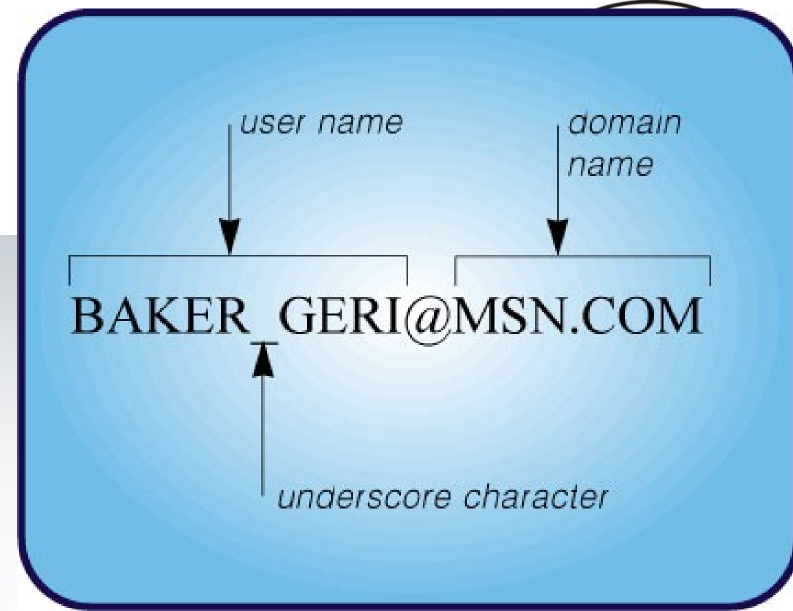
messages per day
users

1969 1994 1995 1996 1997 1998 1999 2000 2001

source: Forrester Research

- Source: "Discovering 2000", Shelly Cashman Series, Thomson Learning

Email address



- What you need is
  - The E-mail address of the recipient.
  - user@host
  - cccheung@cse.cuhk.edu.hk
  - eng70132@leonis.nus.sg

- "cse.cuhk.edu.hk" is the domain name of the mail server which handles the recipient's mail.
- “cccheung" is the user name of the recipient.
- User name and hostname are separated by "@".

Email server and client

- Email client – software / program that can transfer e-mail from a local host to a local e-mail server.
- Email server – software/program that can send/receive e-mail from/to other email servers.
- Mailbox – An electronic mailbox is a disk file which holds email messages.

Cc & Bcc

- Carbon Copy Section
  - Send a message to more than one person, all the recipients will see the list of email addresses.
- Blind Carbon Copy Section
  - The addresses won't be seen by the recipients.
  - When email is sent to a large group of people who don't know each other.

Attachment – MIME
Multi-purpose Internet Mail Extension

- A protocol for transmitting non-text information across the Internet. Basically, non-ASCII data is converted to ASCII for transmission and then converted back at the receiver.
- A specification for automatically sending objects other than text in email messages.
- MIME is usually associated with multimedia, such as images, audio recordings, and movies.
- Additional hardware and helper software are usually required.
- Common MIME-compliant mailers:
  - pine, metamail, Netscape messenger, MS Outlook

Encoding
ASCII & Double-byte

Encoding	Representation
Latin	a
ASCII Hex	0x61
Decimal	97
Binary	0110 0000
Octal	140

Encoding	Representation
Kanji	今
Unicode	\u4ECA
Shift_JIS	8DA1
EUC_JP	BAA3
Big 5	A4B5
UTF-8	E4BB8A

Common MIME Types

Type	Subtype	Description	File extensions
Application	postscript text	Printable postscript document TEX document	.eps, .ps .tex
Audio	midi realaudio wav	Musical Instrument Digital Interface Progressive Networks sound Microsoft sound	.midi, .mid .ra, .ram .wav
Image	gif jpeg png	Grapical Interchange Format Joint Photographic Experts Group Portable Network Graphics	.gif .jpeg, .jpg, .jpe .png
Model	vrml	Virtual Reality Modeling Language	.wrl
Text	html plain	Hypertext Markup Language Unformatted text	.html, .htm .txt
Video	avi mpeg quicktime	Microsoft audio video interleaved Moving Picture Experts Group Apple QuickTime movie	.avi .mpeg, .mpg .qt, .mov

MIME headers

- MIME-Version: 1.0
- Content-type: multipart/mixed: boundary="simple boundary"
- --simple boundary
- ……
- --simple boundary
- ……
- --simple boundary--

How does email works?

HOW AN E-MAIL MESSAGE TRAVELS FROM THE SENDER TO THE RECEIVER

SMTP

- E-mails are transferred across the Internet via Simple Mail Transfer Protocol (SMTP).
- The mail server uses SMTP to determine how to route the message through the Internet and then sends the message.
- When the message arrives at the recipient's mail server, the message is transferred to a POP3 server. POP stands for Post Office Protocol.
- The POP server holds the message until the recipient retrieves it with his/her email software.

SMTP illustration

John's computer

Mary's computer

SMTP

Internet

SMTP

SMTP

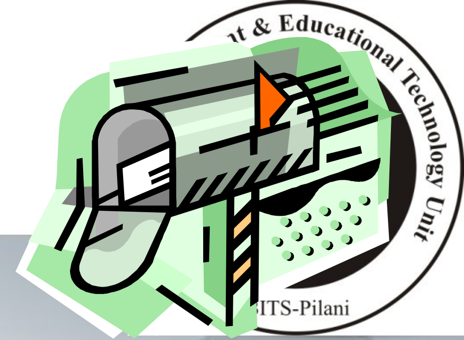
Mail server

Mail server

POP

- Advantages
  - Don’t have to know the name of your machine
  - POP mail server is installed on a computer always ON
  - Use Windows interface to read email
- Disadvantages
  - The email at the mail server is popped to your local machine

POP illustration



John's computer

SMTP

Mail server

SMTP

Internet

SMTP

Mary's computer

Local inbox

POP

Mail server

Mail spool

IMAP

- Another popular method by which users obtain their emails is called a central mail spool.
- Imagine what could happen if Peter where reading emails at his office while his wife was simultaneously trying to read from the same inbox from home.
- Lots of complication can arise in this scenario, and a protocol has been designed to handle many of the relevant issues. It's called the Interactive Mail Access Protocol (IMAP).
- Peter's emails remain on his mail server. The emails are not brought over to the computer from which he is working. That is, leaving the emails in a central location, and allowing access of the emails from several places.
- That is, when Peter wants to read his emails, he must send a password to the mail server to authenticate himself.
- Another advantage of IMAP is that it encrypts passwords so that someone *sniffing* the network cannot directly obtain his password.

IMAP illustration

John's home computer

John's office computer

IMAP link

IMAP link

Incoming/
Outgoing emails

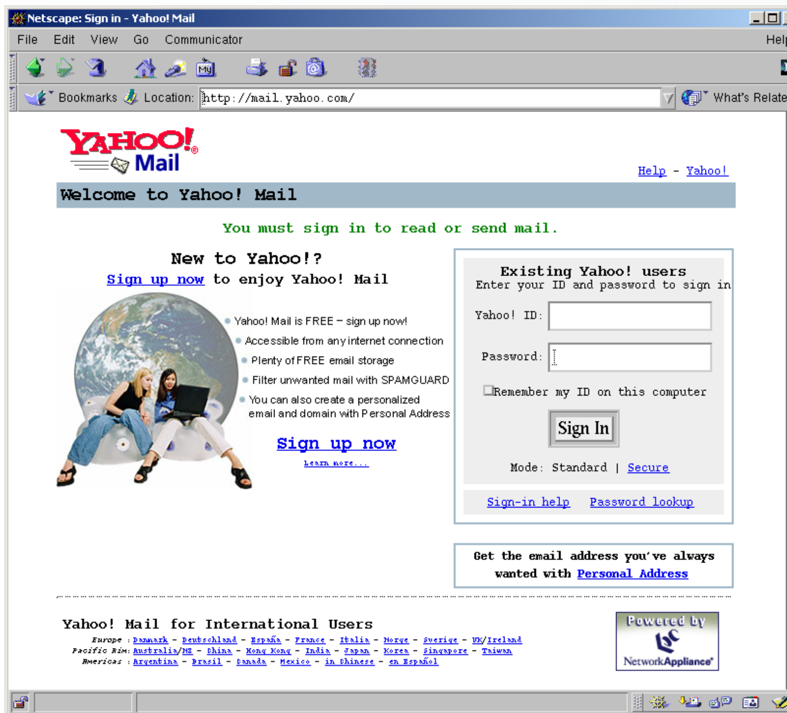
John
inbox

Mail server (SMTP/IMAP)

Web-based e-mail - HTTP

Software Development & Educational Technology Unit
SDET Unit
BITS-Pilani

- Can deliver mail message in web page format.
- More reliable to use POP and IMAP than HTTP mail account.



Email Virus - Melissa

- May 24, 1999, Melissa virus is spreading as RTF files.
- 300 organizations affected, 100,000 hosts.
- One site reported, 32,000 copies of email are received in 45 minutes.
- Not a worm, require user interaction to propagate the virus.
- Why called Melissa?
  - Named by the antivirus software vendors.
- Don’t open attachment → not infected.

Junk Mails

- How can they get into your mailbox?
  - From name card, letter heads, published papers.
  - Use search engine in the newsgroup, bulletin boards, phone books.
  - Dump a full user list in a server.
- How to stop the intrusion of Junk Mails?
  - Mail server providers joint effort
  - Filtering
  - Preview before downloading

SPAM

- SPAM is flooding the Internet with many copies of the same message
  - Force to send message to people
  - Junk electronic mail.
- Why cause problem?
  - Cost-shifting – very cheap to send thousands of emails
  - Fraud – not an advertisement subject
  - Waste of others' resources – stealing bandwidth
  - Displacement of Normal Email – destroy the usefulness and effectiveness of email
  - Ethics problem

CAUCE - Coalition Against Unsolicited Commercial E-mail

- An all volunteer organization which is created by Netizens over the world.

DHCP: *The Dynamic Host Configuration Protocol*

Purpose of DHCP

SDET Unit
Software Development & Educational Technology Unit
BITS-Pilani

The RFC 2131 mentions:

- *The Dynamic Host Configuration Protocol (DHCP) provides a framework for passing configuration information to hosts on a TCP/IP network.*
- *DHCP consists of two components:*
  - *a protocol for delivering host-specific configuration parameters from a DHCP server to a host*
  - *a mechanism for allocation of network addresses to hosts.*

DHCP functional goals

- A host without a valid IP address locates and communicates with a DHCP server
- A DHCP server passes configuration parameters, including an IP address, to the host
- The DHCP server may dynamically allocate addresses to hosts and reuse addresses
- Hosts can detect when they require a new IP address
- Unavailability of DHCP server has minimal effect on operation of hosts

DHCP & BOOTP Compatibility

- Backward compatible packet format for BOOTP interoperation (RFC 1542)
- DHCP can coexist with hosts that have pre-assigned IP addresses and hosts that do not participate in DHCP

Design Goals

- Eliminate manual configuration of hosts
- Prevent use of any IP address by more than one host
- Should not require a server on every subnet
- Allow for multiple servers
- Provide a mechanism, not a policy
- Provide same configuration - including IP address - to a host whenever possible

What can you do with DHCP

- Plug-and-play
- Move desktop PCs between offices
- Renumber
- Other restructuring - change subnet masks
- Mobile IP
- Moving equipment - cartable

What DHCP *doesn't* do

- Support multiple addresses per interface
- Inform running host that parameters have changed
- Propagate new addresses to DNS
- Support inter-server communication
- Provide authenticated message delivery

What DHCP *doesn't* do

- Configure routers and other network equipment
- Design network addressing plan
- Determine other configuration parameters
- Locate other servers

Implementation status

- DHCP is an open standard, with freely available specifications
- Can be (and has been) implemented entirely from the specification
- Commercial implementations are widely available
- Non-commercial implementations are also available

Planning for DHCP

- Preparation for DHCP requires careful planning
- IP addressing strategy
  - Consider current needs
  - Allow for growth
- Network architect configures rules for addressing strategy into DHCP server

Newly installed computer

- Newly installed computer locates DHCP server
- Server consults address scheme rules
  - Picks an address
  - Determines other configuration parameters
- "Plug-and-play"

Relocated computer

- Computer retains address
- When restarted, computer checks with server to confirm address
- If address OK, computer retains old address
- If computer attached to different subnet, obtains new address

Using DHCP with legacy equipment

- DHCP server *not* required to make every address on a subnet available for allocation
- DHCP server *not* required to answer every incoming request
- Network architect can configure server to reserve (not allocate) addresses

DHCP and new computers

- DHCP server will hand out all available addresses
- Limited number of addresses can be shared (if all computers not on simultaneously)
- Eventually, network architect will have to allocate more addresses

Reusing addresses

- Server can reuse abandoned addresses
  - Address initially allocated for fixed time called a *lease*
  - Client can extend lease
- If lease expires, server can reallocate
- Reallocation only when necessary (e.g., LRU) is a good idea…

Reconfiguring the server for multiple networks

- Server configuration file defines multiple subnets and address pools on one physical segment
- Server chooses address from pools for the segment
- Server checks DHCP client address against all subnets on the segment

Using DHCP for renumbering

- Set up plan for renumbering
  - New network architecture
  - Network addresses, server addresses
  - Timing of cutovers
- Force DHCP clients to contact server for notification about new address
  - Set short leases
  - Require all clients be rebooted

Using DHCP for renumbering

- Rebooting, although not elegant, probably most reliable
- Schedule subnet cutover for overnight or weekend, force reboot through “alternate protocol” (e.g.., e–mail to all users)

Address allocation

- *Static* (BOOTP): client must be pre-configured into database
- *Automatic*: server can allocate new address to client
- *Dynamic*: server can allocate and reuse addresses

Leases

- Dynamic addresses are allocated for a period of time known as the lease
- Client is allowed to use the address until the lease expires
- Client MUST NOT use the address after the lease expires, even if there are active connections using the address
- Server MUST NOT reuse the address before the lease expires

Motivation for leases

- An IP internetwork may not always be completely operational; there may not always be connectivity between any two hosts, so:
  - Can’t use distributed (client-based) assignment of addresses
  - Can’t use “address defense” before server reuse of addresses
  - Leases guarantee an agreement as to when an address may be safely reused even if the server can’t contact the client

Address reuse

- Server MAY choose to reuse an address by reassigning it to a different client after the lease has expired
- Server can check using ICMP echo to see if the address is still in use (but no response is not a definitive answer!)
- Allows address sharing
  - From old computers replaced by new ones
  - Among a pool of computers not always using TCP/IP
  - For transient hosts like laptops

Address allocation details

- Clients check on address validity at reboot time (renumbering)
- Clients can extend the lease on an address at startup time
- Clients can extend the lease on an address as expiration time approaches (without closing and restarting existing connections)
- Clients with addresses that have been configured manually can use DHCP to obtain other configuration parameters

Four ways a client uses DHCP

- INIT - acquire an IP address and configuration information
- INIT-REBOOT - confirm validity of previously acquired address and configuration
- RENEWING - extend a lease from the original server
- REBINDING - extend a lease from any server

Obtaining an initial address

- Client broadcasts DISCOVER to locate servers
- Server chooses address and replies
- Client selects a server and sends REQUEST for address
- Server commits allocation and returns ACK

Rebooting client

- Client puts address in REQUEST and broadcasts
- Server checks validity and returns ACK with parameters
- If client address is invalid – e.g., client is attached to a new network – server replies with NAK and client restarts

Extending a lease

- Client puts requested lease extension in REQUEST and sends to server
- Server commits extension and returns ACK with parameters

DHCP options

- Options carry additional configuration information to client
  - DHCP message type
  - Subnet mask, default routers, DNS server
  - *Many* others …
- Carried as fields in DHCP message

Configuration with options

- Network architect configures server to select and return options and values
- Client can explicitly request specific options

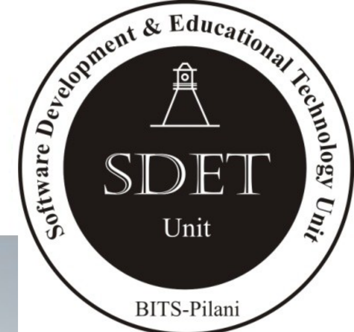
Relay agents

- Using hardware and IP broadcast still limits DHCP message from client to single physical network
- *Relay agent*, on same subnet as client, forwards DHCP messages between clients and servers
- Relay agent and server exchange messages using unicast UDP
  - Servers can be located anywhere on intranet
  - Servers can be centrally located for ease of administration
- Very simple in function, implementation
- Usually, but not necessarily, located in routers

Strategies for using multiple DHCP servers

- Split address pool for each subnet among servers
- Coordinate leases off-line
- Reallocate addresses when needed

Lease times and strategies



- Choice of lease times made by DHCP administrator
- Long lease times decrease traffic and server load, short lease times increase flexibility

Lease times and strategies

- Should choose lease time allow for server unavailability
  - Allows clients to use old addresses
  - For example, long enough to span weekends
- Can assign different leases to desktop computers, cartable systems and laptops

Changing other configuration parameters

- Other configuration parameters such as print servers may change
- Reconfigure DHCP server with new parameters
- At next reconfirmation, clients will get new addresses

Moving a client to a new location

- User may get moved to a new location on a different subnet
- User may arrange to move computer system without contacting network administrator
- DHCP will allocate address for new location
- What about old lease?
  - New server can notify network administrator about address allocation
  - Client can issue RELEASE before moving from old location
- Or, might be appropriate to leave old lease in place…

Replacing a system

- User may get new computer on desktop
- Network administrator wants to allocate same IP address to the new computer – but, new computer will have different hardware address
- Use client id as system identifier and transfer to new system

Coordination among multiple servers

- Becomes a distributed database problem
- Several strategies have been proposed
- “Failover protocol” now in development

Dynamic DNS

- When client is allocated a new address, DNS records need to be updated
  - A record: Name to IP address
  - PTR record: IP address to name
- DHCP to be extended to allow coordination between client and server
  - Which does updates?
  - Error conditions?

Security/Authentication

- Unauthorized – either intentional or accidental – server can cause denial of service problems
- Some sites may want to limit IP address allocation to authorized client
- Authentication based on shared secret key, an authentication ticket and a message digest
- Assures source of message is valid and message hasn't been tampered with en route
- Schiller/Huitema/Droms/Arbaugh proposal in process

New options acceptance

- New options must have non–overlapping option codes
- Codes handed out by *Internet Assigned Numbers Authority* (IANA)
- New mechanism will approve each new option as a separate RFC (like TELNET)

IPv6 & Its impact

- Includes new features for host configuration:
  - Router advertisement
  - Autoconfiguration
  - Link-local addresses

IPv6

- To accommodate sites that want centralized management of addresses, *DHCP for IPv6* (DHCPv6) has been developed by the DHCP WG.

Summary

- DHCP works today as a tool for automatic configuration of TCP/IP hosts
- It is an open Internet standard and interoperable client implementations are widely available

Summary

- Provides automation for routine configuration tasks, once network architect has configured network and addressing plan
- Ongoing work will extend DHCP with authentication, DHCP-DNS interaction and inter-server communication

About Application Client / Server Processes

- **Definition of an Application Server Process:**
  - **A process that provides any set of predefined services to one or more requesting clients is called a Server Process.**
- **Types of Application Server Processes:**
  - **Concurrent Server Process**
    - **A process that simultaneously provides any set of predefined services to one or more requesting clients is called a Concurrent Server Process.**
  - **Iterative Server Process**
    - **A process that provides any set of predefined services to only one requesting client at any point of time is called an Iterative Server Process.**
- **Definition of an Application Client Process:**
  - **A process that solicits any specific service from any designated server is called a Client Process.**

The TCP/IP Access Points

- In a TCP/IP network, services offered by any layer can only be used through parameter / data passing through the various Service Access Points (SAPs) located on Layer-boundaries.
- TSAPs and NSAPs are two major examples of SAPs located at the AL-TCP/UDP Layer-boundary and TCP/UDP-IP Layer-boundary respectively.
  - A typical example of an NSAP is an IP address.
  - A typical example of a TSAP is an IP address + a 16-bit integer called as Port Number.
- Port Numbers permit unique identification of several simultaneous processes using TCP / UDP.
- IETF's RFC 1700 lists select IANA-suggested Port Number Assignments.

- Network Programming
- in MS Windows environments

Methods of Network Programming in MS-Windows

- Windows Network Programming support exists for:
  - TCP/IP (IETF's)
  - Named Pipe <FIFO>, <server creates a pipe and awaits, allowing bi-directional data-flow>
  - Mailslots <uni-directional guaranteed or non-guaranteed data-flow, uses broadcast datagrams, can be created by any process>
  - IPX/SPX (Novell's Netware is based on this stack)
  - NetBEAU (NETBIOS Extended User Interface) <for PC-based networking, exposes nterfaces for reading / writing data, can be used for C/S or P2P, connection-oriented or connection-less way>
  - AppleTalk (Apple's)

Methods of Network Programming in MS-Windows

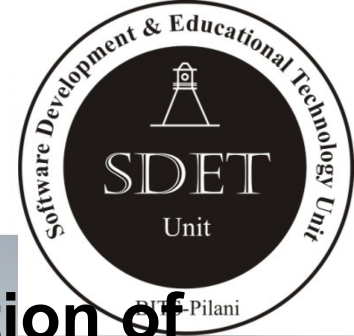
- Methods available:
  - Windows NT Remote Procedure Call (RPC) <uses a collection of libraries / tools with a focus on the Application Procedure alone, targeting execution on a remote machine, independent of transport protocols>
  - Windows Socket (WinSock) <uses Windows Socket interface, extension of BSD Sockets, allows bi-directional data-flow, upon completion of asynchronous I/O, exports functions which transmit messages to the relevant applications: feature fits within the message-driven interface model of MS-Windows>
  - WinNet API <for over-the-MS-Windows-network sharing of file-server / printer etc.>

- Network Programming
- in Linux / UNIX and similar environments

The Sockets and Socket Pairs

- Socket is another name for the TSAP discussed above. Thus, it comprises of IP Address+Port Number. **Sockets are often created by the *socket()* system call.**
- **A Socket Pair refers to a set of socket endpoints at the two communicating ends of a TCP/IP network.** Typically, a Socket-Pair, for TCP as well as UDP, comprises of four components:
  - IP Address-1 & Port Address/Number-1
  - IP Address-2 & Port Address/Number-2
- In case of TCP, a Socket-pair typically uniquely represents a TCP-connection (logical).

Creating a Socket

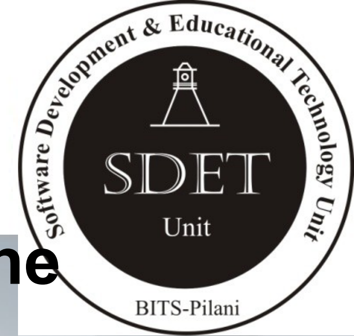


- **Every network protocol has its own definition of *Network Address*.**
- **In C, a protocol implementation provides a**
- ***struct sockaddr***
- **as the elementary form of a *Network Address.***
- **A sample definition of *struct sockaddr***

```
#include <sys/socket.h>

struct sockaddr {
unsigned short sa_family;
char sa_data [MAXSOCKADDRDATA]
}
```

Creating a Socket …



- In UNIX and Linux, Sockets are created by the *socket()* system call.
- This call returns a *file descriptor* for the Socket that is yet to be initialized.
- Then the socket is initialized by binding it to a particular protocol and address using the *bind()* system call.

```
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
        Here,
        the parameter domain specifies the PF;
        parameter type usually specifies either of SOCK_STREAM or SOCK_DGRAM;
        parameter protocol specifies the protocol to be used (0=> default protocol associated).
int bind (int sock, struct sockaddr * my_addr, int addrlen);
```

- Here, the parameter sock is the socket-in-question; parameter sockaddr is the address of protocol; parameter addrlen is the length of the address for the local end-point.

Creating a Socket …

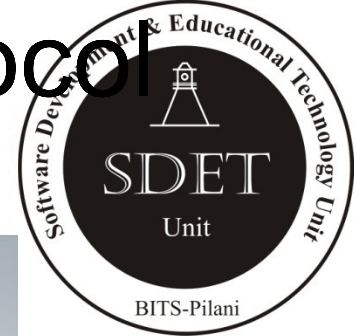
- Next, *listen()* system call is executed for informing the system that the process is now ready to allow other processes establish a connection to this socket at the specified end-point.
- This step does not really establish a connection by itself, however!
- Now, the *accept ()* system call is made for accepting the connection requests.
- *accept ()* is a blocking call as it blocks until a process requests a connection. In case, the socket has been marked as 'non-blocking' by the *fcntl()* call, *accept()* would return an error if no process is requesting it for a connection.

```
#include <sys/socket.h>

int listen (int sock, int backlog);
    Here,
    the parameter sock is the socket-in-question;
    parameter backlog is the number of connection requests that may be pending on the socket before any further
        connection requests are explicitly refused.
int accept (int sock, struct sockaddr * addr, int * addrlen);
```

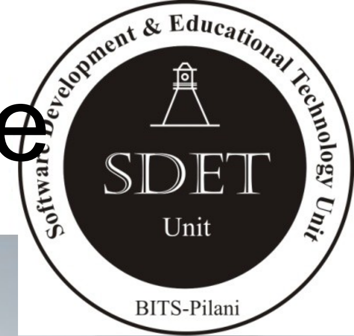
- The *select ()* system call can also be made for determining if any connection request is currently pending to a socket.
- Similarly, a Client attempts to connect to a Client attempts to connect to a Server by creating a socket, binding it to an address (optionally) and making the *connect()* call to the Server at the known address.

A Glimpse of Address and Protocol Families for Various Stacks



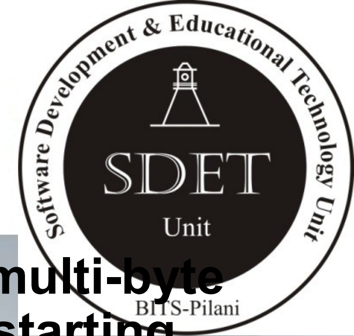
- **Address Families:**
- **Unix / Linux Domain: AF_UNIX**
- **TCP/IPv4 Domain: AF_INET**
- **TCP/IPv6 Domain: AF_INET6**
- **Novell NetWare Domain: AF_IPX**
- **AppleTalk Domain: AF_APPLETALK**
- **Protocol Families:**
- **Unix / Linux Domain: PF_UNIX**
- **TCP/IPv4 Domain: PF_INET**
- **TCP/IPv6 Domain: PF_INET6**
- **Novell NetWare Domain: PF_IPX**
- **AppleTalk Domain: PF_APPLETALK**

The IP Socket Address Structure



- struct in_addr {
  in_addr_t s_addr; /* Big Endian 32-bit IP address */
  };
- struct sockaddr_in {
- uint8_t sin_len; /* 16-byte structure length */
- sa_family_t sin_family; /* AF_INET */
- in_port_t sin_port; /* 16-bit Big Endian TL Port */
- struct in_addr sin_addr; /* 32-bit Big Endian IP address*/
- char sin_zero [8];
- }
- /* This is defined in netinet/in.h file. */

On the Byte-ordering and Byte-manipulation Functions



- **The term Byte-ordering refers to the manner in which a multi-byte / multi-octet number / field is stored in Lower-order (i.e. starting address) and Higher-order Memory Addresses.**
- **If the Lower-order Byte is stored in the Starting / Lower-order Memory Location and Higher-order Memory Location, then it is called the Little Endian scheme.**
- **If the storage is carried out in exactly opposite manner, it is called the Big Endian scheme.**
- **Different manufacturers may choose any one of these Byte-ordering Schemes. For instance, traditionally, Motorola processors have followed the Big Endian scheme whereas the Intel processors have used the Little Endian scheme.**
- **The Network Byte-order is always Big Endian, by convention whereas the Host Byte-order may be of either type.**
- ***None of these two schemes is superior or inferior to the other since they merely represent two possible ways in which Lower and Higher Order Bytes are stored into and retrieved from Memory or transferred over a network link.***

On the Byte-ordering and Byte-manipulation Functions (Contd.)

- **Over a network, between any two nodes, data transfer takes place in Big Endian manner.**
- **There are standard function prototypes for 'Host-Byte-order to Network-Byte-order' conversion and 'Network-Byte-order to Host-Byte-order' conversion. These functions are defined in the netinet/in.h header file in the systems supporting Socket Programming in C and its variations. Their usage has been shown below:**

```
#include <netinet/in.h>
uint16_t htons (uinit16_t        16-bit-host-address)
uint32_t htonl (uinit32_t        32-bit-host-address)
uint16_t ntohs (uinit16_t        16-bit-network-address)
uint32_t ntohl (uinit32_t        32-bit-network-address)
```

- **There exist two categories of functions those are capable of operating on multi-byte / multi-octet numbers / fields.**
- **The first category has BSD functions like bcopy, bzero etc. Functions belonging to this category begin with 'b' (b=> byte).**
- **The second category has ANSI-C functions like memset, memcpy etc. Functions belonging to this category begin with 'mem' (mem=> memory). Both types are defined in the header files strings.h and string.h respectively.**

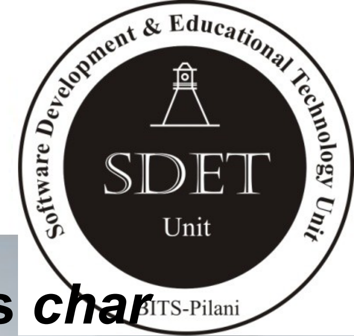
On the Functions used for Address Conversions

- **A few functions used for IPv4 Address conversion from / to ASCII string and Binary strings expressed in the Network Byte-order have been defined in arpa/inet.h header file. These include: inet_pton, inet_ntop, inet_aton, inet_ntoa, inet_addr etc.**
- **Here, inet=> internet, a=> ASCII, n=> network, addr=> address, ntoa and aton refer to network to ASCII and ASCII to network respectively.**
- **The first two of these functions can handle IPv4 as well as IPv6 addresses and are therefore commonly used these days since majority of implementions attempt to offer dual-stack compatibility.**
- **It is advisable to avoid use of inet_addr, inet_aton and inet_ntoa as these have known problems as well as are considered deprecated. In near future, they may not be supported.**
- **Using the preferred functions:**

```
#include <arpa/inet.h>
int inet_pton ( int <addr-family>, const char *<strptr>, void *<addrptr> );
const char *inet_ntop ( <addr-family>, const void *<addrptr>, char *<strptr>, size_t
        <len> );
```

- **/* Here, addr-family may be AF_INET or AF_INET6 whichever needed. */**

On the Functions used for Address Conversions (Contd.)



- **As may be noticed in the syntax, inet_ntop requires *char *<strptr>,* this in turn requires to get the binary address that is stored as a part of a Socket Address Structure. This would mean some degree of protocol dependence of the function to be written since the programmer must know the said structure beforehand.**
-
- **const char *inet_ntop (<addr-family>, const void *<addrptr>, char *<strptr>, size_t <len>);**
- **This problem may be easily taken care of if the programmer writes his own function that extracts this data and its presentation format and supplies it to the standard function.**
- **There are numerous other similar situations in which such custom-built functions, if written by the programmer, increase the level of protocol independence of the resulting code. This, however, comes with its own overheads and may not be a preferable approach where speed and efficiency are the primary requirements.**

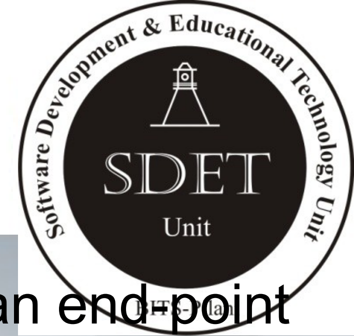
On the Functions used for I/O Operations on Stream Sockets and POSIX-compliant Status Information

- **Two points are important whenever a programmer wishes to perform various read and write operations on Stream Sockets.** ***(Stream here refers to Byte Stream as in case of TCP Sockets.)***
  - **A read operation or a write operation, even without an error, may read or write lesser number of bytes than explicitly mentioned particularly when a Buffer gets full. This does not mean that inaccurate result needs to be accepted; instead, it simply requires that the function is called again in order to read or write the rest of the bytes subsequently.**
  - **This may happen during all read as well as all non-blocking write operations.**
- **The fstat function has been used traditionally for getting the information about any specified descriptor. This function, along with others has a prototype in the sys/stat.h header file.**
- **Another function isfdtype has been more common in use of late. This is used as follows:**

```
#include <sys/stat.h>
int isfdtype (int <sock-fd>, int <fd-type>);
```

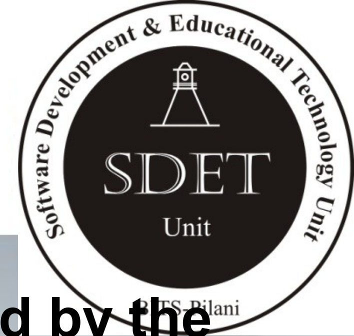
Here, fd=> file descriptor.

The TCP/IP Tips



- Syntax of the **socket** function that is required to create an end-point has been discussed earlier as:
- **#include <sys/socket.h>**
- **int socket (int <add-family or proto-family>, int <socket-type>, int <protocol>);**
- Unlike the **Server**, described earlier, the **Client** need not invoke the **bind** function call. In case of TCP over IPv4, this function, as discussed before, assigns a **32-bit+16-bit address** to a Socket created by the **socket** call. **Server** processes normally prefer to be assigned a standard-convention-based port called **Well Known Port**. Whenever used, syntax of the function call is is:
- **int bind (int <sock-fd>, const struct sockaddr *<assigned-addr>, socklen_t <addr-len>);**
- Syntax of the **connect** function that is needed by a Client to initiate a connection-request to a Server is:
- **int connect (int <sock-fd>, const struct sockaddr *<server-addr>, socklen_t <addr-len>);**

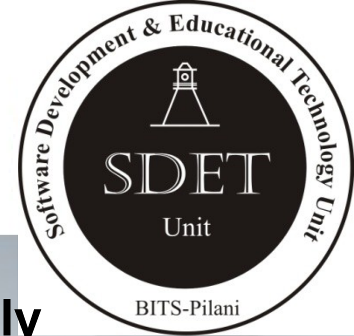
The TCP/IP Tips (Contd.)



- Just like bind, another function that is never invoked by the Client is the listen function. Syntax of the listen function that is required to make an unconnected Active Socket of a Server process behave as a Passive Socket entity has been discussed earlier as:
- #include <sys/socket.h>
- int listen (int <sock-fd>, int <composite-connect-queue-len>);

  Where, composite-connect-queue-len=> sum of complete+incomplete connection queues.
- Like bind and listen, another function that is called only by the Server is the function accept. As shown earlier, its syntax is:
- #include <sys/socket.h>
- int accept (int <sock-fd>, struct sockaddr *<client-addr>, socklen_t *<addr--len>);

Tips on Programming a Concurrent Server



- **Under UNIX and Linux environments, typically, the only available way to create a new process is by invoking the fork function that creates a Child Process. This function returns twice per invocation – once in the Parent Process that invoked it and thereafter in the created Child process. The first time it returns a value to the Parent that is actually the PID of the Child and second time it returns a value of zero to the Child.**
- **Syntax of its usage is:**

```
#include <unistd.h>
pid_t fork(void);
```

- **Interestingly, this function can be used for a different reason altogether, i.e. it can be used in association with the exec function whenever it is desired that a given process must invoke another process. *In this case, first a replica / child process is created by the* fork *call which then replaces itself with the process named as an argument to the* exec *function call.* The exec has six variations (?).**
- **Syntax of use of exec is:**

```
int exec?(const char *<file-or-path-name>, <argument-list>, <…>);
```

Tips on Programming a Concurrent Server (Contd.)

- **Steps involved in writing a simple Concurrent Server are:**
  - **Create a routine that creates a Socket, binds it to a well known address, changes it to listen mode and waits for a connection request at this address from a Client. (socket+bind+listen thus form Step One!)**
  - **Once a connection request is received, based on the queuing status, this routine has to ensure that following events occur:**
    - **The accept call returns,**
    - **The fork call is used to spawn a Child,**
    - **Let the Child close the listening Socket, service the Client etc. While in the mean time, the Parent returns to its passive ('listening') status and awaits the next request.**
    - **Once the Child has serviced the Client, it closes all open descriptors and exits,**
  - **Finally, the routine must ensure that the Parent closes the connected Socket. This completes the cycle.**
  - **Syntax for a UNIX / Linux close is:**

```
#include <unistd.h>
int close (int <sock-fd>);
```

A Few More Functions of Importance

- **Certain other functions of relevance include:**

ioctl, fcntl, recvfrom, sendto, signal,

select, poll, shutdown, pselect,

getsockname, getpeername, getsocketopt,

setsocketopt,

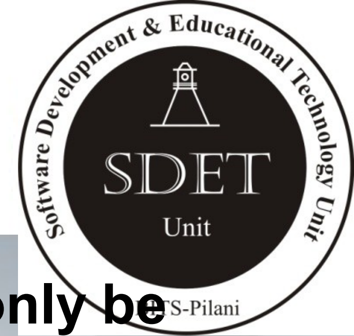
gethostbyname, gethostbyname2, gethostbyaddr,

uname,

gethostname, getservbyname, getservbyport, getaddrinfo, getnameinfo

gai_strerror, freeaddrinfo

Summary



- **In TCP/IP setup, services specific to any layer can only be accessed through Service Access Points located at the layer-boundaries.**
- **A TSAP is an *IP address+ a 16-bit Port Number*.**
- **Typically, a Socket-Pair, for TCP as well as UDP, comprises of Server IP Address, Port Address / Number of the Server, Client IP Address, Port Address / Number of the Client.**
- **In case of TCP, a Socket-pair typically uniquely represents a TCP-connection.**
- **Each network protocol may have its own definition of *Network Address*.**
- **The Network Byte-order is always Big Endian, by convention whereas the Host Byte-order may be of either type.**
- **functions used for IPv4 Address conversion from / to ASCII string and Binary strings expressed in the Network Byte-order have been defined in *arpa/inet.h* header file. These include: inet_pton, inet_ntop, inet_aton, inet_ntoa, inet_addr etc.**

Summary (Contd.)

- **Under UNIX and Linux environments, typically, the only available way to create a new process is by invoking the fork function that creates a Child Process. This function returns twice per invocation – once in the Parent Process that invoked it and thereafter in the created Child process.**
- **The function fork can be used in association with the exec function whenever it is desired that a given process must invoke another process.**
- **Commonly used functions include socket, bind, listen, connect, accept, close, getsockname, getpeername, select, poll, shutdown, pselect, getsocketopt, sesocketopt, ioctl, fcntl, recvfrom, sendto, signal, gethostbyname, gethostbyname2, gethostbyaddr, uname, gethostname, getservbyname, getservbyport, getaddrinfo, gai_strerror, freeaddrinfo, getnameinfo.**
- **If you know how to write a Client-Server pair for TCP-based application, we may easily identify the distinctly simple alterations that we may need to make in this code if we ever need to write a code for UDP-based application.**

Summary of a Quick tour to basics

- **Every network protocol has its own definition of *Network Address*.**
- **In C, a protocol implementation provides a**
- ***struct sockaddr***
- **as the elementary form of a *Network Address.***

- A sample definition of ***struct sockaddr***
- **#include <sys/socket.h>**
- **struct sockaddr {**
- **unsigned short sa_family;**
- **char sa_data [MAXSOCKADDR DATA]**
- **}**

Summary of a Quick tour to basics

- In Linux, sockets are created by the *socket()* system call.
- This call returns a *file descriptor* for the socket that is yet to be initialized.
- Then the socket is initialized by binding it to a particular protocol and address using the *bind()* system call.

- #include <sys/socket.h>
- int socket (int domain, int type, int protocol);
- Here,
- the parameter domain specifies the PF;
- parameter type usually specifies either of SOCK_STREAM or SOCK_DGRAM;
- parameter protocol specifies the protocol to be used (0=> default protocol associated).
- int bind (int sock, struct sockaddr * my_addr, int addrlen);
- Here,
- the parameter sock is the socket-in-question;
- parameter sockaddr is the address of protocol;
- parameter addrlen is the length of the address for the local end-point.

Summary of a Quick tour to basics

- Next, *listen()* system call is executed for informing the system that the process is now ready to allow other processes establish a connection to this socket at the specified end-point.
- This step does not really establish a connection by itself, however!
- Now, the *accept ()* system call is made for accepting the connection requests.
- *accept ()* is a blocking call as it blocks until a process requests a connection. In case, the socket has been marked as 'non-blocking' by the *fcntl()* call, *accept()* would return an error if no process is requesting it for a connection.
- A Client attempts to connect to a Client attempts to connect to a Server by creating a socket, binding it to an address (optionally) and making the *connect()* call to the Server at the known address.

- #include <sys/socket.h>
- int listen (int sock, int backlog);
- Here,
- the parameter sock is the socket-in-question;
- parameter backlog is the number of connection requests that may be pending on the socket before any further connection requests are explicitly refused.
- int accept (int sock, struct sockaddr * addr, int * addrlen);
- The *select ()* system call can also be made for determining if any connection request is currently pending to a socket.

References

1. **W. R. Stevens: UNIX Network Programming, Vols. 1-2, ISE, Addison-Wesley, Mass.**
2. **Alok K. Sinha: Network Programming in Windows NT, Addison-Wesley, Mass.**
3. **W. R. Stevens: TCP/IP, Vol. 1, Addison-Wesley, Mass.**
4. **D. Comer: Internetworking with TCP / IP , Vol.-1, PHI.**
5. **D. Comer & D. L. Stevens: Internetworking with TCP /IP, Vol.. 2-3, PHI.**
6. **M. K. Johnson and E. W. Troan: Linux Application Development, ISE, Addison-Wesley.**
7. **Rahul Banerjee: Lecture Notes in Computer Networking, BITS-Pilani.**