

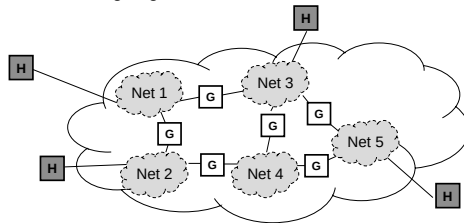
Lecture 2 Applications and Layered Architectures

*TCP/IP Architecture
How the Layers Work Together*



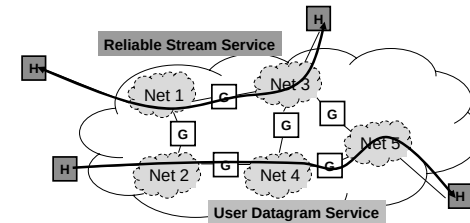
Why Internetworking?

- To build a “network of networks” or internet
 - operating over multiple, coexisting, different network technologies
 - providing ubiquitous connectivity through IP packet transfer
 - achieving huge economies of scale



Why Internetworking?

- To provide *universal communication services*
 - independent of underlying network technologies
 - providing common interface to user applications

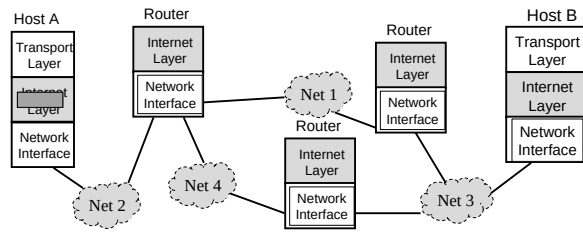


Why Internetworking?

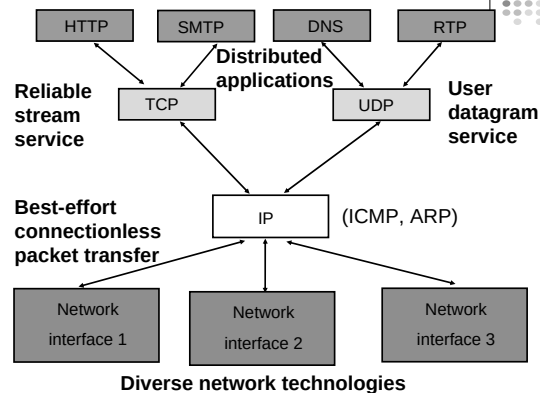
- To provide *distributed applications*
 - Any application designed to operate based on Internet communication services immediately operates across the entire Internet
 - Rapid deployment of new applications
 - Email, WWW, Peer-to-peer
 - Applications independent of network technology
 - New networks can be introduced below
 - Old network technologies can be retired

Internet Protocol Approach

- IP packets transfer information across Internet
Host A IP → router → router... → router → Host B IP
- IP layer in each router determines next hop (router)
- Network interfaces transfer IP packets across networks



TCP/IP Protocol Suite



Internet Names & Addresses

Internet Names

- Each host has a unique name
 - Independent of physical location
 - Facilitate memorization by humans
 - Domain Name
 - Organization under single administrative unit
- Host Name
 - Name given to host computer
- User Name
 - Name assigned to user

ansari@anlserver1.njit.edu

Internet Addresses

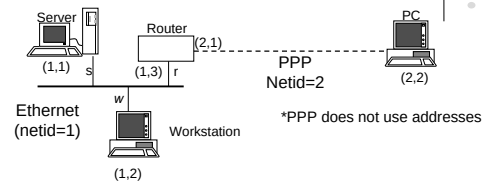
- Each host has globally unique *logical* 32 bit IP address
- Separate address for each physical connection to a network
- Routing decision is done based on destination IP address
- IP address has two parts:
 - *netid* and *hostid*
 - *netid* unique
 - *netid* facilitates routing
- Dotted Decimal Notation:
 - int1.int2.int3.int4 (intj = jth octet)
 - 128.235.10.13

DNS resolves IP name to IP address

Physical Addresses

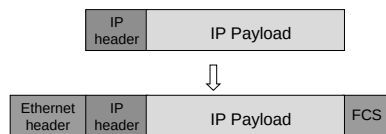
- LANs (and other networks) assign physical addresses to the physical attachment to the network
- The network uses its own address to transfer packets or frames to the appropriate destination
- IP address needs to be resolved to physical address at each IP network interface
- Example: Ethernet uses 48-bit addresses
 - Each Ethernet network interface card (NIC) has globally unique Medium Access Control (MAC) or physical address
 - First 24 bits identify NIC manufacturer; second 24 bits are serial number
 - 00:90:27:96:68:07 12 hex numbers
 - Intel

Example internet



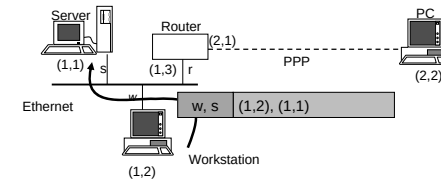
	netid	hostid	Physical address
server	1	1	s
workstation	1	2	w
router	1	3	r
router	2	1	-
PC	2	2	-

Encapsulation



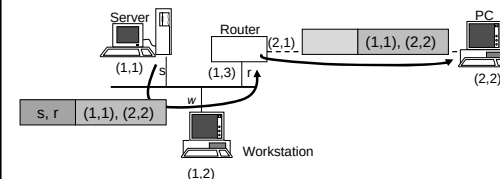
- Ethernet header contains:
 - source and destination physical addresses
 - network protocol type (e.g. IP)

IP packet from workstation to server



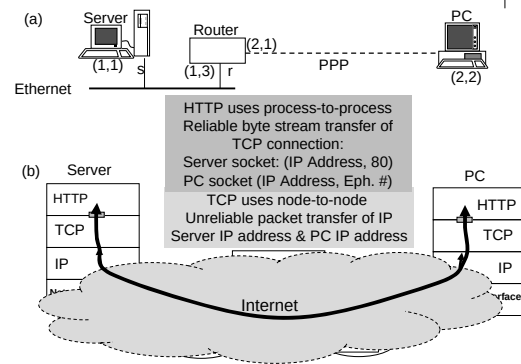
- IP packet has (1,2) IP address for source and (1,1) IP address for destination
- IP table at workstation indicates (1,1) connected to same network, so IP packet is encapsulated in Ethernet frame with addresses w and s
- Ethernet frame is broadcast by workstation NIC and captured by server NIC
- NIC examines protocol type field and then delivers packet to its IP layer

IP packet from server to PC

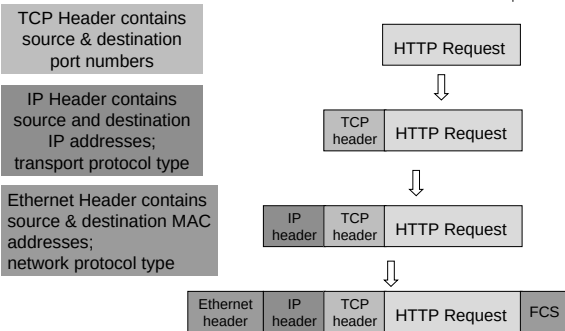


- IP packet has (1,1) and (2,2) as IP source and destination addresses
- IP table at server indicates packet should be sent to router, so IP packet is encapsulated in Ethernet frame with addresses s and r
- Ethernet frame is broadcast by server NIC and captured by router NIC
- NIC examines protocol type field and then delivers packet to its IP layer
- IP layer examines IP packet destination address and determines IP packet should be routed to (2,2)
- Router's table indicates (2,2) is directly connected via PPP link
- IP packet is encapsulated in PPP frame and delivered to PC
- PPP at PC examines protocol type field and delivers packet to PC IP layer

How the layers work together



Encapsulation

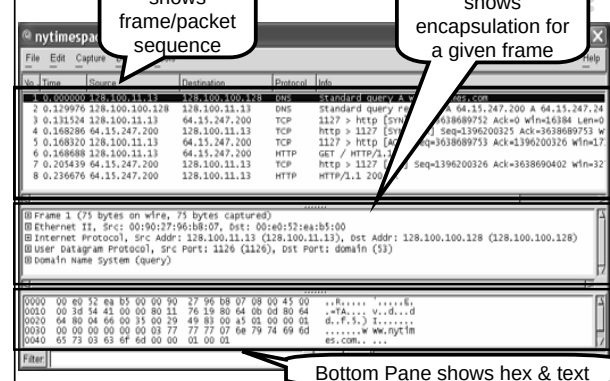


How the layers work together: Network Analyzer Example



- User clicks on <http://www.nytimes.com/>
- *Ethereal* network analyzer captures all frames observed by its Ethernet NIC
- Sequence of frames and contents of frame can be examined in detail down to individual bytes

Ethereal Flows



Top pane: frame sequence

The top pane of Wireshark displays a list of captured packets. Three callouts highlight specific packets:

- DNS Query:** Points to packet 4, an Ethernet II frame containing a DNS query.
- TCP Connection Setup:** Points to packet 5, a TCP segment with the SYN flag set.
- HTTP Request & Response:** Points to packets 6 and 7, which contain an HTTP GET request and its corresponding response.

Middle pane: encapsulation

The middle pane shows the detailed structure of a selected packet (Frame 6). Callouts identify the following components:

- IP Packet:** Points to the Ethernet II frame header.
- IP Source and Destination Addresses:** Points to the IP header fields for source and destination addresses.
- Protocol Type:** Points to the IP header field for protocol type (0x06 for TCP).

Middle pane: Encapsulation

The middle pane shows the detailed structure of a selected packet (Frame 6). Callouts identify the following components:

- Ethernet Frame:** Points to the Ethernet II frame header.
- Ethernet Destination and Source Addresses:** Points to the Ethernet header fields for destination and source MAC addresses.
- Protocol Type:** Points to the Ethernet header field for protocol type (0x06 for TCP).

Middle pane: Encapsulation

The middle pane shows the detailed structure of a selected packet (Frame 6). Callouts identify the following components:

- TCP Segment:** Points to the Transmission Control Protocol (TCP) header.
- Source and Destination Port Numbers:** Points to the TCP header fields for source and destination ports.
- GET:** Points to the application data field, which contains an HTTP GET request.
- HTTP Request:** Points to the application data field, which contains an HTTP GET request.

Summary

- Encapsulation is key to layering
- IP provides for transfer of packets across diverse networks
- TCP and UDP provide universal communications services across the Internet
- Distributed applications that use TCP and UDP can operate over the entire Internet
- Internet names, IP addresses, port numbers, sockets, connections, physical addresses

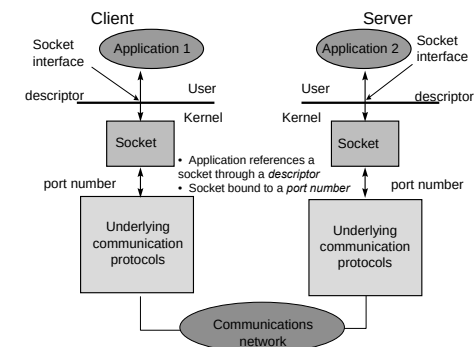
Lecture 2 Applications and Layered Architectures

Sockets

Socket API

- API (Application Programming Interface)
 - Provides a standard set of functions that can be called by applications
- Berkeley UNIX Sockets API
 - Abstraction for applications to send & receive data
 - Applications create sockets that “plug into” network
 - Applications write/read to/from sockets
 - Implemented in the kernel
 - Facilitates development of network applications
 - Hides details of underlying protocols & mechanisms
- Also in Windows, Linux, and other OS's

Communications through Socket Interface



Stream mode of service

Connection-oriented

- First, setup connection between two peer application processes
- Then, reliable bidirectional in-sequence transfer of *byte stream* (boundaries not preserved in transfer)
- Multiple write/read between peer processes
- Finally, connection release
- Uses TCP

Connectionless

- Immediate transfer of one block of information (boundaries preserved)
- No setup overhead & delay
- Destination address with each block
- Send/receive to/from multiple peer processes
- Best-effort service only
 - Possible out-of-order
 - Possible loss
- Uses UDP

Client & Server Differences

• Server

- Specifies well-known port # when creating socket
- May have multiple IP addresses (net interfaces)
- Waits passively for client requests

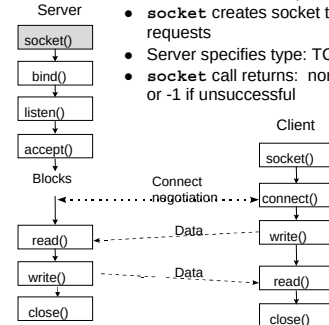
• Client

- Assigned ephemeral port #
- Initiates communications with server
- Needs to know server's IP address & port #
 - DNS for URL & server well-known port #
- Server learns client's address & port #

Socket Calls for Connection-Oriented Mode

Server does Passive Open

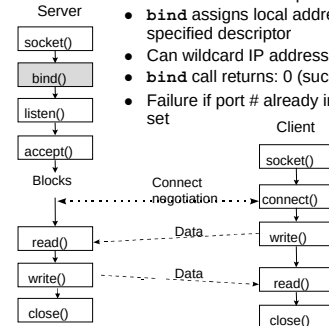
- **socket** creates socket to *listen* for connection requests
- Server specifies type: TCP (stream)
- **socket** call returns: non-negative integer *descriptor*, or -1 if unsuccessful



Socket Calls for Connection-Oriented Mode

Server does Passive Open

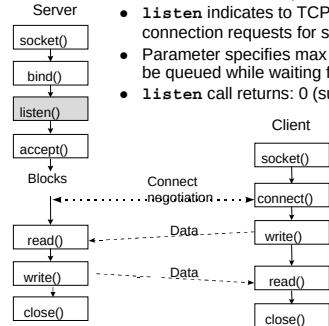
- **bind** assigns local address & port # to socket with specified descriptor
- Can wildcard IP address for multiple net interfaces
- **bind** call returns: 0 (success); or -1 (failure)
- Failure if port # already in use or if reuse option not set



Socket Calls for Connection-Oriented Mode

Server does Passive Open

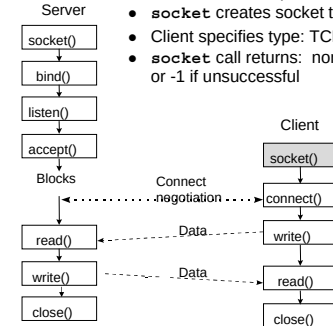
- **listen** indicates to TCP readiness to receive connection requests for socket with given descriptor
- Parameter specifies max number of requests that may be queued while waiting for server to accept them
- **listen** call returns: 0 (success); or -1 (failure)



Socket Calls for Connection-Oriented Mode

Client does Active Open

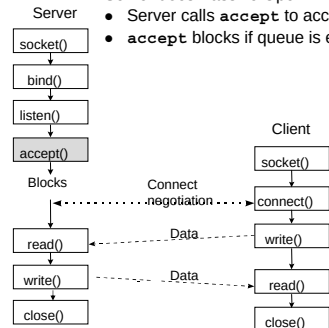
- **socket** creates socket to connect to server
- Client specifies type: TCP (stream)
- **socket** call returns: non-negative integer *descriptor*, or -1 if unsuccessful



Socket Calls for Connection-Oriented Mode

Server does Passive Open

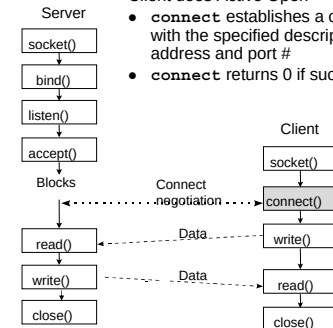
- Server calls **accept** to accept incoming requests
- **accept** blocks if queue is empty



Socket Calls for Connection-Oriented Mode

Client does Active Open

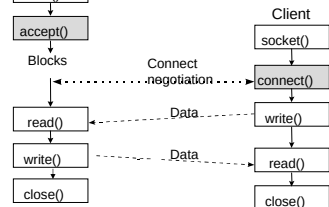
- **connect** establishes a connection on the local socket with the specified descriptor to the specified remote address and port #
- **connect** returns 0 if successful; -1 if unsuccessful



Note: **connect** initiates TCP three-way handshake

Socket Calls for Connection-Oriented Mode

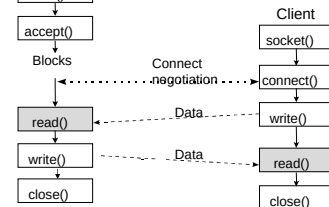
- **accept** wakes with incoming connection request
- **accept** fills client address & port # into address structure
- **accept** call returns: *descriptor of new connection socket* (success); or -1 (failure)
- Client & server use new socket for data transfer
- Original socket continues to listen for new requests



Socket Calls for Connection-Oriented Mode

Data Transfer

- Client or server call **read** to receive data from a connected socket
- **read** specifies: socket descriptor; pointer to a buffer; amount of data
- **read** call returns: # bytes read (success); or -1 (failure); blocks if no data arrives

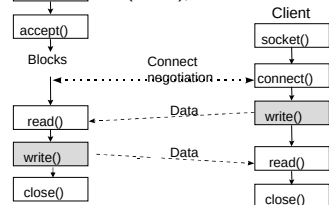


Note: **write** and **read** can be called multiple times to transfer byte streams in both directions

Socket Calls for Connection-Oriented Mode

Data Transfer

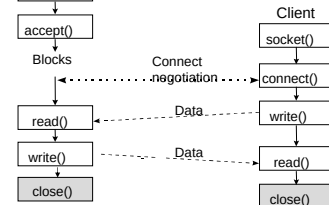
- Client or server call **write** to transmit data into a connected socket
- **write** specifies: socket descriptor; pointer to a buffer; amount of data; flags to control transmission behavior
- **write** call returns: # bytes transferred (success); or -1 (failure); blocks until all data transferred



Socket Calls for Connection-Oriented Mode

Connection Termination

- Client or server call **close** when socket is no longer needed
- **close** specifies the socket descriptor
- **close** call returns: 0 (success); or -1 (failure)



Note: **close** initiates TCP graceful close sequence

Example: TCP Echo Server

```

/* A simple echo server using TCP */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define SERVER_TCP_PORT 3000
#define BUFSIZE 256

int main(int argc, char **argv)
{
    int n, bytes_to_read;
    int fd, new_fd, client_len, port;
    struct sockaddr_in server, client;
    char *bp, buf[BUFSIZE];

    switch(argc) {
        case 1:
            port = SERVER_TCP_PORT;
            break;
        case 2:
            port = atoi(argv[1]);
            break;
        default:
            printf(stderr, "Usage: %s [port]\n", argv[0]);
            exit(1);
    }

    /* Create a stream socket */
    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        printf(stderr, "Can't create a socket\n");
        exit(1);
    }

    /* Bind an address to the socket */
    bzero((char *) &server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if (bind(fd, (struct sockaddr *) &server,
            sizeof(server)) == -1) {
        printf(stderr, "Can't bind name to socket\n");
        exit(1);
    }

    /* Queue up to 5 connect requests */
    listen(fd, 5);

    while (1) {
        client_len = sizeof(client);
        if ((new_fd = accept(fd, (struct sockaddr *) &client,
                           &client_len)) == -1) {
            printf(stderr, "Can't accept client\n");
            exit(1);
        }

        bp = buf;
        bytes_to_read = BUFSIZE;
        while ((n = read(new_fd, bp, bytes_to_read)) > 0) {
            bp += n;
            bytes_to_read -= n;

            printf("Rec'd: %s\n", buf);
            write(new_fd, buf, BUFSIZE);
            printf("Sent: %s\n", buf);
            close(new_fd);
        }

        close(fd);
        return(0);
    }
}

```

Example: TCP Echo Client

```

/* A simple TCP client */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define SERVER_TCP_PORT 3000
#define BUFSIZE 256

int main(int argc, char **argv)
{
    int n, bytes_to_read;
    int fd, port;
    struct hostent *hp;
    struct sockaddr_in server;
    char *host, *bp, buf[BUFSIZE], sbuf[BUFSIZE];

    switch(argc) {
        case 2:
            host = argv[1];
            port = SERVER_TCP_PORT;
            break;
        case 3:
            host = argv[1];
            port = atoi(argv[2]);
            break;
        default:
            printf(stderr, "Usage: %s host [port]\n", argv[0]);
            exit(1);
    }

    /* Create a stream socket */
    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        printf(stderr, "Can't create a socket\n");
        exit(1);
    }

    bzero((char *) &server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if ((hp = gethostbyname(host)) == NULL) {
        printf(stderr, "Can't get server's address\n");
        exit(1);
    }

    strcpy(host, hp->h_addr);
    /* Connecting to the server */
    if (connect(fd, (struct sockaddr *) &server,
               sizeof(server)) == -1) {
        printf(stderr, "Can't connect\n");
        exit(1);
    }

    printf("Connected: server's address is %s\n", hp->h_name);
    printf("Transmit:\n");
    get(sbuf);
    write(fd, sbuf, BUFSIZE);

    printf("Receive:\n");
    bp = sbuf;
    bytes_to_read = BUFSIZE;
    while ((n = read(fd, bp, bytes_to_read)) > 0) {
        bp += n;
        bytes_to_read -= n;
        printf("%s\n", sbuf);
    }

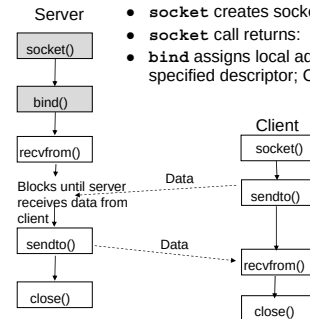
    close(fd);
    return(0);
}

```

Socket Calls for Connection-Less Mode

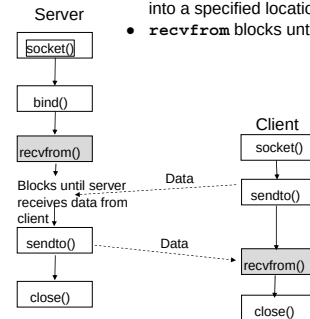
Server started

- **socket** creates socket of type UDP (datagram)
- **socket** call returns: *descriptor*, or -1 if unsuccessful
- **bind** assigns local address & port # to socket with specified descriptor; Can wildcard IP address

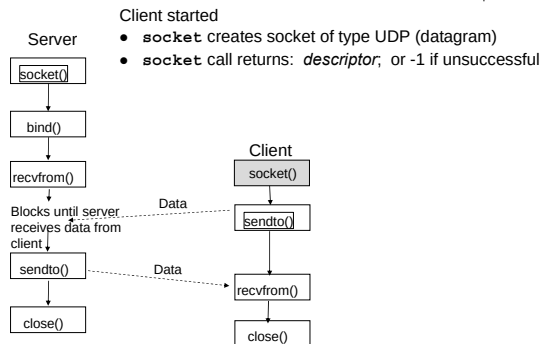


Socket Calls for Connection-Less Mode

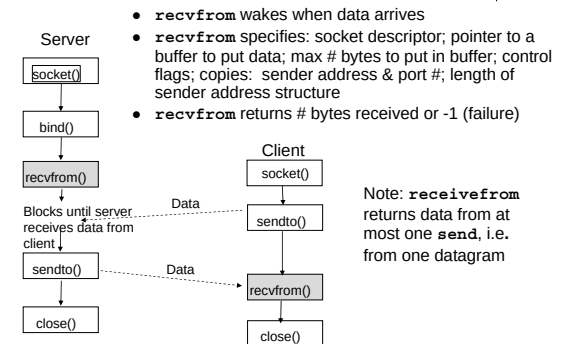
- **recvfrom** copies bytes received in specified socket into a specified location
- **recvfrom** blocks until data arrives



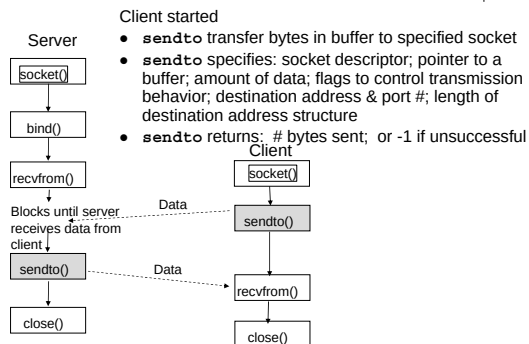
Socket Calls for Connection-Less Mode



Socket Calls for Connection-Less Mode

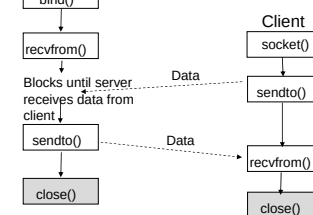


Socket Calls for Connection-Less Mode



Socket Calls for Connection-Less Mode

- Socket Close**
- Client or server call **close** when socket is no longer needed
 - **close** specifies the socket descriptor
 - **close** call returns: 0 (success); or -1 (failure)



bucket:\n*) ;

$$m^*, pname) ;$$

-



Network Virtual Terminal

- *Network Virtual Terminal*
- Lowest common denominator terminal
- Each machine maps characteristics to NVT
- Negotiate options for changes to the NVT
- Data input sent to server & echoed back
- Server control functions : interrupt, abort output, are-you-there, erase character, erase line
- Default requires login & password

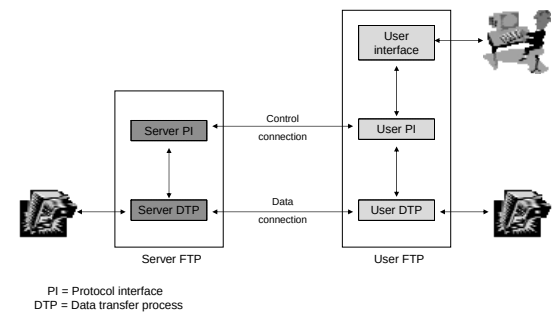
telnet

- A program that uses the Telnet protocol
- Establishes TCP socket
- Sends typed characters to server
- Prints whatever characters arrive
- Try it to retrieve a web page (HTTP) or to send an email (SMTP)

File Transfer Protocol (RFC 959)

- Provides for transfer of file from one machine to another machine
- Designed to hide variations in file storage
- FTP parameter commands specify file info
 - File Type: *ASCII*, EBCDIC, image, local.
 - Data Structure: *file*, record, or page
 - Transmission Mode: *stream*, block, compressed
- Other FTP commands
 - Access Control: USER, PASS, CWD, QUIT, ...
 - Service: RETR, STOR, PWD, LIST, ...

FTP File Transfer



Two TCP Connections

Control connection

- Set up using Telnet protocol on well-known port 21
- FTP commands & replies between protocol interpreters
- PIs control the data transfer process
- User requests close of control connection; server performs the close

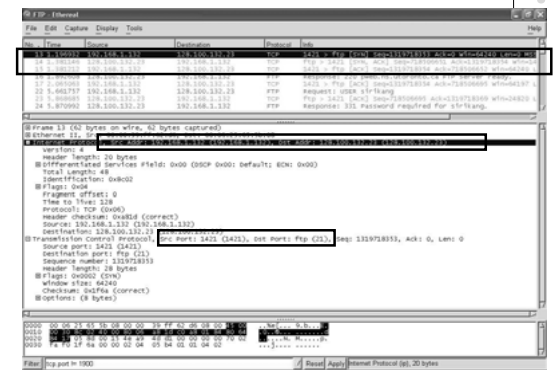
Data connection

- To perform file transfer, obtain lists of files, directories
- Each transfer requires new data connection
- Passive open by user PI with ephemeral port #
- Port # sent over control connection
- Active open by server using port 20

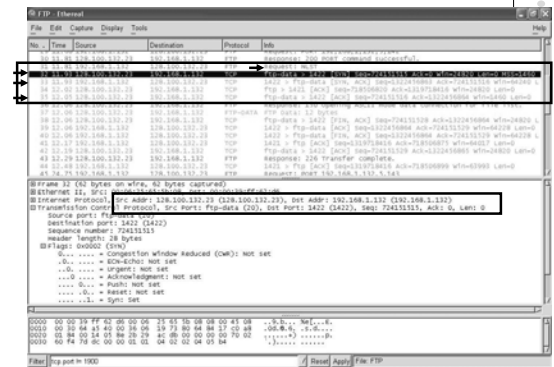
FTP Replies

Reply	Meaning
1yz	Positive preliminary reply (action has begun, but wait for another reply before sending a new command).
2yz	Positive completion reply (action completed successfully; new command may be sent).
3yz	Positive intermediary reply (command accepted, but action cannot be performed without additional information; user should send a command with the necessary information).
4yz	Transient negative completion reply (action currently cannot be performed; resend command later).
5yz	Permanent negative completion reply (action cannot be performed; do not resend it).
x0z	Syntax errors.
x1z	Information (replies to requests for status or help).
x2z	Connections (replies referring to the control and data connections).
x3z	Authentication and accounting (replies for the login process and accounting procedures).
x4z	Unspecified.
x5z	File system status.

FTP Client (192.168.1.132: 1421) establishes Control Connection to FTP Server (128.100.132.23: 21)



User types /s to list files in directory (frame 31 on control) FTP Server (128.100.132.23: 20) establishes Data Connection to FTP Client (192.168.1.132: 1422)



User types *get index.html* to request file transfer in control connection (frame 47 request); File transfer on new data connection (port 1423, fr. 48, 49, 51)

Wireshark capture of an FTP session. The packet list shows a series of FTP control and data packets. Packet 47 is the request for index.html. Packets 48, 49, and 51 show the data transfer on a new connection.

Hypertext Transfer Protocol

- RFC 1945 (HTTP 1.0), RFC 2616 (HTTP 1.1)
- HTTP provides communications between web browsers & web servers
- Web: framework for accessing documents & resources through the Internet
- Hypertext documents: text, graphics, images, hyperlinks
- Documents prepared using Hypertext Markup Language (HTML)

HTTP Protocol

- HTTP servers use well-known port 80
- Client request / Server reply
- Stateless: server does not keep any information about client
- HTTP 1.0 new TCP connection per request/reply (non-persistent)
- HTTP 1.1 persistent operation is default

HTTP Typical Exchange

Wireshark capture of a typical HTTP exchange. It shows a GET request for a webpage and the corresponding response with status 200 OK.

HTTP Message Formats

- HTTP messages written in ASCII text
- Request Message Format
 - *Request Line (Each line ends with carriage return)*
 - *Method URL HTTP-Version \r\n*
 - Method specifies action to apply to object
 - URL specifies object
 - *Header Lines (Ea. line ends with carriage return)*
 - *Attribute Name: Attribute Value*
 - E.g. type of client, content, identity of requester, ...
 - Last header line has extra carriage return)
 - *Entity Body (Content)*
 - Additional information to server

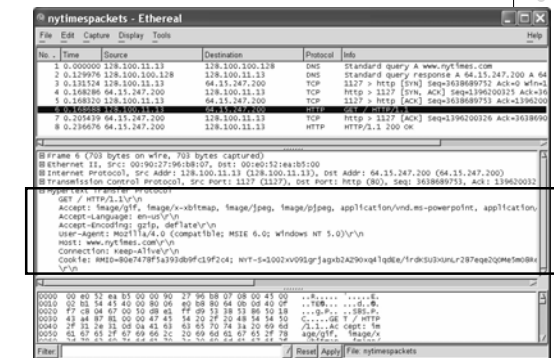
HTTP Request Methods

Request method	Meaning
GET	Retrieve information (object) identified by the URL.
HEAD	Retrieve meta-information about the object, but do not transfer the object; Can be used to find out if a document has changed.
POST	Send information to a URL (using the entity body) and retrieve result; used when a user fills out a form in a browser.
PUT	Store information in location named by URL.
DELETE	Remove object identified by URL.
TRACE	Trace HTTP forwarding through proxies, tunnels, etc.
OPTIONS	Used to determine the capabilities of the server, or characteristics of a named resource.

Universal Resource Locator

- Absolute URL
 - `scheme://hostname[:port]/path`
 - `http://www.nytimes.com/`
- Relative URL
 - `/path`
 - `/`

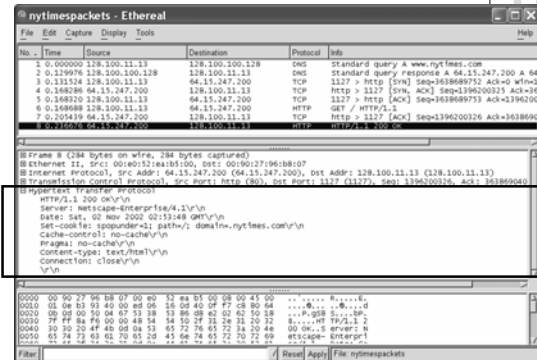
HTTP Request Message



HTTP Response Message

- Response Message Format
 - *Status Line*
 - *HTTP-Version Status-Code Message*
 - Status Code: 3-digit code indicating result
 - E.g. HTTP/1.0 200 OK
 - *Headers Section*
 - Information about object transferred to client
 - E.g. server type, content length, content type, ...
 - *Content*
 - Object (document)

HTTP Response Message



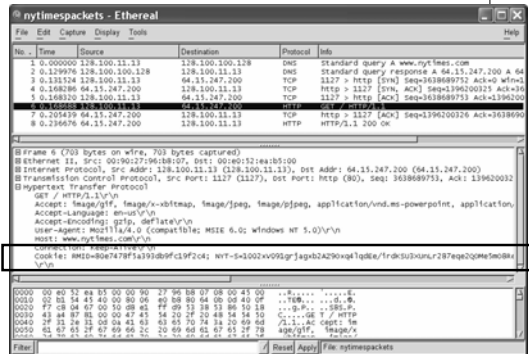
HTTP Proxy Server & Caching

- Web users generate large traffic volumes
- Traffic causes congestion & delay
- Can improve delay performance and reduce traffic in Internet by moving content to servers closer to the user
- Web proxy servers cache web information
 - Deployed by ISPs
 - Customer browsers configured to first access ISPs proxy servers
 - Proxy replies immediately when it has requested object or retrieves the object if it does not

Cookies and Web Sessions

- Cookies are data exchanged by clients & servers as header lines
- Since HTTP stateless, cookies can provide context for HTTP interaction
- *Set cookie* header line in reply message from server + unique ID number for client
- If client accepts cookie, cookie added to client's cookie file (must include expiration date)
- Henceforth client requests include ID
- Server site can track client interactions, store these in a separate database, and access database to prepare appropriate responses

Cookie Header Line; ID is 24 hexadecimal numeral



PING

- Application to determine if host is reachable
- Based on Internet Control Message Protocol
 - ICMP informs source host about errors encountered in IP packet processing by routers or by destination host
 - ICMP Echo message requests reply from destination host
- PING sends echo message & sequence #
- Determines reachability & round-trip delay
- Sometimes disabled for security reasons

PING from NAL host

```
Microsoft(R) Windows DOS
(c)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ping nal.toronto.edu

Pinging nal.toronto.edu [128.100.244.3] with 32 bytes of data:

Reply from 128.100.244.3: bytes=32 time=84ms TTL=240
Reply from 128.100.244.3: bytes=32 time=110ms TTL=240
Reply from 128.100.244.3: bytes=32 time=81ms TTL=240
Reply from 128.100.244.3: bytes=32 time=79ms TTL=240

Ping statistics for 128.100.244.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 79ms, Maximum = 110ms, Average = 88ms

C:\DOCUME~1\1>
```

Traceroute

- Find route from local host to a remote host
- Time-to-Live (TTL)
 - IP packets have TTL field that specifies maximum # hops traversed before packet discarded
 - Each router decrements TTL by 1
 - When TTL reaches 0 packet is discarded
- Traceroute
 - Send UDP to remote host with TTL=1
 - First router will reply ICMP Time Exceeded Msg
 - Send UDP to remote host with TTL=2, ...
 - Each step reveals next router in path to remote host

Traceroute from home PC to university host

Tracing route to www.comm.utoronto.ca [128.100.11.60]
over a maximum of 30 hops:

1	1 ms	<10 ms	<10 ms	192.168.2.1	Home Network
2	3 ms	3 ms	3 ms	10.202.128.1	
3	4 ms	3 ms	3 ms	gw04.ym.phub.net.cable.rogers.com [66.185.83.142]	
4	*	*	*	Request timed out.	
5	47 ms	59 ms	66 ms	gw01.bloor.phub.net.cable.rogers.com [66.185.80.230]	
6	3 ms	3 ms	38 ms	gw02.bloor.phub.net.cable.rogers.com [66.185.80.242]	
7	8 ms	3 ms	5 ms	gw01.wildie.phub.net.cable.rogers.com [66.185.80.2]	Rogers Cable
8	8 ms	7 ms	7 ms	gw02.wildie.phub.net.cable.rogers.com [66.185.80.142]	ISP
9	4 ms	10 ms	4 ms	gw01.front.phub.net.cable.rogers.com [66.185.81.18]	
10	6 ms	4 ms	5 ms	ralesb-ge3-4.mt.bigpipeinc.com [66.244.223.237]	Shaw Net
11	16 ms	17 ms	13 ms	rx0sh-hydro-one-telecom.mt.bigpipeinc.com [66.244.223.246]	Hydro One
12	7 ms	14 ms	8 ms	142.46.4.2	
13	10 ms	7 ms	6 ms	utorgw.onet.on.ca [206.249.221.6]	Ontario Net
14	7 ms	6 ms	11 ms	mcl-gateway.gw.utoronto.ca [128.100.96.101]	
15	7 ms	5 ms	8 ms	sf-ggb.gw.utoronto.ca [128.100.96.17]	University of Toronto
16	7 ms	7 ms	10 ms	bl15000.ece.utoronto.ca [128.100.96.236]	
17	7 ms	9 ms	9 ms	www.comm.utoronto.ca [128.100.11.60]	

Trace complete.

netstat

- Queries a host about TCP/IP network status
- Status of network drivers & their interface cards
 - #packets in, #packets out, errored packets, ...
- State of routing table in host
- TCP/IP active server processes
- TCP active connections

ipconfig

- Utility in Microsoft® Windows to display TCP/IP information about a host
- Many options
 - Simplest: IP address, subnet mask, default gateway for the host
 - Information about each IP interface of a host
 - DNS hostname, IP addresses of DNS servers, physical address of network card, IP address, ...
 - Renew IP address from DHCP server

netstat protocol statistics

IPv4 Statistics		ICMPv4 Statistics	
Packets Received	= 71271	Received	Sent
Received Header Errors	= 0	Messages	10 6
Received Address Errors	= 9	Errors	0 0
Datagrams Forwarded	= 0	Destination Unreachable	8 1
Unknown Protocols Received	= 0	Time Exceeded	0 0
Received Packets Discarded	= 0	Parameter Problems	0 0
Received Packets Delivered	= 71271	Source Quenches	0 0
Output Requests	= 70138	Redirects	0 0
Routing Discards	= 0	Echos	0 2
Discarded Output Packets	= 0	Echo Replies	2 0
Output Packet No Route	= 0	Timestamps	0 0
Reassembly Required	= 0	Timestamp Replies	0 0
Reassembly Successful	= 0	Address Masks	0 0
Reassembly Failures	= 0	Address Mask Replies	0 0
Datagrams Successfully Fragmented	= 0	TCP Statistics for IPv4	
Datagrams Failing Fragmentation	= 0	Active Opens	= 798
Fragments Created	= 0	Passive Opens	= 17
UDP Statistics for IPv4		Failed Connection Attempts	= 13
Datagrams Received	= 6810	Reset Connections	= 467
No Ports	= 15	Current Connections	= 0
Receive Errors	= 0	Segments Received	= 64443
Datagrams Sent	= 6309	Segments Sent	= 63724
		Segments Retransmitted	= 80

tcpdump and Network Protocol Analyzers



- `tcpdump` program captures IP packets on a network interface (usually Ethernet NIC)
- Filtering used to select packets of interest
- Packets & higher-layer messages can be displayed and analyzed
- `tcpdump` basis for many network protocol analyzers for troubleshooting networks
- We use the open source Ethereal analyzer to generate examples
 - www.ethereal.com