Application Layer

Application Layer

Application layer

"The user runs an application program. The program selects the data transport service it needs — either a sequence of data messages or a continuous stream of data — and hands application data to the Transport Layer for delivery."

RFC 4949 — Internet Security Glossary

Application layer issues

What data transport service does application need?

Quality of service (QoS)

Measurable network parameters

QoS requirements determine transport requirements

How does application hand data to transport layer for delivery?

Socket interface

OS call opens Internet connection to program on remote node

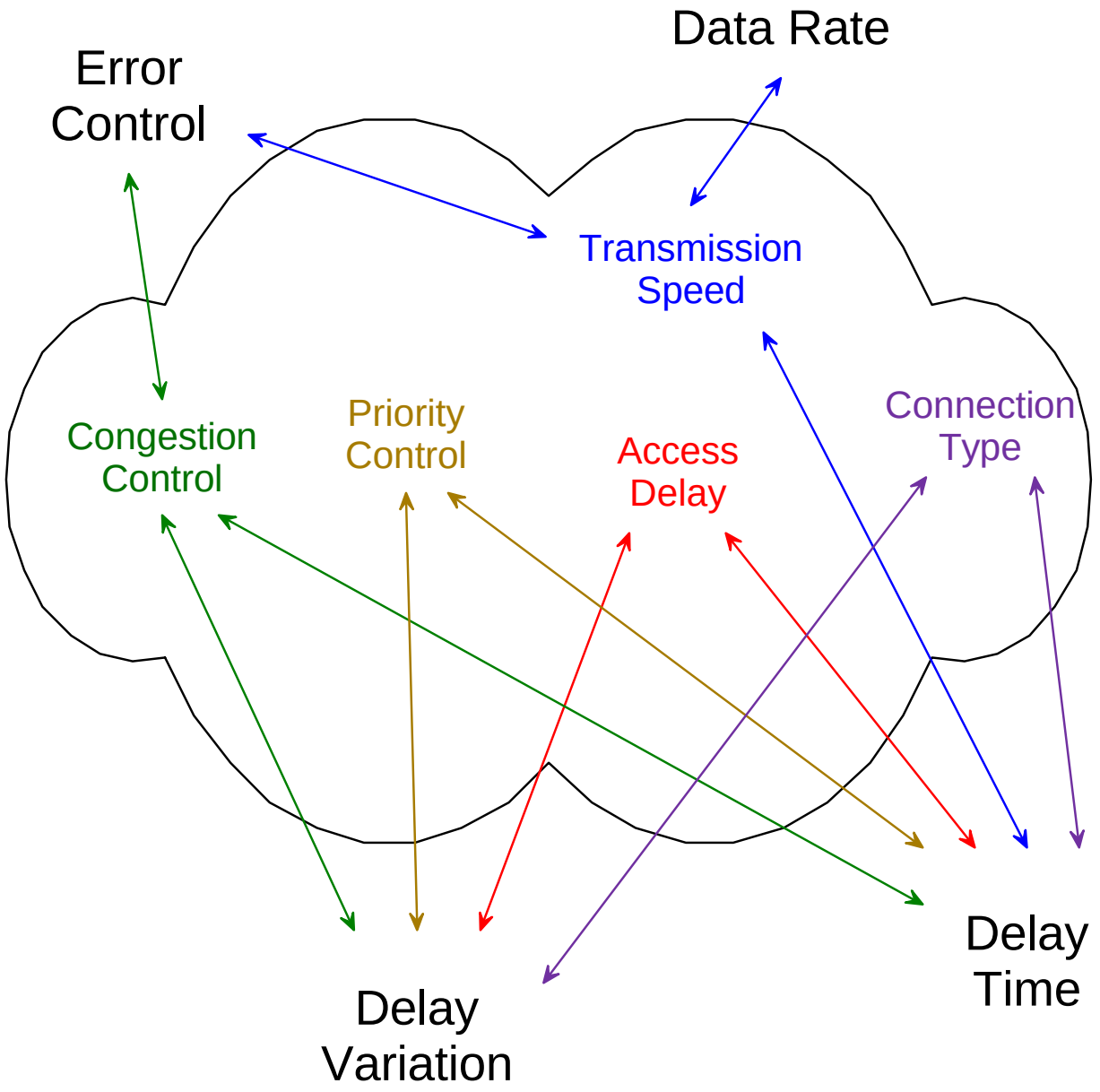
Socket API — application sees remote program as open file

Quality of Service (QoS)

Network parameters

Data rate	Physical transmission speed in bps
Error control	Bit error rate (BER) $< 10^{-9}$ on fiber optic cable $< 10^{-3}$ on wireless Packet loss rate Depends on congestion control policy
Delay	End-to-end transmission time for one bit High data rate $\Rightarrow$ lower transmission delay T_{trans} Congestion / priority $\Rightarrow$ longer queuing delay T_Q Datagram service $\Rightarrow$ longer processing delay T_{proc}
Delay variation (jitter)	Dynamic variations in Data rate / propagation delay ΔT_{trans} Queuing delay ΔT_Q Processing delay ΔT_{proc}

QoS and Network Policies



QoS Requirements for Various Services

Service	Speed	Error Control	Delay	Delay Variation
e-mail	—	good	—	—
file transfer	reasonable	maximum	reasonable	reasonable
NFS				
database access				
voice	fast	good	very small	very small
video	very fast	good	very small	very small
real time control				

QoS and Transport Trade-Off

	Reliable Transport	Unreliable Transport
Protocol	TCP	UDP
Error control	Error-free	No error correction
Connection	Connection-oriented	Connectionless
Socket type	Stream	Datagram
Processing overhead	Error correction, packet ordering, congestion control, session state management	None

Reliable Transport

Generally preferred when possible

Unreliable Transport

Used when

- Some data loss tolerable
- Delay or jitter intolerable
- Example — video delivery

Application / Transport Interface

Application requests service from transport layer

Application Layer	Prepare Transport service requirements Data for transport Local endpoint — node address + service port Remote endpoint — node address + service port
Transport Layer	Receives Application layer PDU — data for transport Interface Control Information (ICI) Service requirements Local + remote endpoints

Berkeley Sockets

Networking API

De facto standard on Unix / Linux / Apple systems

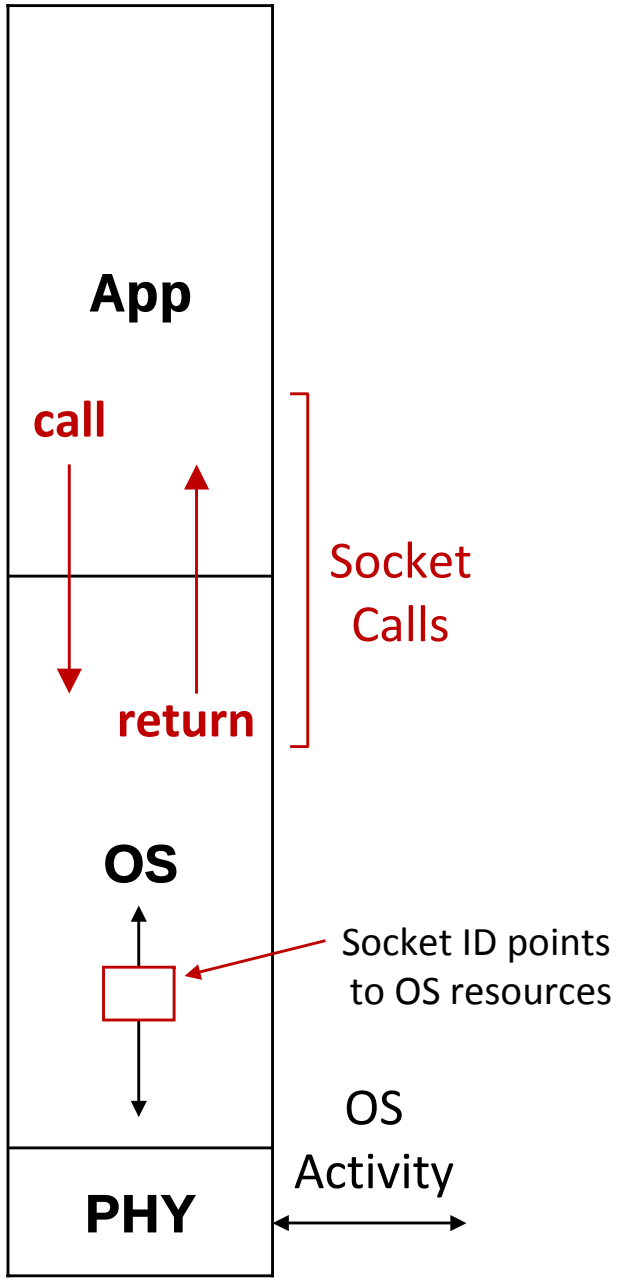
Application performs communication like file I/O

Address	Internet address of local / remote node
Port	ID number of client / server software
Source	Local node
Destination	Remote (foreign) node
Endpoint	Address + port of software process (local / remote)
Association	Description of connection between client / server processes Protocol + source endpoint + destination endpoint
Socket	OS resources associated with protocol + endpoint Server side — socket ID points to service offered to remote nodes Client side — socket ID points to connection to remote service

Socket System Calls

Client and Server	
Socket	Create new socket associated with protocol at local address
Bind	Associate port with socket
Send	Local process sends data to socket (OS forwards to remote node)
Receive	Local process reads data from socket (OS buffer from remote node)
Close	Terminate socket
Client Side	
Connect	Open connection to remote service and associate with socket
Server Side	
Listen	Prepare to accept remote connections to local service socket
Accept	Accept remote connection to local service socket

Socket Calls — Between App and OS



Socket, Bind

OS creates resources for network connection

OS returns to App socket descriptor (socket ID)

Listen

Server OS makes service available

Connect

Client OS attempts connection to service

Accept

Server OS creates new connection socket

Server OS returns new descriptor to App

Listen socket continues to listen

Send

App sends data to OS, pointed at socket

OS transmits data on associated connection

Receive

App requests data from OS socket buffer

OS returns data to App

Connection

Client **Server**

```
s_ID = socket(domain, type, protocol)
```
address type, service type

```
bind(s_ID, s_endpoint, s_endpoint_len)
listen(s_ID, backlog)
```

connection set-up →

```
c_ID = socket(domain, type, protocol)
bind(c_ID, c_endpoint, c_endpoint_len)
connect(c_ID, service_endpoint)
```

If client skips **bind()**
OS sets a default port

```
s = accept(s_ID, c_endpoint, endpoint_len)
```

← data →

```
send(s, data, len, flags)
receive(c_ID, buffer, len, flags)
```

Node Addresses

IPv4 — Internet Protocol version 4

- 32-bit address
- 4 octets (bytes) written in decimal — A.B.C.D
- $0 \leq A, B, C, D \leq 255$
- Disconnected intranet addresses 10.x.x.x and 192.168.0.x
  - Usable in any local intranet
  - Not used in global Internet (between intranets)
  - Network Address Translation (NAT) at gateway router
    - Permits sharing one external IP address for entire Intranet
- 127.0.0.1
  - Loopback address used for testing
  - Address alias for "this node"

IPv6 — Internet Protocol version 6

- 128-bit address
- Repairs service flaws in IPv4

Ports

Client (source) port

Client can assign any unused port number $1024 \leq \text{port} \leq 65{,}535$

Well-known port (destination port) — standard services

Assigned by Internet Assigned Numbers Authority (IANA)

Examples

Service	Port	Comment
ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
ssh	22	The Secure Shell (SSH) Protocol
telnet	23	Telnet
smtp	25	Simple Mail Transfer
dns	53	Domain Name Service
http	80	World Wide Web HTTP
pop3	110	Post Office Protocol - Version 3
imap	143	Internet Message Access Protocol
https	443	http protocol over TLS/SSL

Telnet

Telnet Protocol — RFC 854

- Text-oriented reliable communication over TCP
- Telnet client forwards user text to server
- Telnet server provides remote login
  - Command line user interface (CLI) on remote node

Telnet program

- Implements telnet client protocol
- Connect to telnet server (port 23) on server node

```
$ telnet node
```

- Connect to port **port** on server node

```
$ telnet node port
```

- Example
  - Connect to Google web server on HTTP (port 80)

```
$ telnet www.google.com 80
```

HTTP — Hypertext Transfer Protocol

Stateless transaction protocol

- Client request formulated in ASCII (printable) text
- Server response formulated in MIME structures
- No stored data on prior transactions

MIME — Multipurpose Internet Mail Extensions

- Text in non-ASCII character sets
- Non-text attachments
- Multipart message bodies

HTTP/1.0

- Non-persistent connection
  - One transaction per TCP connection

HTTP/1.1

- Persistent connection
  - Multiple transactions on same TCP socket
- Pipelined transactions
  - Multiple requests without waiting for response

HTTP Requests

Simple requests

Defined in HTTP1.1 for HTTP/1.0 compatibility

Request: single line — GET, POST, HEAD

Response: raw page without MIME headers

Full requests

Multi-line request format

Line	Incidence
Request-Line	One
Message-header CRLF	Zero or more
CRLF	Blank line
Message-body	Optional

Method	Function
GET	get page
HEAD	get page header
PUT	store page
POST	append to resource
DELETE	delete page
TRACE	echo back message
CONNECT	tunneling via proxy

Request-Line format

Method	Sp	Request-URI	Sp	HTTP-Version	CRLF

Full Request Example

```
GET /index.html HTTP/1.1\r\n
```

Request line

```
Host: www-net.cs.umass.edu\r\n

User-Agent: Firefox/3.6.10\r\n

Accept: text/html,application/xhtml+xml\r\n

Accept-Language: en-us,en;q=0.5\r\n

Accept-Encoding: gzip,deflate\r\n

Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n

Keep-Alive: 115\r\n

Connection: keep-alive\r\n
```

Header lines

```
\r\n
```

CRLF

HTTP Responses

General structure

Line	Incidence
Status Line	One
message-header CRLF	Zero or more
Data	Zero or more

Status line structure

HTTP version	Sp	status code	Sp	description	CRLF

Example

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Status line

Headers

Data

HTTP Response Codes

100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden

404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
416	Requested range not satisfiable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

Telnet into Web Server

```
$ telnet www.hadassah.ac.il 80
Trying 212.179.79.228...
Connected to hathi.hadassah.ac.il.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 301 Moved Permanently
Content-Length: 158
Content-Type: text/html
Location: http://www.hadassah.ac.il/main.html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Tue, 05 Jul 2011 09:01:35 GMT
Connection: close

<head><title>Document Moved</title></head>
<body><h1>Object Moved</h1>This document may be found <a
    HREF="http://www.hadassah.ac.il/main.html">here</a></bo
    dy>Connection closed by foreign host.
```

open connection to Hadassah web server

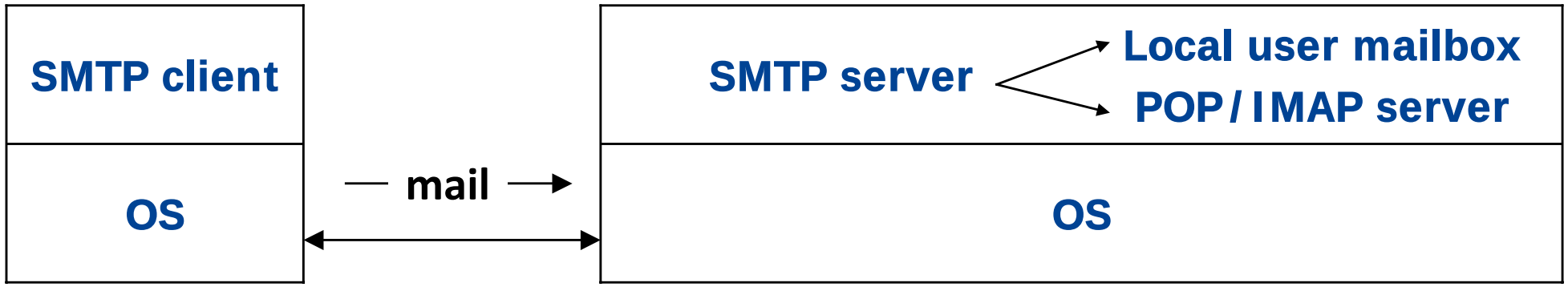
Simple request for index page CRLF

Server response

email

Simple Mail Transfer Protocol (SMTP)

Defined in RFC 5321



Post Office Protocol (POP3)

Defined in RFC 1939 + 2449 + 1734

Server access with user authentication

Download / delete messages

Internet Message Access Protocol (IMAP)

Defined in RFC 2060

Server access with user authentication

Read / delete / organize messages

email Use Cases

Unix/Linux mail

Mailbox assigned to each user account

SMTP client sends message → SMTP server → local user mailbox

Outlook/Thunderbird mail

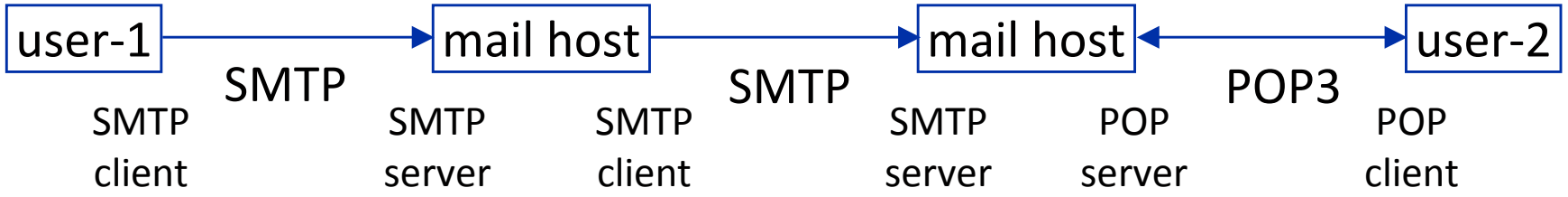
User mailboxes on mail server (host)

Outlook/Thunderbird runs SMTP client + POP3/IMAP clients

SMTP client sends message → SMTP server on user-1 mail host

SMTP client sends message → SMTP server on user-2 mail host

POP3/IMAP client gets message ←→ POP3/IMAP server on mail host



Webmail

User composes mail on web service

Web service sends mail from SMTP client → user-2 SMTP server

User read mail on web service IMAP client

Telnet to SMTP Server

```
$ telnet mail.bezeqint.net 25
Trying 192.115.106.15...
Connected to mail.bezeqint.net.
Escape character is '^]'.
220 sa14.bezeqint.net ESMTP Bezeq International SMTP out Mail Server
helo martin@multinet.net.il
250 sa14.bezeqint.net
mail from:<martin@multinet.net.il>
250 2.1.0 Ok
rcpt to:<martin@multinet.net.il>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
this is a line of a message

.

250 2.7.1 Ok, discarded, id=13560-10 - SPAM
500 5.5.2 Error: bad syntax
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

open connection to Bezeqint SMTP server

introduce user

begin new message

recipient

begin message body

message body

busted!

give up

Telnet to POP3 Server — 1

```
$ telnet mail.bezeqint.net 110
Trying 192.115.106.15...
Connected to mail.bezeqint.net.
Escape character is '^]'.
+OK Bezeqint POP3PROXY Ready 00020c4f
user **********
+OK USER ********** set, mate
pass **********
+OK Maildrop locked and ready
list
+OK scan listing follows
1 500780
2 4432
.
```

open connection to Bezeqint POP3 server

POP3 commands	
USER	user id
PASS	password
STAT	number of messages
LIST	size of messages
RETR msg	download message
TOP msg lines	download header + lines
DELE msg	delete
RSET	undelete
QUIT	end session

Telnet to POP3 Server — 2

request header of message 1 with 0 lines of message body

```
top 1 0
+OK 500780 octets
Return-Path: <simcha@hadassah.ac.il>
Received: from m5.bezeqint.net (m5.bezeqint.net [192.115.104.85])
        by mas27.bezeqint.net (MOS 3.8.6-GA)
        with ESMTP id HRK58842;
        Mon, 4 Jul 2011 14:19:51 +0300 (IDT)
Received: from baloo-new.hadassah-col.ac.il (EHLO bagheera.hadassah.ac.il)
      ([212.179.79.229])
        by m5.bezeqint.net (MOS 4.1.9-GA FastPath queued)
        with ESMTP id KHN24377;
        Mon, 04 Jul 2011 14:19:50 +0300 (IDT)
Content-class: urn:content-classes:message
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="----_=_NextPart_001_01CC3A3C.603293DC"
Subject: =?windows-1255?B?5Obu8OQ=?=
Date: Mon, 4 Jul 2011 14:18:25 +0300
Message-ID: <27B903C4A7521E47B76F1C36B1CE5ED903044469@arathorn.hadassah-col.ac.il>
From: "Simcha Rozen" <simcha@hadassah.ac.il>
To: <solange@cs.huji.ac.il>, <alhanaty@bezeqint.net>, <martin@multinet.net.il>,
        <yoram.yekutieli@gmail.com>
.
quit
+OK
Connection closed by foreign host.
```

end session

File Transfer

File Transfer Protocol (FTP)

- Transfer files between client and server
- Separate control (port 21) and data (port 20) connections

FTP server

- Accesses file system on server node
- Provides Unix-like directory services to client
- Opens, reads, copies file contents to service socket
- Creates, writes file contents from client socket

FTP client

- Requests directory services
- Sends / receives files

Active mode

- Client sends client IP address and port number to server
- Server initiates TCP data connection

Passive mode

- Server sends IP address and port number to client
- Client initiates TCP data connection

FTP Commands

About 80 commands

```
ftp> help
Commands may be abbreviated.  Commands are:

!               dir             mdelete         qc              site
$               disconnect      mdir            sendport        size
account         exit            mget            put             status
append          form            mkdir           pwd             struct
ascii           get             mls             quit            system
bell            glob            mode            quote           sunique
binary          hash            modtime         recv            tenex
bye             help            mput            reget           tick
case            idle            newer           rstatus         trace
cd              image           nmap            rhelp           type
cdup            ipany           nlist           rename          user
chmod           ipv4            ntrans          reset           umask
close           ipv6            open            restart         verbose
cr              lcd             prompt          rmdir           ?
delete          ls              passive         runique
debug           macdef          proxy           send
```

Some common commands

`!`	Send command to local system
`open`	Connect to remote ftp
`close`	Terminate ftp session
`get`	Transfer remote file to local system
`put`	Transfer local file to remote system
`mget / mput`	Transfer multiple files using wildcards
`lcd`	Change local directory

Domain Name System (DNS)

Address resolution

Converts between node name and node IP address

```
$ nslookup www.hadassah.ac.il
Server:         10.0.0.180
Address:        10.0.0.180#53

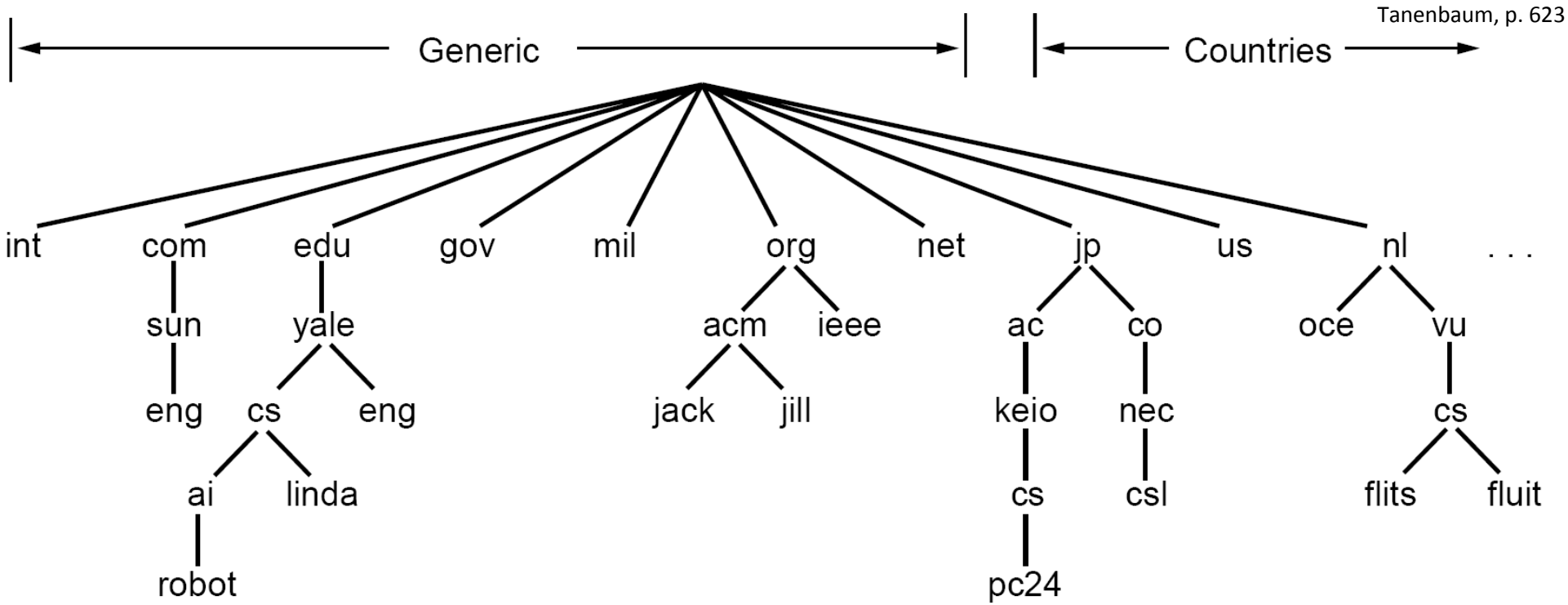
Non-authoritative answer:
Name:   hathi.hadassah.ac.il
Address: 212.179.79.228

$ nslookup 212.179.79.228
Server:         10.0.0.180
Address:        10.0.0.180#53

Non-authoritative answer:
228.79.179.212.in-addr.arpa
name = bzq-179-79-228.static.bezeqint.net
```

Domain Name Space

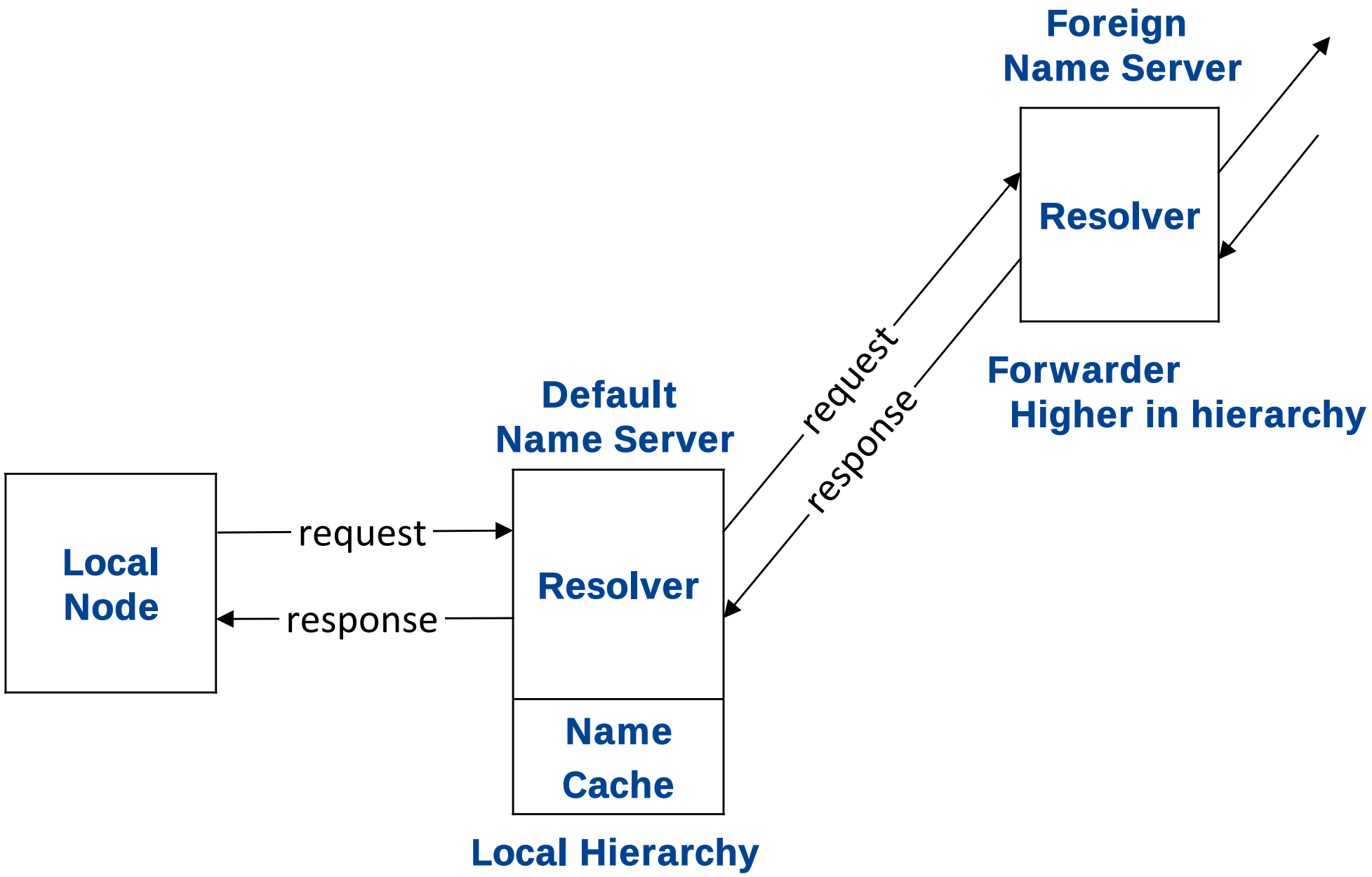
Hierarchal tree



`hathi.hadassah.ac.il`

`hathi`	server	Named server in Hadassah College domain
`hadassah`	organization	Hadassah College domain
`ac`	academic	מרכז החישובים הבינאוניברסיטאי (מחב"א)
`il`	country	איגוד האינטרנט הישראלי (ISOC-IL)

Address Resolution Hierarchy



Hierarchical Name Resolution

Host OS

Resolves addresses using default name server

Windows XP

```
c:\>ipconfig /all
Ethernet adapter Ethernet:
          ...
        IP Address. . . . . . . . . . : 10.0.0.5
        Default Gateway . . . . . . . : 10.0.0.138
        DNS Servers . . . . . . . . . : 10.0.0.11

c:\>nslookup www.hadassah.ac.il
Server:  dns.home
Address:  10.0.0.11
Non-authoritative answer:
Address:  212.179.79.228
Name:  www.hadassah.ac.il
```

Linux

```
$ cat /etc/resolv.conf
# Generated by NetworkManager
search home
nameserver 10.0.0.180
nameserver 10.0.0.11

$ nslookup www.hadassah.ac.il
Server:         10.0.0.11
Address:        10.0.0.11#53

Non-authoritative answer:
www.hadassah.ac.il
Name: www.hadassah.ac.il
Address: 212.179.79.228
```

DNS server

Resolves refers unknown address up the tree

```
$ cat /etc/bind/named.conf.options
forwarders {    192.115.106.10;
                62.219.186.7;
};
```

Resources Records (RR)

RR Field	Description	Length (octets)
NAME	Node name	Length of node name
TYPE	RR type (see table)	2
CLASS	Class code (class = Internet)	2
TTL	Time to live	4
RDLENGTH	Length of RDATA field	2
RDATA	RR-specific data	RDLENGTH

Type Value	Field Name	Description	Function
1	A	address	32-bit IPv4 address
2	NS	name server	Authoritative DNS name servers for domain
5	CNAME	name	Canonical node name
6	SOA	start of authority	Authoritative information about DNS zone: primary name server, email of domain, ...
12	PTR	pointer	Pointer to canonical name (for reverse DNS)
15	MX	mail exchange	List of email agents for that domain
16	TXT	text	Additional information

DNS Request / Response Packets

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Message ID								supplied by questioner and echoed unchanged by responder							
QR	OPCODE				AA	TC	RD	RA	res1	res2	res3	RCODE			
QDCOUNT					number of entries in question section										
ANCOUNT					number of resource records in answer section										
NSCOUNT					number of name server resource records in authority section										
ARCOUNT					number of resource records in additional records section										

QR	Query = 0 / Response = 1
OPCODE	Request operation type 0 = standard query 1 = inverse query 2 = status request
AA	1 = Authoritative Answer
TC	1 = Message truncated (too long for channel)
RD	1 = Recursion Desired
RA	1 = Recursion Available
RCODE	Response type 0 No error condition 1 Format error 2 Server failure 3 Name Error 4 Not Implemented 5 Refused

Client Socket in Perl

Perl IO::Socket package

Provides simplified implementation of socket API

Open new socket

```
use IO::Socket;
$sock = IO::Socket::INET->new(
                Proto    => "Protocol",
                PeerAddr => "Destination Address",
                PeerPort => "Port Number",
                )
        or die "cannot connect to port at localhost";
```

Read data from server

```
$data = <$sock>;
print $data;
```

Send data to server

```
print $sock "hello server!\n";
```

Perl Client Socket Example

Read Hadassah College homepage

```
#!/usr/bin/perl
use IO::Socket;


$sock = IO::Socket::INET->new(
                    Proto    => "tcp",
                    PeerAddr => "www.hadassah.ac.il",
                    PeerPort => "80",
                    )
      or die "Couldn't connect!";

print $sock "GET /main.html HTTP/1.0\n\n";

print <$sock>;

close($sock);
```

include perl Socket package
open connection to Hadassah web server
OS returns socket descriptor to variable **$sock**

send server request for home page

display on screen data returned from server

close connection

Perl Client Socket Example

Read Hadassah College homepage

```
#!/usr/bin/perl
use IO::Socket;
$sock = IO::Socket::INET->new(
                        Proto    => "tcp",
                        PeerAddr => "www.hadassah.ac.il",
                        PeerPort => "80",
                        )
           or die "Couldn't connect!";
print $sock "GET /main.html HTTP/1.0\n\n";
print <$sock>;
close($sock);
```

Save file as hadassah.pl

At command line

```
$ perl hadassah.pl
HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=utf-8
Location: http://www.hadassah.ac.il/en/error-page/
X-AspNet-Version: 4.0.30319
Date: Wed, 19 Nov 2014 17:47:18 GMT
Connection: close

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a
href="http://www.hadassah.ac.il/en/error-page/">here</a>.</h2>
</body></html>
```

Server Socket in Perl

```
#!/usr/bin/perl
use IO::Socket;

$sock = IO::Socket::INET->new(
   LocalHost => 'IP_address',

   LocalPort => port_number,

   Listen => 1,

   Reuse => 1,

   Proto => 'tcp') || die "Error\n";

$client = $sock->accept();
```

open listening socket on server

assign server address

assign service port number

listen system call

reuse port number when process stops

protocol = TCP

accept system call

Perl Server Socket Example

```
#!/usr/bin/perl
$flag = "1";
use IO::Socket;
$sock = IO::Socket::INET->new(
        LocalHost => '127.0.0.1',
        LocalPort => 1234,
        Listen => 1,
        Reuse => 1,
        Proto => 'tcp') || die "Error creating socket\n";
$client = $sock->accept();
while($flag == "1") {
  $line = <$client>;
  print $line;
  print $client "Received\n";
  if ($line =~ /bye/) {$flag = "0";}
}
close($sock);
```

In CLI run perl script

```
$perl servlet.pl
```

In second CLI enter:

```
$telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

Server

- Prints data from client Echoes `Received`
- Closes on data `bye`

LOOPBACK ADDRESS (for testing)

`127.0.0.1`

Local calls on this machine

Sockets in C
Overview

Header library

Constants, data type, data structures, system calls

Constants

Protocol parameters / sizes, address parameters /sizes, …

Data structures

`struct` templates for addresses, ports, protocols, …

`addrinfo`

`struct` template for connection information

Programmer provides partial endpoint information to instance **`hints`**

System provides remaining information in instance **`res`**

System calls

Byte order for network standard

Endpoint parameters based on OS operations

Standard socket calls

Socket, bind, listen, accept, send, receive

Sockets in C
Headers

`<sys/socket.h>`	Socket functions and data structures
`<netinet/in.h>`	**`PF_INET / PF_INET6`** protocol families **`AF_INET / AF_INET6`** address families Theory — multiple AF addresses per protocol family Practice — AF = FP
`<sys/un.h>`	**`PF_UNIX / PF_LOCAL`** address family For communication between programs on same computer
`<arpa/inet.h >`	Functions for manipulating numeric IP addresses
`<netdb.h>`	Functions for translating protocol and host names into numeric addresses

Sockets in C

Some data types and constants

`size_t`	Data type for length of data message
`socklen_t`	Data type for size of endpoint structure
`SOCK_STREAM`	Stream (connection oriented) service
`SOCK_DGRAM`	Datagram (connectionless) service
`AI_PASSIVE`	Allow OS to set local IP address
`INET_ADDRSTRLEN`	Length of IP address string on OS
`PF_INET = AF_INET`	IP protocol family
`PF_INET6 = AF_INET6`	IP version 6
`addrinfo`	**`struct`** template for description of connection
`sockaddr`	**`struct`** template for endpoint (address+port)
`sockaddr_in`	**`struct`** template for building **`sockaddr`**

Sockets in C

Predefined address information (**ai**) template

```
struct addrinfo {

    int     ai_flags;


    int     ai_family;

    int     ai_socktype;

    int     ai_protocol;

    socklen_t  ai_addrlen;

    struct  sockaddr *ai_addr;

    char    *ai_canonname;

    struct  addrinfo *ai_next;

};
```

AI_PASSIVE — OS sets address

AI_CANONNAME — use hostname

AF_INET for IPv4

SOCK_STREAM / SOCK_DGRAM

protocol / 0

size of `ai_addr` in bytes

pointer to endpoint `struct`

canonical hostname

pointer to linked AI `struct`

Sockets in C

Predefined endpoint information templates

```
struct sockaddr {
    unsigned short      sa_family;

    char                sa_data[14];



};

struct sockaddr_in {
    short int           sin_family;
    unsigned short int  sin_port;
    struct in_addr      sin_addr;
    unsigned char       sin_zero[8];
};
```

Socket address family

AF_INET (for IPv4)

protocol address

Port = 2 bytes

IP address = 4 bytes (IPv4)

Padding = 8 bytes (zeros)

Access `sockaddr` fields

AF_INET

Port number

IP address as `struct`

zeros (padding)

Sockets in C
`getaddrinfo()`

```
int getaddrinfo(const char *node,
                const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

System call

Input instance (**`hints`**) ⟶ output instance (**`res`**) using OS information

`node`	Remote node address NULL (OS provides local node address)
`service`	Port number (client's requested service or server's offered service) NULL leaves service uninitialized
`hints`	Input instance of **`addrinfo`** holding info set by App
`res`	Output instance of **`addrinfo`** holding info set by OS resolution
`Return`	Status integer — check with function **`gai_strerror()`**

Sockets in C

Who uses address information (**ai**) templates?

addrinfo

Client

Requires **addrinfo** instance to describe remote service

Writes remote server address string + remote service to **hints**

getaddrinfo converts **hints** ⟶ **res** using OS information

Server

Requires **addrinfo** instance to describe local service

Writes NULL address + local service to **hints**

getaddrinfo converts **hints** ⟶ **res** using OS information

sockaddr

getaddrinfo writes **addrinfo.sockaddr** as HEX endpoint

sockaddr_in

User copies **addrinfo.sockaddr** ⟶ **sockaddr_in**

Reads port and IP address as separate members

Sockets in C

`getaddrinfo()` example — setting endpoint for LOCAL HTTP server

```
int status;
struct addrinfo hints;
struct addrinfo *local_res;
```

hints = input struct

local_res = output struct

fill **hints** with partial information

```
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
```

initialize hints to zero

IPv4

TCP stream

OS provides IP address

system fills **local_res** with local service endpoint = **`local_IP:http`** using **hints** and OS data

```
status = getaddrinfo(NULL, "http", &hints, &local_res)
```

nonzero status — check error

```
fprintf(stderr, "getaddrinfo error: %s\n",
        gai_strerror(status));
```

Sockets in C

`getaddrinfo()` example — setting endpoint for REMOTE server

```
int status;
struct addrinfo hints;
struct addrinfo *remote_res;
```

hints = input struct

remote_res = output struct

fill **hints** with partial information

```
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
```

initialize hints to zero

IPv4

TCP stream

OS provides IP address

system fills **remote_res** with local service endpoint = **`remote_IP:80`** using **hints** and OS data

```
status = getaddrinfo(remote_IP, 80, &hints, & remote _res)
```

nonzero status — check error

```
fprintf(stderr, "getaddrinfo error: %s\n",
        gai_strerror(status));
```

Sockets in C
Address conversion

Convert text form to struct (presentation to network)

`inet_pton()`

Example

```
struct sockaddr_in sa;

inet_pton(AF_INET, "10.0.0.1", &(sa.sin_addr));
```

define instance sa of struct sockaddr_in

convert address to struct and write into sa

Convert struct to text form (network to presentation)

`inet_ntop()`

Example

```
char ip4[INET_ADDRSTRLEN];
struct sockaddr_in sa;

inet_ntop(AF_INET, &(sa.sin_addr), ip4,
    INET_ADDRSTRLEN);
```

INET_ADDRSTRLEN preset by system

convert address and write into ip4

Sockets in C
Byte order conversion

Big-endian

Integer stored in memory with MSB in lowest address

Standard for network operations (Network Order)

Little-endian

Integer stored in memory with LSB in lowest address

Standard on Intel processors

Conversion functions

Converts Host Order to Network Order

No change if Host Order = big-endian

`htons()`	host to network short (short = 2 bytes = 16 bits)
`htonl()`	host to network long (long = 4 bytes = 32 bits)
`ntohs()`	network to host short
`ntohl()`	network to host long

Sockets in C

Example for setting remote endpoint — part 1

```
/* showip.c show IP addresses for remote host */
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    struct addrinfo hints, *remote_res, *p;
    int status;
    char ipstr[INET_ADDRSTRLEN];
    if (argc != 2) {
        fprintf(stderr,"usage: showip hostname\n");
        return 1;
    }
```

3 instances

character array to hold IP addresses

Sockets in C

Example for setting remote endpoint — part 2

initialize hints to zero

```
memset(&hints, 0, sizeof hints);
```

set known hints
- IPv4
- TCP

```
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
```

set **remote_res** to system-provided info
- argv[1] = hostname
- port = 0 (not set)

```
if ((status = getaddrinfo(argv[1], NULL, &hints,
                                    &remote_res)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n",
            gai_strerror(status));
    return 2;
}
```

print error message on fail

print heading for list of addresses

```
    printf("IP addresses for %s:\n\n", argv[1]);
```

Sockets in C

Example for setting remote endpoint — part 3

loop over linked list

```
for (p = remote_res; p != NULL; p = p->ai_next) {
    void *addr;
    char *ipver;

    struct sockaddr_in *ipv4 =
                    (struct sockaddr_in *)p->ai_addr;

    addr = &(ipv4->sin_addr);
    ipver = "IPv4";

    inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
    printf("  %s: %s\n", ipver, ipstr);
}
    freeaddrinfo(res);
    return 0;
}
```

instance **ipv4** ← instance **p.ai_addr**
ai_addr points to **sockaddr** = endpoint = IP:port

addr ← IP address in **p.ai_addr**

convert **addr** = IP address to **ipstr** = character string

print IPv4: **ipstr**

free linked list

Sockets in C

Running **showip**

```
$ showip www.google.co.il
IP addresses for www.google.co.il:

   IPv4: 74.125.230.148
   IPv4: 74.125.230.144
   IPv4: 74.125.230.145
   IPv4: 74.125.230.146
   IPv4: 74.125.230.147
```

Sockets in C

socket() with hard coded **endpoint type**

```
int socket(int domain, int type, int protocol);
```

domain	Address family
type	Service type
protocol	Specify protocol or 0 default — TCP for STREAM or UDP for DGRAM

```
int s;
s = socket(PF_INET, SOCK_STREAM, 0);
```

socket descriptor

PF_INET = AF_INET = IPv4
SOCK_STREAM = reliable transport
0 = use default service protocol — TCP for STREAM or UDP for DGRAM

No local or remote endpoint required to open socket

Sockets in C

socket() using fields in `struct res`

```
int socket(int domain, int type, int protocol);
```

domain	Address family
type	Service type
protocol	Specify protocol or 0 default — TCP for STREAM or UDP for DGRAM

```
int s;
struct addrinfo hints, *res;
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;

getaddrinfo("www.hadassah.ac.il", "http", &hints, &res);

// getaddrinfo(NULL, "http", &hints, &res);

s = socket(res->ai_family, res->ai_socktype,
           res->ai_protocol);
```

socket descriptor

set up **hints**

client side for remote server

server side for local server

socket call using **res**

Sockets in C
bind()

```
int bind(int sockfd, const struct sockaddr *my_addr,
         socklen_t addrlen);
```

prepare **hints**

```
int s;
struct addrinfo hints, *local_res;
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
```

OS sets node IP address

resolve local address and port to **local_res** , open socket as server

```
getaddrinfo(NULL, "http", &hints, &local_res);
s = socket(local_res->ai_family, local_res->ai_socktype,
     local_res->ai_protocol);
```

bind socket to local service using **ai_address** endpoint (instance of **sockaddr**)

```
bind(s, local_res->ai_addr, local_res->ai_addrlen);
```

Sockets in C
listen() and accept()

```
int listen(int sockfd, int backlog);
```

```
listen(s, backlog);
```

listen on socket **s**
allow connections ≤ **backlog**

```
struct sockaddr_storage cliaddr;
socklen_t addr_size;
addr_size = sizeof cliaddr;
```

sockaddr_storage
struct for connected endpoint
designed for IPv4 and IPv6
will be cast as **sockaddr** in **accept()**

```
int accept(int sockfd, struct sockaddr *cliaddr,
           socklen_t *addrlen);
```

```
int c_s;
```

create copy **c_s** of socket **s** for connection
accept connection on socket **c_s**
cast **cliaddr** as **sockaddr**
write client endpoint information to **cliaddr**

```
c_s = accept(s, (struct sockaddr *)&cliaddr,&addr_size);
```

Sockets in C
connect()

```
int connect(int sockfd,
            const struct sockaddr *serv_addr,
            socklen_t addrlen);
```

```
int s;
struct addrinfo hints, *remote_res;
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;

getaddrinfo("www.google.com, "http", &hints, &remote_res);
```

prepare **hints**, resolve remote **addrinfo**

resolve remote address and port to **remote_res**

open socket **s** for connecting to remote service using **ai_address** endpoint

```
s = socket(remote_res->ai_family, remote_res->ai_socktype,
     remote_res->ai_protocol);

connect(s, remote_res->ai_addr, remote_res->ai_addrlen)
```

Sockets in C
send()

```
int send(int sockfd, const void *msg, size_t len,
         int flags);
```

send string on socket **c_s**
returns number of bytes actually sent

```
char *msg = "Hello, world!";
size_t s_len,
int bytes_sent;
s_len = strlen(msg);
bytes_sent = send(c_s, msg, s_len, 0);
```

Sockets in C
recv()

```
int recv(int sockfd, void *buf, size_t len, int flags);
```

receive string on socket **c_s**
returns number of bytes actually received

choose desired receive buffer size

```
#define BUFFER_SIZE 100
char buffer[BUFFER_SIZE];
size_t r_len = BUFFER_SIZE;
int bytes_received;
bytes_received = recv(c_s, buffer, r_len, 0);
```

Sockets in C

Connectionless data transfer — sendto() and recvfrom()

```
int send(int sockfd, const void *msg, size_t len,
         int flags);
```



No connect() for local sockfd
Define destination endpoint on the fly

```
int sendto(int sockfd, const void *msg, size_t len,
           int flags, const struct sockaddr *to,
           socklen_t tolen);
```

```
int recv(int sockfd, void *buf, size_t len, int flags);
```



No accept() for local sockfd
Define source endpoint on the fly

```
int recvfrom(int sockfd, void *buf, size_t len,
             int flags, struct sockaddr *from,
             size_t *fromlen);
```