Sockets
Simple Web Request
How do we find the server?

- Every computer on the Internet has an Internet address.
- Called an IP address (Internet Protocol)
- An IP address is 4 8-bit numbers separated by dots.

www.ktu.edu.tr = 193.140.168.225
Domain Name Servers

```
browser

www.eecg.utoronto.ca?

www.eecg.utoronto.ca?

www.eecg.utoronto.ca?

root name server

c server

utoronto server

“ca” name server

local name server

128.100.10.235

“eeecg” name server

www.eecg.utoronto.ca?

“utoronto” name server

www.eecg.utoronto.ca?
```
This is getting complicated!

Number of messages? 10-12
Protocols
TCP/IP

- Transmission Control Protocol.
- Tells us how to package up the data.

<table>
<thead>
<tr>
<th>source address</th>
<th>dest. address</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes</td>
<td>ack</td>
</tr>
<tr>
<td></td>
<td>port</td>
</tr>
</tbody>
</table>

data
TCP Connection

Hi 128.100.10.25
Connection port 1?
okay
Send me a file
Got it
ack
Here’s some data
Got it
ack
Here’s some more
Got it
ack
I’m done
fin
I’m done too
fin

Hi 128.100.10.128
Let’s talk

3-way handshake SYN
Routing

- dis17-toronto64-fe10.bellnexxia.net
- dis1-toronto-64-pos2.bellnexxia.net
- onet-gw.dis1-toronto.bellnexxia.net
- utorgw-border-if.onet.on.ca
- sf2-bbup.gw.utoronto.ca

browser

server
Putting it together
How many messages?

- It depends on the size of the web page we retrieve.
- If the web page is 75 Kbytes (small!) it will be broken up into 103 IP packets.
- Remember DNS took 10 messages

\[10 + 103 \times 7 \text{ hops} = 731 \text{ messages!}\]
Sockets

- One form of communication between processes.
- Similar to pipes, except sockets can be used between processes on different machines.
- Use file descriptors to refer to sockets.
- Built on top of TCP layer
TCP: Three-way handshake

- **Sequence number = J**
  - socket
  - connect (blocks)
- **Sequence number = K**
  - socket, bind, listen
  - accept (blocks)
  - SYN K, ack J+1
  - connect
  - returns
  - accept
  - returns
  - ack K+1
TCP Server

```
socket()
bind()
listen()
accept()
```

block until connection from client

```
read()
write()
```

end-of-file notification

TCP Client

```
socket()
connect()
write()
```

Connection establishment (3-way handshake)

data transfer

```
read()
close()
```

Chapter Fifteen Sockets
Connection-Oriented

Server
- Create a socket: `socket()`
- Assign a name to a socket: `bind()`
- Establish a queue for connections: `listen()`
- Get a connection from the queue: `accept()`

Client
- Create a socket: `socket()`
- Initiate a connection: `connect()`
Socket Types

- Two main categories of sockets
  - UNIX domain: both processes on the same machine
  - INET domain: processes on different machines

- Three main types of sockets:
  - SOCK_STREAM: the one we will use
  - SOCK_DGRAM: for connectionless sockets
  - SOCK_RAW
Addresses and Ports

- A **socket pair** is the two endpoints of the connection.
- An endpoint is identified by an **IP address** and a **port**.
- IPv4 addresses are 4 8-bit numbers:
  - 128.100.31.156 = penguin
  - 128.100.31.4 = eddie
- **Ports**
  - because multiple processes can communicate with a single machine we need another identifier.
More on Ports

- Well-known ports: 0-1023
  - 80 = web
  - 21 = ftp
  - 22 = ssh
  - 25 = smtp (mail)
  - 23 = telnet
  - 194 = irc

- Registered ports: 1024-49151
  - 2709 = supermon
  - 26000 = quake

- Dynamic (private) ports: 49152-65535
  - You should pick ports in this range to avoid overlap
TCP Server

- socket()
- bind()
- listen()
- accept()
  block until connection
  from client

TCP Client

- socket()
- connect()
- write()
- read()
- close()

Connection establishment
(3-way handshake)

data transfer

echo-of-file notification

Server side

int socket(int family, int type, int protocol);

- family specifies protocol family:
  - PF_INET - IPv4
  - PF_LOCAL - Unix domain

- type
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW

- protocol
  - set to 0 except for RAW sockets

- returns a socket descriptor
bind to a name

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

- **sockfd** - returned by socket

```c
struct sockaddr_in{
    short   sin_family; /*PF_INET */
    u_short sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
}
```

- **sin_addr** can be set to INADDR_ANY to communicate with any host
Set up queue in kernel

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

- **after calling `listen`, a socket is ready to accept connections**
- prepares a queue in the kernel where partially completed connections wait to be accepted.
- **backlog** is the maximum number of partially completed connections that the kernel should queue.
Complete the connection

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

- blocks waiting for a connection (from the queue)
- returns a new descriptor which refers to the TCP connection with the client
  - `sockfd` is the listening socket
  - `cliaddr` is the address of the client
- reads and writes on the connection will use the socket returned by accept
Client side

- `socket()` - same as server, to say “how” we are going to talk

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

- the kernel will choose a dynamic port and source IP address.
- returns 0 on success and -1 on failure setting `errno`.
- initiates the three-way handshake.
Byte order

- **Big-endian**

91,329

<table>
<thead>
<tr>
<th>A</th>
<th>A+1</th>
<th>A+2</th>
<th>A+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>64</td>
<td>C1</td>
</tr>
</tbody>
</table>

- **Little-endian**

91,329

<table>
<thead>
<tr>
<th>A+3</th>
<th>A+2</th>
<th>A+1</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>64</td>
<td>C1</td>
</tr>
</tbody>
</table>

- **Intel is little-endian, and Sparc is big-endian**
Network byte order

- To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (bigendian) before we send them.

- There are functions provided to do this:
  - unsigned long htonl(unsigned long)
  - unsigned short htons(unsigned short)
  - unsigned long ntohl(unsigned long)
  - unsigned short ntohs(unsigned short)