Networking via TCP

machine\textsubscript{1}

process\textsubscript{1}

machine\textsubscript{2}

process\textsubscript{2}
Networking via TCP

machine₁

process₁

machine₂

process₂
Networking via TCP

machine_1

process_1

machine_2

process_2

= tcp_open(...);

= tcp_open(...);
Networking via TCP

machine\textsubscript{1} \quad \text{process}_{1} \quad \text{hello} \quad \text{machine}_{2} \quad \text{process}_{2}

\text{Write}(\leftarrow \rightarrow, "hello", 5); \quad \text{Read}(\leftarrow \rightarrow, \text{buffer}, 5);

= \text{tcp\_open}(\ldots); \quad = \text{tcp\_open}(\ldots);
Networking via TCP

= tcp_open(...);

Write( ↔ , "hello", 5);
Read( ↔ , buffer, 2);

Read( ↔ , buffer, 5);
Write( ↔ , "hi", 2);
Networking via TCP

```
lnr = Open_listenfd(...);
= Accept(lnr, ...);

Write( , "hello", 5);
Read( , buffer, 5);

Read( , buffer, 2);
Write( , "hi", 2);
```
Networking via TCP

```c
lnr = Open_listentfd(....);

server

= Accept(lnr, ....);

client

Write(↔️, "hello", 5);

Read(↔️, buffer, 5);

Read(↔️, buffer, 2);

Write(↔️, "hi", 2);
```
Networking via TCP

machine\textsubscript{1}

server

machine\textsubscript{2}

client

machine\textsubscript{3}

client\textsubscript{2}
Networking via UDP

machine$_1$

process$_1$

machine$_2$

process$_2$
Networking via UDP

\[ \uparrow = \text{Socket(....)}; \]
\[ \text{Bind}(\uparrow, \text{....}); \]

\[ \downarrow = \text{Socket(....)}; \]
\[ \text{Bind}(\downarrow, \text{....}); \]
Networking via UDP

```
# on machine1

process1

strup = Socket(...);
_bind(strup, ...);
_sendto(strup, "hello", 5, ...);

# on machine2

process2

recv = Socket(...);
_bind(recv, ...);
_recv(recv, buffer, 5, 0);
```
Networking via UDP

\[
\begin{align*}
\text{machine}_1 & \quad \text{machine}_2 \\
\text{process}_1 & \quad \text{process}_2 \\
\end{align*}
\]

\(\Rightarrow\) = Socket(....);  \\
Bind(\Rightarrow, ....);  \\
Recv(\Rightarrow, buffer, 2);  \\

\(\Leftarrow\) = Socket(....);  \\
Bind(\Leftarrow, ....);  \\
Sendto(\Leftarrow, "hi", 2, ...);
TCP vs. UDP

**TCP**

Connection- and stream-oriented

Reliable

The most widely used networking protocol

**UDP**

Connectionless and packet-oriented

Best-effort

Minimal structure over next primitive layer

Both built on **IP**
Finding Hosts on a Network

Using **IP**, a **host** is named by a 32-bit value

More precisely, this is IPv4

Written as dot-separated, unsigned 8-bit values

155.98.111.50

129.10.116.81
Finding Hosts on a Network

Using **IP**, a **host** is named by a 32-bit value

More precisely, this is IPv4

Written as dot-separated, unsigned 8-bit values

```
155.98.111.50
129.10.116.81
```
Finding Hosts on a Network

Using **IP**, a **host** is named by a 32-bit value

More precisely, this is **IPv4**

A **port** plus **protocol** identifies an endpoint within a host
Finding Hosts on a Network

Using **IP**, a **host** is named by a 32-bit value

More precisely, this is IPv4

A **port** plus **protocol** identifies an endpoint within a host
Message Transport

client process  155.98.111.50

server process  129.10.116.81
               80  tcp
Message Transport

client process 155.98.111.50

HTTP/1.1...

payload

server process 129.10.116.81

80 tcp
Message Transport

```
155.98.111.50
```

```
129.10.116.81
```

**packet header**
Message Transport

```
client process  155.98.111.50

HTTP/1.1...

kernel

server process  129.10.116.81
                80 tcp

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

network interface controller
```
Message Transport

client process

155.98.111.50

HTTP/1.1...

kernel

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

server process

129.10.116.81 80 tcp

NIC

router

NIC

routing tables

NIC

NIC

NIC

NIC
Message Transport

client process

HTTP/1.1...

kernel

NIC

server process

129.10.116.81:80 tcp

frame header

155.98.111.50

129.10.116.81:80 tcp

155.98.111.50:7365 tcp

NIC

routing tables

NIC
Message Transport

Client process

155.98.111.50

HTTP/1.1

server process

129.10.116.81 80 tcp

kernel

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1

NIC

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1

NIC

3B:92:...

3B:92:...

NIC

3B:92:...

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1

router

A5:78:...

NIC

routing tables

NIC

NIC

NIC

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition
Message Transport

client process

155.98.111.50

kernel

HTTP/1.1...

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

server process

129.10.116.81

80 tcp

155.98.111.50:7365 tcp

HTTP/1.1...

kernel

HTTP/1.1...

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

3B:92:...

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

3B:92:...

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

router

routing tables

NIC

NIC

NIC

Message Transport

client process
192.168.1.100

HTTP/1.1...

kernel

129.10.116.81:80 tcp
192.168.1.100:6022 tcp

NIC

server process
129.10.116.81:80 tcp
192.168.1.100:6022 tcp

HTTP/1.1...

kernel

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

NIC

155.98.111.50

NAT router

129.10.116.81:80 tcp
155.98.111.50:7365 tcp

HTTP/1.1...

A5:78:

A5:78:

3B:92:

3B:92:

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition
Name Resolution

IP locates hosts by number

\[ \text{e.g., 155.98.111.50} \]

```
$ ssh 155.98.111.50
```

For many purposes, names are obviously better

\[ \text{e.g., lab1-1.eng.utah.edu} \]

```
$ ssh lab1-1.eng.utah.edu
```
**Name Resolution**

**DNS** (Domain Name System) maps names to remote addresses:
- identify DNS server by address
- DNS server maps names to addresses and vice versa

155.98.111.50

lab1-1.eng.utah.edu?
Ask 8.8.8.8...

8.8.8.8
**DNS** (Domain Name System) maps names to remote addresses
- identify DNS server by address
- DNS server maps names to addresses and vice versa

```
155.98.111.50
```

```
lab1-1.eng.utah.edu?
Ask 8.8.8.8...
```

```
lab1-1.eng.utah.edu = 155.98.111.50
```

```
8.8.8.8
```
Name Resolution

Multiple names can map to the same address

$ ./hostinfo www.eng.utah.edu
155.98.110.30
$ ./hostinfo www.cade.utah.edu
155.98.110.30

We’ll implement hostinfo...
Name Resolution

A single name can map to multiple addresses

$ ./hostinfo twitter.com
104.244.42.129
104.244.42.65
104.244.42.193
104.244.42.1
Name Resolution

Naming is not just about finding remote hosts
Name Resolution

Naming is not just about finding remote hosts

- process
- process
- kernel
- localhost
- 127.0.0.1
- 155.98.111.50
- loopback NIC
- ethernet NIC
Name Resolution

Naming is not just about finding remote hosts
Name Resolution

Naming is not just about IPv4

- localhost via IPv4 = 127.0.0.1
- localhost via IPv6 = ::1

System calls need to support many protocols
C Library for Name Resolution

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *hostname, const char *servname, 
                 const struct addrinfo *hints, 
                 struct addrinfo **res);
```

Converts a combination of name and port to one or more protocol-specific addresses

- **hostname** can be **NULL** for “any here”
- **servname** is a port number or alias, **NULL** for “any”
- **hints** can request IPv4, TCP, etc.
- **res** is the result: set to a linked list of addresses
C Library for Name Resolution

```
struct addrinfo {
    int ai_flags;
    int ai_family;       /* protocol family */
    int ai_socktype;     /* socket type */
    int ai_protocol;     /* protocol */
    socklen_t ai_addrlen; /* length of ai_addr */
    struct sockaddr *ai_addr; /* address */
    char *ai_canonname;
    struct addrinfo *ai_next; /* next in list */
};
```

Represents an IPv4 address when `ai_family = AF_INET`
C Library for Name Resolution

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                 char *host, socklen_t hostlen,
                 char *serv, socklen_t servlen,
                 int flags);
```

The reverse of `getaddrinfo`

Set flags to

```
NI_NUMERICHOST | NI_NUMERICSERV
```

for numeric address and port
```c
#include "csapp.h"

int main(int argc, char **argv, char **envp) {
    struct addrinfo hints, *addrs, *addr;
    char host[256];

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* Request IPv4 */
    hints.ai_socktype = SOCK_STREAM; /* TCP connection */
    Getaddrinfo(argv[1], NULL, &hints, &addrs);

    for (addr = addrs; addr != NULL; addr = addr->ai_next) {
        Getnameinfo(addr->ai_addr, addr->ai_addrlen,
                    host, sizeof(host),
                    NULL, 0,
                    NI_NUMERICHOST);
        printf("%s\n", host);
    }
    ....
}
```
C Library for Name Resolution

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
```

Frees options allocated by `getaddrinfo`
UDP/IP and TCP/IP

**IP** is the addressing and packet-transfer layer
- Packets can get lost
- Packets can get reordered

**UDP** is a thin layer on IP
- Packets can get lost
- Packets can get reordered

**TCP** is a substantial layer on IP
- Mostly hides packet nature behind a stream interface
- Retries as needed to get data sent
- Tags packets with sequence numbers for ordering
Sockets

A generic communication is a **socket**

A socket is represented as an **int**
Sockets

sockets

file descriptors
Sockets

sockets

file descriptors

files
Sockets

sockets

file descriptors

files

UDP

TCP
Sockets

sockets

file descriptors

files

IP

UDP

TCP
#include <sys/socket.h>

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

Sockets

Creates a new socket

- **domain** is a protocol family; **PF_INET** means IPv4
- **type** is
  - **SOCK_DGRAM** for UDP
  - **SOCK_STREAM** for TCP
- **protocol** is a kind of subtype

For portable code, get arguments from the result of
**getaddrinfo**
#include <sys/socket.h>

int bind(int socket,
          struct sockaddr *addr, socklen_t addr_len);

Attaches a socket to a specific address

If other processes know the address, they can send a message to the socket

The **addr** and **addr_len** arguments come from **getaddrinfo**
Using UDP

Diagram showing the process of using UDP with calls to `getaddrinfo`, `socket`, `bind`, `sendto`, `recvfrom`, and `close`.
Using UDP

Automatically selected ports are in the *ephemeral port* range
Using UDP

[Diagram showing the process of using UDP with functions like `getaddrinfo`, `socket`, `sendto`, `recv`, and `close`.]

Use `recv` if source of message isn’t needed.
#include "csapp.h"

int main(int argc, char **argv) { /* argv[0] == portno */
    struct addrinfo hints, *addrs;
    int s;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* Request IPv4 */
    hints.ai_socktype = SOCK_DGRAM;  /* Accept UDP connections */
    hints.ai_flags = AI_PASSIVE;     /* ... on any IP address */
    Getaddrinfo(NULL, argv[0], &hints, &addrs);

    s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
    Bind(s, addrs->ai_addr, addrs->ai_addrlen);
    Freeaddrinfo(addrs);

    while (1) {
        char buffer[MAXBUF];
        size_t amt;
        amt = Recv(s, buffer, MAXBUF, 0);
        Write(1, buffer, amt);
        Write(1, "\n", 1);
    }

    return 0;
}
#include "csapp.h"

int main(int argc, char **argv, char **envp) {
    char *hostname = argv[1], *portno = argv[2];
    struct addrinfo hints, *addrs;
    char host[256], serv[32];
    int s;
    size_t amt;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;      /* Request IPv4 */
    hints.ai_socktype = SOCK_DGRAM; /* UDP connection */
    Getaddrinfo(hostname, portno, &hints, &addrs);

    Getnameinfo(addr->ai_addr, addr->ai_addrlen,
                host, sizeof(host), serv, sizeof(serv),
                NI_NUMERICHOST | NI_NUMERICSERV);
    printf("sending to %s:%s\n", host, serv);

    s = Socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
    amt = Sendto(s, argv[3], strlen(argv[3]), 0,
                 addr->ai_addr, addr->ai_addrlen);
    Freeaddrinfo(addr);

    return (amt != strlen(argv[3]));
}
#include "csapp.h"

int main(int argc, char **argv) {
    /* argv[0] == portno */
    struct addrinfo hints, *addrs;
    int s;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* Request IPv4 */
    hints.ai_socktype = SOCK_DGRAM;  /* Accept UDP connections */
    hints.ai_flags = AI_PASSIVE;     /* ... on any IP address */
    Getaddrinfo(NULL, argv[0], &hints, &addrs);

    s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
    Bind(s, addrs->ai_addr, addrs->ai_addrlen);
    Freeaddrinfo(addrs);

    while (1) {
        char buffer[MAXBUF];
        size_t amt;
        amt = Recv(s, buffer, MAXBUF, 0);
        Write(1, buffer, amt);
        Write(1, "\n", 1);
    }

    return 0;
}
Revised UDP Server

```c
....
int counter = 0;
....

while (1) {
    char buffer[MAXBUF];
    size_t amt;
    struct sockaddr_in from_addr;
    unsigned int from_len = sizeof(from_addr);

    amt = recvfrom(s, buffer, MAXBUF, 0,
        (struct sockaddr *)&from_addr, &from_len);
    Write(1, buffer, amt);
    Write(1, "\n", 1);

    Getnameinfo((struct sockaddr *)&from_addr, from_len,
        host, sizeof(host),
        serv, sizeof(serv),
        NI_NUMERICHOST | NI_NUMERICSERV);
    printf(" from %s:%s [%d]\n", host, serv, ++counter);
}
....
```
Revised UDP Client

```
char *myportno = argv[1];
char *hostname = argv[2];
char *portno = argv[3];
....
Getaddrinfo(NULL, myportno, &hints, &my_addrs);
....

if (argc == 6)
   copies = atoi(argv[5]);
else
   copies = 1;

while (copies--)
   amt = Sendto(s, argv[4], strlen(argv[4]), 0,
                addrs->ai_addr, addrs->ai_addrlen);
....
```