

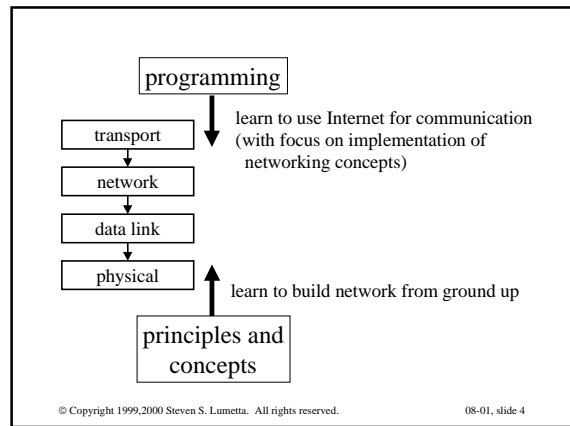
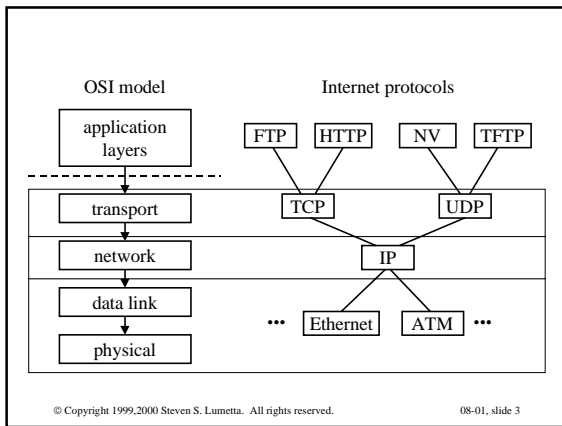
ECE/CS 338, CSE 325
Communication Networks for Computers

Prof. Steve Lumetta and Prof. Bruce Hajek

**Introduction to
 Network Programming**

Why Programming, and Why Sockets?

- things to learn
 - Internet protocols (IP, TCP, UDP, and more)
 - sockets API (application programming interface)
- why?
 - Internet Protocol (IP) is standard
 - allows a common namespace across most of Internet
 - reduces number of translations, which incur overhead
 - sockets: reasonably simple and elegant, Unix interface (most servers run Unix), available on EWS machines



Network Programming with Sockets

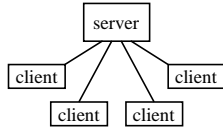
- reading: Stevens 2nd ed., Ch. 1-6 or 1st ed., Ch. 1-3, 6
- sockets API: a transport layer service interface
 - introduced in 1981 by BSD 4.1
 - implemented as library and/or system calls
 - similar interfaces to TCP and UDP
 - can also serve as interface to IP (for super-user); known as “raw sockets”

Outline

- client-server model
- TCP connections
- UDP services
- addresses and data
- sockets API
- example of use

Client-Server Model

- asymmetric relationship
- server/daemon
 - well-known name
 - waits for contact
 - process requests, sends replies
- client
 - initiates contact
 - waits for response



© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 7

- bidirectional communication model
- service models
 - concurrent: server processes multiple clients' requests simultaneously
 - sequential: server processes only one client's requests at a time
 - hybrid: server maintains multiple connections, but processes requests sequentially
- server and client categories not disjoint
 - server can be client of another server
 - server can be client of its own client (peer-to-peer architecture)

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 8

TCP Connections

- Transmission Control Protocol, at OSI transport layer
- recall: each protocol provides service interface
- aspects of TCP service
 - transfers a stream of bytes (interpreted by application)
 - connection-oriented
 - set up connection before communicating
 - tear down connection when done
 - in-order delivery of data: if A sends M1 followed by M2 to B, B never receives M2 before M1

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

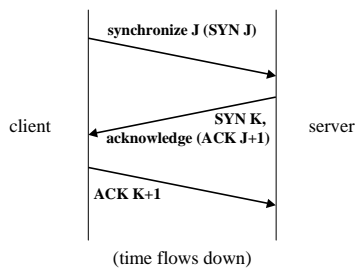
08-01, slide 9

- aspects of TCP service (continued)
 - reliable
 - data delivered at most once
 - exactly once if no catastrophic failures
 - flow control
 - prevents senders from wasting bandwidth
 - reduces global congestion problems
 - full-duplex: send or receive data at any time
 - 16-bit port space allows multiple connections on a single host

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 10

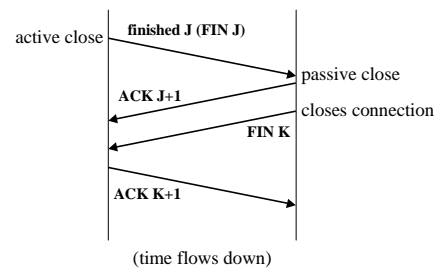
- TCP connection setup via 3-way handshake
 - J and K are sequence numbers for messages



© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 11

- TCP connection teardown (4 steps)
 - (either client or server can initiate connection teardown)



© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 12

UDP Services

- User Datagram Protocol, at OSI transport layer
- thin veneer over IP
- aspects of services
 - unit of transfer is a datagram (variable length packet)
 - unreliable, drops packets silently
 - no ordering guarantees
 - no flow control
 - 16-bit port space (distinct from TCP ports) allows multiple recipients on a single host

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 13

Addresses and Data

- Internet domain names: human readable
 - mnemonic
 - variable length
 - e.g., mercenrike.crhc.uiuc.edu
- IP addresses: easily handled by routers/computers
 - fixed length
 - tied (loosely) to geography
 - e.g., 131.126.143.82

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 14

- machines on Internet have different endianness
 - little-endian (Intel, DEC): least significant byte of word stored in lowest memory address
 - big-endian (Sun, SGI, HP): most significant byte...
 - network byte order is big-endian
 - use of network byte order
 - imperative for some data (e.g., IP addresses)
 - good form for all binary data (e.g., application-specific)
 - ASCII/Unicode acceptable alternatives

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 15

- 16- and 32-bit conversion functions (for platform independence)

```
int m, n;  
short int s, t;
```

```
m = ntohl (n)    net-to-host long (32-bit) translation  
s = ntohs (t)    net-to-host short (16-bit) translation
```

```
n = htonl (m)    host-to-net long (32-bit) translation  
t = htons (s)    host-to-net short (16-bit) translation
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 16

- socket address structures (all but `sin_family` in network byte order)

IP address

```
struct in_addr {  
    in_addr_t s_addr; /* 32-bit IP address */  
};
```

TCP or UDP address

```
struct sockaddr_in {  
    short sin_family; /* e.g., AF_INET */  
    ushort sin_port; /* TCP/UDP port */  
    struct in_addr; /* IP address */  
};
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 17

- address conversion; all binary values used and returned by these functions are network byte ordered

```
struct hostent* gethostbyname (const char* hostname);  
translate English host name to IP address (uses DNS)
```

```
struct hostent* gethostbyaddr (const char* addr,  
    size_t len, int family);  
translate IP address to English host name (not secure)
```

```
char* inet_ntoa (struct in_addr inaddr);  
translate IP address to ASCII dotted-decimal notation (e.g.,  
    "128.32.36.37"); not thread-safe
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 18

- address conversion (continued)

```
in_addr_t inet_addr (const char* strptr);
translate dotted-decimal notation to IP address;
returns -1 on failure, thus cannot handle
broadcast value "255.255.255.255"
```

```
int inet_aton (const char* strptr,
              struct in_addr inaddr);
translate dotted-decimal notation to IP address;
returns 1 on success, 0 on failure
```

```
int gethostname (char* name, size_t namelen);
read host's name (use with gethostbyname to find local IP)
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 19

framing problem

- approach
 - think about the problem for a minute or two
 - introduce yourself to 2-3 people near you (form groups of 3-4)
 - discuss the problem and agree on a solution
 - pick a representative
 - I'll ask a couple groups to present their answers

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 20

Sockets API

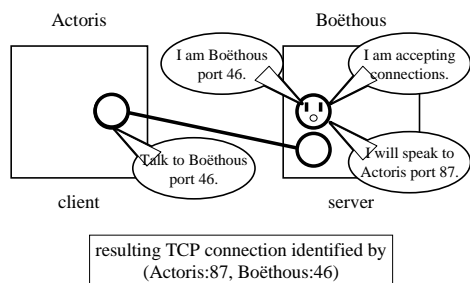
- basic Unix concepts
- creation and setup
- establishing a connection (TCP only)
- sending and receiving data
- tearing down a connection (TCP only)
- advanced sockets

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 21

- basic Unix concepts

- I/O
 - per-process table of I/O channels
 - table entries can describe files, sockets, devices, pipes, etc.
 - unifies I/O interface
 - table entry/index into table called "file descriptor"
- error model
 - return value
 - 0 on success, -1 on failure
 - NULL on failure for routines returning pointers
 - `errno` variable

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 22



© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 23

- socket creation and setup (sys/socket.h)

```
int socket (int family, int type, int protocol);
Create a socket. Returns file descriptor or -1.
```

```
int bind (int sockfd, struct sockaddr* myaddr,
         int addrlen);
Bind a socket to a local IP address and port number.
```

```
int listen (int sockfd, int backlog);
Put socket into passive state (wait for connections rather
than initiate a connection).
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 24

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

Create a socket. Returns file descriptor or -1. Also sets `errno` on failure.

family: address family (namespace)
 AF_INET for IPv4
 other possibilities: AF_INET6 (IPv6), AF_UNIX or AF_LOCAL (Unix socket), AF_ROUTE (routing)

type: style of communication
 SOCK_STREAM for TCP (with AF_INET)
 SOCK_DGRAM for UDP (with AF_INET)

protocol: protocol within family typically 0

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 25

```
int bind (int sockfd, struct sockaddr* myaddr, int addrlen);
```

Bind a socket to a local IP address and port number. Returns 0 on success, -1 and sets `errno` on failure.

sockfd: socket file descriptor (returned from `socket`)
myaddr: includes IP address and port number
 IP address: set by kernel if value passed is INADDR_ANY, else set by caller
 port number: set by kernel if value passed is 0, else set by caller
addrlen: length of address structure = sizeof (struct sockaddr_in)

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 26

notes on TCP and UDP port namespaces

allocated and assigned by the Internet Assigned Numbers Authority (see RFC 1700 or <ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers>)

1-512 standard services (see /etc/services); super-user only
 513-1023 registered and controlled, also used for identity verification; super-user only
 1024-49151 registered services/ephemeral ports
 49152-65535 private/ephemeral ports

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 27

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

Put socket into passive state (wait for connections rather than initiate a connection). Returns 0 on success, -1 and sets `errno` on failure.

sockfd: socket file descriptor (returned from `socket`)
backlog: bound on length of unaccepted connection queue (connection backlog); kernel will cap, thus better to set high

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 28

- establishing a connection (sys/socket.h)

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

Connect to another socket.

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

Accept a new connection. Returns file descriptor or -1.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 29

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

Connect to another socket. Returns 0 on success, -1 and sets `errno` on failure.

sockfd: socket file descriptor (returned from `socket`)
servaddr: IP address and port number of server
addrlen: length of address structure = sizeof (struct sockaddr_in)

Can use with UDP to restrict incoming datagrams and to obtain asynchronous errors.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 30

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

Accept a new connection. Returns file descriptor or -1. Also sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`cliaddr`: IP address and port number of client (returned from call)

`addrlen`: length of address structure = pointer to int set to sizeof (struct `sockaddr_in`)

`addrlen` is a **value-result** argument: the caller passes the size of the address structure, the kernel returns the size of the client's address (the number of bytes written)

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 31

- sending and receiving data

```
int write (int sockfd, char* buf, size_t nbytes);
```

Write data to a stream (TCP) or "connected" datagram (UDP) socket. Returns number of bytes written or -1.

```
int read (int sockfd, char* buf, size_t nbytes);
```

Read data from a stream (TCP) or "connected" datagram (UDP) socket. Returns number of bytes read or -1.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 32

- sending and receiving data (continued)

```
int sendto (int sockfd, char* buf, size_t nbytes,
            int flags, struct sockaddr* destaddr,
            int addrlen);
```

Send a datagram to another UDP socket. Returns number of bytes written or -1.

```
int recvfrom (int sockfd, char* buf, size_t nbytes,
              int flags, struct sockaddr* srcaddr,
              int* addrlen);
```

Read a datagram from a UDP socket. Returns number of bytes read or -1.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 33

```
int write (int sockfd, char* buf, size_t nbytes);
```

Write data to a stream (TCP) or "connected" datagram (UDP) socket. Returns number of bytes written or -1. Also sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`buf`: data buffer

`nbytes`: number of bytes to try to write

some reasons for failure or partial writes:

process received interrupt or signal

kernel resources unavailable (*e.g.*, buffers)

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 34

```
int read (int sockfd, char* buf, size_t nbytes);
```

Read data from a stream (TCP) or "connected" datagram (UDP) socket. Returns number of bytes read or -1. Also sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`buf`: data buffer

`nbytes`: number of bytes to try to read

Returns 0 if socket closed.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 35

```
int sendto (int sockfd, char* buf, size_t nbytes,
            int flags, struct sockaddr* destaddr,
            int addrlen);
```

Send a datagram to another UDP socket. Returns number of bytes written or -1. Also sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`buf`: data buffer

`nbytes`: number of bytes to try to read

`flags`: see man page for details; typically use 0

`destaddr`: IP address and port number of destination socket

`addrlen`: length of address structure = sizeof (struct `sockaddr_in`)

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 36

```
int recvfrom (int sockfd, char* buf, size_t nbytes,
             int flags, struct sockaddr* srcaddr,
             int* addrlen);
```

Read a datagram from a UDP socket. Returns number of bytes read (0 is valid) or -1. Also sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`buf`: data buffer

`nbytes`: number of bytes to try to read

`flags`: see man page for details; typically use 0

`srcaddr`: IP address and port number of sending socket (returned from call)

`addrlen`: length of address structure = pointer to int set to `sizeof (struct sockaddr_in)`

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 37

- tearing down a connection

```
int close (int sockfd);
```

Close a socket. Returns 0 on success, -1 and sets `errno` on failure.

```
int shutdown (int sockfd, int howto);
```

Force termination of communication across a socket in one or both directions. Returns 0 on success, -1 and sets `errno` on failure.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 38

```
int close (int sockfd);
```

Close a socket. Returns 0 on success, -1 and sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

Closes communication on socket in both directions. All data sent before close are delivered to other side (although this aspect can be overridden).

After `close`, `sockfd` is not valid for reading or writing.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 39

```
int shutdown (int sockfd, int howto);
```

Force termination of communication across a socket in one or both directions. Returns 0 on success, -1 and sets `errno` on failure.

`sockfd`: socket file descriptor (returned from `socket`)

`howto`: SHUT_RD to stop reading

SHUT_WR to stop writing

SHUT_RDWR to stop both

`shutdown` overrides the usual rules regarding duplicated sockets, in which TCP teardown does not occur until all copies have closed the socket.

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 40

Advanced Sockets

- managing multiple connections
 - fork/exec: multiple server processes
 - pthread_create: multi-threaded server process
 - (no calls): event-based server process
- detecting data arrival
 - select and poll functions
- synchronous vs. asynchronous connections
- other socket options

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 41

Example of Use

- taken from Beej's Guide to Network Programming (see the class home page)
- client-server example using TCP
- for each client
 - server forks new process to handle connection
 - sends "Hello, world"

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 42

server.c (from Beej)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#define PORT 3490 /* well-known port */
#define BACKLOG 10 /* how many pending
connections queue will hold */
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 43

server.c (from Beej)

```
main()
{
    int sockfd, new_fd;
    /* listen on sock_fd, new connection on new_fd */
    struct sockaddr_in my_addr; /* my address */
    struct sockaddr_in their_addr; /* connector addr */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0))
        == -1) {
        perror("socket");
        exit(1);
    }
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 44

server.c (from Beej)

```
my_addr.sin_family = AF_INET; /* host byte order */
my_addr.sin_port = htons(MYPORT);
/* short, network byte order */
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
/* automatically fill with my IP (w/o Beej's bug) */
bzero(&my_addr.sin_zero, 8);
/* zero the rest of the struct */

if (bind(sockfd, (struct sockaddr *)&my_addr,
        sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(1);
}
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 45

server.c (from Beej)

```
if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}
while(1) { /* main accept() loop */
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)
                        &their_addr, &sin_size)) == -1) {
        perror("accept");
        continue;
    }
    printf("server: got connection from %s\n",
        inet_ntoa(their_addr.sin_addr));
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 46

server.c (from Beej)

```
if (ifork()) { /* this is the child process */
    if (send(new_fd, "Hello, world!\n", 14, 0)
        == -1)
        perror("send");
    close(new_fd);
    exit(0);
}
close(new_fd); /* parent doesn't need this */

/* clean up all child processes */
while(waitpid(-1, NULL, WNOHANG) > 0);
}
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 47

client.c (from Beej)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define PORT 3490 /* well-known port */
#define MAXDATASIZE 100 /* max number of
bytes we can get at once */
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved. 08-01, slide 48

client.c (from Beej)

```
int main (int argc, char* argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent* he;
    struct sockaddr_in their_addr;
        /* connector's address information */
    if (argc != 2) {
        fprintf (stderr, "usage: client hostname\n");
        exit (1);
    }
    if ((he = gethostbyname (argv[1])) == NULL) {
        /* get the host info */
        perror ("gethostbyname");
        exit (1);
    }
}
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 49

- the `hostent` data structure (from `/usr/include/netdb.h`)
 - canonical domain name and aliases
 - list of addresses associated with machine
 - also address type and length information

```
struct hostent {
    char*    h_name;        /* official name of host */
    char**   h_aliases;    /* NULL-terminated alias list */
    int      h_addrtype;   /* address type (AF_INET) */
    int      h_length;     /* length of addresses (4B) */
    char**   h_addr_list;  /* NULL-terminated
                           address list */
#define h_addr h_addr_list[0]; /* backward-compatibility */
};
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 50

client.c (from Beej)

```
if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
    perror ("socket");
    exit (1);
}
their_addr.sin_family = AF_INET; /* interp'd by host */
their_addr.sin_port = htons (PORT);
their_addr.sin_addr = *((struct in_addr*)he->h_addr);
bzero (&(their_addr.sin_zero), 8);
        /* zero rest of struct */

if (connect (sockfd, (struct sockaddr*)&their_addr,
            sizeof (struct sockaddr)) == -1) {
    perror ("connect");
    exit (1);
}
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 51

client.c (from Beej)

```
if ((numbytes = recv (sockfd, buf, MAXDATASIZE, 0))
    == -1) {
    perror ("recv");
    exit (1);
}

buf[numbytes] = '\0';

printf ("Received: %s", buf);

close (sockfd);

return 0;
}
```

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 52

Summary in Notes

- TCP connection setup handshake annotated with socket API calls
- client-server interaction
 - TCP version
 - UDP version

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 53

Food for Thought

framing messages on a byte stream

- problem
 - pass logical messages using a TCP connection
 - `read` may return partial or multiple messages
 - how can receiver identify the end of a message?
- try to come up with two or three methods
- hints
 - string storage in C and Pascal
 - format strings with `printf`

© Copyright 1999,2000 Steven S. Lumetta. All rights reserved.

08-01, slide 54

