10. Network Interface

- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

OS Abstraction for Distributed I/O

Challenges

- Abstract multiple *layers* of multiple *networking protocol stacks*
- Cross-system synchronization and communication primitives
- Extend classical I/O primitives to distributed systems

10. Network Interface

- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Open Systems Interconnection (OSI)

Basic Reference Model

- Layer 7: Application layer — RPC, FTP, HTTP, NFS
- Layer 6: Presentation layer — XDR, SOAP XML, Java socket API
- *Layer 5: Session layer* — *TCP, DNS, DHCP*
- *Layer 4: Transport layer* — *TCP, UDP, RAW*
- *Layer 3: Network layer* — *IP*
- Layer 2: Data Link layer — Ethernet protocol
- Layer 1: Physical layer — Ethernet digital signal processing

OS Interface

- Abstract layers 3, 4 and 5 through special files: *sockets*

Socket Abstraction

What?

- Bidirectional communication channel across systems called *hosts*

Socket Abstraction

What?

- Bidirectional communication channel across systems called *hosts*

Networking Domains

- *INET*: Internet Protocol (IP)
- *UNIX*: efficient host-local communication
- And many others (IPv6, X.25, etc.)

- `$ man 7 socket`
- `$ man 7 ip` or `$ man 7 ipv6` (for INET sockets)
- `$ man 7 unix`

Socket Abstraction

What?

- Bidirectional communication channel across systems called *hosts*

Socket Types

- STREAM: *connected* FIFO streams, reliable (error detection and replay), without message boundaries, much like `pipe`s across hosts
- DGRAM: *connection-less*, unreliable (duplication, reorder, loss) exchange of messages of fixed length (datagrams)
- RAW: direct access to the raw protocol (not for UNIX sockets)
- Mechanism to *address* remote sockets depends on the socket type
  - `$ man 7 tcp` Transmission Control Protocol (TCP): for STREAM sockets
  - `$ man 7 udp` User Datagram Protocol (UDP): for DGRAM sockets
  - `$ man 7 raw` for RAW sockets
- Two classes of INET sockets
  - IPv4: 32-bit address and 16-bit port
  - IPv6: 128-bit address and 16-bit port

10. Network Interface

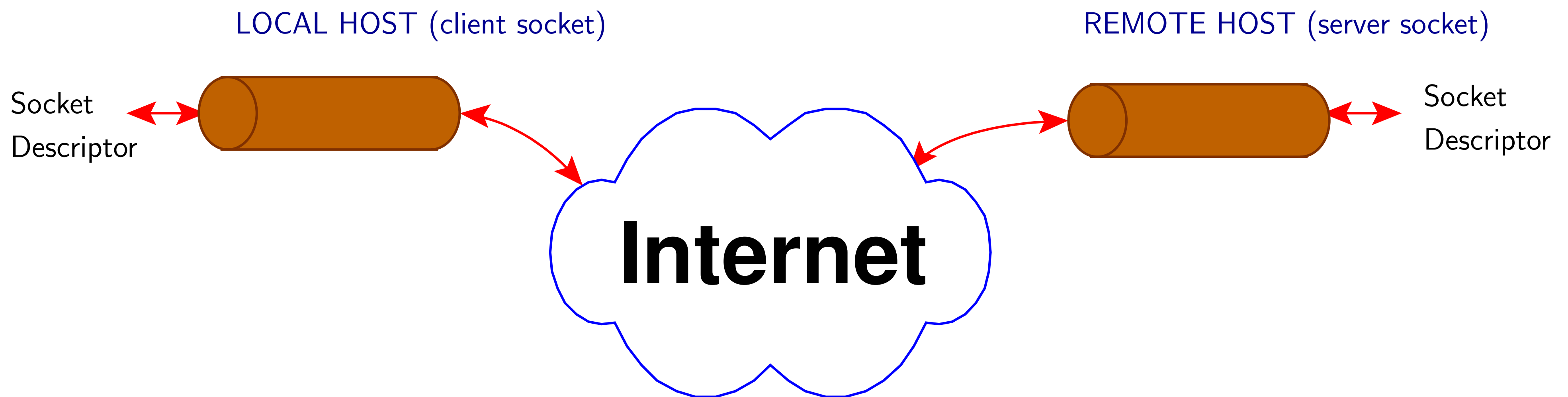
- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Scenarios for Socket-to-Socket Connection

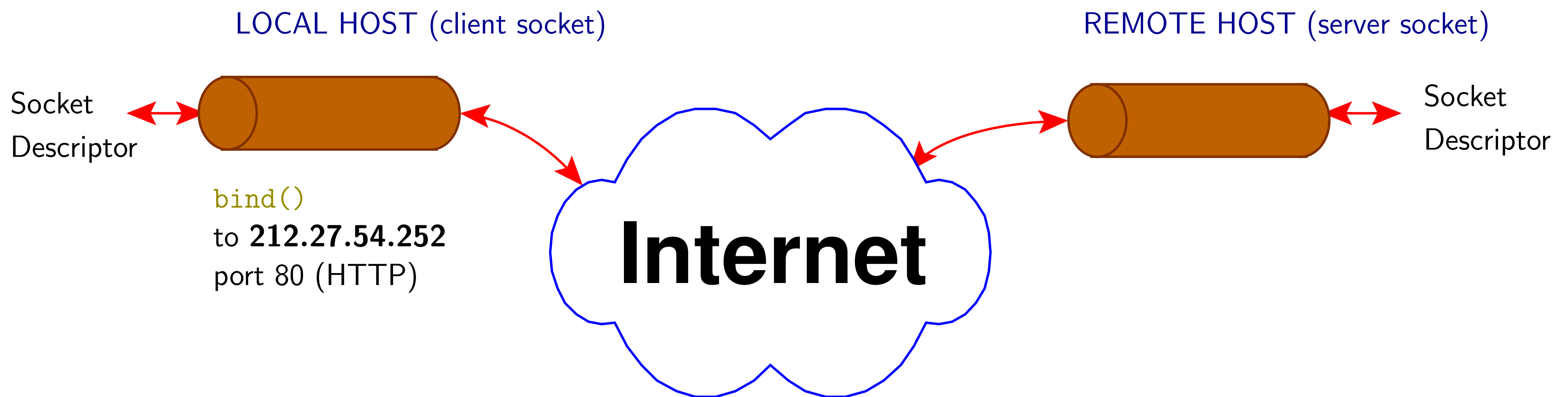
Direct Communication Scenario

- Create a socket with `socket()`
- Bind to a local address with `bind()`
- In the remote host, go through the first **2** steps exchanging the roles of local and remote addresses
- Only DGRAM (UDP) sockets can be operated that way
- Note: port numbers only provide a partial support for a rendez-vous protocol: unlike named FIFOs, no synchronization is enforced

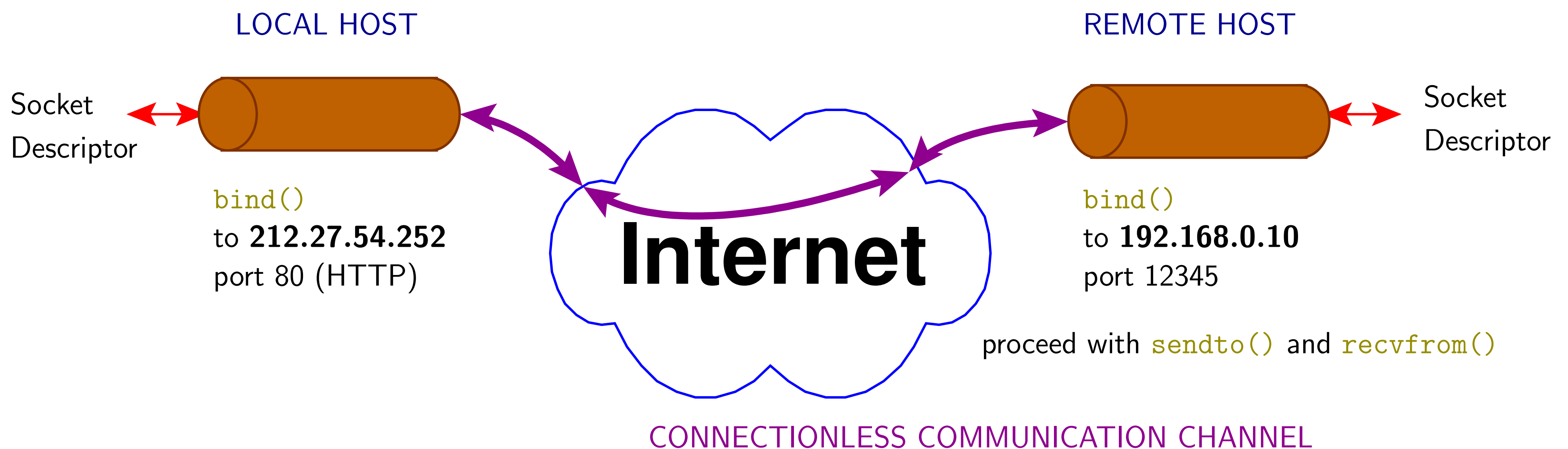
Establishing a Connectionless Channel



Establishing a Connectionless Channel



Establishing a Connectionless Channel



10. Network Interface

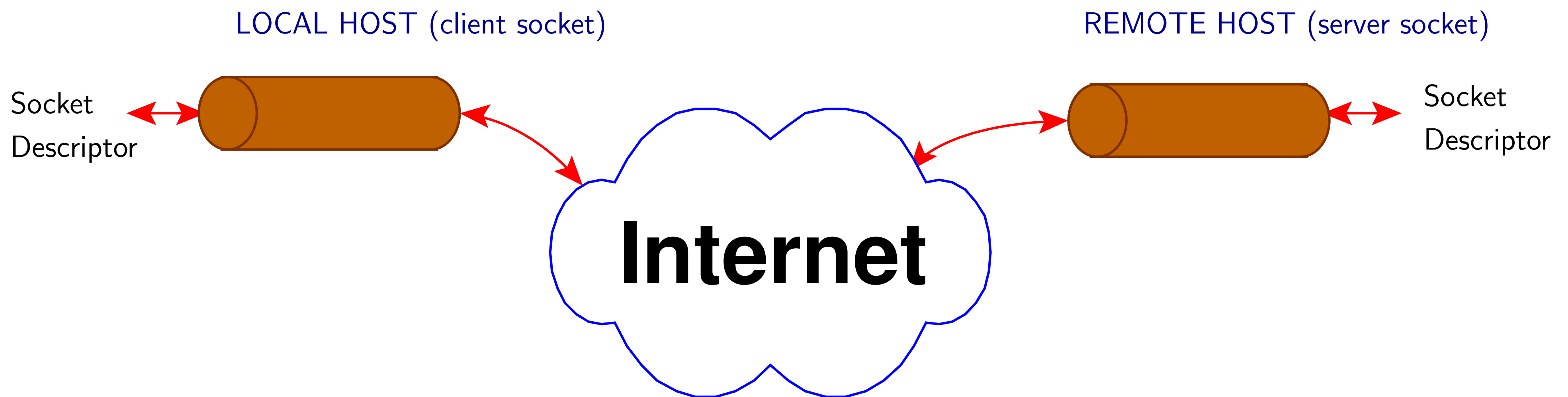
- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Scenarios for Socket-to-Socket Connection

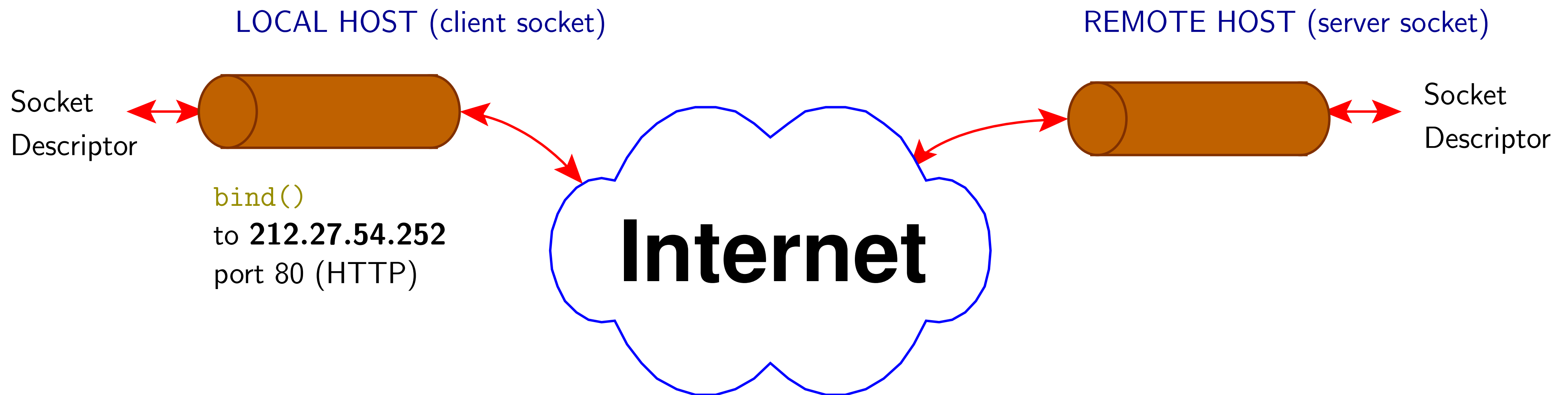
TCP Abstraction: Creation of a Private Channel

- Create a socket with `socket()`
- Bind to a local address with `bind()`
- Call `listen()` to tell the socket that new connections shall be accepted
- Call `accept()` to wait for an incoming connection, returning a new socket associated with a private channel (or "session") for this connection
- In the remote host, go through the first two steps exchanging the roles of local and remote addresses, and calling `connect()` instead of `bind()`
- The original pair of sockets can be reused to create more private channels
- Reading or writing from a "yet unconnected" connection-based socket raises `SIGPIPE` (like writing to a pipe without readers)

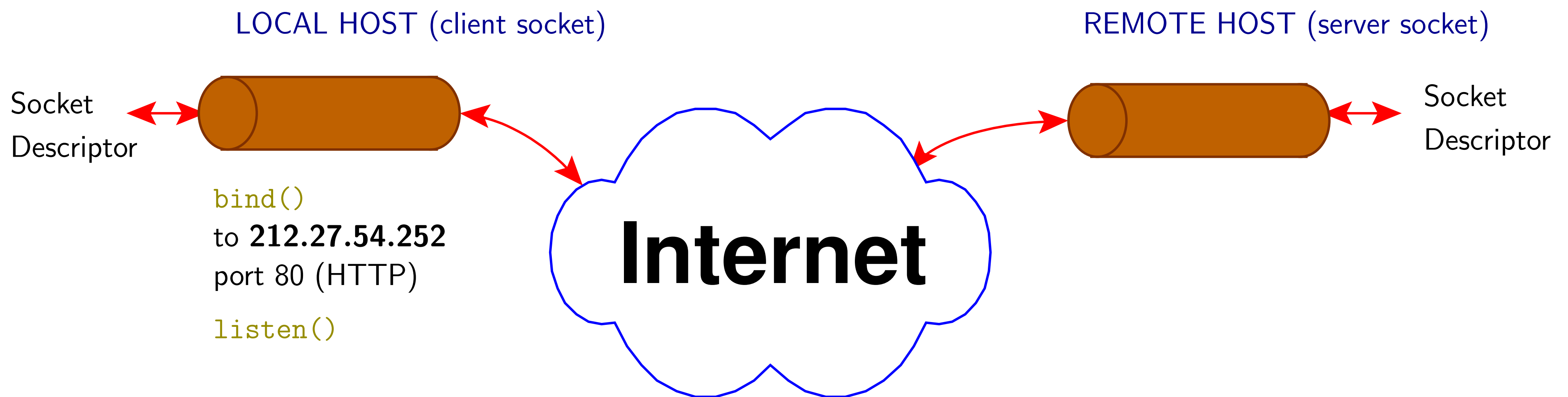
Establishing a Connection-Based Channel



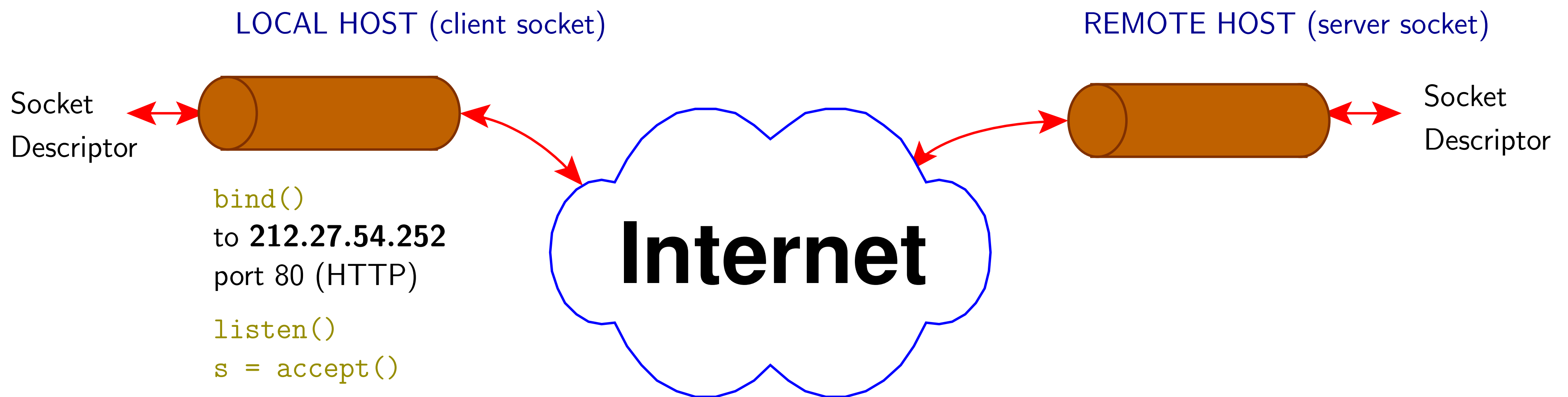
Establishing a Connection-Based Channel



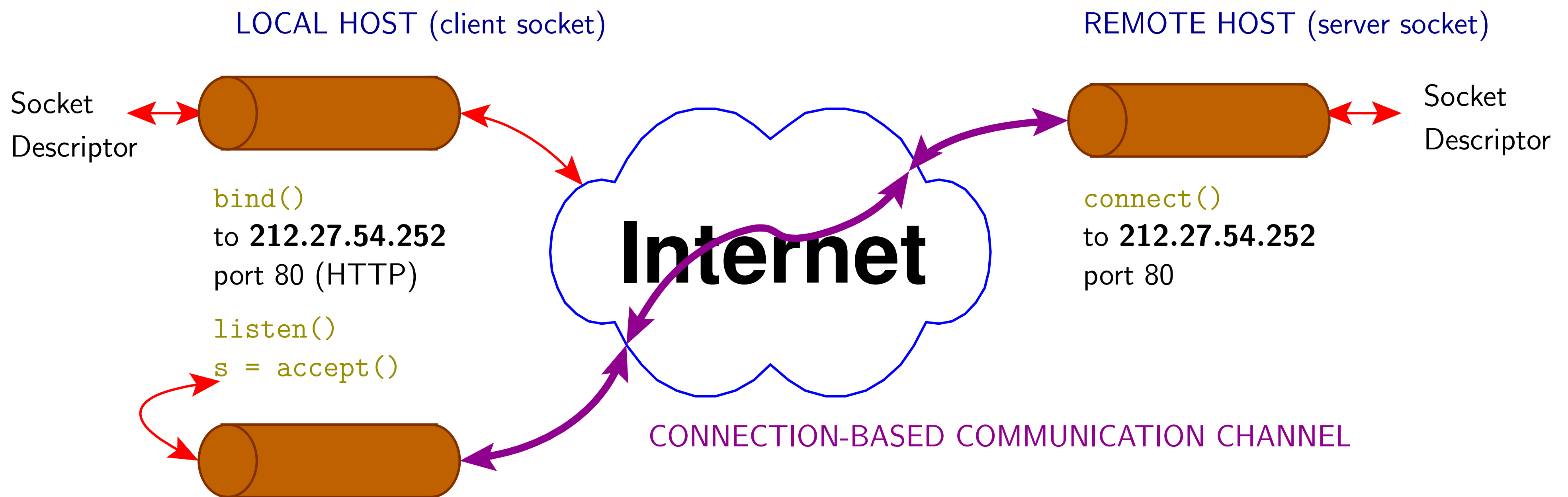
Establishing a Connection-Based Channel



Establishing a Connection-Based Channel



Establishing a Connection-Based Channel



10. Network Interface

- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Establishing a Socket for Incoming Connections

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

int establish(unsigned short portnum) {
  int s; char myname[MAXHOSTNAME+1]; struct sockaddr_in sa;
  memset(&sa, 0, sizeof(struct sockaddr_in));    // Clear our address
  gethostname(myname, MAXHOSTNAME);              // Who are we?
  struct hostent *hp = gethostbyname(myname);    // Get our address info
  if (hp == NULL) return -1;                     // We do not exist!?
  memcpy(&sa.sin_addr.s_addr, hp->h_addr, hp->h_length); // Host address
  sa.sin_family = hp->h_addrtype;                // And address type
  sa.sin_port = htons(portnum);                  // And our big-endian port
  s = socket(AF_INET, SOCK_STREAM, 0))           // Create socket
  if (s < 0) return -1;                          // Socket creation failed
  int my_true = 1; // Immediate reuse of the local port after closing socket
  int so_r = setsockopt(*s, SOL_SOCKET, SO_REUSEADDR, &my_true, sizeof(my_true));
  // Wait 10 seconds after closing socket for reliable transmission
  struct linger my_linger = { .l_onoff = 1, .l_linger = 10 };
  so_r |= setsockopt(*s, SOL_SOCKET, SO_LINGER, &my_linger, sizeof(my_linger));
  if (so_r) { perror ("setsockopt"); close (*s); return -1; }
  if (bind(s, &sa, sizeof(sa), 0) < 0)           // Bind address to socket
    { close(s); return -1; }
  return s;
}
```

Waiting for Incoming Connections

```
int wait_for_connections(int s) {      // Socket created with establish()
  struct sockaddr_in sa;               // Address of socket
  int i = sizeof (sa);                 // Size of address
  int t;                               // Socket of connection

  listen(s, 3);                        // Max # of queued connections
  if ((t = accept(s, &sa, &i)) < 0)    // Accept connection if there is one
    return -1;
  return t;
}
```

Opening an Outgoing Connection

```
int call_socket(char *hostname, unsigned short portnum) {
  struct sockaddr_in sa;
  struct hostent *hp;
  int a, s;

  if ((hp = gethostbyname(hostname)) == NULL) {          // Do we know
    errno = ECONNREFUSED;                                // The host's address?
    return -1;                                           // No
  }

  memset(&sa, 0, sizeof(sa));
  memcpy(&sa.sin_addr, hp->h_addr, hp->h_length);        // Set address
  sa.sin_family = hp->h_addrtype;                        // And type
  sa.sin_port = htons(portnum);                          // And big-endian port

  if ((s = socket(hp->h_addrtype, SOCK_STREAM, 0)) < 0)  // Get socket
    return -1;
  if (connect(s, &sa, sizeof(sa)) < 0) {                 // Connect
    close(s); return -1;
  }
  return s;
}
```

Communicating Through a Pair of Sockets

Connected Socket I/O

- System calls `read()` and `write()` work as usual on *connected* sockets (otherwise raise `SIGPIPE`)
- System calls `recv()` and `send()` refine the semantics of `read()` and `write()` with additional flags to control socket-specific I/O (out-of-band, message boundaries, etc.)
- System call `shutdown(int sockfd, int how)` causes all or part of a full-duplex TCP connection to shut down, according to the value of `how`: `SHUT_RD`, `SHUT_WR`, `SHUT_RDWR` respectively disallow receptions, transmissions, and both receptions and transmissions; this call is important to avoid dead-locks or to simulate end-of-file through TCP connections (analog to selectively closing pipe descriptors)

Connection-Less Socket I/O

- A *single* DGRAM (UDP) socket can be used to communicate
- System calls: `recvfrom()` and `sendto()`

10. Network Interface

- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Application: Threaded Server Model

Dynamic Thread Creation

1. A *main thread* `listen`s for a `connect`ion request on a *predefined port*
2. After `accept`ing the request, the server creates a thread to handle the request and resumes `listen`ing for another request
3. The thread `detach`es itself, performs the request, closes the `socket` in response to the client's closing and returns

→ the thread function takes the socket returned from the `accept()` system call as a parameter

Application: Threaded Server Model

Dynamic Thread Creation

1. A *main thread* `listen`s for a `connect`ion request on a *predefined port*
2. After `accept`ing the request, the server creates a thread to handle the request and resumes `listen`ing for another request
3. The thread `detach`es itself, performs the request, closes the `socket` in response to the client's closing and returns

→ the thread function takes the socket returned from the `accept()` system call as a parameter

Worker Pool

1. A *main thread* plays the role of a *producer*
2. A *bounded* number of *worker threads* play the role of *consumers*
3. The main thread `listen`s for `connect`ion requests and asks the workers to process them, e.g., enqueuing a task/coroutine on the worker's work list

Application: Threaded Server Model

Dynamic Thread Creation

1. A *main thread* `listen`s for a `connect`ion request on a *predefined port*
2. After `accept`ing the request, the server creates a thread to handle the request and resumes `listen`ing for another request
3. The thread `detach`es itself, performs the request, closes the `socket` in response to the client's closing and returns

→ the thread function takes the socket returned from the `accept()` system call as a parameter

Worker Pool

1. A *main thread* plays the role of a *producer*
2. A *bounded* number of *worker threads* play the role of *consumers*
3. The main thread `listen`s for `connect`ion requests and asks the workers to process them, e.g., enqueuing a task/coroutine on the worker's work list

More Information and Optimizations: `http://www.kegel.com/c10k.html`

10. Network Interface

- Principles
- Connectionless Communications
- Connection-Based Communications
- Programmer Interface
- Threaded Server Model
- Distributed Systems

Distributed Systems and Protocols

RFC: Request for Comments

- RFC-Editor: `http://www.rfc-editor.org/rfc.html`
- IETF: Internet Engineering Task Force
- IANA: Internet Assigned Numbers Authority

Open Systems Interconnection (OSI)

Basic Reference Model

- *Layer 7: Application layer* — *RPC, FTP, HTTP, NFS*
- *Layer 6: Presentation layer* — *XDR, SOAP XML, Java socket API*
- *Layer 5: Session layer* — *TCP, DNS, DHCP*
- Layer 4: Transport layer — TCP, UDP, RAW
- Layer 3: Network layer — IP
- Layer 2: Data Link layer — Ethernet protocol
- Layer 1: Physical layer — Ethernet digital signal processing

Interface

- Abstract layers 4, 5 and 6 through dedicated protocols
- Virtualize distributed system resources over these protocols
  Cloud services: storage and computation resources, applications

More Information on Distributed Systems

- Look for information on each individual protocol
- Overview of distributed Computing:
  http://en.wikipedia.org/wiki/Distributed_computing
- Distributed Operating Systems (Andrew Tanenbaum):
  http://www.cs.vu.nl/pub/amoeba/amoeba.html
- Peer to peer: http://en.wikipedia.org/wiki/Peer-to-peer

→ See INF570 course