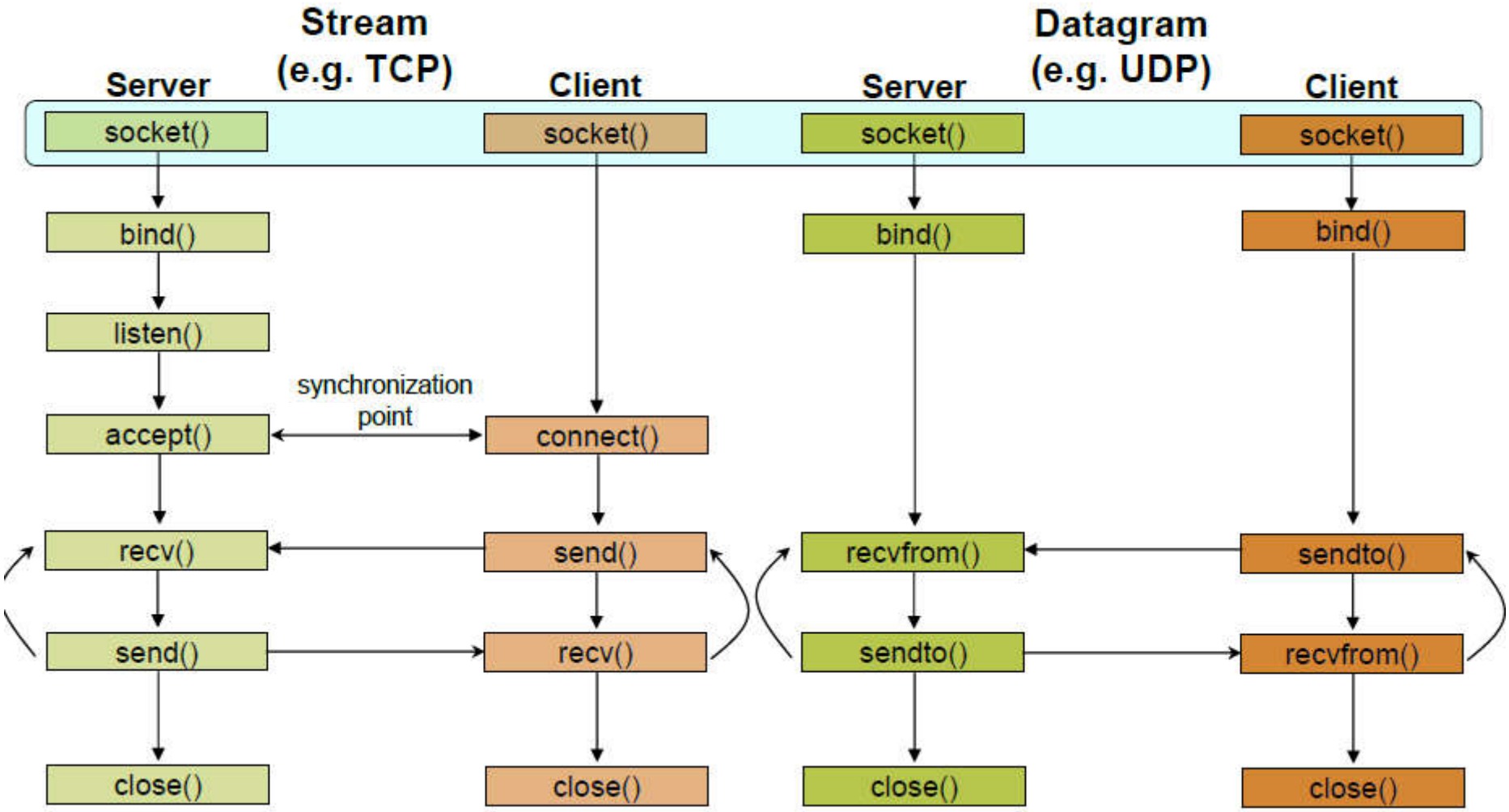
Socket programming basics

Client-Server communication

- **Server**
  - passively waits for and responds to clients
  - **passive** socket
- **Client**
  - initiates the communication
  - must know the address and the port of the server
  - **active** socket

Sockets - Procedures

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection


Stream
(e.g. TCP)
Server
Client
socket()
socket()
bind()
listen()
accept()
synchronization
point
connect()
recv()
send()
send()
recv()
close()
close()
Datagram
(e.g. UDP)
Server
Client
socket()
socket()
bind()
bind()
recvfrom()
sendto()
sendto()
recvfrom()
close()
close()

Using UDP Sockets

- User Datagram Protocol (UDP): end-to-end service different from TCP
- UDP performs only two functions:
    - It adds another layer of addressing (ports) to that of IP, and
    - it detects data corruption that may occur in transit and discards any corrupted datagrams.
- UDP sockets do not have to be connected before being used.
    - TCP is analogous to telephone communication
    - UDP is analogous to communicating by mail:
- do not have to "connect" before send a package,
- do have to specify the destination address for each one.
- In receiving, a UDP socket is like a mailbox into packages from many different sources can be placed.
- As soon as it is created, a UDP socket can be used to send/receive messages to/from any address and to/from many *different addresses in succession.*

Using UDP Sockets

- To allow the destination address to be specified for each message, the sockets API provides a different sending routine that is generally used with UDP sockets: sendto().
- Similarly, the recvfrom() routine returns the source address of each received message in addition to the message itself.

Using UDP Sockets

int sendto (int *socket, const void *msg, unsigned int msgLength, int flags,* struct sockaddr **destAddr, unsigned int addrLen**)*

int recvfrom (int *socket, void *msg, unsigned int msgLength, int flags,* struct sockaddr **srcAddr, unsigned int *addrLen*)*

- The first four parameters to *sendto()* are the same as those for *send().*
- The two additional parameters specify the message's destination. They will invariably be a pointer to a
  - ***struct sockaddr_in*,** and
  - ***sizeof(struct sockaddr_in*),** respectively**.**

- ***recvfrom()*** takes the same parameters 3 as recv().
  - *addrLen* is an **inout** parameter in *recvfrom():*
    - On input it specifies the size of the address buffer *srcAddr*;
    - On output it specifies the size of the address that was copied into the buffer.

Using UDP Sockets

Two typical errors
(1)passing an integer value instead of a pointer to
an integer for *addrLen*
(2)forgetting to initialize the pointed-to length
variable to contain sizeof(struetc sockaddr_in).

UDP Client

- UDP echo client- similar to TCP echo
- except that it does not call connect(), and
- it only needs to do a single receive, because UDP sockets preserve message boundaries, unlike TCP's byte-stream service.
- UDP client only communicates with a UDP server.
- Many systems include a UDP echo server for debugging and testing purposes; the server simply echoes whatever messages it receives back to wherever they came from.
- echo client performs the following steps:

  (1) it sends the echo string to the server,
  (2) it receives the echo, and
  (3) it shuts down the program.

UDP Server

UDP version of the echo server

• It loops forever, receiving a message and then sending the same message back to wherever it came from
• the server only receives and sends back the first 255 characters of the message;
• any excess is silently discarded by the sockets implementation.

• UDP client is example for UDP socket calls
• not suitable for production - if a message is lost going to or from the server, the call to recvfrom() blocks forever, and the program does not terminate.
• Clients generally deal with this problem through the use of *timeouts*

Local Broadcast and Directed Broadcast

- A local broadcast address (255.255.255.255) sends the message to every host on the same broadcast network.
- Local broadcast messages are never forwarded by routers.

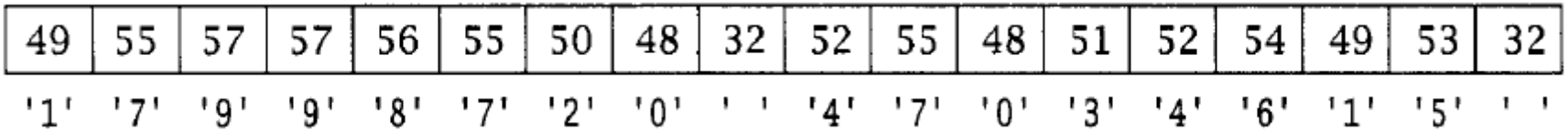
- Directed broadcast allows broadcast to all hostson a specific network.
- IP addresses have two parts: the network and the host identifier.
- If the network identifier is X, a directed broadcast address for that network is an IP address with the high-order bits set to X and the remaining bits set to 1 (i.e., X111 . . . 111).
- For example, the directed broadcast address for a network with network identifier 169.125 (first two bytes) is 169.125.255.255.

Directed Broadcast

With subnetting, we consider the subnet identifier part of the network identifier, so the definition of a directed broadcast address for a subnet is the same.
For example, if a network with subnet mask 255.255.255.0 has a subnet 169.125.134, the directed
broadcast address for that subnet is 169.125.134.255.

Enkodovanje podataka

17,998,720 i 47,034,615,



sprintf(msgBuffer, "%d %d ", x, y) ;
send(s, msgBuffer, strlen(msgBuffer), 0);

```
#define BUFSIZE 132
char msgBuf[BUFSIZE];
.
.
.
sprintf(msgBuffer, "Nd Nd ", deposits, withdrawals);
send(s, msgBuffer, BUFSIZE, 0);
```

Direktno slanje

```
int x;
int y;

send(s, &x, sizeof(x), 0);
send(s, &y, sizeof(y), 0);
```

With UDP this does not work because it results in two separate datagrams.

Redosled kodiranja

17,998,720 i 47,034,615,

1	18	163	128	2	205	176	247
128	163	18	1	247	176	205	2

Standard routines to convert two- and four-byte integers from native to network byte order

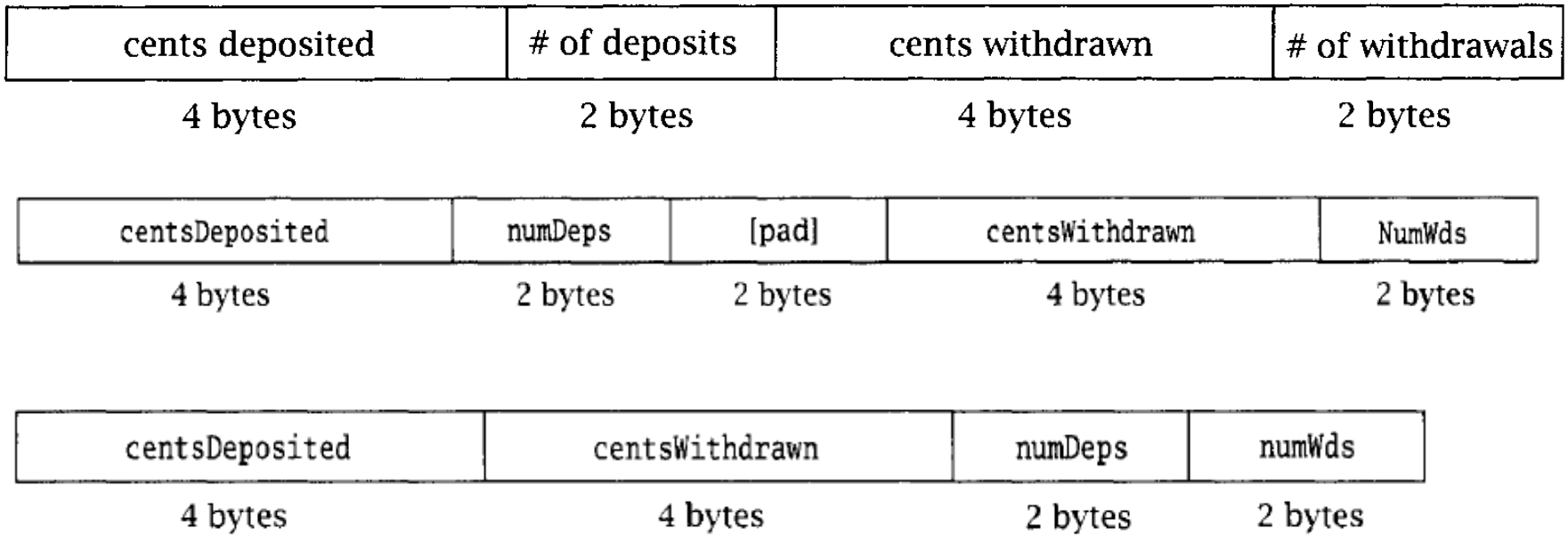
long int htonl(long int *hostLong)*

short int htons(short int *hostShort)*

long int ntohl(long int *netLong)*

short int ntohs(short int *netShort)*

Alignment and Padding



- ASCII poruke
- Framing and Parsing