UDP Protocol Specification

CPSC 441 - TUTORIAL 5

WINTER 2020

UNIVERSITY OF CALGARY

*Modified from slides by Reza Gholizadeh

What is UDP?

User **D**atagram **P**rotocol

A transport layer protocol – like TCP

Features

UDP is a minimal transport layer protocol.

Unreliable and connection-less:

- UDP provides no guarantees to the upper layer protocol for message delivery
- The UDP protocol retains no state of UDP messages once sent
- Messages may be delivered out of order.

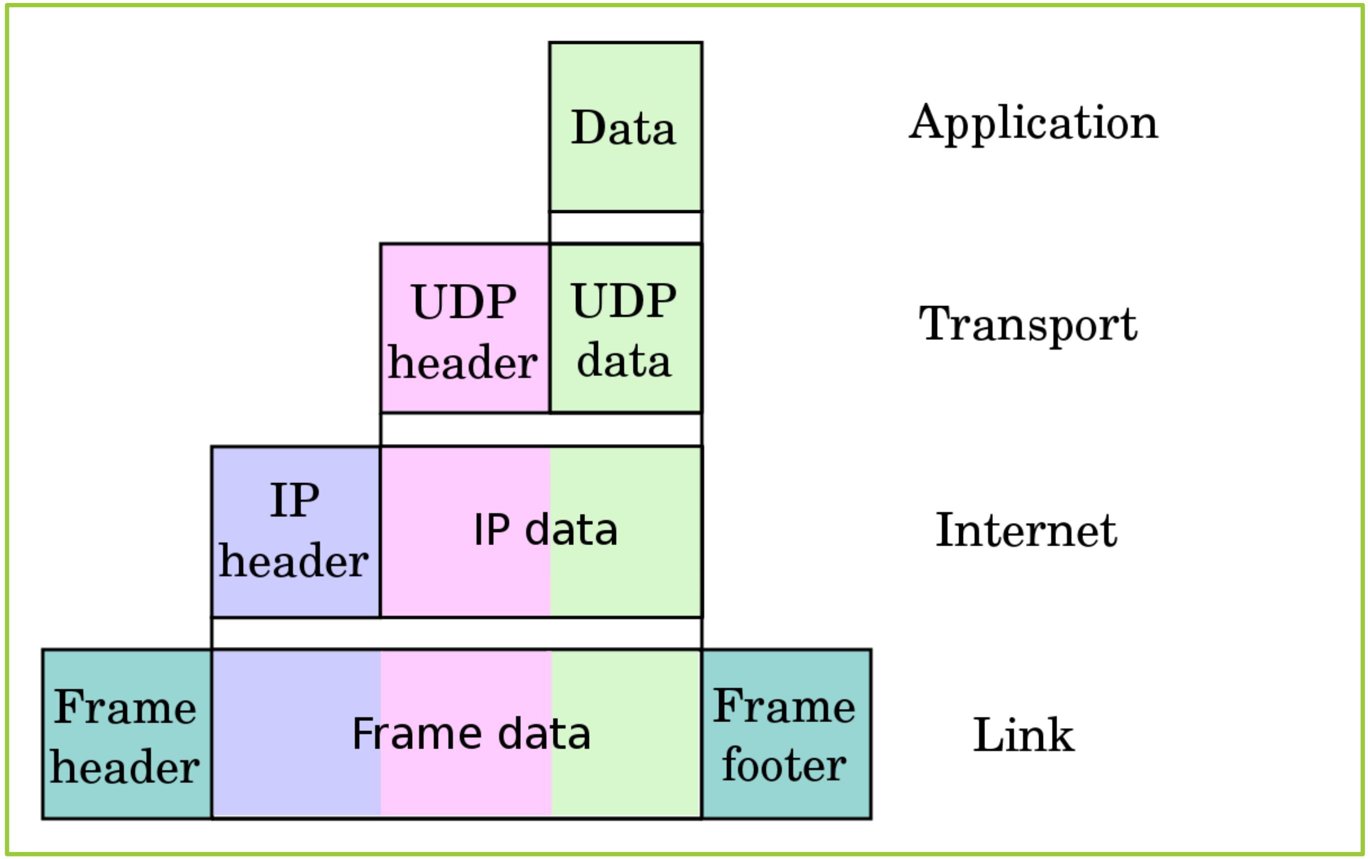
Provide integrity verification

- (via checksum) of the header and payload.

UDP provides application multiplexing (via port numbers)

Encapsulation and Decapsulation

Encapsulation



Decapsulation

From: https://en.wikipedia.org/wiki/Encapsulation_(networking)

UDP Header

The UDP header consists of 4 fields, each of which is 2 bytes.

In IPv4, source port and checksum are optional. In IPv6 only the source port is optional.

offset (bits)	0-15	16-31
0	Source Port Number	Destination Port Number
32	Length	Checksum
64+	Data	

Source port number

- Identifies the sender's port , should be assumed to be the port to reply to if needed.
- If not used, then it should be zero.

Destination port number

- Identifies the receiver's port and is required

Length

- Specifies the length in bytes; considers the entire datagram: header and data
- The minimum length is 8 bytes = the length of the header.
- The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram

UDP Header

The UDP header consists of 4 fields, each of which is 2 bytes.

In IPv4, source port and checksum are optional. In IPv6 only the source port is optional.

offset (bits)	0-15	16-31
0	Source Port Number	Destination Port Number
32	Length	Checksum
64+	Data	

Checksum

- The checksum field is used for error-checking of the header and data. This field is mandatory for IPv6.
- If no checksum is generated by the transmitter, the field should be set to all-zeros.
- UDP uses pseudo header to define the checksum. It is calculated over the combination of pseudo header and UDP message.
- The pseudo header contains: the IP Source Address field, the IP Destination Address field, the IP Protocol field and the UDP Length field.

TCP	UDP
Reliable	Unreliable
Connection-Oriented	Connectionless
Segment Retransmission and Flow Control Through Windowing	No Windowing or Retransmission
Segment Sequencing	No Sequencing
Acknowledge Segments	No Acknowledgement

TCP vs. UDP

UDP advantages

UDP’s header is much smaller than TCP’s. The header is being applied to every segments, and adds up!

Generating a UDP header has much simpler processing steps.

UDP has no connection setup overhead, while TCP requires a 3-way handshake.

UDP Use-Cases

UDP is widely used and recommended for cases where:

- Speed Is more important than reliability. An application values timely delivery over reliable delivery
- Data exchanges are short and the order of reception of datagram does not matter
- A best-effort for delivery is enough
- Applications require multicast or broadcast transmissions, not supported by TCP.

Who Uses UDP?

Domain Name System (DNS)

Simple Network Management Protocol (SNMP)

Dynamic Host Configuration Protocol (DHCP)

Routing Information Protocol (RIP)

Checksum

“Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.”

[RFC 768]

Checksum

Checksum is calculated for UDP header and data

- IP header also has a checksum, but it doesn’t cover data.

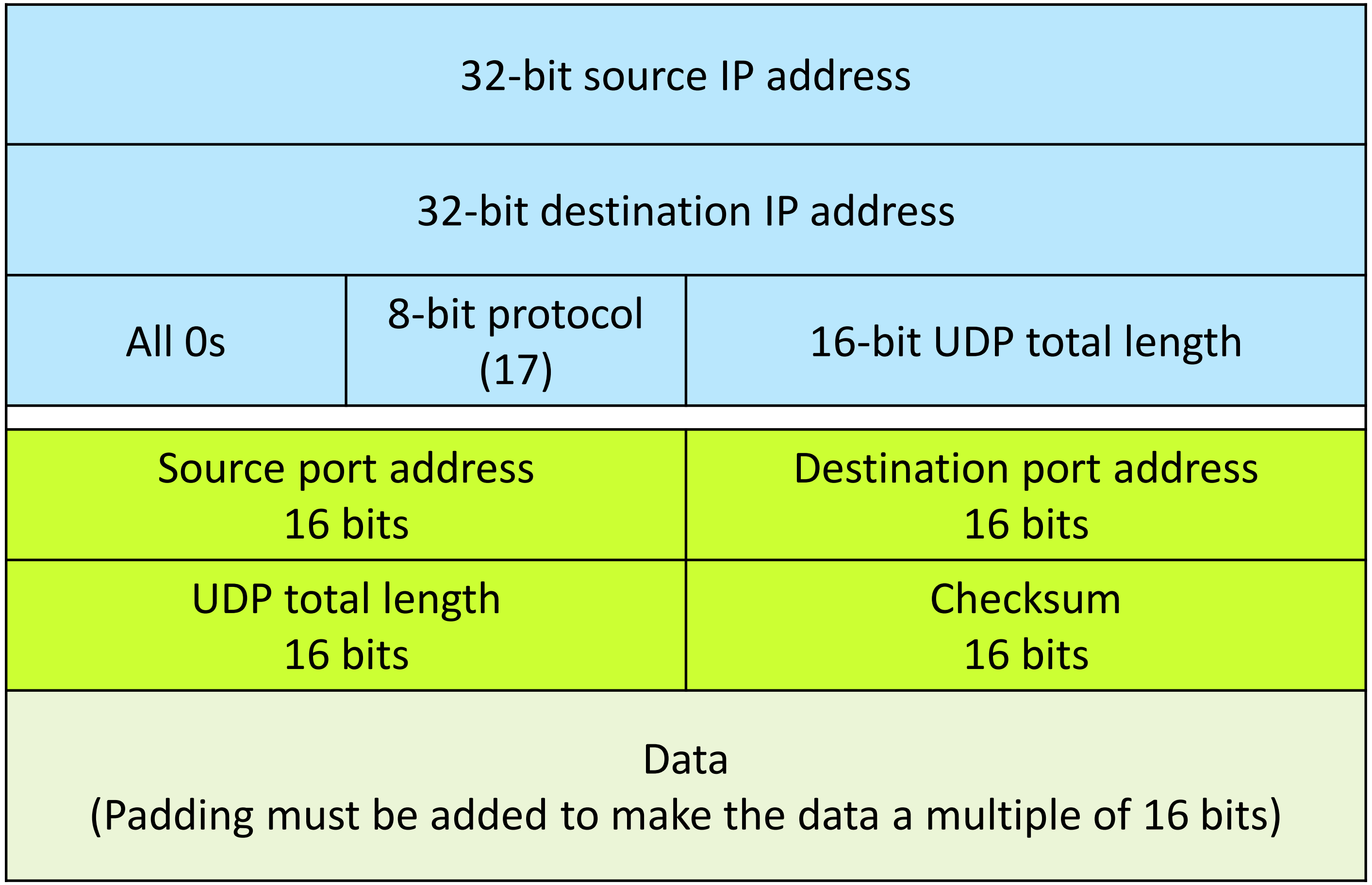
UDP checksum test is performed only at the sender and receiver end stations

- The IP checksum test is performed in every intermediate node (router).

UDP check sum is performed over a pseudo header.

- In addition to UDP header and data + the source and the destination IP address
- This prevent misrouting: in case the destination IP address in IP header was corrupted, and it was not discovered by the IP checksum test, the UDP datagram would arrive to the wrong IP address. UDP can detect this and silently drop the datagram.

Pseudo-Header

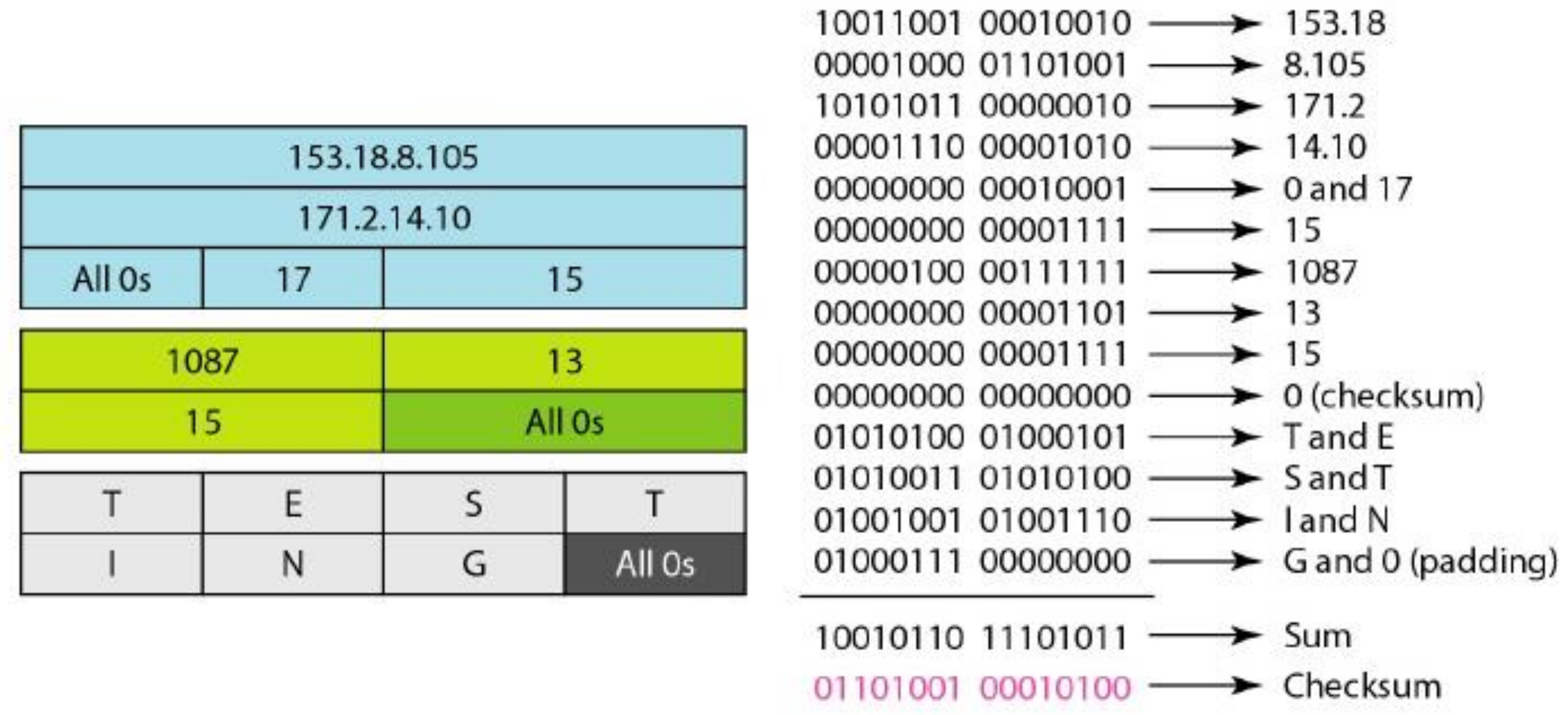


32-bit source IP address		
32-bit destination IP address		
All 0s	8-bit protocol (17)	16-bit UDP total length

Source port address 16 bits	Destination port address 16 bits
UDP total length 16 bits	Checksum 16 bits
Data (Padding must be added to make the data a multiple of 16 bits)	

Ref:
https://www.brainkart.com/article/User-Datagram-Protocol-(UDP)_13485/

Sample Checksum Calculation



153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		All 0s	
T	E	S	T
I	N	G	All 0s

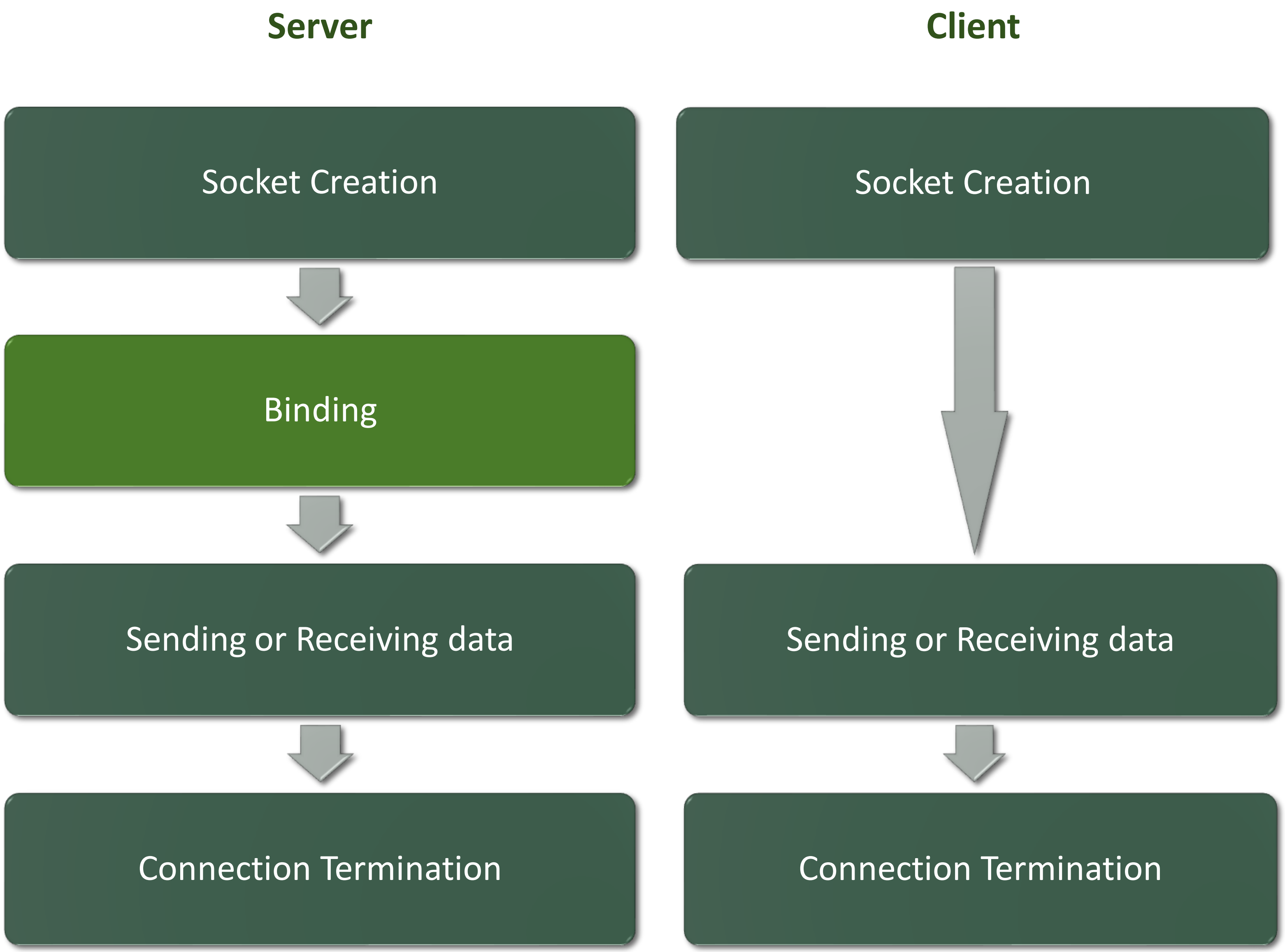
10011001 00010010	153.18
00001000 01101001	8.105
10101011 00000010	171.2
00001110 00001010	14.10
00000000 00010001	0 and 17
00000000 00001111	15
00000100 00111111	1087
00000000 00001101	13
00000000 00001111	15
00000000 00000000	0 (checksum)
01010100 01000101	T and E
01010011 01010100	S and T
01001001 01001110	I and N
01000111 00000000	G and 0 (padding)
10010110 11101011	Sum
01101001 00010100	Checksum

Sample from: https://www.brainkart.com/article/User-Datagram-Protocol-(UDP)_13485/

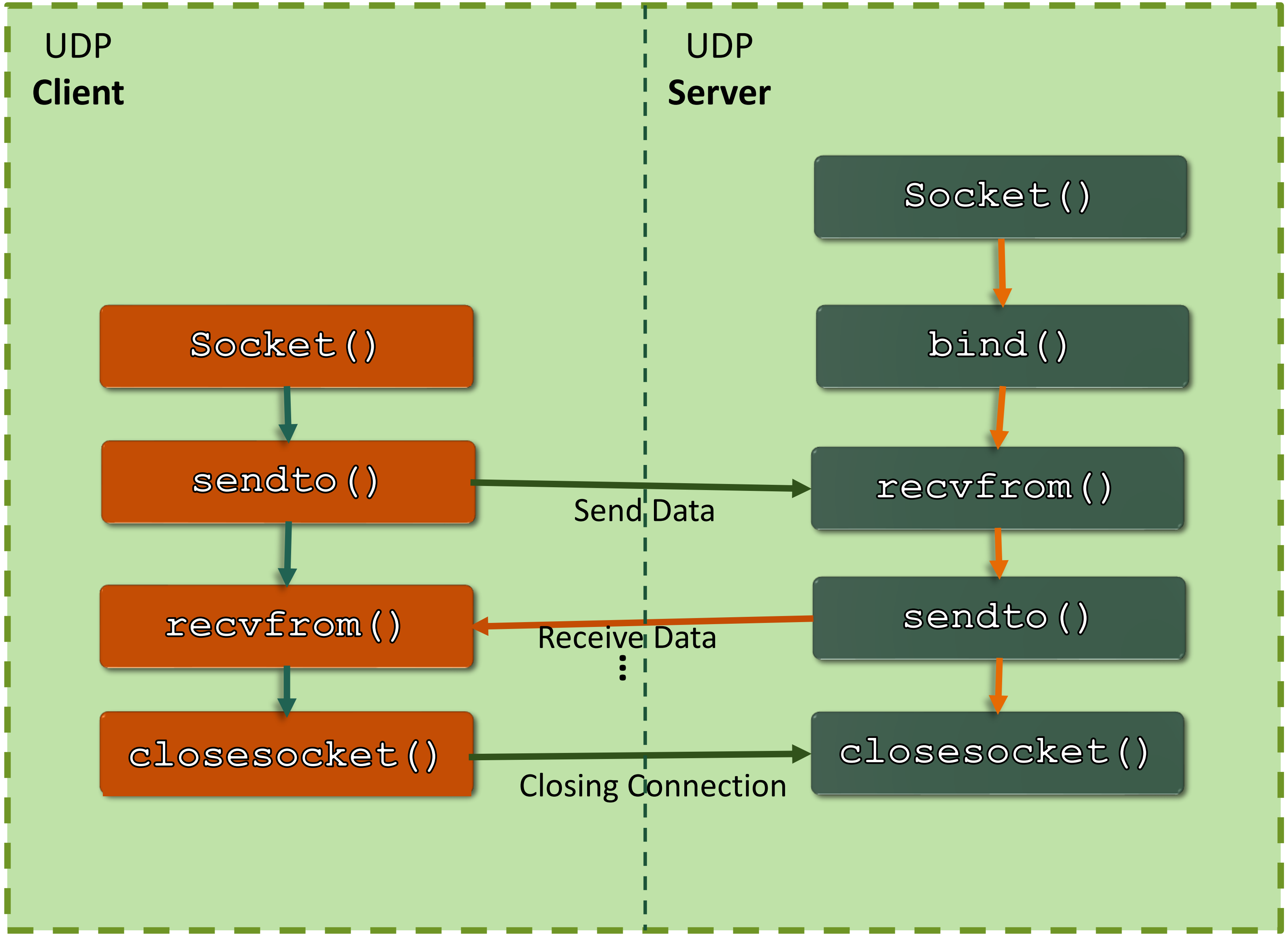
Socket Programming with UDP

Two sides of socket programming:

- Server side
- Client side



Server and Client Interactions



UDP **Client**

UDP **Server**

Socket Creation

`socket(domain, type, protocol)`

- **`domain`**: communication domain - integer
- **`type`**: type of connection – SOCK_DGRAM for UDP
- **`protocol`**: for using UDP over IP, should be set to IPPROTO_UDP

```
int mySocket;
mySocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if(mySocket == -1){
    printf(“Could not setup a socket”);
}
```

Binding

bind(socketid, &addrport, size)

- **sockid**: socket descriptor - integer
- **type**: the (IP) address and port of the machine – **struct sockaddr**
- **size**: the size of the **addrport** structure.

```
int status;
struct sockaddr_in ip_server;
struct sockaddr *server;
memset ((char*) &ip_server, 0, sizeof(ip_server));
ip_server.sin_family = AF_INET;
ip_server.sin_port = htons(PORT);
ip_server.sin_addr.s_addr = htonl(INADDR_ANY);
server = (struct sockaddr *) &ip_server;
status = bind(mySocket, server,  sizeof(ip_server));
if(status == -1){
    printf(“Could not bind to port\n”);
    return -1;
}
```

Sending

```
sendto(int socketid,
const void *sendbuf, int
sendlen, int flags,
const struct sockaddr
*to, int tolen)
```

- **sockid**: socket descriptor - integer
- **sendbuf**: buffer containing the data to be transmitted
- **Sendlen**: size in bytes of the data in the buffer
- **flags**: indicator specifying the way in which the call is made
- **to**: the address of the target - **struct sockaddr**
- **tolen**: the size of the **addrport** structure

```
struct sockaddr *client;
struct sockaddr_in ip_client;
client = (struct sockaddr *) &ip_client;
int num_bytes = sendto(mySocket, SendBuff,
strlen(SendBuff), 0, client, sizeof(client));
if(num_bytes == -1)
    printf(“Unsuccessful send\n”)
else
    printf(“number of sent bytes = %d\n”,num_bytes);
```

UNIVERSITY OF CALGARY

Receiving

```
recvfrom(int socketid,
const void *recvbuf, int
recvlen, int flags,
const struct sockaddr
*from, int tolen)
```

- **`sockid`**: socket descriptor - integer
- **`recvbuf`**: buffer containing the receiving data
- **`recvlen`**: size in bytes of the data in the buffer
- **`flags`**: indicator specifying the way in which the call is made
- **`to`**: the address of the target - **`struct sockaddr`**
- **`tolen`**: the size of the **`addrport`** structure

```
if ( readbytes = recvfrom(mySocket, messagein,
   strlen(messagein), 0, client, sizeof(client))<0)
 {
     printf(“Read error\n”);
     return -1;
 }
```

References

http://en.wikipedia.org/wiki/User_Datagram_Protocol

https://beej.us/guide/bgnet/html/

http://ipv6.com/articles/general/User-Datagram-Protocol.htm

http://msdn.microsoft.com/en-us/library/ms881658.aspx

https://www.brainkart.com/article/User-Datagram-Protocol-(UDP)_13485/