CPSC 317
COMPUTER NETWORKING

2023W2: Transport – Day 1 – Introduction and UDP

READING

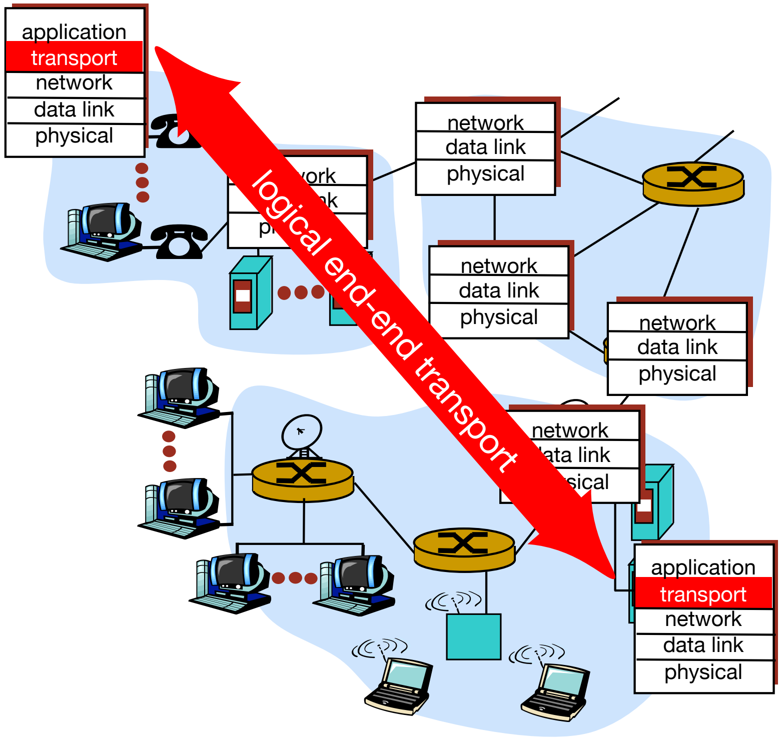
- Reading: 3.1, 3.2, 3.3

LEARNING GOALS

- Explain the need and main purpose of the transport layer
- Define multiplexing at the transport level (i.e., ports)
- Understand the types of services that transport can support
- Compare and contrast the important services provided by UDP and TCP
- Identify applications that (can) make use of TCP (UDP) and explain why
- Explain the purpose of the fields of the UDP header
- Use UDP sockets in Java

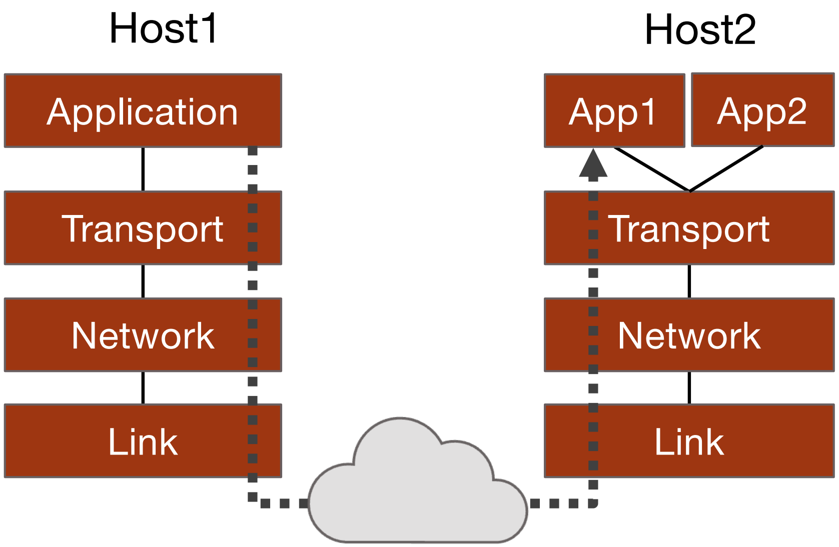
TRANSPORT PROTOCOL PURPOSES

- Provide *logical communication* between application processes running on different hosts
- Multiplexing of communication to different applications on end hosts
- Provide services to applications



LOGICAL END-TO-END COMMUNICATION

- *Network layer* provides logical communication between hosts (terminated at the interface)
- *Transport layer:* provides logical communication between processes (terminated at the application)
- Transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - recv side: reassembles segments into messages, passes to app layer



MULTIPLEXING APPLICATIONS OVER TRANSPORT

A application is identified by a transport layer address: <IP address, port>

IP address: gets you to the host (technically the interface, but the interface is part of the host)

Port number: gets you to some application process or thread on that host

- Historically a 16 bit unsigned number (0 – 65535)
- DICT servers – 2628, DNS servers – 53, HTTP servers (conventionally) – 80
- And there are hundreds more: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

LIST OF POSSIBLE SERVICES

- Partial delivery
- Reliable delivery
- Ordered delivery
- Flow control
- Congestion control
- Bidirectional
- Unidirectional

- Connection-oriented
- Connection-less
- Segmentation
- Stream-oriented
- Message-oriented
- Non-duplication

Do not memorize this!

THE TRANSPORT BIG PICTURE

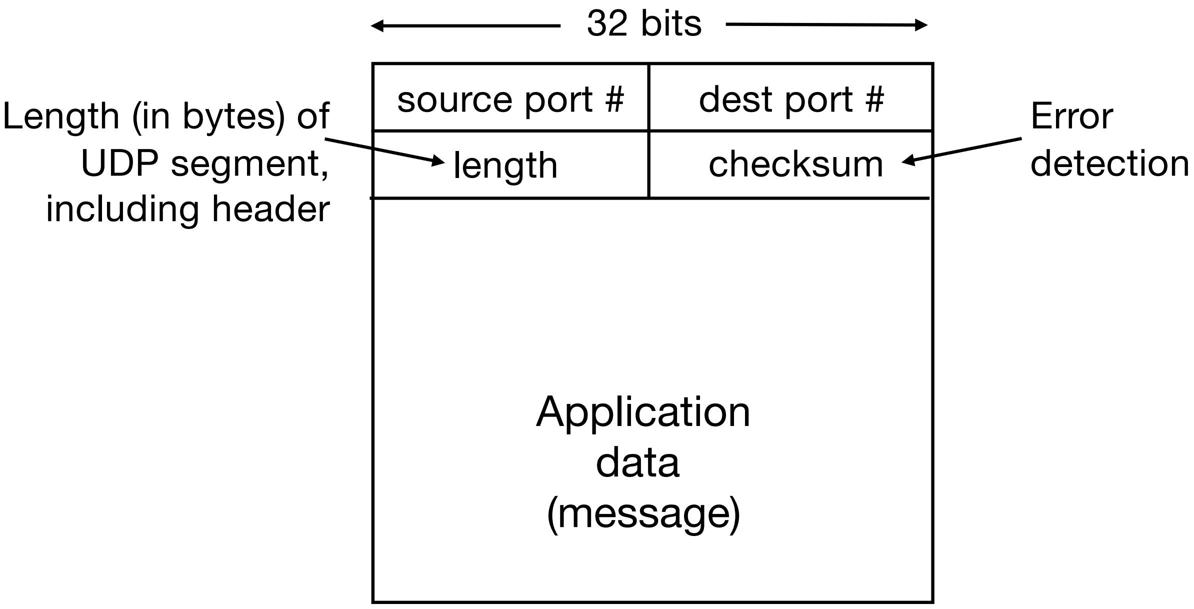
Reliable stream	Unreliable packet
Connection	No connection
Reliable ordered delivery	Best effort
Flow/Congestion control	Nope
Possible delays	No (transport level) delay

INTERNET APPLICATIONS AND TRANSPORT PROTOCOLS

Application	Application layer protocol	Transport protocol
Email	SMTP [RFC 2821]	
Remote shell access	Telnet [RFC 854] SSH [RFC 4253]	
Web	HTTP [RFC 2616]	
Real-time multimedia	Proprietary (Zoom)	
Internet telephony	Proprietary (Skype)	
Domain name	DNS [RFC 1035]	
Dictionary lookup	DICT [RFC 2229]	

UDP

UDP SEGMENT FORMAT


32 bits
source port #
dest port #
length
checksum
Application data (message)
Length (in bytes) of UDP segment, including header
Error detection

UDP segment format

CHECKSUMS

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- Computes some function on the data
- Adds the checksum value to the data
- Sends the data and checksum

Receiver:

- Computes the same function on the received data
- Check if computed checksum equals received checksum value
  - NO - error detected
  - YES - no error detected
- Not all errors can be detected

CHECKSUMS

- Appear at transport layer, network layer, and link layer
- Serve a different purpose at each layer

- Same algorithm at transport and network layer

CHECKSUM ALGORITHM

- Treat the data as a sequence of 16-bit integers
- Function: addition (1's complement sum, carry out added back in) of all these 16 bit integers
- Checksum is the 1's complement of the computed value (flip all the bits)
- Verifying is computing the same function over the data and checksum (correct if 0)

CHECKSUM EXAMPLE (5 BIT INTEGERS)

```
1 1 0 1 0
0 1 0 0 1
1 0 1 1 0
1 0 0 1 1
```

CHECKSUM EXAMPLE (5 BIT INTEGERS)

```
    1 1 0 1 0
    0 1 0 0 1
    1 0 1 1 0
    1 0 0 1 1
    ---------
1 0 0 1 1 0 0   Sum and carry
```

CHECKSUM EXAMPLE (5 BIT INTEGERS)

```
1 1 0 1 0
0 1 0 0 1
1 0 1 1 0
1 0 0 1 1
---------
0 1 1 0 0   Sum
      1 0
---------   Add the carry
0 1 1 1 0
```

CHECKSUM EXAMPLE (5 BIT INTEGERS)

```
1 1 0 1 0
0 1 0 0 1
1 0 1 1 0
1 0 0 1 1
----------
0 1 1 0 0    Sum
      1 0
----------   Add the carry
0 1 1 1 0

1 0 0 0 1    1’s complement
```

VERIFYING THE CHECKSUM

```
    1 1 0 1 0
    0 1 0 0 1
    1 0 1 1 0
    1 0 0 1 1
    1 0 0 0 1
    ---------
1 0 1 1 1 0 1   Sum and carry
          1 0
                Add the carry
    ---------
    1 1 1 1 1

    0 0 0 0 0   1’s complement
```

CLICKER QUESTION

Will the Internet checksum be able to detect when one bit has been erroneously changed?

A. Yes

B. No

C. Sometimes

CLICKER QUESTION

Will the Internet checksum be able to detect when two bits have been erroneously changed?

A. Yes

B. No

C. Sometimes

VERIFYING THE CHECKSUM

```
    1 1 0 1 0
    0 1 0 0 1
    1 0 1 1 0
    1 0 0 1 1
    1 0 0 0 1
    ---------
1 0 1 1 1 0 1   Sum and carry
          1 0
    ---------   Add the carry
    1 1 1 1 1

    0 0 0 0 0   1's complement
```

VERIFYING THE CHECKSUM

```
    1 1 0 1 0
    0 1 1 0 1
    1 0 0 1 0
    1 0 0 1 1
    1 0 0 0 1
    ---------
1 0 1 1 1 0 1   Sum and carry
          1 0
    ---------   Add the carry
    1 1 1 1 1

    0 0 0 0 0   1's complement
```

UDP SOCKETS

UDP COMMUNICATION IN JAVA

- One more Socket class to learn about
  - `DatagramSocket`
- And one new message class
  - `DatagramPacket`

DATAGRAM SOCKET CLASS

- Two commonly used constructors
  - DatagramSocket(int port)
  - DatagramSocket() – let the system choose any available port
- Send a message using send
  - socket.send(DatagramPacket packet)
- Receive a message using receive
  - socket.receive(DatagramPacket packet)
  - The incoming information is stored in the provided packet

DATAGRAM PACKET CLASS

- Two commonly used constructors
  - DatagramPacket(byte[] buf, int length)
    - for receiving messages
  - DatagramPacket(byte[] buf, int length, InetAddress addr, int port)
    - for sending messages
- The data comes from or is stored into the provided buffer

AN EXAMPLE SERVER

```
DatagramSocket socket = new DatagramSocket(4445);

void echo() {
  byte[] buf = new byte[256];
  DatagramPacket packet = new DatagramPacket(buf, buf.length);
  socket.receive(packet);
  InetAddress address = packet.getAddress();
  int port = packet.getPort();
  int length = packet.getLength();
  packet = new DatagramPacket(buf, length, address, port);
  socket.send(packet);
}
```

AN EXAMPLE CLIENT

```
DatagramSocket socket = new DatagramSocket();

String ping(String hostname, int port, String msg) {
  byte[] buf = msg.getBytes();
  byte[] recvbuf = new byte[256];
  InetAddress address = InetAddress.getByName(hostname);
  DatagramPacket packet = new DatagramPacket(buf, buf.length, address, port);
  socket.send(packet);
  packet = new DatagramPacket(recvbuf, recvbuf.length);
  socket.receive(packet);
  String received = new String(packet.getData(), 0, packet.getLength()));
  return received;
}
```

IN-CLASS ACTIVITY

- ICA41