Lesson 3.1:
Transport Layer

CSC450 – COMPUTER NETWORKS | WINTER 2019-20

DR. ANDREY TIMOFEYEV

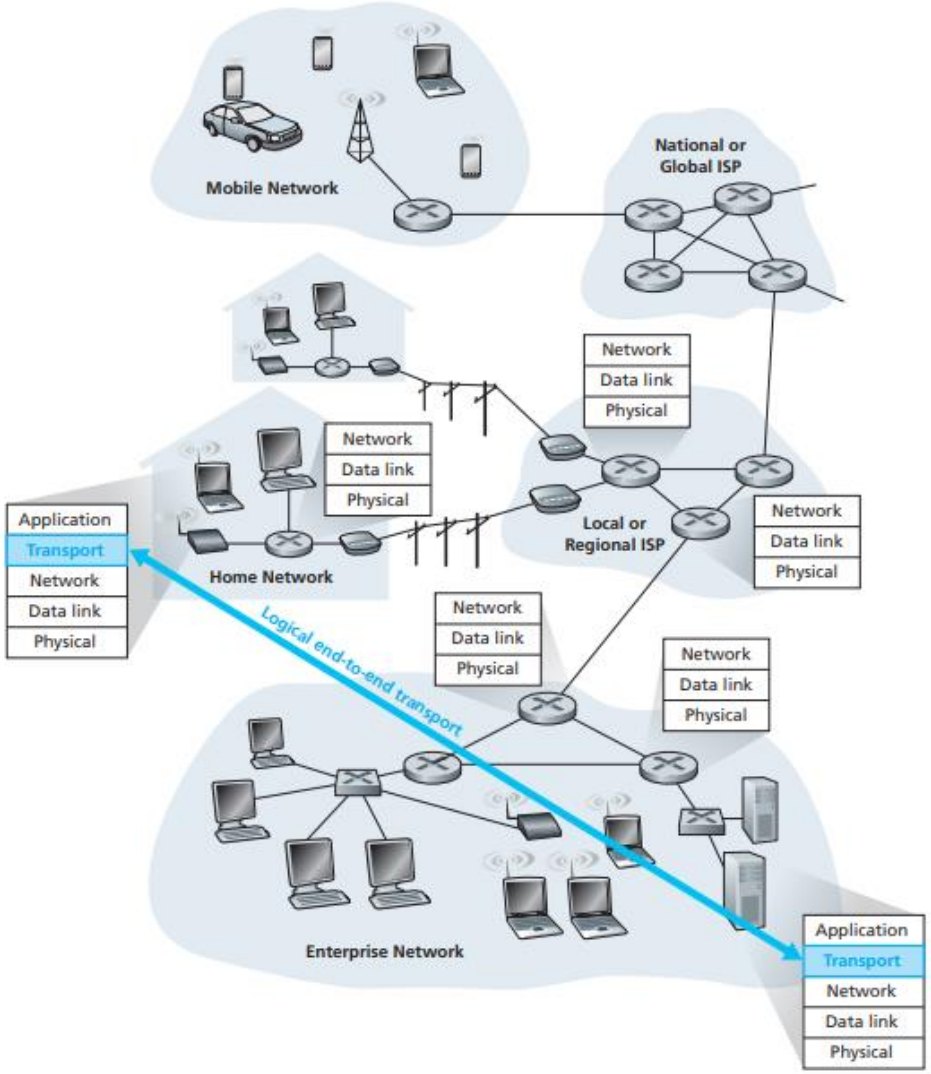
OUTLINE

- Introduction.
  - Transport layer services & protocols.
  - Overview of UDP & TCP protocols.
- Multiplexing & demultiplexing.
- UDP protocol.
  - Segment structure.
  - Checksum.

INTRO: SERVICES & PROTOCOLS

- **Transport layer** protocols provides **logical communication** between **application processes** running on different hosts.
  - **Logical communication** – from application perspective, hosts are connected directly.

- **Transport layer** protocols are implemented in the **end systems**, but not in **routers**.
  - **Sender side** – **breaks** application messages into segments and **passes** them to network layer.
  - **Receiver side** – **reassembles** segments into messages and **passes** them to application layer.



Logical communication between processes

INTRO: TRANSPORT VS. NETWORK

- **Network layer** provides **logical** communication between **hosts**.
- **Transport layer** provides **logical** communication between **processes**.
- **Transport** layer services **constrained** on services provided by **network** layer.
  - Additional **services** could be implemented as an **enhancements**.
- **Analogy example**.
  - 10 kids in Jane’s house sending letters to 10 kids in Bill’s house:
    - Hosts = Houses
    - Processes = Kids
    - Application messages = Letters in envelopes
    - Transport-layer protocol = Jane & Bill who direct mail to in-house siblings
    - Network-layer protocol = Postal service

INTRO: UDP & TCP OVERVIEW

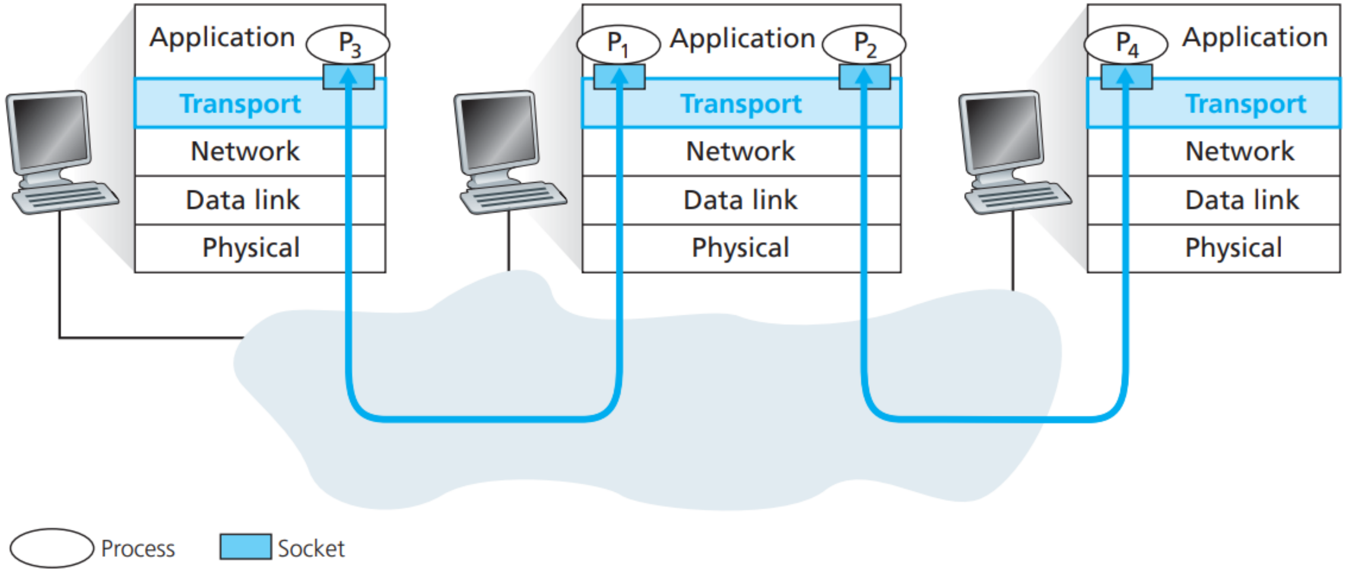
- Current Internet network architecture offers **two** transport layer **protocols**:
  - **User Datagram Protocol (UDP)**.
    - Unreliable, unordered delivery.
    - Process-to-process delivery and error-detection on top of IP.
  - **Transmission Control Protocol (TCP)**.
    - Reliable, in-order delivery.
    - Process-to-process delivery, connection setup, flow control, and congestion control on top of IP.

MUX/DEMUX (1)

- Transport layer **extends** network layer **host-to-host** delivery to **process-to-process** delivery using **multiplexing/demultiplexing**.

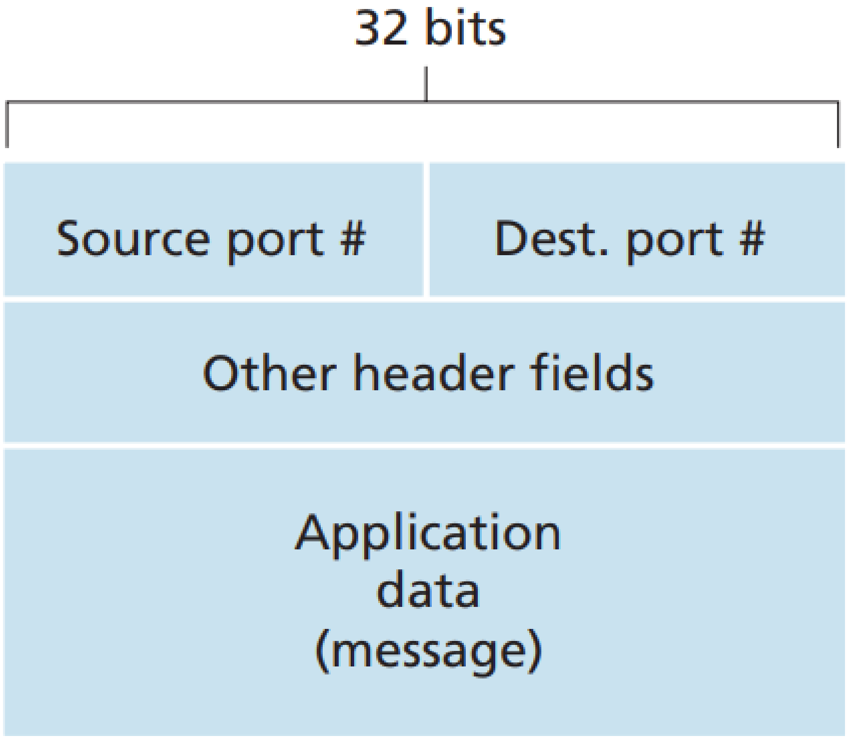
- **Multiplexing** at **sender**:
  - **Gathering** data chunks from different sockets;
  - **Encapsulating** each data chunk with header (creating segments);
  - **Passing** segments to the network layer.

- **Demultiplexing** at receiver:
  - **Checking** segment's header to identify the receiving socket;
  - **Directing** segment to that socket.



MUX/DEMUX (2)

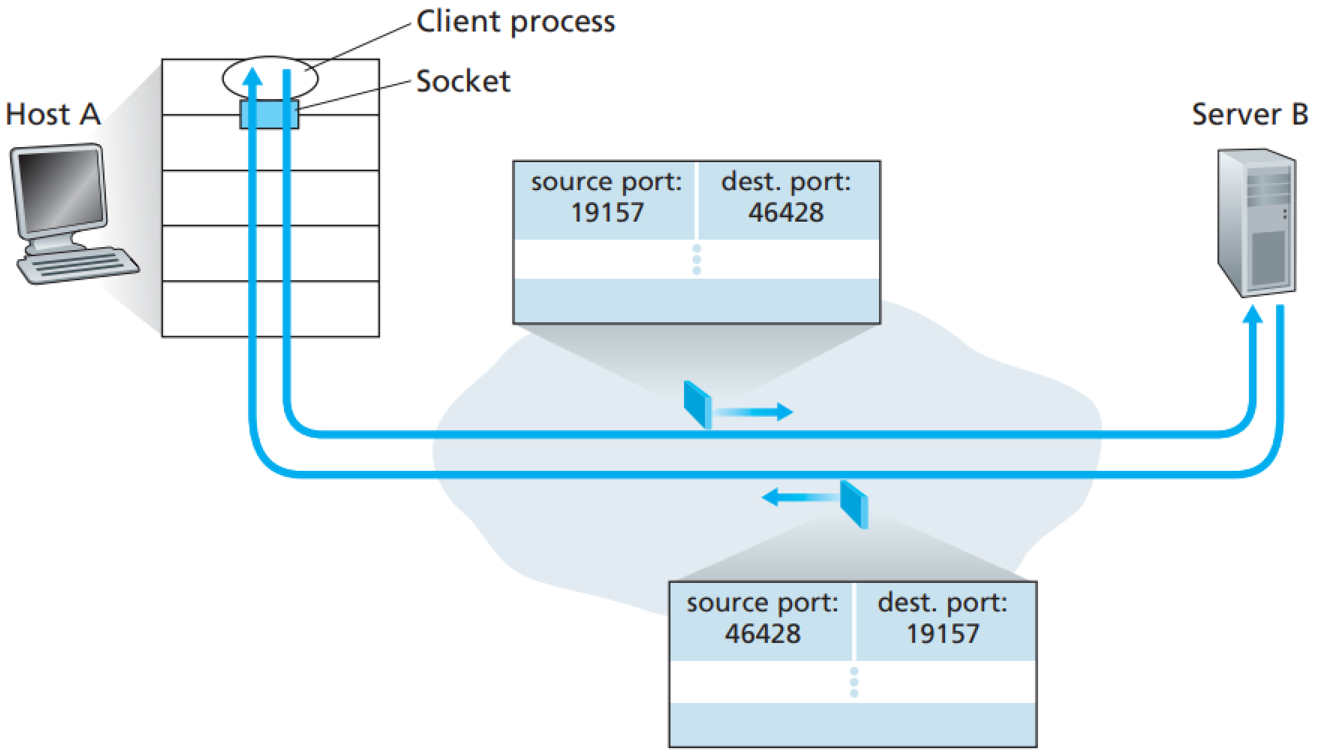
- Transport layer **mux/demux requires**:
  - **Sockets** to have unique **identifiers**.
  - **Segments** to have special **fields** indicating the **destination** and **source** sockets.
    - **Source** port number field.
    - **Destination** port number field.
- Each port number field is **16-bit** long.
  - *65535* **unique** port numbers.
    - *0-1023* reserved for **well-known** port numbers.



Transport-layer general segment fields

MUX/DEMUX: UDP

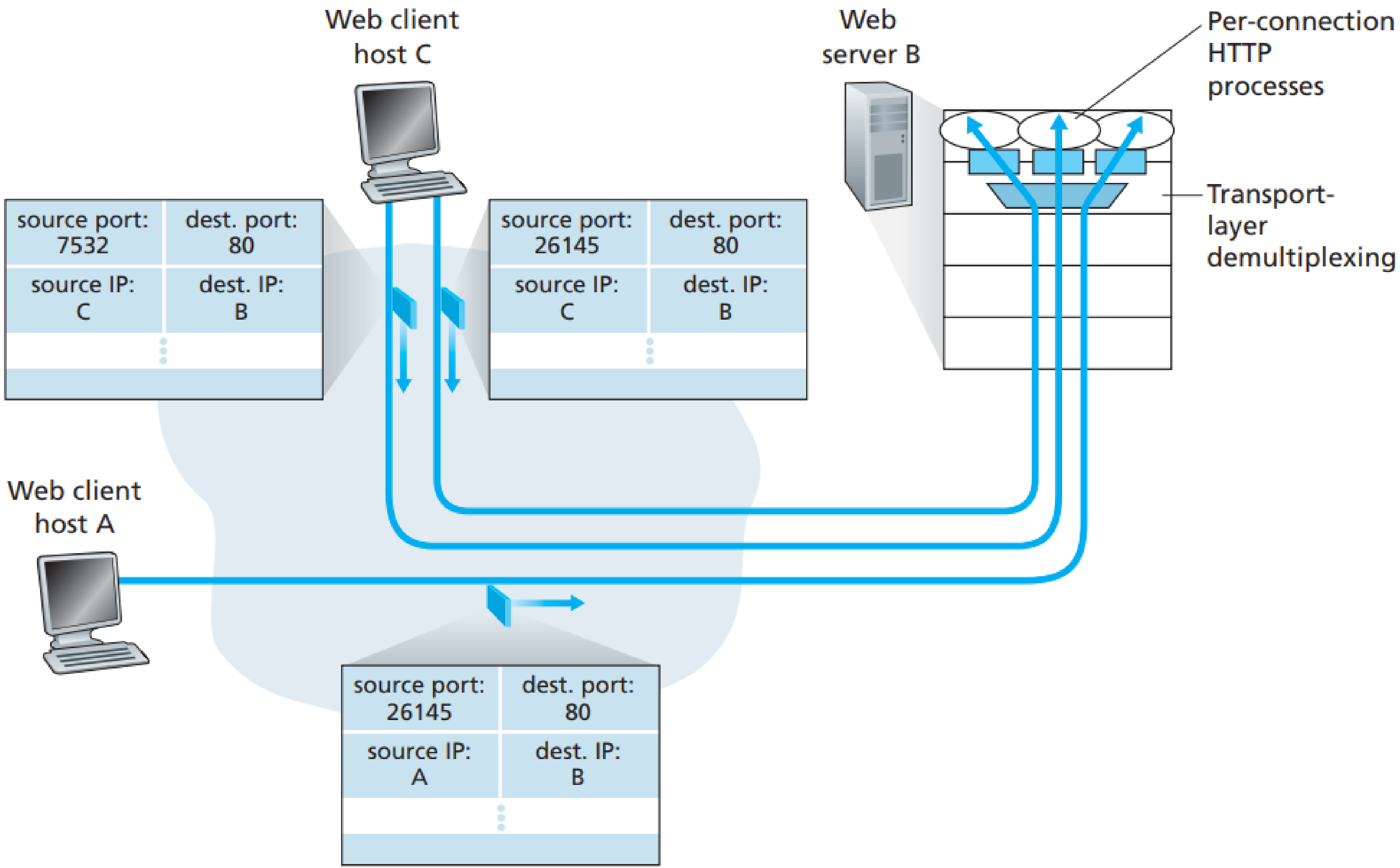
- **UDP socket** is assigned with a **port number** when created.
- To **send** data into **UDP socket** we must specify (dest **IP address**, dest **port number**) 2-tuple.
- When host **receives** UDP segment:
  - **Checks** destination **port number** in segment;
  - **Directs** UDP segment to **socket** with that port number.



MUX/DEMUX: TCP (1)

- **TCP socket** is identified by **4-tuple**:
  - Source IP address;
  - Source port number;
  - Destination IP address;
  - Destination port number.
- **Demultiplexer receiver** uses all **four** values to direct **segment** to appropriate socket.
- **Server** host may support many **simultaneous** TCP sockets.
  - Each socket is **identified** by its own 4-tuple.

MUX/DEMUX: TCP (2)



source port: 7532	dest. port: 80
source IP: C	dest. IP: B

source port: 26145	dest. port: 80
source IP: C	dest. IP: B

source port: 26145	dest. port: 80
source IP: A	dest. IP: B

TCP multiplexing animation

UDP: OVERVIEW (1)

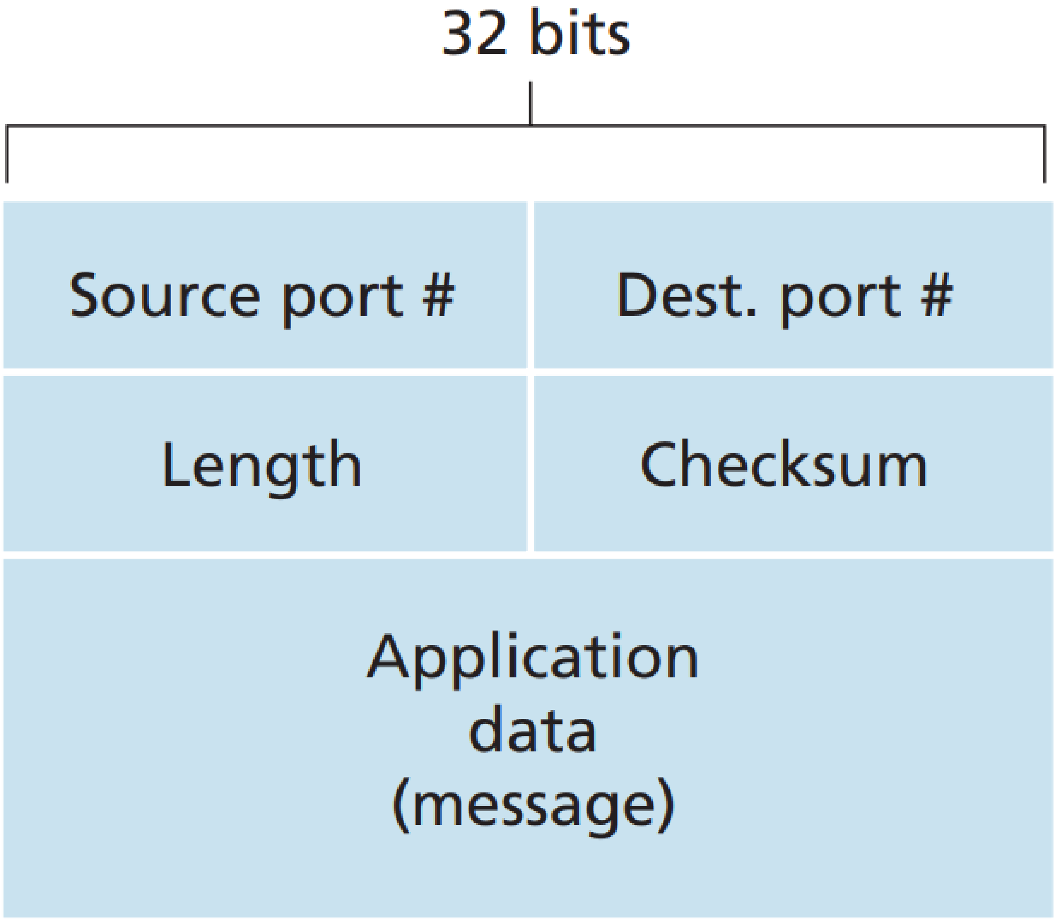
- **User Datagram Protocol** (**UDP**).
  - “**No frills**”, “**bare bones**” Internet transport protocol.
  - “**Best effort**” service. UDP segments may be:
    - **Corrupted** or **lost**;
    - Delivered **out-of-order** to the application.
  - **Connectionless**.
    - No **handshaking** between UDP sender and receiver.
    - Each UDP segment handled **independently** of others.
  - Adds **mux/demux** and simple **error-checking** services on top of IP.

UDP: OVERVIEW (2)

- **Why** UDP?
  - **No connection establishment**.
    - Less delay.
  - **No connection state** at sender & receiver sides.
    - Simplicity.
    - More active clients.
  - **Smaller header size**.
    - TCP header size = 20 bytes.
    - UDP header size = 8 bytes.
      - More space for data.
  - **No congestion control**.
    - Can blast data as fast as desired.

UDP: SEGMENT STRUCTURE

- **UDP header** consists of **four** fields (each 2 bytes long):
  - **Source** port number.
  - **Destination** port number.
  - **Length**.
    - Number of bytes in segment (header + data).
    - Needed since the size of the data field may vary.
  - **Checksum**.
    - Used by receiving host to check for bit errors.



UDP segment structure

UDP: CHECKSUM

- **Checksum** allows **detecting** if error has been introduces during **segment transmission**.
  - **Sender** side:
    - Takes 1’s complement of the sum of three 16-bit words in segment.
      - Overflow of sum is wrapped around.
    - Result is placed in checksum field.
  - **Receiver** side:
    - All 16-bit words are added, including checksum itself.
    - If sum = all 1s → no errors.
    - If at least one 0 → segment has an error.

Source port	0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0
Dest port	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Length	1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0

- Just **error-check**, NO **error-recovery**.
  - Two options:
    - **Discard** damaged segment.
    - **Pass** damaged segment with a **warning**.

UDP: CHECKSUM

- **Checksum** allows **detecting** if error has been introduces during **segment transmission**.
  - **Sender** side:
    - Takes 1’s complement of the sum of three 16-bit words in segment.
      - Overflow of sum is wrapped around.
    - Result is placed in checksum field.
  - **Receiver** side:
    - All 16-bit words are added, including checksum itself.
    - If sum = all 1s → no errors.
    - If at least one 0 → segment has an error.
- Just **error-check**, NO **error-recovery**.
  - Two options:
    - **Discard** damaged segment.
    - **Pass** damaged segment with a **warning**.

Source port	0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0
Dest port	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Length	1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0
Sum	1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 1
Wrap around	0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0

Checksum	1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1
Source port	0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0
Dest port	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Length	1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0
Sum	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
Wrap around	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

SUMMARY

- Transport layer services.
- Multiplexing & demultiplexing.
- UDP segment structure.
- UDP checksum.