Computer Networks
Homework #7

December 13th, 2012

Transport Layer

Q1:

- What entities does the transport layer connect in contrast to the network layer?

Transport vs. network layer

A1:

- *Network layer:*
  logical communication between hosts.
- *Transport layer:*
  logical communication between processes

Multiplexing

Q2:

- Why is **multiplexing** and **demultiplexing** used for at the transport layer and what has the concept of **ports** to do with this?

Multiplexing

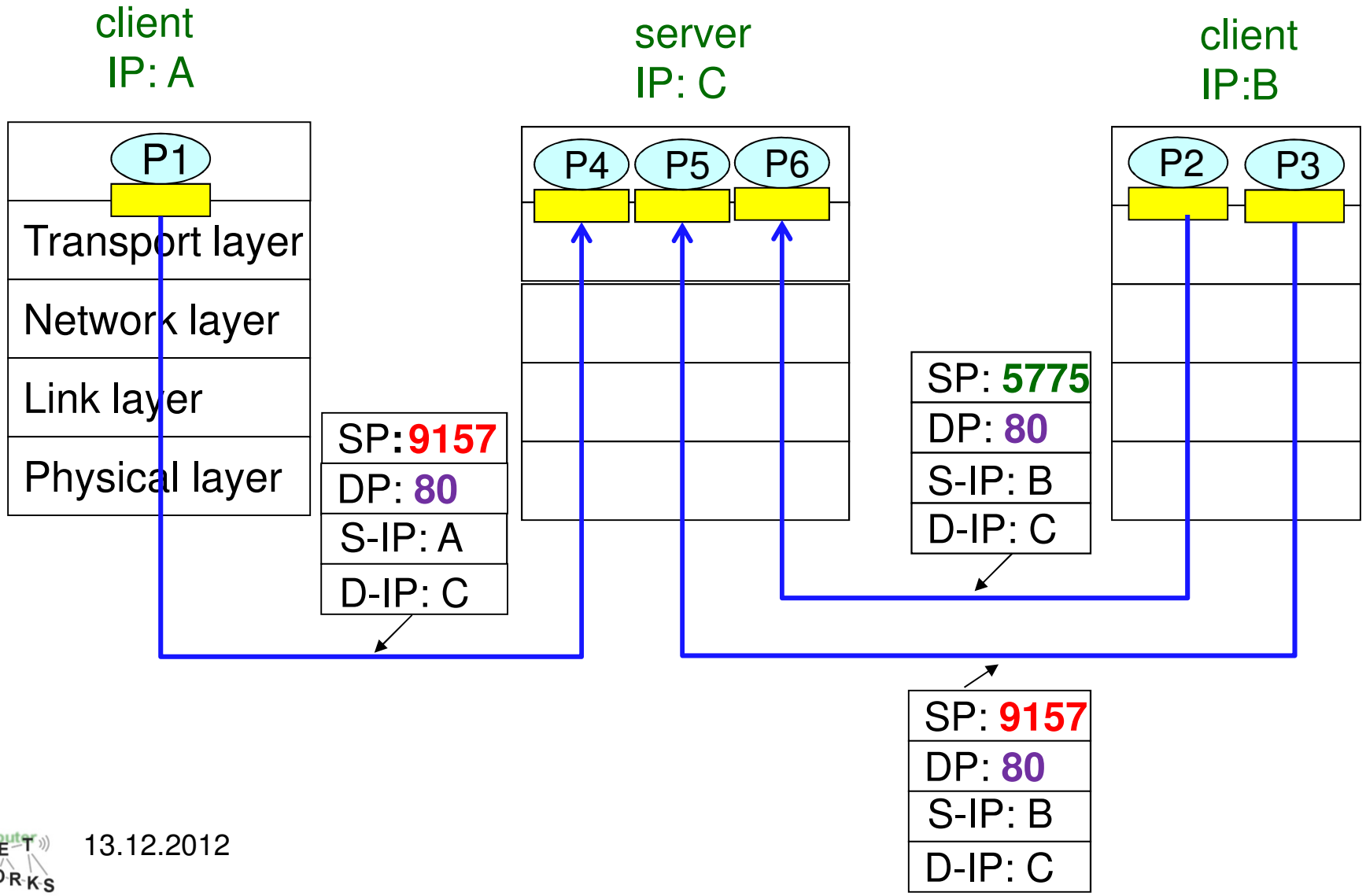
A2:

- **Demultiplexing**:
  - delivering the data in a transport-layer segment to the correct **socket**.
- **Multiplexing**:
  1. gathering data chunks at the source host from different **sockets**;
  2. encapsulating each data chunk with header information to create segments;
  3. passing the segments to the network layer.

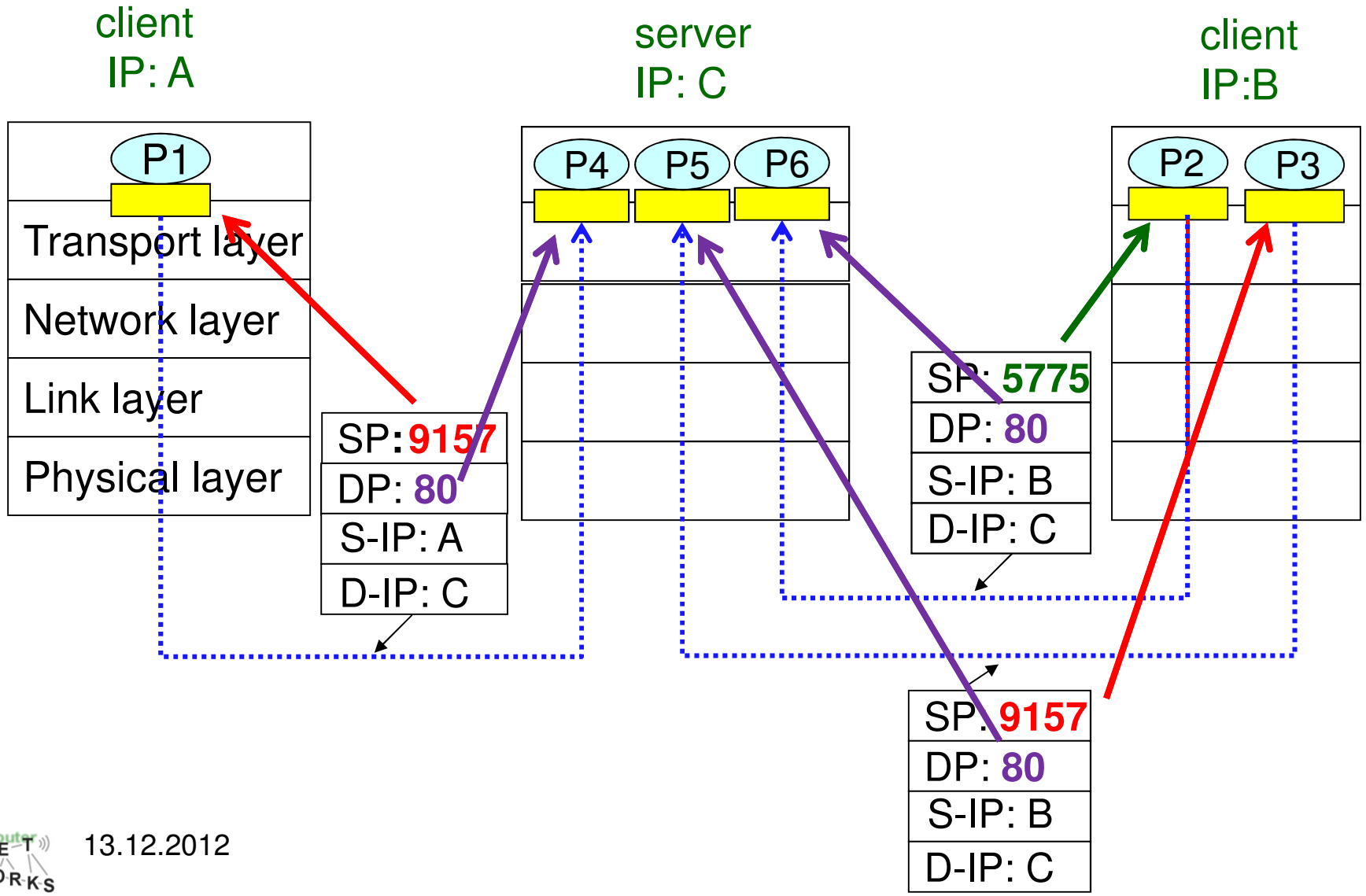
- **Sockets**:

  Doors
  - through which data passes from the network to the process;
  - through which data passes from the process to the network
- **Port Numbers**:
  - unique identifiers of sockets.

Multiplexing



Multiplexing

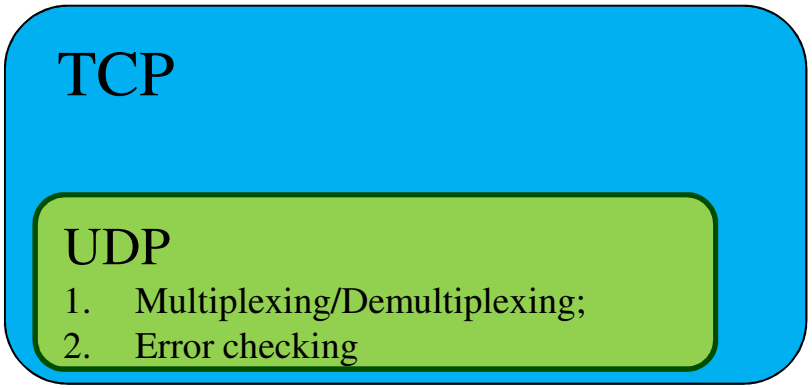


TCP vs. UDP

Q3:

- Please compare TCP and UDP in terms of the services they offer to the application layer.

TCP vs. UDP



A3:

- TCP
  - Reliable data transfer
  - Connection-oriented transport
  - In-order delivery
  - Congestion control
- UDP
  - Best-effort data transfer
  - Connectionless
  - Possible out-of-order delivery
  - No congestion control

TCP vs. UDP (II)

Q4:

- For which kinds of applications would you prefer UDP over TCP.

TCP vs. UDP (II)

A4:

- Transmission rate constraints enforced by TCP are not wanted/needed
- Reliability of TCP is not wanted/needed

Examples:

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

UDP checksums

Q5:

- Assume a UDP transport has received a datagram which consists of the following 16-bit words and the given checksum. Please verify the checksum.

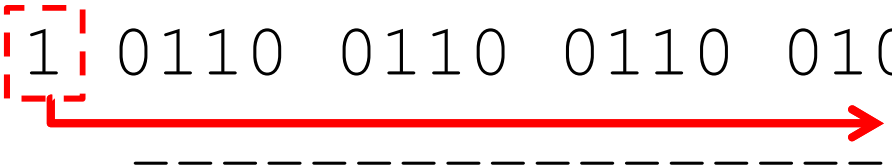
- 1010 1010 1010 1010 (1st 16 bit word) **a1**
- 1011 1011 1011 1011 (2nd 16 bit word) **a2**
- 1100 1100 1100 1100 (3rd 16 bit word) **a3**
- 1110 1100 1100 1100 (recvd. checksum) **$2^{16} - (a1+a2+a3)$**

UDP checksums (cont'd)

A5:

- 1) one's complement sum of first two words:

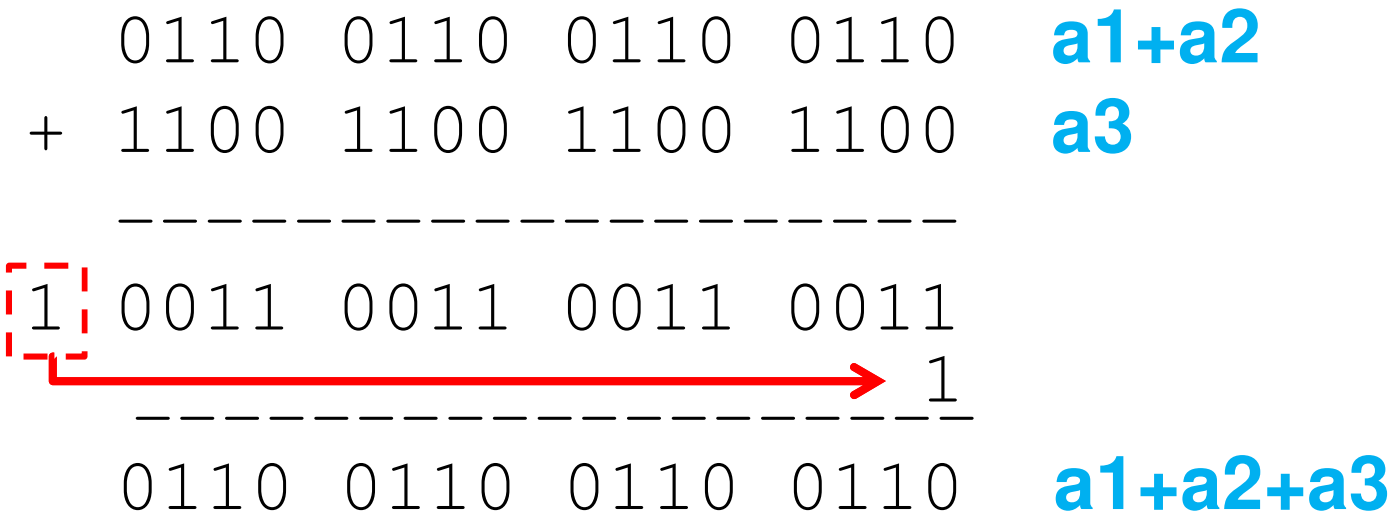
```
  1010 1010 1010 1010   a1
+ 1011 1011 1011 1011   a2
  ---------------------
1 0110 0110 0110 0101
                    1
  ---------------------
  0110 0110 0110 0110   a12
```



- Why 0 at last digit? The carry is added in the last step!

UDP checksums (cont‘d)

- 2) add third word:



UDP checksums (cont'd)

- 3) add to received checksum:

```
  0011 0011 0011 0011
+ 1110 1100 1100 1100
  ---------------------
1 0001 1111 1111 1111
```

a1+a2+a3

checksum: $2^{16} - (a1+a2+a3)$

$= 2^{16}$?

- 4) Result ≠ 1111 1111 1111 1111 therefore verification failed

Reliable data transfer

Q6:

Assume you want to reliably transfer data over a channel with **bit errors** but **no loss**. An error detection mechanisms is already implemented.

- Which **simple mechanism** can you use to recover from errors?
- What **flaw** does this simple mechanism have?

Reliable data transfer (cont'd)

A6:

rdt2.0: reliable data transfer 2.0

- Send **ACK** if packet was received without errors,

  ACK: "**OK**!"

- Send **NAK** otherwise

  NAK: "**Please repeat that**!"

- Flaw: If **ACK**/**NAK** gets corrupted, sender doesn't know if packet was received without errors.

Reliable data transfer (II)

Q7:

Assume you want reliably transfer data over a channel with **bit errors** and **loss**.

- What **additional mechanism** do you need to introduce?
- Give an **example** of how this mechanism can recover from the loss of a packet.

Reliable data transfer (II) (cont'd)

A7:

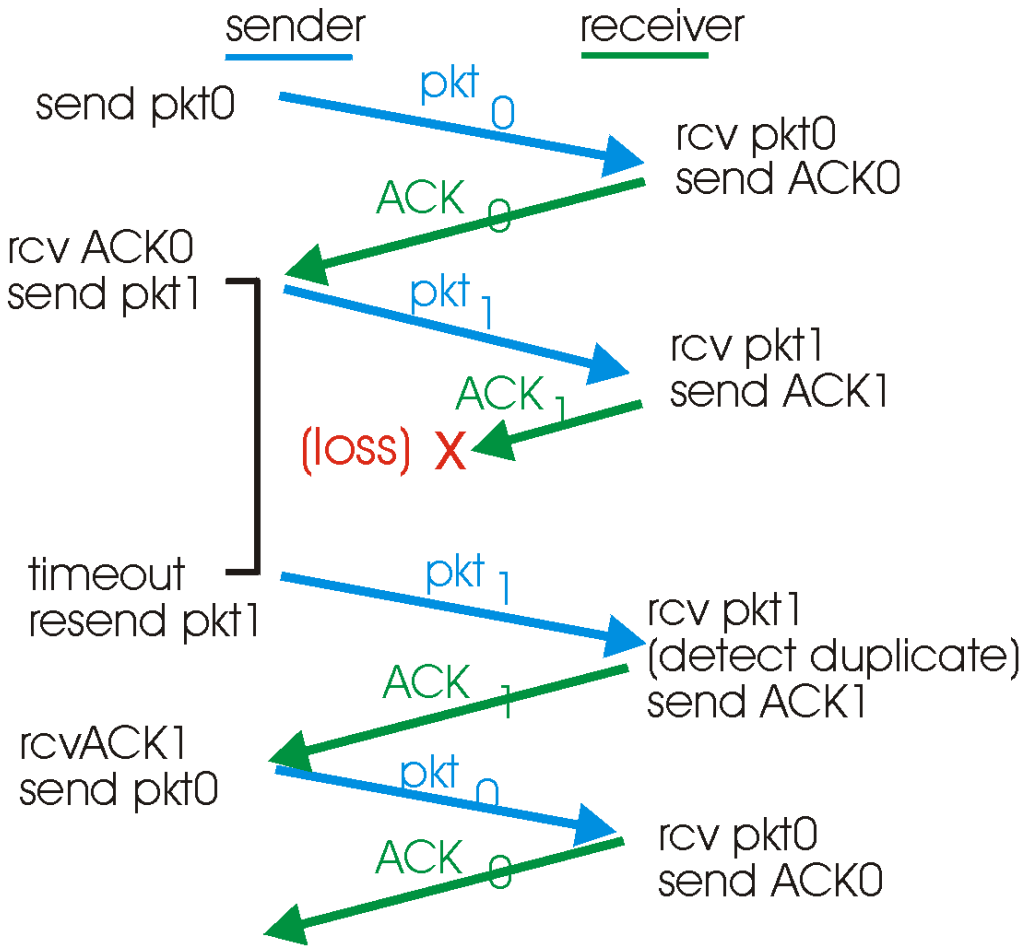
rdt3.0: reliable data transfer 3.0

- Informal:
  - Sender **wait** long enough **time** to *certain that a* packet has been lost, then **retransmit**.

- Formal:
  - Sender needs to maintain a timer for unacknowledged packets so it can re-transmit if an ACK is not received within a certain timeframe.

Reliable data transfer (II) (cont'd)

Example:



- That’s all and thanks for your attention!