- DA343A -
Objektorienterad programutveckling, trådar och datakommunikation

Föreläsning 11
Ports and Connections

Ben Blamey

Recap – Last Lecture:

- As application developers, we see a TCP connection as a pair of continuous streams (each travelling a different way).
  - All the ”fiddliness” (lost/corrupt/misordered packets, routing, WiFi/Ethernet/5G) is hidden from us.
- Layers of the TCP Stack
  - Data actually sent as separate “packets”
  - Different layers of the TCP stack pre-pend their headers to packets (to implement the “fiddleness”)
- HTTP – an example of an existing application layer protocol (used for transfering files on the web and REST Servers).
- We used a high-level Java library to download web resources (HTML, Images).
  - the implementation details of the application protocol, and handling of the connection were encapsulated in the library.

Recap - TCP/IP Stack

TCP/IP Stack

Layer	Unit of Data	Protocols	Concerns
Application Layer	Byte Stream or Message	HTTP SMTP FTP DNS (proprietary)	→ Web Request + Response → Transfer of Emails → Transfer of Files → Domain Name System → (application specific protocols)
Transport Layer	Segment	TCP UDP	→ Connection-orientated, guaranted delivery, congestion, ordering. → Connectionless, no control.
Nework Layer (IP Layer)	Datagram	IP Protocol, Routing	Routing
Data-Link Layer	Frame	Ethernet, WiFi (802.11), Point-to-Point Protocol (PPP)	Binary data sent between nodes.
Physical Layer	Bits	e.g. Ethernet. Wifi: 802.11	Ethernet: Which color wires used for what? Voltages, frequency. WiFi: Format of radio messages? What radio frequency?

Lecture Summary

- Key networking concepts:
  - Network Interfaces
  - Localhost
  - Ports
  - Sockets
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
  - Implementing our own application-layer protocols.
  - Examples in Java.

Network Interfaces

- A device on the network ("***host***") has one or more ***network interfaces***.
- WiFi and Ethernet connections are each represented by their own *network interface*.
- (Physical) interfaces have a ***medium access control*** (MAC) address.
- An active network interface typically has an ***IP address*** associated with it.
- We can list all the interfaces on the system with the ifconfig (mac/linux) or ipconfig (win) tool:

```
$ ifconfig -Lu
...
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
ether 6c:7e:67:cb:04:98
inet6 fe80::1808:b08e:ca49:dafb%en0 prefixlen 64 secured scopeid 0xf
inet 10.3.5.244 netmask 0xffff0000 broadcast 10.3.255.255
nd6 options=201<PERFORMNUD,DAD>
media: autoselect
status: active
```

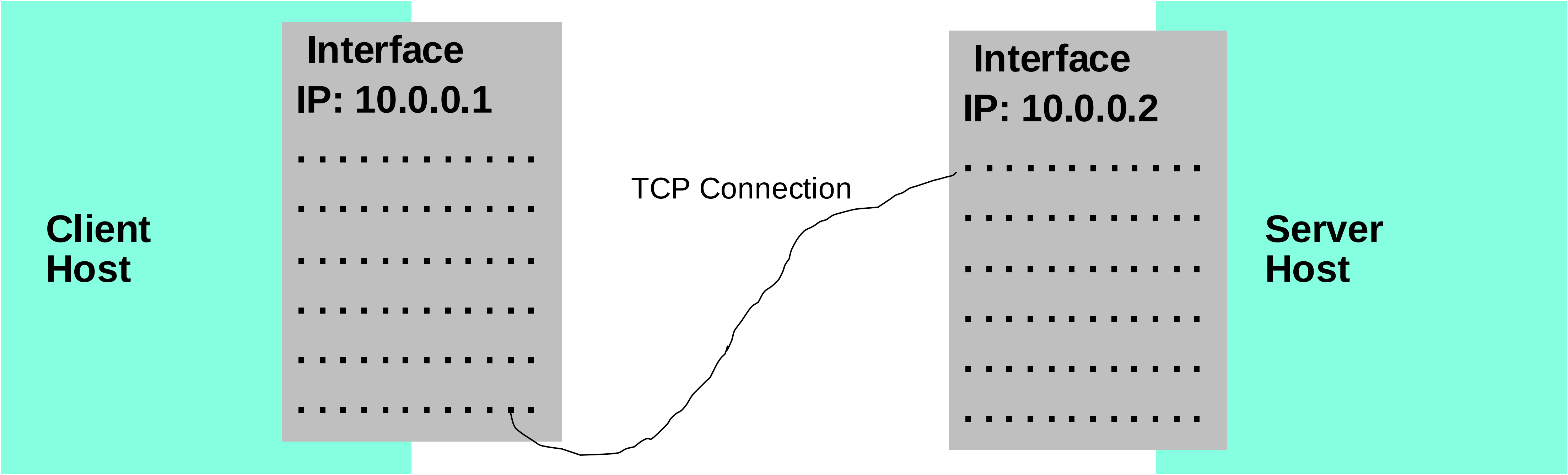
Localhost / Loopback Interface

- There is always a special network “loopback” interface, with the hostname ***localhost***.
- It always has the IPv4 address **127.0.0.1** and **(**::1 in IPv6)
- It is not associated with any hardware, and is **inaccessable from outside the host**.
- It can be used for applications/components on the same machine to communicate with each other.
- We can also use it for testing our applications, running local web servers, etc.

```
$ ifconfig -Lu
...
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
inet 127.0.0.1 netmask 0xff000000
inet6 ::1 prefixlen 128
inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
nd6 options=201<PERFORMNUD,DAD>
```

Ports

- Each interface has ***ports*** used for sending and receiving data.
- Using a specific **port number** on a specific **interface** with a specific **transport protocol** (TCP or UDP) it is called a ***socket.***
- The ports on each interface on the same machine are independent from eachother.
- Ports are numbered 1 to 65535 (0xFFFF).
- The ports between 1…1024 are by convention used for specific services - ”well known”.
- For example, TCP connections use a specific port on the client and a port on the server:



Command Line Utilities

- Following examples use the following command line utitilies.
- On Windows, they can be accessed using the “Windows Subsystem for Linux” https://learn.microsoft.com/en-us/windows/wsl/install
- Documentation, (search for the name of the command)
- Linux: https://linux.die.net/man/
- BSD (e.g. mac): https://man.freebsd.org/cgi/man.cgi

- ifconfig
- watch (can be installed on MacOS with “brew”)
- grep
- nc

- These commands are only used for demonstration purposes – you do not need to revise these commands for the exam. They might be helpful for troubleshooting in your utvecklingsprojekt.

Listing Connections / Sockets

We can list all the sockets with the netstat tool.
This example shows a sample, filtering only TCPv4 sockets (in this example):

```
$ netstat -f inet -n
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  10.3.5.244.51235       162.125.70.13.443      ESTABLISHED
tcp4       0      0  10.3.5.244.51233       13.69.239.72.443       ESTABLISHED
tcp4       0      0  10.3.5.244.51229       20.50.73.9.443         ESTABLISHED
tcp4       0      0  10.3.5.244.51228       20.50.73.13.443        ESTABLISHED
tcp4       0      0  10.3.5.244.51227       20.50.73.13.443        ESTABLISHED
tcp4       0      0  10.3.5.244.51226       74.112.186.146.443     ESTABLISHED
tcp4       0      0  10.3.5.244.51222       185.15.59.240.443      ESTABLISHED
tcp4       0      0  10.3.5.244.51221       185.15.59.224.443      ESTABLISHED
tcp4       0      0  10.3.5.244.51210       52.111.209.3.443       ESTABLISHED
...
```

Table 2-1. Well-known port assignments

Protocol	Port	Protocol	Purpose
echo	7	TCP/UDP	Echo is a test protocol used to verify that two machines are able to connect by having one echo back the other's input.
discard	9	TCP/UDP	Discard is a less useful test protocol in which all data received by the server is ignored.
daytime	13	TCP/UDP	Provides an ASCII representation of the current time on the server.
FTP data	20	TCP	FTP uses two well-known ports. This port is used to transfer files.
FTP	21	TCP	This port is used to send FTP commands like `put` and `get`.
SSH	22	TCP	Used for encrypted, remote logins.
telnet	23	TCP	Used for interactive, remote command-line sessions.
smtp	25	TCP	The Simple Mail Transfer Protocol is used to send email between machines.
time	37	TCP/UDP	A time server returns the number of seconds that have elapsed on the server since midnight, January 1, 1900, as a four-byte, signed, big-endian integer.
whois	43	TCP	A simple directory service for Internet network administrators.
finger	79	TCP	A service that returns information about a user or users on the local system.
HTTP	80	TCP	The underlying protocol of the World Wide Web.
POP3	110	TCP	Post Office Protocol Version 3 is a protocol for the transfer of accumulated email from the host to sporadically connected clients.
NNTP	119	TCP	Usenet news transfer; more formally known as the "Network News Transfer Protocol".
IMAP	143	TCP	Internet Message Access Protocol is a protocol for accessing mailboxes stored on a server.
RMI Registry	1099	TCP	The registry service for Java remote objects. This will be discussed in Chapter 18.

Lästips

Java Network Programming
Chapter 2 ("*Basic Network Concepts: Client/Server Model*")

- Server sockets generally use lower numbers, and typically use their "well-known" ports, but this is only a convention.

- Client sockets generally use any available (high) port number. The port number used by the client typically doesn't matter.

“Socket” vs “Port”

- Casually we use “port” and “socket” interchangably. We talk about “open ports” “closed ports”
- Technically:
  - Port = A number between 1 and 65535
  - Socket = Port + Transport Protocol + Inferface(s) (on a Host)
- Note: we often open a listening socket on all interfaces at once.
- We often talk about “sockets” to mean the Socket API that we use as application developers.
- When configuring firewalls, we talk about “ports” – because we filter on the port number.
- A “listening socket” or ”server socket” can be listening for new connections (or messages, in UDP).
- In TCP, a “connection” is between a “client socket” and a “server socket”. Once the socket is established, there is no special meaning for the client and server – either end can close the connection whenever it likes.

TCP

TCP Connections

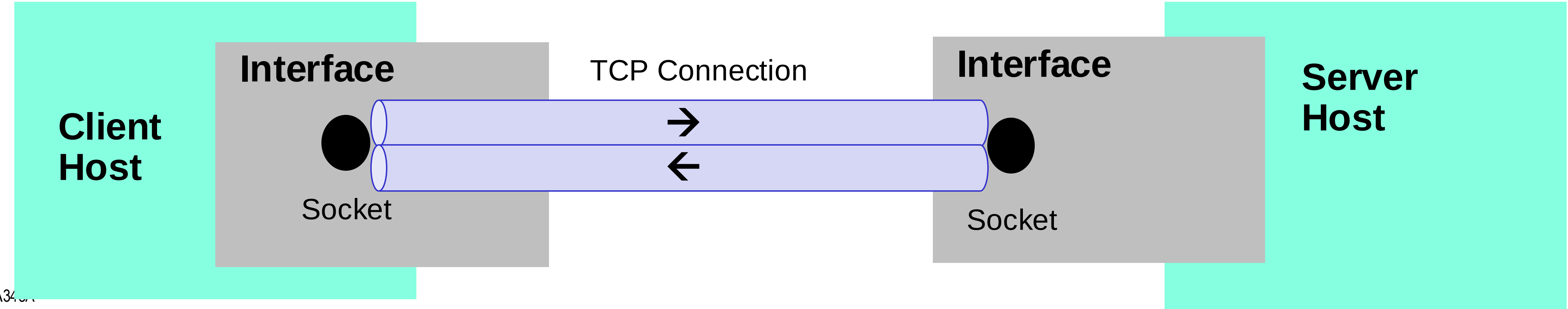
TCP (**transmission control protocol**) is a ***transport protocol***, *it is* ***connection-orientated***.

1. A **process** on a host needs to open a **socket** (on a specific **protocol + interface(s) + port**) in the **listening** state, ready to accept new connections. This is the **server**.
2. A **process** on a host connects to the socket on the server, this is the **client**.
3. While the connection is open, data can be transferred, as a sequence of **bytes.**
4. The connection is **bi-directional**. Either side can transfer data whenever they like.

    We agree an application-layer protocol for synchronization and message formatting.

5. Either the client or the server can **close** the connection at any time. No more data can be sent.

- Mulitple clients can connect to the same server socket, establishing their own independent connection.



Example TCP Connection with netstat

We can use the netcat tool to establish a TCP connection and send text through it (as bytes).

Server:

```
$ nc -l 5000
hej!
this is DA343...
```

Client:

```
$ nc -v localhost 5000
Connection to localhost port 5000 [tcp/commplex-main] succeeded!
hej!
this is DA343...
```

Example TCP Connection with netstat

```
$ netstat -f inet -n –a
tcp4       0      0  10.3.5.244.51847      13.248.245.213.443     ESTABLISHED
tcp4       0      0  10.3.5.244.51846      143.204.237.54.443     ESTABLISHED
tcp4       0      0  10.3.5.244.51845      185.184.8.90.443       ESTABLISHED
tcp4       0      0  127.0.0.1.5000        127.0.0.1.51775        ESTABLISHED
tcp4       0      0  127.0.0.1.51775       127.0.0.1.5000         ESTABLISHED
tcp4       0      0  10.3.5.244.51850      35.190.0.66.443        ESTABLISHED
tcp4       0      0  10.3.5.244.51848      172.217.21.162.443     ESTABLISHED
tcp4       0      0  *.5000                *.*                    LISTEN
tcp4       0      0  10.3.5.244.51844      185.64.190.78.443      ESTABLISHED
tcp4      31      0  10.3.5.244.51841      54.229.79.103.443      CLOSE_WAIT
tcp4       0      0  10.3.5.244.51839      142.250.74.70.443      ESTABLISHED
tcp4       0      0  10.3.5.244.51837      216.58.211.1.443       ESTABLISHED
tcp4       0      0  10.3.5.244.51836      216.58.211.1.443       ESTABLISHED
tcp4       0      0  10.3.5.244.51834      142.250.74.65.443      ESTABLISHED
tcp4       0      0  10.3.5.244.51832      34.102.146.192.443     ESTABLISHED
tcp4       0      0  10.3.5.244.51831      104.22.52.86.443       ESTABLISHED
tcp4       0      0  10.3.5.244.51830      143.204.237.28.443     ESTABLISHED
…
```

Example TCP Connection with netstat

With two clients:

```
$ watch netstat -f inet -n -a | grep 5000
tcp4    36    0  127.0.0.1.5000     127.0.0.1.51789    ESTABLISHED
tcp4     0    0  127.0.0.1.51789    127.0.0.1.5000     ESTABLISHED
tcp4     0    0  127.0.0.1.5000     127.0.0.1.51775    ESTABLISHED
tcp4     0    0  127.0.0.1.51775    127.0.0.1.5000     ESTABLISHED
tcp4     0    0  *.5000             *.*                LISTEN
```

TCP Connections

- The TCP Socket API presents an abstracted **bidirectional continous stream of bytes** (8 bits).
- We read/write to these streams just like files (we need to think about e.g. encodings).
- The guarantees are:
  - Reliability – we know whether bytes have been delivered.
  - Error-checked – bytes received have been checked for corruption.
  - Ordering – bytes received in the same order as they are sent.
- From our "view" of the connection has no concept of "messages", "request / response"
  - we need to design these into our application protocol

Remember:

- "Underneath", separate messages are sent, these can be corrupted, fragmented, delayed, lost, or arrive in the wrong order – but all we see at the Socket API level is a byte stream.
- TCP handles re-sending of corrupt or missing messages, and reassembles the messages in the correct order to re-create the byte stream. This creates **latency.**

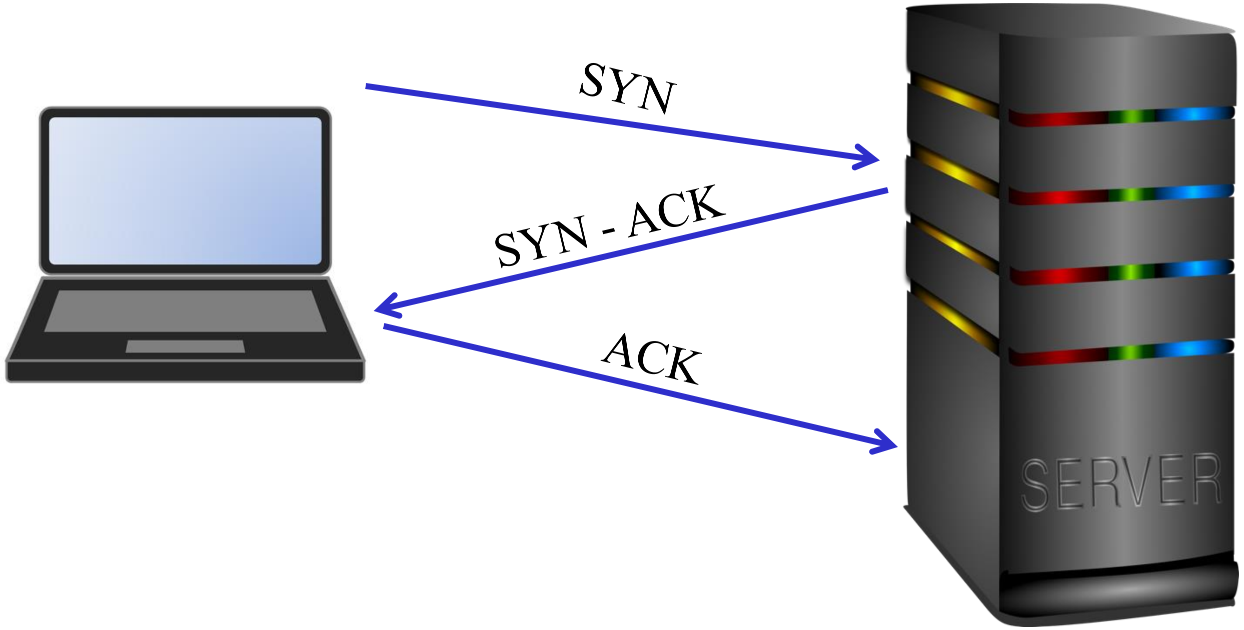
TCP – how to create a connection

The ”abstracted” view of the connection as a cts byte stream, starts with a ”handshake” exchange of packets.

Step 1: The client initiates a contact with the server. A SYN (synchronization) message is sent, among other things. with information about the client.

Step 2: The server responds to the client's request with a SYN-ACK message with additional information and confirmation of receipt of a request.

Step 3: The client sends an ACK on the server's response and a stable connection has been created.



Step 4: Now data transfer can start. Lost or corrupt data is re-sent. Packets are put back in the right order and assembled back into the ”continous” stream. (using the packet header info)

Application Protocols

To make the byte stream more manageable, we use an **Application Protocol.**

We either use an existing one (like HTTP), or we can agree on our own, protocol can stipulate…

- use a request + response pattern, (or publish/subscribe), etc.
- define types of "messages" with a header including a length (maybe a version):
  - <message type (e.g. integer)> <body length in bytes (integer)> <message body (byte array)>
  - …or include this information in string format.
  - Why do we need to know the length?
- or, use special characters to break up messages, such as the line end characters (as in HTTP):

  This is a message.\r\nHej!\r\nAnother message!\r\n
- agree whether to close the connection after one request-response cycle, or leave it open.
- close connections if we receive data we don't understand.
- close connections with servers if they do not respond quickly enough (timeout).
- stipulate the character encoding (if sending strings).

TCP Socket with Java Examples

Java classes for Sockets/Connections

“Socket” (i.e. “client socket” – connects to a “Server Socket”). TCP

https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html

“ServerSocket” (i.e. “listening socket” – listens for client connections). TCP.

https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

“DatagramSocket” (i.e. UDP socket for receiving messages)

https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html

java.net.Socket Class (client socket), constructors:

Constructors

Modifier	Constructor and Description
	`Socket()` Creates an unconnected socket, with the system-default type of SocketImpl.
	`Socket(InetAddress address, int port)` Creates a stream socket and connects it to the specified port number at the specified IP address.
	`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote address on the specified remote port.
	`Socket(Proxy proxy)` Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
`protected`	`Socket(SocketImpl impl)` Creates an unconnected Socket with a user-specified SocketImpl.
	`Socket(String host, int port)` Creates a stream socket and connects it to the specified port number on the named host.
	`Socket(String host, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote host on the specified remote port.

https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html

java.net.Socket Class, methods:

Methods

Modifier and Type	Method and Description
void	close() Closes this socket.
InetAddress	getInetAddress() Returns the address to which the socket is connected.
InputStream	getInputStream() Returns an input stream for this socket.
OutputStream	getOutputStream() Returns an output stream for this socket.
int	getPort() Returns the remote port number to which this socket is connected.
void	setSoTimeout(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
String	toString() Converts this socket to a String.

TCP Client Connections - Examples

Application Protocol	Example	Classes	
Text-based.	TCP_Example1_Strings.java - A message is a single line. - We use the new line character to delimit messages.	java.io.BufferedReader java.io.BufferedWriter	Pro: Easy to debug. Cons: Verbose for lots of data. Sending binary data with strings can be fiddly/inefficient.
Binary.	TCP_Example2_Bytes.java - We send messages using its binary representation. - We write a message body length in the header. - There is no strings involved, and there is no delimiter character.	java.io.DataInputStream java.io.DataOutputStream	Pro: more efficient. Portable. Con: harder to debug. Easy to mix up datatypes, endianess.
Serialized Java Objects	(No example)	java.io. ObjectInputStream java.io. ObjectOutputStream	Pro: can use Serializable interface, implicit message formats, less code. Con: Java-specific, not portable. Security.

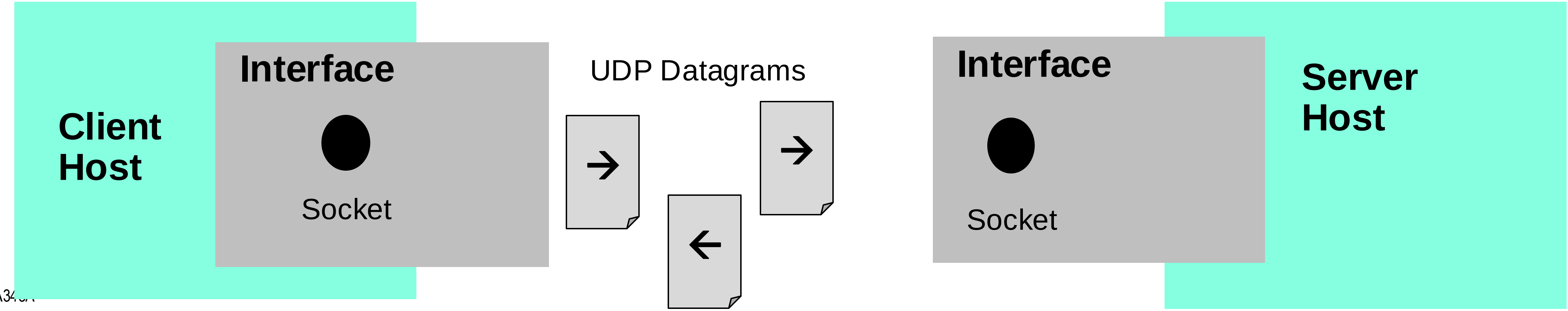
UDP

UDP Datagrams

User Datagram Protocol (UDP) is a ***transport protocol (like TCP)***, *it is* ***message-orientated***.

1. A **process** on a host needs to open a **socket** (on a specific **protocol + interface + port**) in the **listening** state, ready to accept new connections. Otherwise the port is **closed.** This is the **server**.
2. A **process** on a host connects sends data from the client socket as **messages.**
3. A message is simply a **byte array**.

- There is no connection, no handshaking, and no message ordering guarantees, no protection against duplicate messages. (contrast with TCP). Message size is limited. There is a checksum for integrity.
- Often used for streaming applications.
- Special **broadcast** messages can be sent to all hosts on a network. Useful for service discovery.



UDP Socket with Java Examples

UDP – DatagramSocket & DatagramPacket

https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html

https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

Homework

- If you have completed the utvecklingsprojekt….
- Try to create (e.g.) a chat application, adapting these code examples.
- Try with different transport protocols and serialization techniques.
  - UDP and/or TCP
  - Using different serialization techniques:
    - Bytes.
    - Text/Line based.
    - Java.io.Serializable https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html

3.6 TCP och UDP

Denna fråga handlar om TCP och UDP. Välje Sant eller Falskt för varje påstående.

"Host A skapar en **TCP**-anslutning (connection) till Host B. Data kan bara skickas från Host A till Host B. Att skicka tillbaka data åt andra hållet kräver en separat anslutning."

Välj ett alternativ

- Sant
- Falskt

"Data som skickas med **UDP** kommer garanterat fram i samma ordning som de skickas."

Välj ett alternativ

- Sant
- Falskt

"Data som skickas med **TCP** kommer garanterat fram i samma ordning som de skickas."

Välj ett alternativ

- Sant
- Falskt

"Om data som skickas med **UDP** försvinner på väg, skickas den automatiskt igen."

Välj ett alternativ

- Sant
- Falskt

"Om data som skickas med **TCP** försvinner på väg, skickas den automatiskt igen."

Välj ett alternativ

- Sant
- Falskt

Questions?

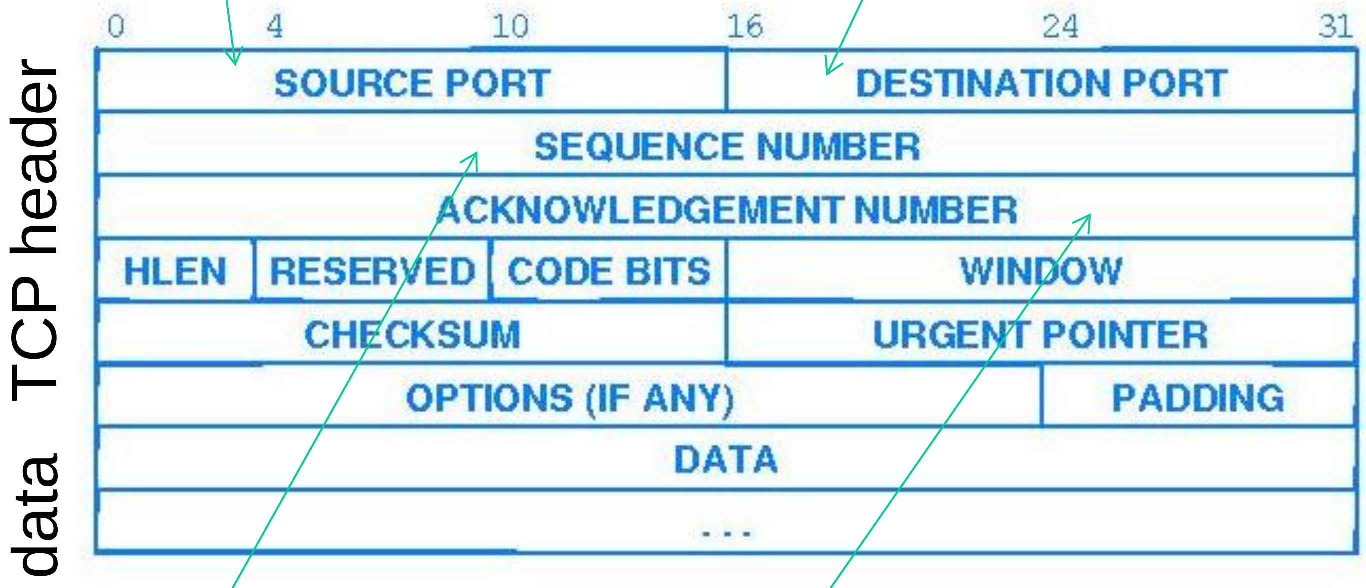
Appendix: Packet Formats

- Not needed to revise for the exam.

Formatet av TCP segment

Name of layer	Name of info-packet	Protocols	Services
Application layer	"message" (URL)	HTTP ------------> SMTP ------------> FTP ------------> DNS -------------->	Web document request & transf. Transfer of e-mail messages Transfer of files between 2 end users Domain name system
Transport layer	"segment" (Ports)	TCP ---------------> UDP -------------->	Connection-oriented transport of messages, guaranteed delivery, flow & congestion control Connection-less service, no reliability, no control
Network layer (IP-layer)	"datagram" (IP-Adr)	IP protocol -------> Routing protocols	Delivers the segment to the destination host by defining the fields in the datagram and how the end system and the routers act on these fields Determine the routes that datagrams take between sources and destinations

TCP port numbers at the ends of the connection



Position of the sender's byte stream of the data in the segment

Identifies the number of the octet that the source expects to receive next

Segments are exchanged to:
-establisch connections,
-Transfer data
-Send acknowledgements
-Advertise windows sizes
-Close connetions

IPv4 Datagram

Name of layer	Name of info-packet	Protocols	Services
Application layer	"message" (URL)	HTTP ------------> SMTP ------------> FTP ------------> DNS -------------->	Web document request & transf. Transfer of e-mail messages Transfer of files between 2 end users Domain name system
Transport layer	"segment" (Ports)	TCP ---------------> UDP -------------->	Connection-oriented transport of messages, guaranteed delivery, flow & congestion control Connection-less service, no reliability, no control
Network layer (IP-layer)	"datagram" (IP-Adr)	IP protocol -------> Routing protocols	Delivers the segment to the destination host by defining the fields in the datagram and how the end system and the routers act on these fields Determine the routes that datagrams take between sources and destinations

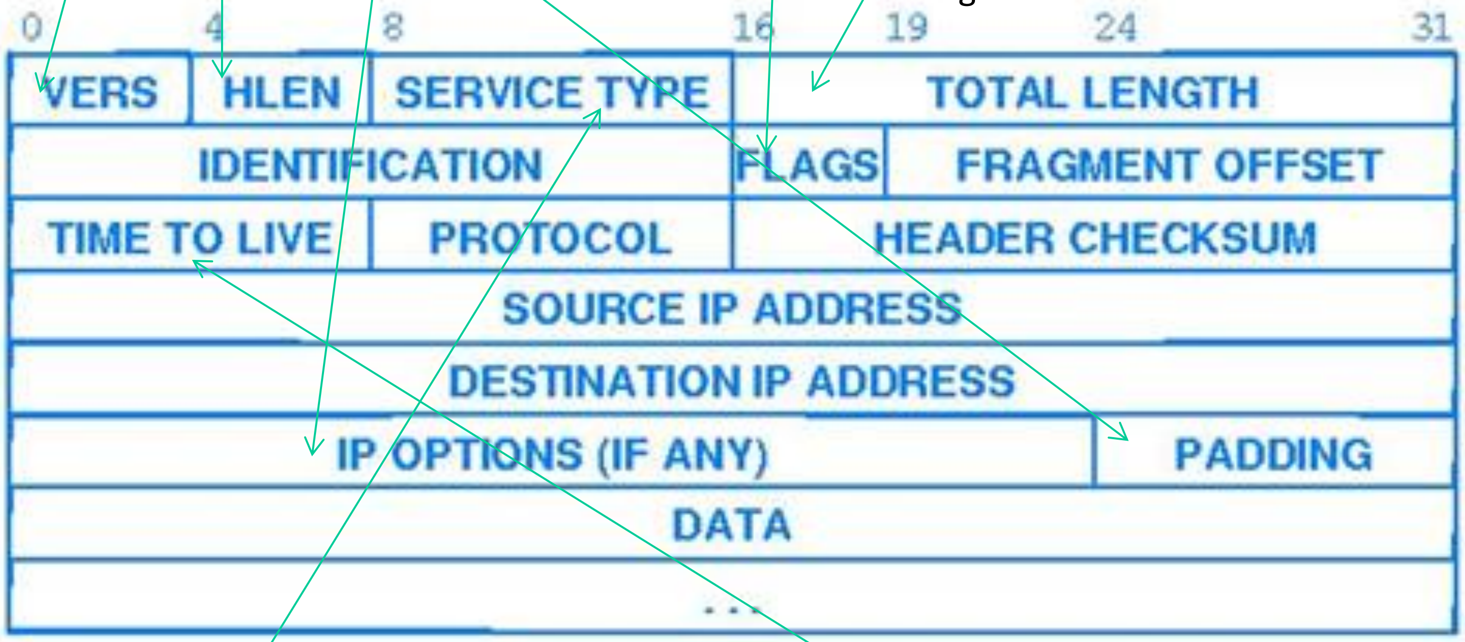
Version of the IP protocol

Header length measured in 32-bit words

Variable in length

Flag to indicate a fragmented datagram

Length of the whole datagram in octets,
16 bits long => max size is 65535 octets



=32bit=4 octet

8 octet

12 octet

16 octet

20 octet

24 octet=6words

8 bit = 1 octet ≈ 1 byte

32 bits=1 word=4octet ≈ 4byte

Type of Service (TOS)
To select forwarding algorithm which
helps to choose among various paths to a destination

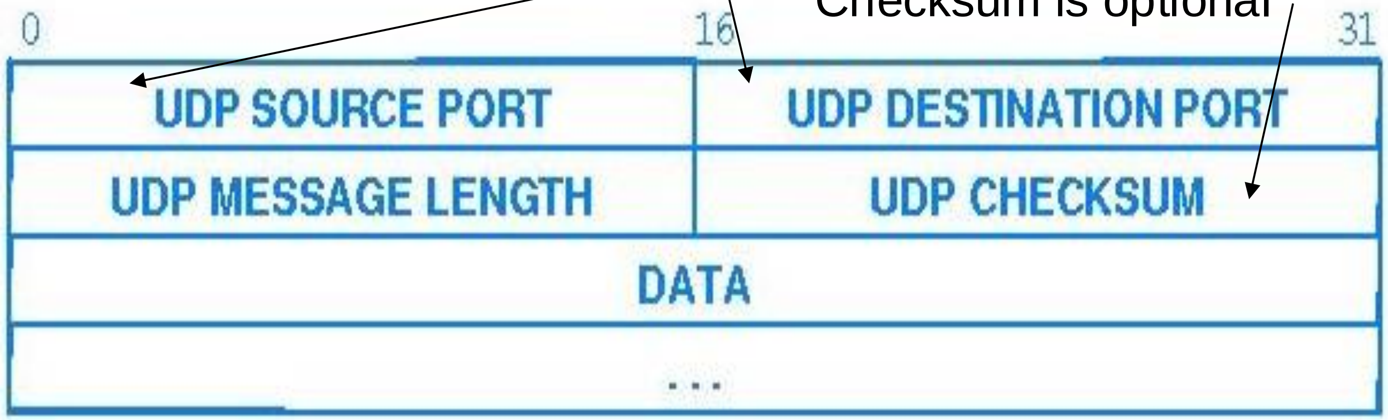
Specifies how long, in seconds, the
Datagram is allowed to remain in the internet system
Works also as a fail-safe mechanism.

Formatet av UDP Datagram

Name of layer	Name of info-packet	Protocols	Services
Application layer	”message” (URL)	HTTP ------------> SMTP ------------> FTP ------------> DNS -------------->	Web document request & transf. Transfer of e-mail messages Transfer of files between 2 end users Domain name system
Transport layer	”segment” (Ports)	TCP ---------------> UDP -------------->	Connection-oriented transport of messages, guaranteed delivery, flow & congestion control Connection-less service, no reliability, no control
Network layer (IP-layer)	”datagram” (IP-Adr)	IP protocol -------> Routing protocols	Delivers the segment to the destination host by defining the fields in the datagram and how the end system and the routers act on these fields Determine the routes that datagrams take between sources and destinations

UDP port numbers at the ends of the connection

Checksum is optional



Man kan kommunicera med Datagram över nätverk. Kommunikationen sker över bestämda portnummer.
Men denna kommunikation garanterar ej att:

- innehållet är oförändrat när det kommer fram.
- datagrammet når mottagaren.
- datagram kommer fram i samma ordning som de sänts.