Filesystem sharing

- Connecting a foreign filesystem transparently into local one
- Network File System (NFS)
  - originally developed at Sun Microsystems, today IETF
  - current version 4.1, RFC 8881, port 2049 (UDP, TCP)
  - source identification: server:path
  - authentication: Kerberos
  - note: relation (RPC) and presentation (XDR) layer
- Server Message Block (SMB)
  - originally developed at IBM, later adopted by Microsoft
  - open implementation Samba (UNIX)
  - source identification: UNC (\\server_name\source_name)
  - authentication: usually username and password

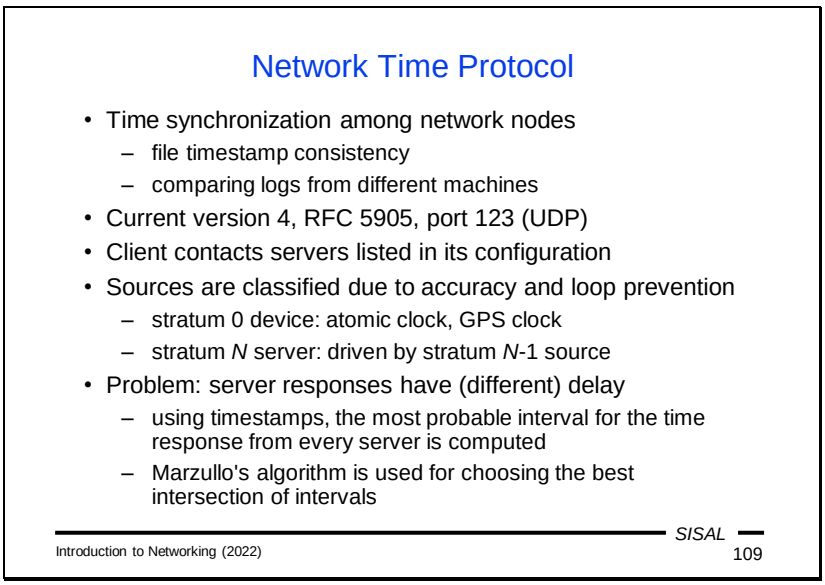
The last group of application protocols we will talk about are protocols that usually remain somewhat hidden from users.

A very important communication tool is the ability to connect a disk from another computer to your computer and work with it as if it were a local disk. The most used technologies for this purpose include NFS and SMB.

The Network File System protocol was born at Sun Microsystems, but over time it has become an open protocol described as RFC. It is still one of the most widely used systems of its kind in the UNIX world, unless the local environment requires a more sophisticated file permission system than the traditional UNIX system (in which case NFS is replaced e.g. by AFS, Andrew File System). NFS refers to a mounted disk as a *server*: *path* pair. For users this connection is transparent and they see a mounted disk as part of the local directory tree. Kerberos is usually used as the authentication mechanism. The NFS protocol works over UDP by default, although it can also be run over TCP. An interesting fact is that the internal structure of the protocol copies the OSI model in more detail and we can isolate the relational layer (Remote Procedure Call, RPC) and the presentation layer (Exchange Data Representation, XDR). RPC is a general mechanism used by other services – a client sends a request to a server to call a function, including a list of arguments, and the server performs the operation and sends a response to the client.

The Server Message Block protocol also comes from the commercial sphere, specifically from IBM. Its development was later taken over by Microsoft, so it was not published in the form of an RFC. However, an effort to connect the world of UNIX and Microsoft Windows led the Samba project to reverse engineer this protocol and their free implementation of SMB actually enabled such a connection: clients from both

families of systems can mount a disk from servers running on any system. A disk that is mounted via SMB is referred to using a notation *server**path*; authentication is performed by the system itself using a username and password.

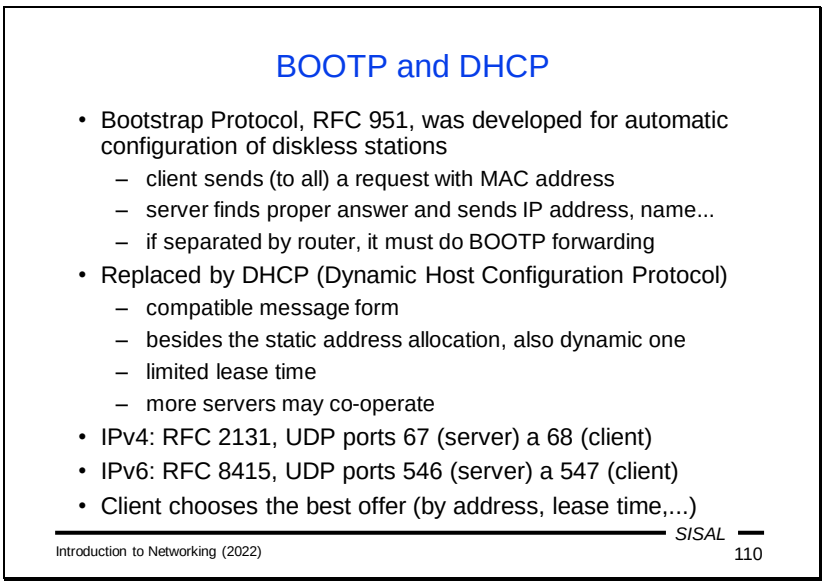


A very important feature of a local network is the synchronization of time on individual nodes. It is not necessary for the proper functioning of a network – if that were the case, then any desynchronization, which can occur very easily, would lead to a network breakdown. However, time synchronization is essential from the user's point of view, especially in two situations:

- If users transfer files between network nodes (and especially when sharing disks), it is essential that both computers have the same time. Otherwise, a file stored on one computer will be considered out of date on the other. Such a discrepancy could cause e.g. repeated update attempts or repeated unnecessary compilations of modules that have already been compiled modules, etc.
- Network administrators very often solve problems by comparing log records from different computers. If these records are written on machines with asynchronous clocks, it is a nightmare if for each line, the administrator must think about how many minutes he must add or subtract to which time to reconstruct the correct sequence.

One of the protocols that perform synchronization is the Network Time Protocol. The basis of the protocol is the fact that somewhere there are sources with absolutely accurate time (e.g. an atomic clock). These are called the source of *stratum 0*. From them, the time is taken by other servers (the source of *stratum 1*), which again serve as a source for the next level. The idea of marking a source according to its "logical distance" from an absolutely accurate source serves both to estimate the degree of accuracy of a particular source and as a protection against loops (the stratum *N* source will ignore the time from the stratum *N* + 1 source). A typical configuration for a LAN is based on running one or more NTP servers, they are synchronized with the NTP server(s) provided by the ISP, and all clients in the local network are synchronized using these servers.

Network latency must be taken into account when implementing the calculation algorithm. A client cannot simply take a time that it receives in a response verbatim because a certain amount of time elapses between sending a time and receiving it. The client calculation therefore uses timestamps from the response, which determine an interval in which the actual time probably lies. Using a technique called Marzullo's algorithm, these intervals are used for the calculation of the most accurate time with the highest possible probability. Since each client is not able to determine the time exactly, inaccuracy increases between the individual levels of NTP servers. From a practical point of view, however, these inaccuracies are fairly below the resolution of users.



The last technical application protocols we will deal with are BOOTP and DHCP. They are used by clients connecting to a network for obtaining an IP address that they can use and other information about the local network.

BOOTP, the Bootstrap Protocol was developed a long time ago, at a time when it was economically advantageous to acquire *diskless workstations*, i.e. computers that had no devices for permanent data storage (disks were relatively expensive at the time). It was not possible to save the configuration, including the IP address, anywhere on such a machine. The BOOTP protocol allowed each workstation to send an address assignment request, which the dedicated BOOTP server checked and possibly replied to. The server considered each request by verifying its MAC address against a whitelist – the MAC address was the only data that the client knew and that it could use for identification, because it was burned into its network card. If the client was on the whitelist, the server determined the IP address to be sent to it (the address assignment was fixed at that moment) and sent a response. Once the client received the response, it could start using the address.

From a technical point of view, it is important to mention that the client must send its request unaddressed, because it has no information about the addresses in the network where it is located. It will therefore use the so-called *limited broadcast* as the destination IP address, which means that the request will reach all nodes in the network (but they will ignore it). In order to prevent such requests from spreading meaninglessly throughout the Internet, routers do not pass them outside the network where they originated. This causes a small complication if we have a more complex subnet structure in the LAN, separated by routers. Then we must either have a BOOTP server in each subnet, or run so-called *BOOTP forwarding* on the routers,

where the BOOTP router forwards queries from the network where they originated to a certain BOOTP server and then forwards its response back to the client.

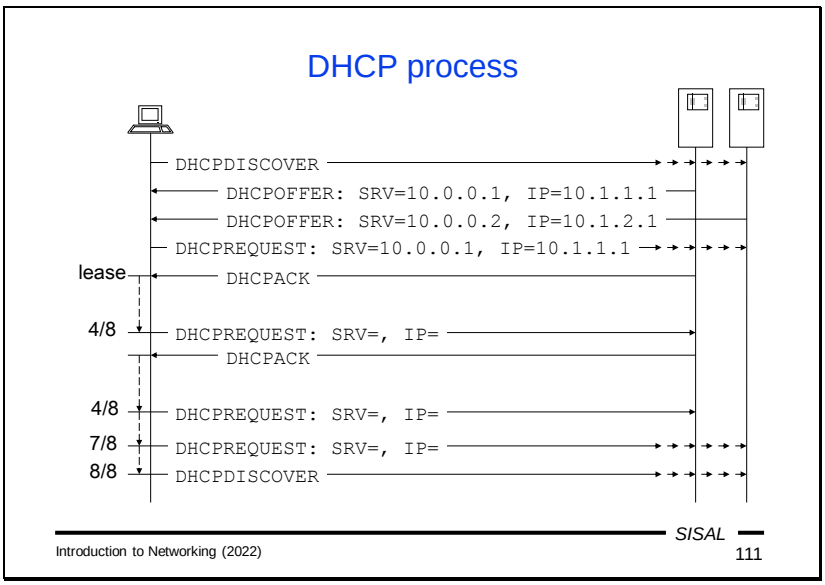
Over time, it turned out that sending just an IP address is not enough. We have already seen several examples of additional information useful for clients: addresses of routers, name servers, NTP servers, mail forwarders, etc. The protocol gradually expanded several times, until in a major change it evolved into DHCP, the Dynamic Host Configuration Protocol. The main DHCP extensions include:

- Dynamic address allocation. Fixed address assignment has lost its security significance today due to rewritable MAC addresses. In addition, network management often does not even know about all clients. And last but not least, with dynamic allocation a network can offer significantly fewer addresses than the number of potential clients.
- Limited rental time. Due to dynamic allocation, it is necessary to restrict clients' access to addresses for a certain period of time. The client therefore receives its address only for a certain period of time, the so-called *lease-time*, and after its expiration it must stop using this address.
- Cooperation of multiple servers. Multiple DHCP servers can operate in a network; they can differ in their purpose and the range of addresses provided, and the client may choose from their offers.

DHCP is backward compatible with BOOTP, which allows BOOTP clients to communicate with and receive the necessary information from DHCP servers.

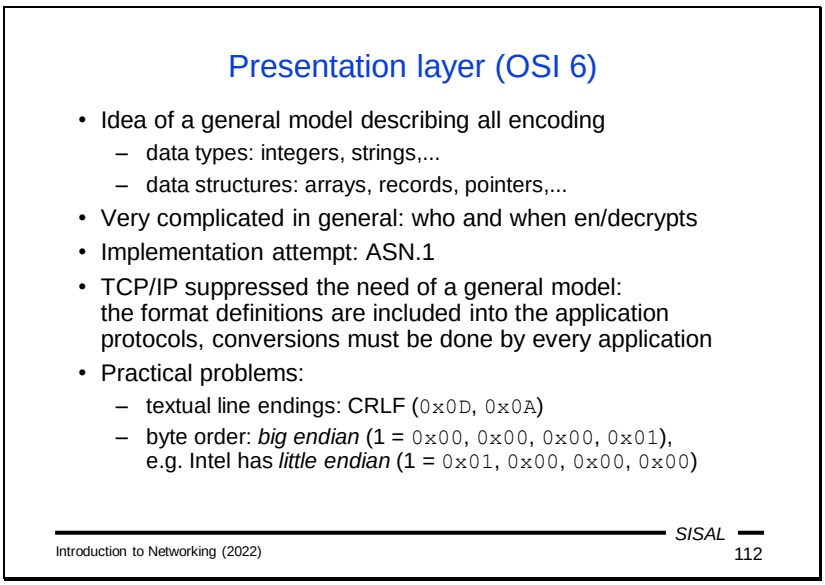
Today, DHCP is the most common way clients connect to a network. For instance, on Microsoft Windows systems, this option can be found under the somewhat obscure name "obtain IP address automatically".

Slide 111



Example of communication in DHCP:

- The client sends a DHCPDISCOVER request using the limited broadcast address.
- All DHCP servers in the network send their offers (DHCPOFFER).
- The client waits for a specific timeout, collects and checks responses.
- It chooses the best offer it has received. The specific algorithm may vary from client to client, but usually the client primarily prefers if a server offers an address that the client already uses (or has recently used). If it does not receive such an offer, it usually chooses according to the length of the lease-time.
- The client sends a DHCPREQUEST message that contains the address it has chosen. This message is still a broadcast because all servers must receive it. After sending their offer, they block the offered address for a while, so that they do not offer it to two clients. If a client has not chosen their offer, they must unblock the address again.
- The server, whose offer the client has chosen will then confirm with a DHCPACK message that the address is really still available.
- From this moment, the lease-time period begins.
- However, halfway through this time, the client should send a DHCPREQUEST message, this time only to the selected server, to make sure the address is still available.
- If it receives a response, a new lease-time period is started.
- If the client does not receive a response, it sends a new DHCPREQUEST within seven eighths of the lease-time period, but this time again by a broadcast.
- If it still does not receive an address, it must start the procedure again from scratch after the leas-time period has expired.



We move from the application layer to the OSI model layer number 6, the **presentation layer**.

The intent of the design was to create a general mechanism to shield the specific architecture of the network node from the format used by the lower layers. However, creating such a mechanism for completely general data communications, i.e. encoding various data types or structures, is relatively complicated. One of the attempts to solve it was ASN.1, which we talked about in connection with H.323. As we mentioned at the time, in terms of description it was a good attempt, but the implementation as a universal tool was far less successful due to its enormous complexity.

Therefore, the TCP/IP design did not go that way and embedded the necessary coding directly into the application protocols, so it moved the concern for encoding and decoding to the application implementation. Fortunately for the programmer, there are simple tools to do this. The most crucial differences between platforms lie in two aspects:

- Different operating systems use different characters or combinations of characters to indicate the **end of text lines**. Microsoft DOS followed the principle used by teletypewriters, mechanical typewriters and old printers: to establish a new line, it was necessary to move the so-called *carriage* to the beginning of the line and then turn the cylinder by one line. The result was the use of a pair of characters CR (carriage return, hex code 0x0D) and LF (line feed, hex code 0x0A) at the end of the each line. Microsoft Windows continues this tradition. The authors of UNIX evaluated the impracticality of this combination and introduced the use of only a LF character. For interest, we can mention old Apple systems, which, on the other hand, used only the CR character. Unfortunately, TCP/IP protocols began to use

the CR + LF model. This has some annoying consequences – e.g. how should a node behave when a counterparty sends the CR character? Should it wait to see if a LF will arrive as well? How long?

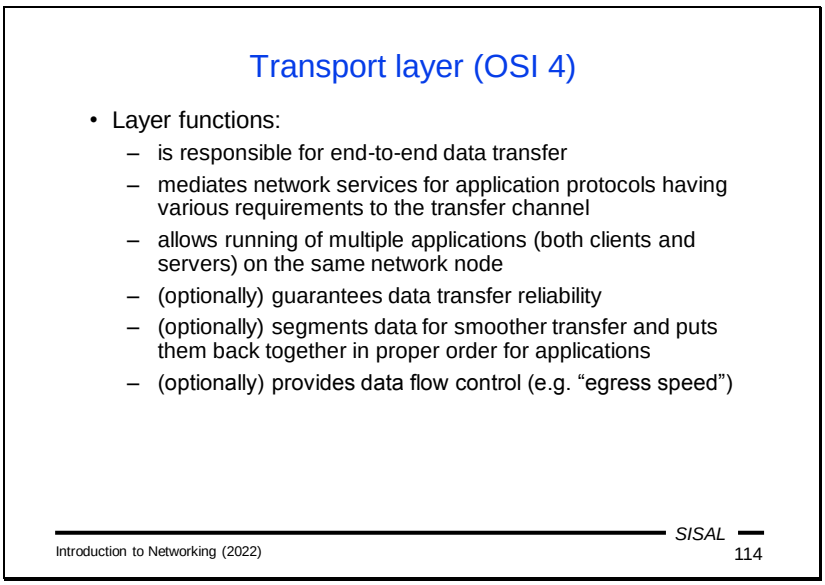
- Different hardware architectures use different byte orders for multi-byte values to store them in memory. With the exception of some obscure alternatives, there are two basic approaches: either the bytes are stored by placing the least significant byte first ("little end" first) and then the next, or vice versa ("big end" first). TCP/IP protocols use a **big-endian** system, i.e. the first byte of an IP address goes through the network first. Libraries may help a programmer to perform an appropriate conversion if the compilation is performed on a little-endian system.

Session layer (OSI 5)

- Idea of a general dialog model
  - one dialog can consist of more connections
  - one connection can carry more dialogs
- TCP/IP suppressed the need of a general model: the dialog principle has been included directly into the application protocols, e.g.:
  - within one SMTP connection a client can send several mails to the server
  - SIP initializes dialog using more partial media data channels

The **relational layer** met a similar fate as the presentation layer. The idea of a general tool that can control the dialogue of communicating parties for all types of application protocols has not proved to be practically realizable, although we can find this functionality in some protocols (like RPC in NFS).

In the TCP/IP model, therefore, the functions of OSI 5 as well as OSI 6 are integrated directly into the application protocol. Examples of situations where the logical dialog of the communicating parties does not correspond to one connection can be found in SMTP (the client transmits several separate messages to the server within the same TCP connection) or in SIP (one video call can be realized by one control connection and two data channels for audio and video). All application protocols then describe the actual control procedure of the dialog flow.



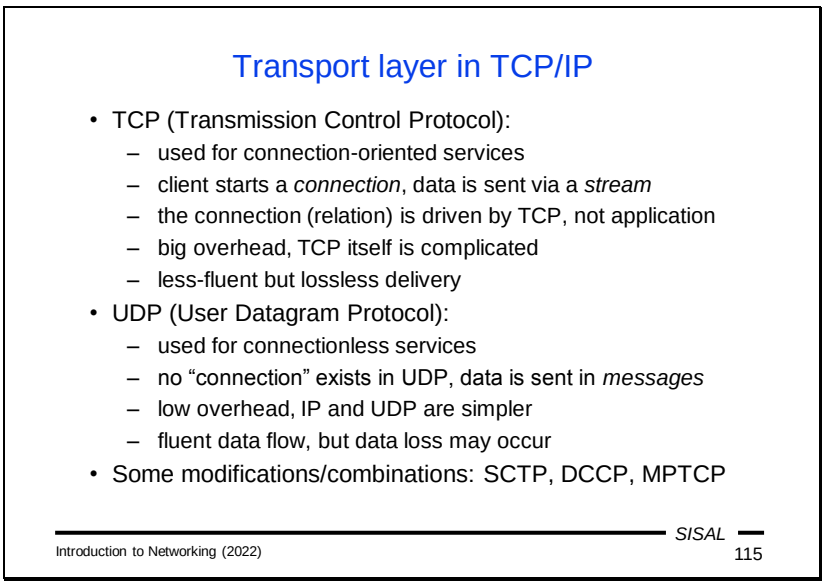
A task of the **transport layer** is to provide applications with a communication channel for the transfer of data units between applications running on end devices. It can be implemented using different protocols, which differ in the scope of services provided.

The interface between the transport and application layers generally gives an application the ability to send a block of data to the counterparty; the specific parameters and conditions depends on the needs of the application. The fundamental difference between various transport layer protocols lies in the delivery guarantee. Some protocols (such as TCP) provide a so-called *reliable* service, which means that the interface function returns a value that indicates whether the data was delivered successfully or not. Some protocols (such as UDP) provide an *unreliable* service, so the function returns only the result of **sending**, not **delivering** data. Delivery detection and possible response to a delivery failure must be handled by the application itself.

An obvious function of the layer is *multiplexing*, in this case access to the network for various connections between local and remote sockets (clients and servers). The individual communication channels are distinguished via **ports**.

Some protocols (e.g. TCP again) perform data *segmentation*. They can receive blocks exceeding the size of data units that can be sent over the connected network, divide the data into smaller blocks (segments) and send them separately. The receiving side reassembles them into their original form and informs the sending party whether the data has arrived, so that it is able to resolve a possible failure of a segment.

An additional protocol function may be *flow control*, a mechanism for changing the data transmission rate so that the communication channel is used efficiently without congestion.



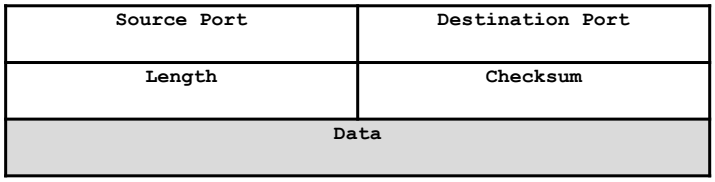
The major protocols used at the TCP/IP transport layer are TCP and UDP.

TCP is used for **connected applications**. To illustrate, a connected application can be compared to a phone call. The caller dials the number, his telephone operator establishes a connection to the target device, and as soon as the call is connected, the user starts an "application", i.e. starts talking and listening. The application has a very simple job: it does not have to worry about whether individual sentences arrived and whether they arrived in the correct order. Applications working over TCP behave similarly – the client establishes a connection and a so-called *stream* is created, a channel through which data flows in both directions. TCP is *reliable*, which means that if an application does not receive an error return code, it is sure that all data has arrived in order. The overhead for this guarantee is fully on the TCP side, and is therefore quite complicated and robust; applications, on the other hand, can be relatively simple. Of course, TCP overhead (acknowledging delivery, waiting for acknowledgements, forwarding lost data) reduces logical transmission capacity and can cause fluctuations in delivery frequency. A disadvantage of TCP is also that an application has a very little ability to react to the state of the network – once it calls a transfer function, it has no choice but to wait until the data is transferred or the appropriate timeouts expire and the function announces a failed delivery.

UDP is used for **connectionless** applications. An analogy is communication using traditional mail. If we drop a letter into a mailbox, it corresponds to calling the UDP send function and the result is only “succeeded to send" or "failed to send". If we send more letters, we have no guarantee that they will all arrive and that they will arrive in the correct order (and in the computer world it may even happen that some messages will be delivered twice). If we want to build a reliable channel in this way, we (the "application") must take control. In UDP, separate messages are sent; UDP

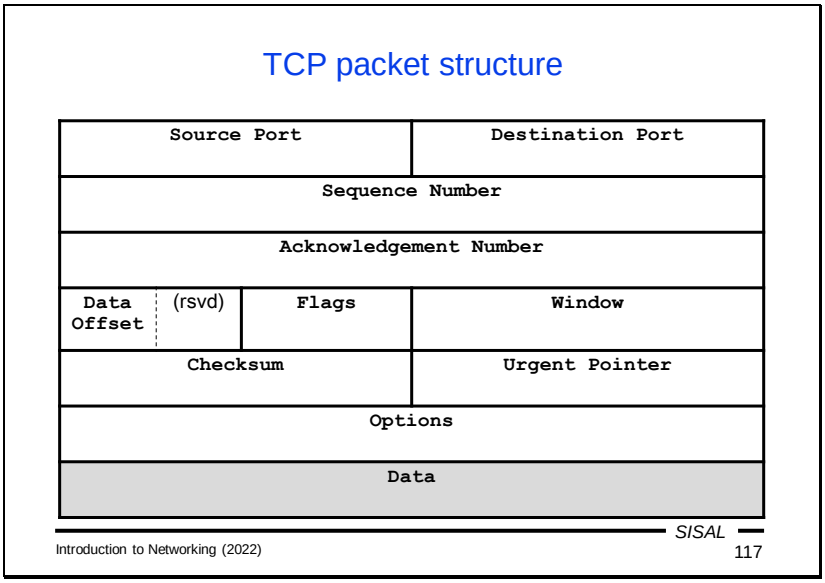
itself handles only transmission, so it has very little overhead. All overhead is transferred to the application, which also has the advantage that the application can respond more flexibly to the current state of the network. Unlike TCP, data in UDP can flow more regularly, and any loss must be resolved by the application – either it should arrange the retransmission itself, or it must be able to work without the missing part of the data.

UDP datagram structure



From the previous description it is clear that UDP does not need to add a lot of information to the data from the application layer for its work and the encapsulation of application data into the PDU of the transport layer is very simple.

In the UDP header, only multiplexing information, i.e. a source and destination port, and control information (length and checksum) are transmitted.



The TCP header contains a number of additional information fields compared to UDP.

In order for TCP to guarantee the completeness of the transmission, a unique identification number must be added to each segment. Roughly said, each segment contains a relative **offset** to the beginning of the stream (in the header we see the so-called *Sequence number*). In order to be able to confirm delivery as well, we need an analogous field for the opposite direction (the *Acknowledgement number*).

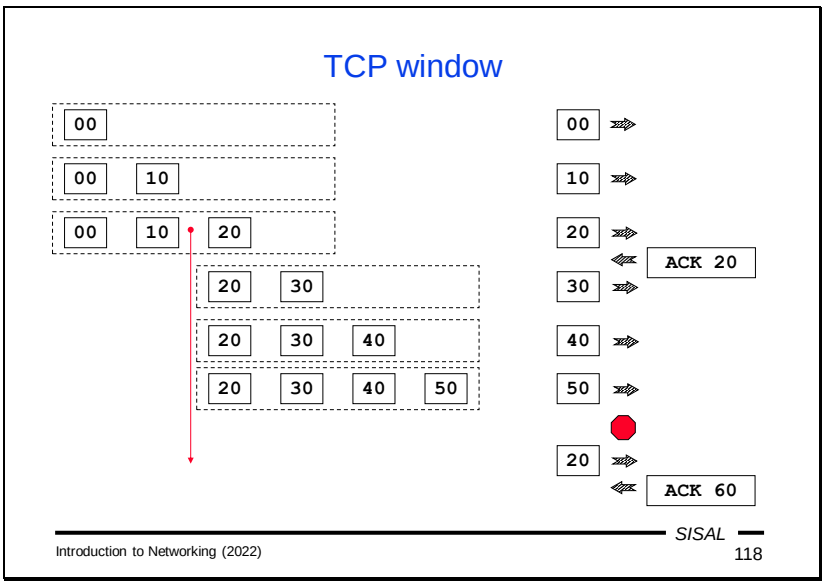
The *Flags* field contains flags, most of which we will discuss immediately.

The *Urgent pointer* field is used for an *out-of-band* transfer function. The application can mark certain data as **urgent** with the URG flag. For instance, the command by which a FTP client wants to interrupt a data transfer is sent as urgent. Such data is then inserted into the normal communication and its relative address within the data block is written in this field.

We will get to the other important fields of the header in more detail later.

Note: I present the structure of the TCP packet here to demonstrate its functions; I am definitely not going to examine the position of individual fields.

Slide 118



Let's take a closer look at how delivery acknowledgment and data flow control are handled in TCP.

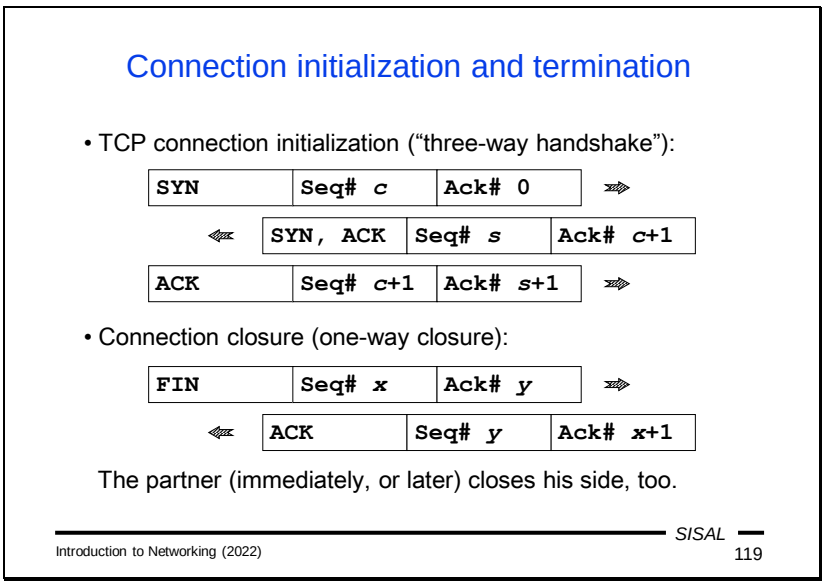
As already mentioned, TCP acknowledges the delivery of data. The recipient acknowledges the delivery of a block by sending a packet to the sender with the ACK flag set and the Acknowledgement number value set to the end of the data offset that was delivered. However, if these acknowledgements were sent for each packet in a separate packet, it would make unnecessary traffic. Therefore, it is possible to pack the confirmation (flag + offset) into some other data that the recipient needs to send back. The usual implementation is that a computer will wait a certain amount of time before sending an ACK as long as another data item that will need to be sent to the other side does not appear to connect the ACK to. If this does not happen within a given limit, it sends the ACK separately.

Similarly, the communication channel would be used inefficiently if the sender had to wait for confirmation of the previous packet before sending any new packets. Therefore, TCP allows both parties to agree on a range of data that the sender may send without waiting for confirmation. This limit is called a **window size**, and a node may announce a proposed size in the *Window* field in the TCP header.

Consider a situation where one side of the connection starts sending blocks with a data size of 10 bytes (in reality, of course, larger blocks are used) with a window size set to 40 bytes. The sender starts sending blocks without waiting for confirmation, but monitors the maximum window capacity. If a delivery receipt arrives in the meantime, it moves the "window" to the position it contains. For instance, if the acknowledged offset was 20, it can now send data up to offset 60. If no further acknowledgement arrives by then, it must stop sending and start waiting for a new acknowledgement. If

the acknowledgement does not arrive, it must resend the first unconfirmed block of data.

Slide 119



The term "TCP connection" refers to a situation where the client and server have confirmed their willingness to communicate with each other and have agreed on the sequence numbers to be used. In fact, sequential numbers do not start from zero, for security reasons, but from a **random number** chosen by the sending party. Therefore, it is necessary to send the initial value to the counterparty. This agreement takes place at the beginning of the connection using three special packets that have an empty data part and carry information only in the header, and is called a *three-way handshake*.

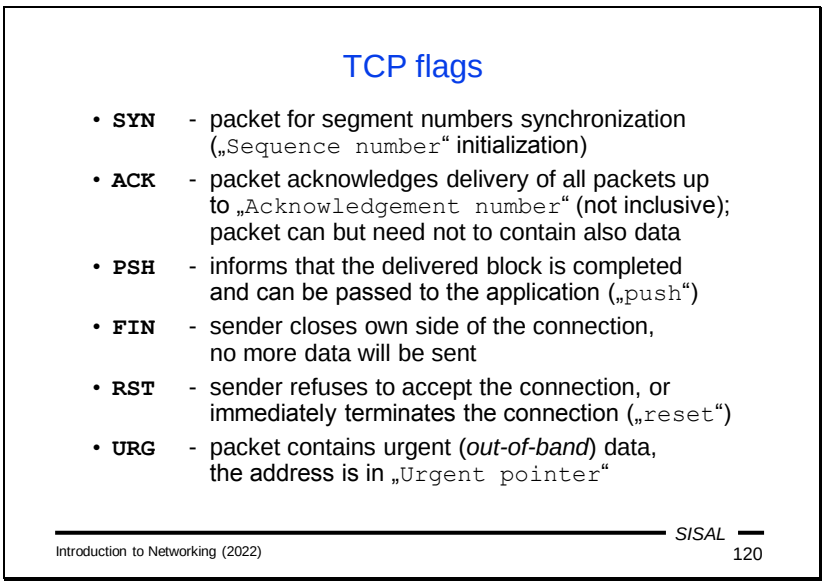
The first of these packets has the SYN (synchronization packet) flag set and has the initial value selected by the client as the Sequence number; let's call it *c*. The server acknowledges receipt by sending a packet with the ACK flag and the Acknowledgement number set to *c* + 1. At the same time it generates its initial value of the sequence number (*s*) and adds the SYN flag as well. The client then completes the procedure by sending a packet in which it acknowledges receipt by sending an ACK with a value of *s* + 1.

From now on, both parties can send data and gradually increase the values of the sequence numbers by the data length.

If either party wants to end the connection, it sends a packet containing the FIN flag. This tells the other party that the sender no longer intends to send **any data**. Usually, the other party immediately sends a FIN packet as well, but if it does not want to terminate the connection, it can continue to send its data. In that case, the party that FIN has already sent will have to forward its ACK **packets** so that the connection does not break. We can easily imagine this situation in HTTP, for example. After a client sends its request, it can close the connection and just wait for a server to

respond. This condition is called a *half-closed* connection and is valid, although not fully recommended, as various control mechanisms along the way can consider the connection as dead and terminate it. Therefore, for example, the mentioned HTTP client should leave the connection open and close it only when it has already received the entire response.

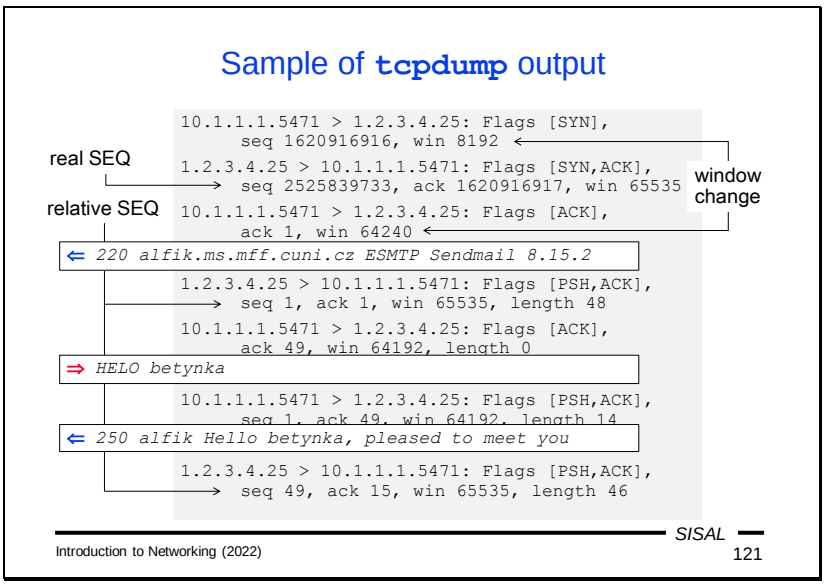
Slide 120



In addition to the already mentioned flags URG, SYN, ACK and FIN, the following are also important:

- PSH (push) - With this flag, the sending party notifies the other party of the last segment of a data block. For the recipient, this means that if it has already received all the other segments of the block, it can pass the block on to the application for further processing.
- RST (reset) - With this flag, the sender notifies the other party that it does not intend to continue communication. The flag can be used by a server during the three-way handshake as a response in case it does not want to accept the connection, or by either party at any time during the communication. Unlike termination with FIN, RST means an immediate termination without waiting for confirmation or for FIN from the counterparty.

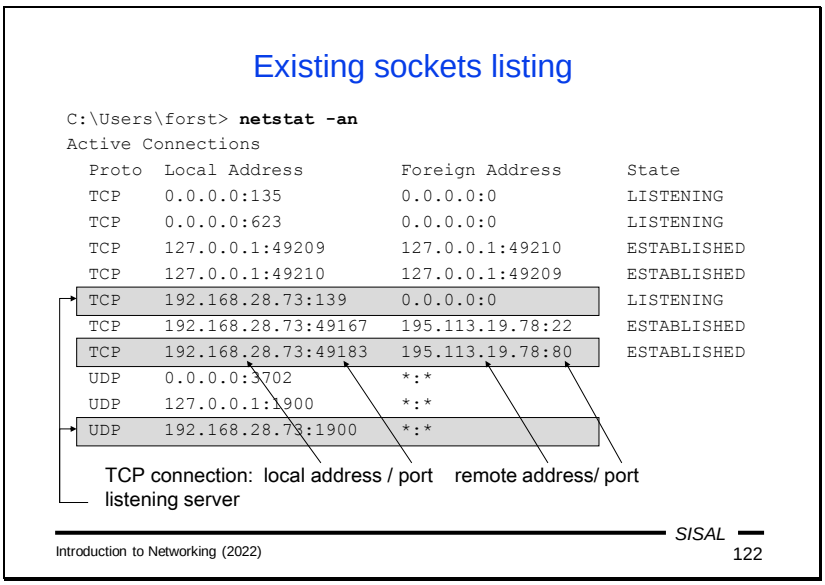
Slide 121



If we want to look at a TCP/IP communication in more detail, we can use a program that can capture and display it in a readable form (e.g. tcpdump, Wireshark, WinPcap). Let's look at an example of the output of the first of these programs…

- The first packet is the first packet of a three-way handshake, which we can recognize by the fact that it carries only the SYN flag. At the beginning of the line, we see the source IP address and the dot-separated port number, and after the arrow, the destination IP address and port. This is port 25, so this will obviously be the first packet of an SMTP connection. We also see the initial sequence number (**seq**) and the window size (**win**).
- The second packet is the server's response (it carries the SYN and ACK flags); we see the confirmed client sequence number (**ack**) and also a proposal of a larger window.
- In the last three-way handshake packet we already see the value of the Acknowledgement number **relatively**. Tools such as tcpdump display sequence and acknowledgement numbers as offsets relative to the beginning of the communication (subtracting the initial value) for better readability.
- The next packet is the server's introductory message. For the first time, we see a data length printed (**length**, 48 characters) and the PSH flag (push), because the line is complete.
- The client apparently lingered a bit with the sending of its first command, so the TCP software sent a separate delivery acknowledgement for the start line from the server, in the fifth packet. It used the initial value plus the length of the previous data (in relative numbers 1 + 48 = 49) as the Acknowledgement number value.
- In the sixth packet, we see the first client command.

- The seventh packet contains the server's response. In addition to the data itself, it also carries the ACK flag and a value that confirms to the client’s TCP layer the receipt of the previous 14 bytes (in relative numbers 1 + 14 = 15). The server therefore combined the sending of the acknowledgement with its own data.



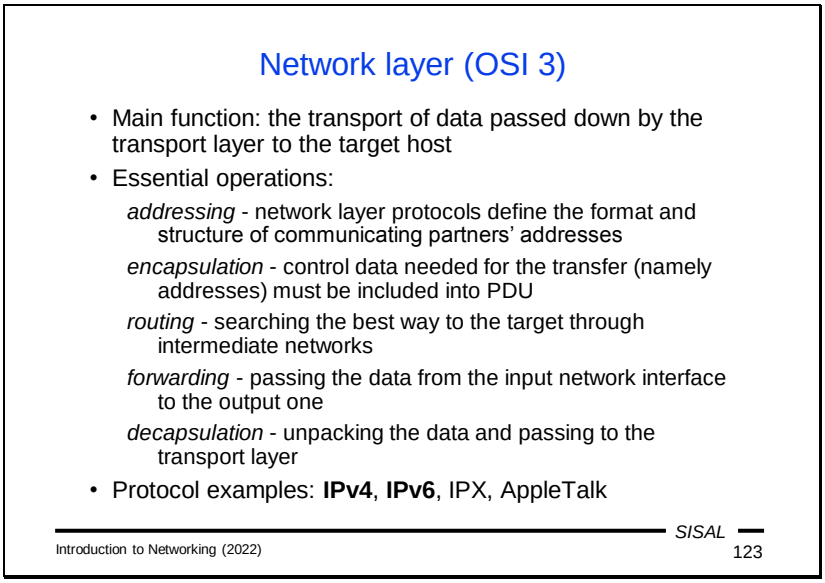
If we want to get an idea of what connections are currently open on our computer, we can use the **netstat** command. With the **-a** parameter, it lists all TCP and UDP servers and open TCP connections (there are no connections in UDP).

For TCP as a state protocol, the listing also contains the state of the connection in the last column; in the example we see LISTENING (a listening server) and ESTABLISHED (an open connection), but a number of other states exist.

In the second column we have the address of the local socket; there we see either the actual address of a network interface or the address 0.0.0.0, which means that the server is listening on all interfaces that are on our computer.

The third column is the socket address of the counterparty, here is either the actual address and port of the remote socket, or in the case of the server the value 0.0.0.0 or *, which means that any client can connect.

In the case of UDP, we only have running **servers** in the list, because UDP has **no connections**, so netstat has no information about the exchange of messages between clients and servers.



The task of the **network layer** is to find a path to a destination computer so that it is possible to deliver data passed by the transport layer regardless of technologies used in the lower layers.

The layer stands on two basic pillars:
- Address scheme – a way to identify individual nodes in the network so that it is possible to distinguish which network they belong to.
- Routing – a way to find a valid path from a source to a destination network based on the address.

Other features of the layer include encapsulation, which we have mentioned before, and **forwarding**; that is the principle that a router can receive a packet even though it is not the packet's final destination and tries to deliver it one step further to the destination as if the packet originated directly on the router.

Internet Protocol (IP)

- Properties:
  - connectionless (datagrams are delivered independently)
  - best effort (unreliable, logic delegated to higher layers)
  - media independent (higher layers need not to bother with it)
- Addresses:
  - contain network address part and node address part
  - IPv4: 4 bytes, IPv6: 16 bytes
- Assignment:
  - central: IANA (Internet Assigned Numbers Authority), department of ICANN
  - regions: RIR (5x, Europe: RIPE NCC)
  - further: ISP
  - local network: network management (manually/automatically)

In the TCP/IP protocol suite, the Internet Protocol (IP) is used at the network layer. This service is

- connectionless – individual datagrams are delivered independently, no formal connections are created,
- unreliable – the network layer does not guarantee delivery (as we have seen with UDP, which takes this property from IP),
- independent of the medium – the transport layer generally does not have to deal with the details of the technology used (except for the MTU, which we will talk about later).

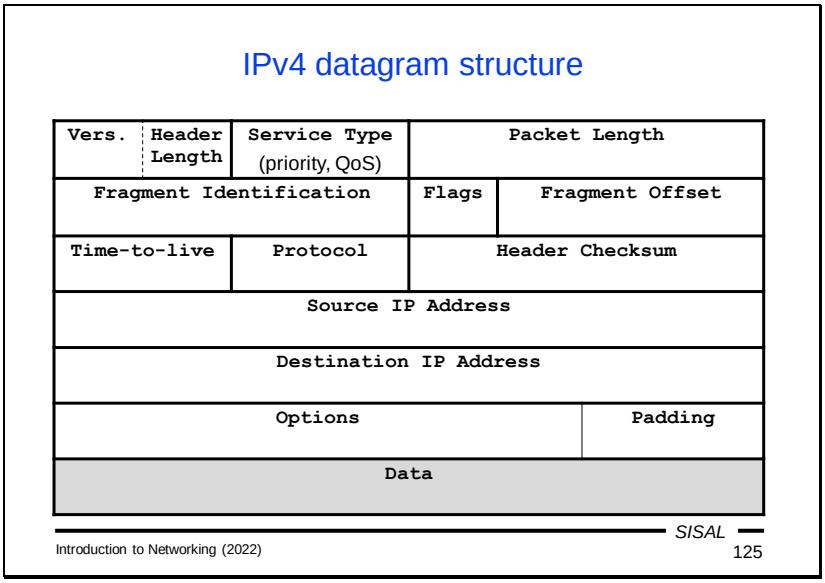
There are currently two versions of the protocol: version 4 has an address length of 4 bytes, and in version 6 is it 16 bytes. For both versions, there is a method to determine the **network address part** (which is responsible for routing) and the **host address part** (which is responsible for delivery to the destination node) of any address.

Each node that wants to communicate in a TCP/IP network must have an IP address. The method of assigning addresses to connected end stations is decided by the network administration. Each network uses certain ranges of addresses for its nodes – either private ones (these are chosen by the network administration itself) or public ones (these are assigned by the ISP that connects the network to the Internet). ISPs receive blocks of addresses from superior ISPs; at the top of the hierarchy is IANA (the Internet Assigned Numbers Authority, part of ICANN), which has five regional registries under it:

- RIPE NCC manages Europe, Russia and West Asia
- APNIC manages the rest of Asia, Australia and Oceania
- ARIN administers the USA, Canada, Antarctica and part of the Caribbean

- LACNIC manages Latin America and the rest of the Caribbean
- AFRINIC manages Africa.

Slide 125



Encapsulation at the network layer adds an IP header that contains information important for delivery. In the case of the version 4 protocol, the header contains the following information:

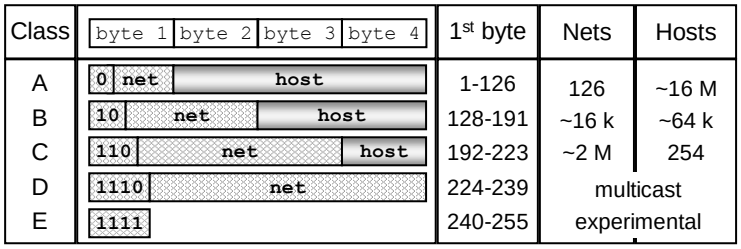
- Version. Half a byte is reserved for the version, so a different format will be needed for IP version 16.
- Header length. It takes up the second half of the first byte. Looking at the diagram, it is probably clear that a maximum value of 15 would not be enough for the entire length. This is really the case, which is why the length is given in **32-bit words** and not in bytes. This means that the maximum length of the header is 60 bytes, and if its content is not a multiple of 4 bytes, it must be padded.
- The second byte contains QoS information and the rest of the first word is filled with the length of the entire datagram.
- The second word is dedicated to **fragmentation**. This is a procedure that applies when the network layer receives a packet so long that when it is encapsulated, we would get a frame longer than the maximum allowed for a given link layer (the so-called Maximum Transmission Unit, MTU). In this case, the network layer must fragment the packet into multiple datagrams and send them sequentially. Fragmentation is an unnecessary complication if it can be avoided. This is the case with TCP, which segments the data itself, so it is possible to instruct it to use a correct segment size. Therefore, a **Path MTU** search method is used – packets are sent with the *Do not fragment* flag, thanks to which the sender learns about possible problems with the size of the MTU along the path and can correctly determine the segment size to be passed to the transport layer.
- The third word contains the Time-to-live value, to which we will return, the protocol number that is encapsulated in the datagram, and the checksum of the header.
- This is followed by the IP addresses of the sender and recipient.
- Additionally, the header may contain options.

Note: I present the structure of the TCP packet here to demonstrate its functions; I am definitely not going to examine the position of individual fields.

Slide 126

IPv4 addresses

- Originally: one byte
- 1975 (RFC 687): three bytes („*This expansion is adequate for any foreseeable ARPA Network growth.*“)
- 1976 (RFC 717): one byte (network) + three bytes (host)
- 1981 (RFC 791): classes A, B and C



The situation in which computer networks began to be built is nicely illustrated by the fact that originally **a single byte** was reserved for the computer address! Even the authors of the original proposal clearly could not imagine how far our daily lives will be affected by what they were just beginning to build.

In 1975, it was decided that the address should be extended. The new length was set to three bytes, and RFC 687 contains a note that this extension was considered **adequate for any foreseeable growth** of the network, which is amusing from today's perspective!

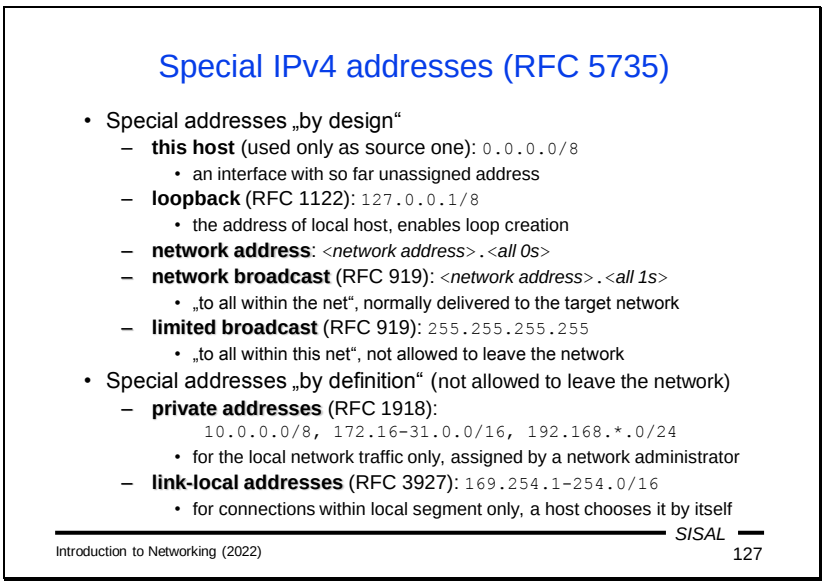
A year later, the address size has expanded again. Not that the addresses were running out so quickly, but addresses were supplemented with a section for a network address (1 byte) and the original 3 bytes remained for a host address.

Another five years later, the number of networks turned out to be growing much faster than expected, the original model was refined and three classes of addresses were created for networks of various sizes. Class A covers the lower half of the address space and uses the original 1:3 division. Class B occupies the third quarter of the space and each class B address is divided into 2-byte portions. Class C occupies the penultimate eighth of the address space and divides addresses by a ratio of 3:1, which means that it offers about 2 million networks each containing a maximum of 254 computers.

Note: You might expect 256 instead of 254 for the number of addresses on a C network, but the first and last addresses on each network are reserved.

The last eighth of the address space was reserved for further development, and in 1986 class D for so-called *multicast* addresses was separated from it. Class D addresses completely lack a host address part, because they are used to address a group of recipients – for special services (e.g. NTP uses the address 224.0.1.1), video conferencing, etc.

Slide 127



Some IP addresses have a special meaning according to the protocol definition:

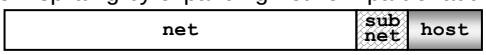
- An address with all zeros is used as the source in a situation where we need to communicate but we do not yet know our address.
- The address 127.0.0.1/8 is reserved for a special interface existing on each node of the network and representing "**this computer**", the so-called *loopback address*. When we need to establish communication between a client and a server both running on our computer, they can both use this address.
- An address with an all-zeros host part is the *network address*.
- The last IP address in the address block of a (sub)network represents a *network broadcast*, an address that we can use if we want to address all computers in the network.
- In addition to the network broadcast address, which is normally delivered across networks, there is also a ***limited broadcast*** (255.255.255.255) that must not leave the network where it originated.

In addition to these special addresses, there are two categories of addresses whose meaning is assigned by an organizational decision, not by the IP architecture:

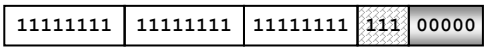
- The first group are *private* addresses, which include one class A network, 16 class B networks and 256 class C networks. As we already know, these are used in local networks and a packet with a private address can only leave the network in special cases. Otherwise, address translation (NAT) must be used
- The second group are *link-local* addresses. These addresses are freely usable; each computer can take any of these addresses and start communication on the network segment where it is connected. Unlike private addresses, however, they cannot be used to communicate outside the network at all.

Subnetting

- Network splitting by expanding network part of address:



using so called network mask (*netmask*), in this case 255.255.255.224:



- Subnets "all-zeros" and "all-ones" are not recommended, so we have here just 6 x 30 addresses (70%)
- Non contiguous mask is possible, but usually not used
- Nowadays, the classes are often ignored (*classless* mode), using only the prefix length in bytes (e.g. 193.84.56.71/27)
- The term *variable length subnet mask* (VLSM) describes situation when various masks are used in a network
- Moving the border in opposite direction: *supernetting*

Over time, it turned out that even a finer division into classes was not fine enough, and especially in local networks, it is necessary to move the boundary between the two parts of the address even further "rightward", to create several **subnets** from one network. In this case, however, it is not enough to set an address when configuring a network interface; you must also specify the network size. One way to do this is to add a so-called **netmask**, a number that contains ones just where the network address is. For example, if we want to shift the network boundary of a class C address by three bits, we use the value 255.255.255.224. The choice of input method is quite logical – if we have a host address and want to know if it belongs to our network, we just take the netmask and the host address, perform a *bit AND* operation, and if the result is our network address, the host address is "ours". However, calculating a netmask is a bit complicated, so today it is often replaced by the so-called *classless* format, where we write only the number of bits that make up the network address part after a slash.

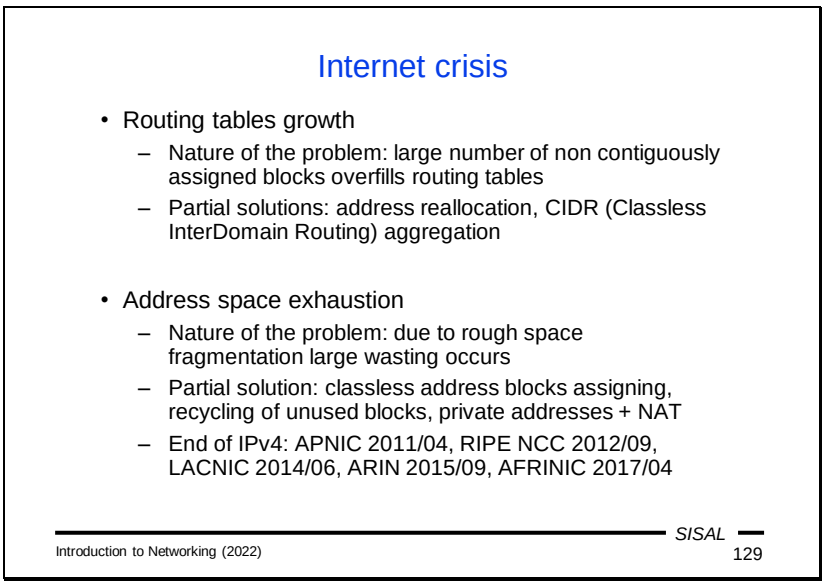
Using subnetting allows you to divide a network into smaller parts, but reduces the number of addresses available. Since the use of a subnet numbers with only zeros and only ones is not fully recommended, a subnet mode in our example will reduce the network to 6 subnets of 30 computers.

Moreover, this division is again very rough, and if we need a more flexible division, we will not be able to do with one mask. For instance, if one of our 6 subnets has more than 30 computers, but two have at most 14 computers, one class C address will not be enough for us with subnetting using a fixed mask. We will have to use the so-called *Variable Length Subnet Mask* (VLSM) mode and use the /28 mask in two small subnets, and /26 in the large network.

The natural question is whether the border can also be moved in the opposite direction, i.e. to join networks instead of dividing them. This is useful e.g. in a situation where I have more than 254 computers in a LAN and I have two class C networks for them. If I did not use *supernetting* and connect them to one network with the /23 mask, I would have to deal with unnecessary complications when communicating between two computers which, although in one LAN, happen to have addresses from different class C networks.

Fortunately, private addresses are used extensively in LANs today, where there are a sufficient number of networks of all classes, so subnetting does not need to be addressed as often.

Note: If we are handling an IP address that does not belong to our network, of course, we cannot know what subnetting its local network uses. But that doesn't matter, because we don't care about it. From the point of view of routing, we communicate with a foreign address as if it does not use any subnetting – our task is to deliver the packet to the target network and the final delivery is already fully under control of that network.



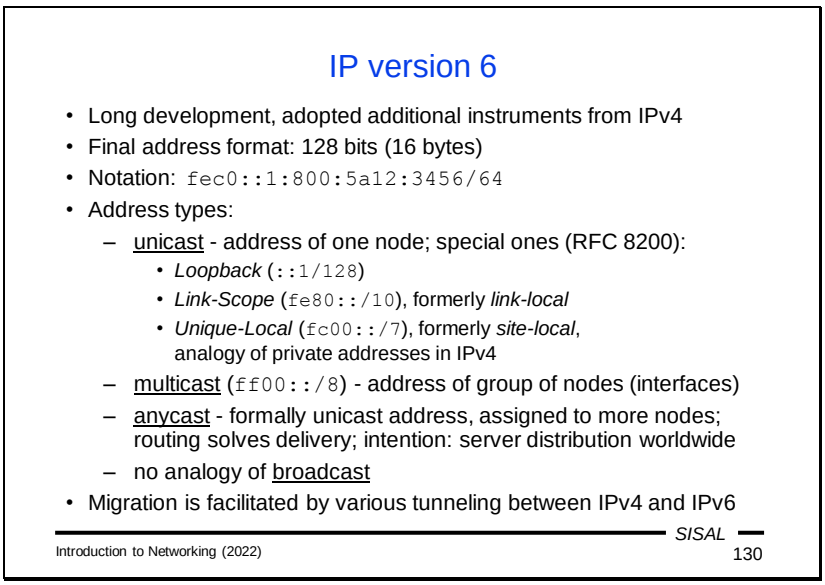
The general feeling of IP address sufficiency in the spirit of the daring sentence from RFC 687 meant that IP address allocation was not governed by any strict rules at that time. At the turn of the 80s and 90s, however, it began to appear that this was beginning to bring problems in two ways.

Tables of central routers began to overflow. This could be prevented by the approach that if in reality some networks were adjacent to each other and they also had adjacent numerical values, it was possible to **aggregate** their records on most routers, i.e. actually do supernetting over them and combine both records into one with a shorter network mask. This principle is the so-called Classless InterDomain Routing (CIDR), but its effective implementation had to be preceded by massive renumbering of many networks on the Internet, so that they could be effectively aggregated at all.

At the same time, it became clear that allocating arbitrary ranges to any network would sooner or later lead to exhaustion of the IPv4 address space. ISPs started to save addresses, renumber networks, allocate blocks that did not correspond to IP classes, etc. Fortunately, the use of private addresses and NAT began to be massively introduced, which dramatically reduced the need for public IP addresses (a network with one thousand computers needs with NAT only one address instead of one thousand).

In spite of that, however, it was clear that these methods are able only to postpone the problem, not to solve it, and intensive work began on the preparation of the so-called IP next generation (IPng).

Slide 130



The creation of a new version of IP took quite a long time, as well as its subsequent introduction into various operating systems and debugging problems. Fortunately, CIDR and NAT postponed the problem, so even before IPv6 could actually be deployed, both the protocol and network software was well prepared. In addition, the long co-existence of both versions was expected and the transition is facilitated by various variants of IPv6 tunneling to IPv4 and vice versa. In addition, the authors of IPv6 took a number of additional tools and features from the previous version and, after a thorough analysis, modified them and incorporated them directly into the IPv6 design.

In its final form, an IPv6 address has 16 bytes and is normally written in 16-bit words in hexadecimal with CIDR mask notation. In order to be able to shorten the notation of current addresses containing long sequences of zeros, a single series of null words can be omitted (so that there are two colons in a row in the notation).

In terms of address types, we find many identical and different features in IPv6. Basic, *unicast* addresses include, in addition to public ranges, a loopback address, link-local addresses and unique-local addresses. The latter group is a certain analogy of private addresses in IPv4. Multicast addresses play a greater role than in IPv4. IPv6 lacks **broadcast** addresses and they are de facto replaced by the multicast address ff02::1 ("All IPv6 devices").

Anycast addresses are a complete novelty. They are used in various situations where a resource is available in geographically separated locations and we would like the network to choose the most suitable resource by itself. One anycast address can be set for all affected computers, and a packet sent to that address will be routed to the nearest destination without the client having to explicitly select it.

Summary 6

- Why and how is time synchronized on hosts on the network?
- What is DHCP for?

- How does TCP/IP deal with the missing presentation layer?

- Explain the difference between TCP and UDP.
- What is TCP window for?
- What is a three-way handshake?

- How are the IPv4 classes, subnetting, and CIDR related?
- Why are there only 254 host addresses in a class C network?
- What types of broadcast addresses do we know about?