

# *RCOMP - Redes de Computadores (Computer Networks)*

*2021/2022*

## *Lecture 05*

- The TCP/IP protocol stack.
- IPv4, ARP, UDP, BOOTP/DHCP, ICMP, TCP, IGMP.

# The IP layer

The protocol stack commonly known as TCP/IP, has currently an almost total dominance in computer communications, this makes possible the direct dialogue between almost any type of equipment interconnected by a network.

At the base of this stack lies the Internet Protocol (IP), for now, we will focus on version four (IPv4), currently it's still the most widely used.

When many proprietary technologies were competing with each other, and the OSI model failed to affirm itself, IPv4 was imposed by the growing user acceptance. One advantage of IPv4 was not being a new protocol, meaning it had already suffered a significant evolution under practical use.

Some IPv4 key features are:

- Defines a packet format (the IPv4 packet).
- Defines network addresses, and within each network, node addresses.
- Packet lifetime (TTL) and an identifier for data multiplexing.
- Intermediate nodes fragmentation and reassembly.
- Error detection (for the header only). QoS parameters.

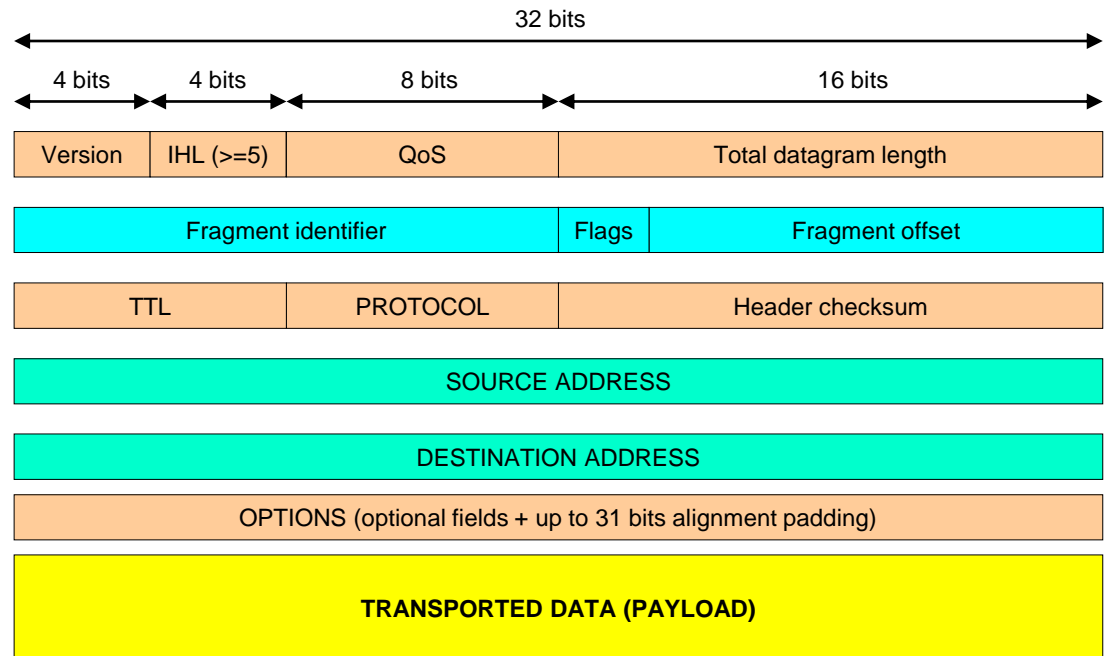
# IPv4 packets (datagrams)

One key point for a protocol to be able to ensure universal interconnection is the definition of a packet format for carrying data. Once set it can hardly be changed, in this domain IPv4 benefited from the maturity it already had at the time its use began to spread.

IPv4 packets may be up to 64 Kbytes long, the header has a 32-bits alignment:

Every IPv4 packet starts with bits 0100, this stands for version four in the version field.

The IHL (Internet Header Length) defines the size of the packet header in 32-bits units, this is required because the header length may vary depending on options being used or not.



The minimum value for IHL is therefore five, meaning no options are used.

# IPv4 header fields

The QoS 8-bits field is intended for Quality of Service parameters, allowing differentiated treatment of traffic, its use has changed along time. Currently, it's divided into two fields: the first six bits are called Differentiated Services Code Point (DSCP) and the remaining two bits, Explicit Congestion Notification (ECN).

The next 16-bits field is used to specify the total packet length (header included), and that's up to 64 Kbytes.

For now, we will skip the next three fields about fragmentation. The following 8-bits field is TTL (Time To Live) actually it doesn't specify any time value, instead it represents a hop countdown, whenever a router forwards the packet it decrements the TTL, if it reaches zero the packet is discarded and an ICMP error message is issued. This allows a sender to specify how far the packet will be propagated, meaning how many routers it will be able to cross.

The protocol 8-bits field is used for multiplexing, it holds a number that identifies to whom the transported data (the payload) belongs to. This allows the IPv4 layer to be used by several different upper layers, for instance, value six means data belongs to TCP, value one means it belongs to ICMP.

# IPv4 header checksum

The 16-bits header checksum field is used for error detection over the header. When a packet is received, the checksum of the header is calculated, if it doesn't match the value in this field, the packet is discarded. The checksum field itself is not included in the checksum calculation and is assumed to have the value zero for that purpose.

Because routers change the IPv4 header content (for instance the TTL field) they must resetttle the correct checksum before retransmitting.

Neither IPv4 nor IPv6 care about error detection over transported data (payload). If required, that's left to be done for upper-level protocols.

Regarding TCP, it always carries out its own error detection on data, UDP may or not implement error detection.

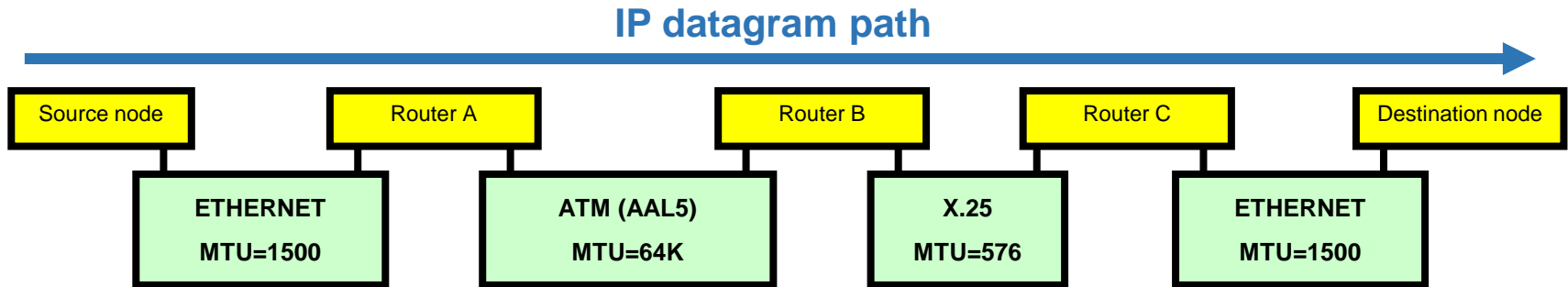
We may also regard error detection at these levels as somewhat redundant, by default the TCP/IP stack assumes there is no error detection at lower layers, but that's hardly ever true.

Take for instance IPv4 over PPP over Ethernet, then we already have two layers with error detection below IPv4, one by Ethernet and another by PPP.

# The IPv4 packet size issue

Even though IPv4 datagrams may be up to 64 Kbytes long, ultimately, they must be transported by a layer two frame. Yet, frames aren't able to carry arbitrary amounts of data. The maximum amount of data each layer two frame can carry is known as the MTU (Maximum Transmission Unit). For instance, the MTU of an Ethernet-II frame is 1500 bytes, and the MTU of an AAL5 PDU is 64 Kbytes.

This issue is complex because the path of a given datagram may include many different layer two technology types, each with a different MTU value.



IPv4 offers two approaches to solve this issue:

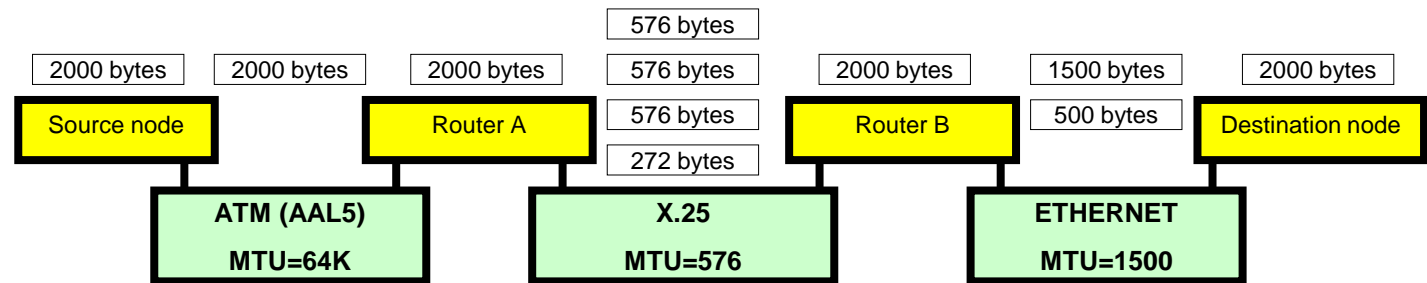
- The original approach: **fragmentation**
- **PMTUD** (Path Maximum Transmission Unit Discovery).

# IPv4 packet fragmentation

Theoretically, the ideal solution is fragmentation and it's directly supported by IPv4. The idea is, if an IPv4 packet is larger than a network MTU, then break it into several shorter fragments to fit the MTU, and then, after overcoming that network, reassemble the packet again.

Fragment identifier	Flags	Fragment offset
---------------------	-------	-----------------

To fragment a datagram, a unique **Fragment identifier** is generated, it's used to identify the fragments as belonging to a same datagram. The **fragment offset** field represents the position of the fragment in the original datagram. The first bit of the flags field is used to indicate if there are more fragments (value 1) the value zero indicates that's the last fragment.



At first glance fragmentation is the ideal solution because it always takes the best advantage from each available MTU. In practice however, fragmentation causes delays on routers that must wait for fragments and reassemble packets.

# PMTUD (Path Maximum Transmission Unit Discovery)

To overcome the fragmentation performance issue, an alternative solution was developed, and it's now widely used. The goal is discovering the **smallest MTU** along a path, that will be the **path MTU**. By not sending packets larger than the determined **path MTU** it's guaranteed fragmentation will never be required along the path.

To obtain the **path MTU**, a try and error mechanism is used:

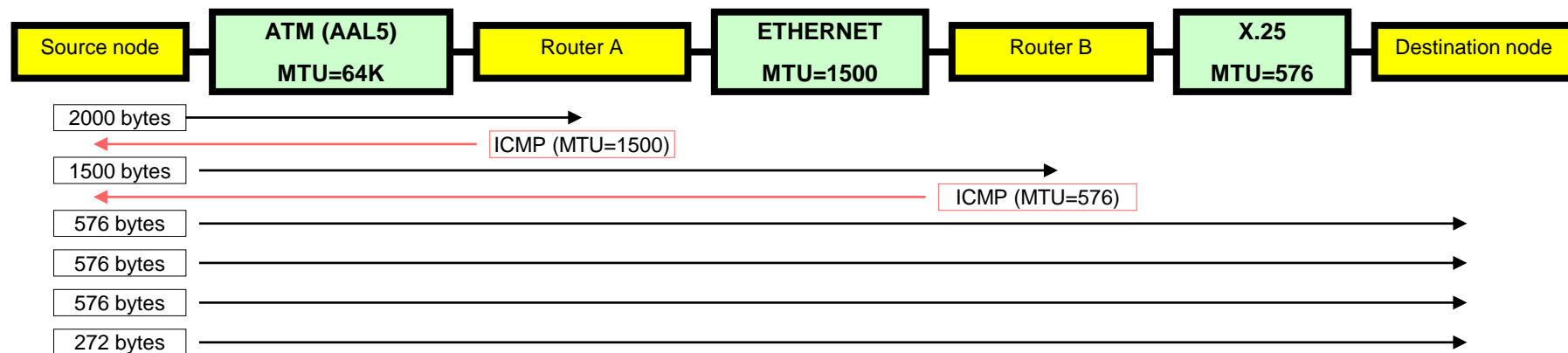
The second bit of the flags field is also known as DF flag, it means **Don't Fragment** and prohibits routers to fragment the packet. When a router receives a packet with DF set and the packet is bigger than the MTU it will have no option other than discarding it. When doing so, the router also sends an **ICMP error message** of type **Destination Unreachable** with code **Fragmentation needed and DF set**, this error message also specifies the MTU value causing the problem (RFC1191).

This means, **if a node sends packets with DF set** it will have a feedback from routers and that feedback can be used to send a smaller version of the same packet matching the reported problematic MTU value.



# PMTUD (Path Maximum Transmission Unit Discovery)

The sender starts by creating a packet with a size matching the local network MTU. If along the path a smaller MTU exists, and because DF is set, the packet is discarded. As a result, an ICMP error message is delivered to the sender informing about the required MTU. Of course, this may happen again if further away along the path there is an even smaller MTU.



In the image above we can see that, after two errors, the sender finally sets the **path MTU** to 576 bytes, following communications to that destination node will always use packets smaller than 576 bytes. Of course, the path can change and may include a smaller MTU, then another correction will be required.

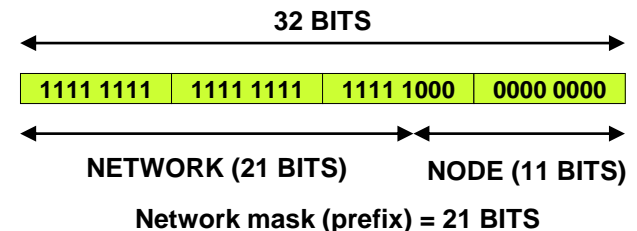
Typically, the local MTU is 1500 bytes (Ethernet) and most networks can cope with that, thus, the need for path MTU corrections is just sporadic.

# IPv4 node addressing

Each IPv4 node has a unique 32-bits address to identify it. Each IPv4 packet header carries two IPv4 node addresses:

- The IPv4 source address, identifies the node that has originally sent the packet.
- The IPv4 destination address, identifies the node to which the packet should be delivered.

Even though an IPv4 address identifies a node, it also identifies the network the node is in. Most significant bits are used to identify the network, the remaining bits are used to identify the node within that network. The network mask defines how many bits are used to identify the network, it's a sequence of bits one (network part) followed by bits zero (host part), for instance:



Thus, if we have an IPv4 node address **and the network mask** we can immediately tell the network address that node belongs to.

# IPv4 networks and routers

When an IPv4 packet is to be sent, one of two things may happen:

- The destination IPv4 address belongs to the network the sender is in – in this case, it can be sent directly to the destination node.
- The destination IPv4 address belongs to a different network – then it must be sent to a router.

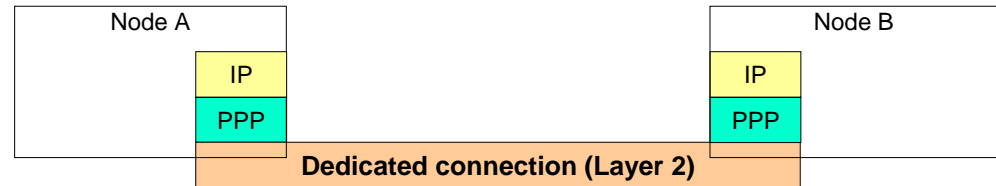
Routers are layer three devices that interconnect different IPv4 networks. When a router receives an IPv4 packet not intended to it (the destination IPv4 address in the packet header is not the router's IPv4 address) it's supposed to retransmit it. The router must first figure out to where it should send the packet, it does so by looking at the packet's destination address and try to figure out to which network it belongs to.

Once the destination network is identified, the **routing table** tells the router to where the packet should be sent in order to reach that network.

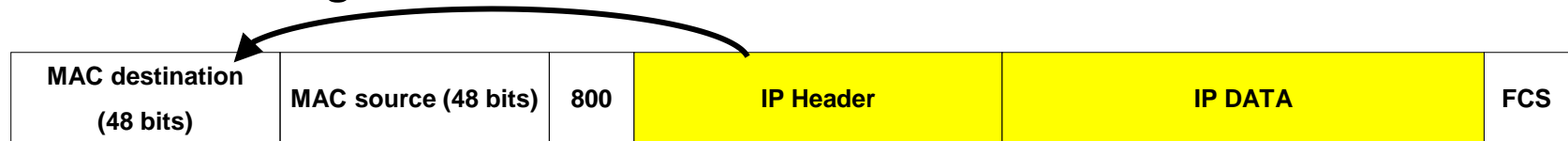
Often, the destination address doesn't belong to a network to which the router is connected, if so, the packet has to be sent to another router. If along the path all routers operate properly, the packet will ultimately be delivered to the destination node, as defined in the packet header.

# IPv4 packets transport

For datagrams transfer between IPv4 nodes, the TCP/IP protocol stack uses an external packet transport service (layer two frames). If the used layer two is a dedicated connection (two nodes network) addressing is implicit and typically the PPP protocol is used to control IPv4 datagrams transport.



However, if IPv4 is using a network with several nodes (switching or shared medium), it must establish an equivalence between the layer two addresses and layer three (IPv4) addresses. When an IPv4 datagram is placed inside a layer two frame for transportation, the layer two destination address for the frame has to match the node having the desired destination IPv4 address.



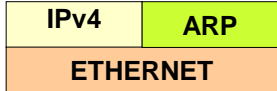
**In fact, this is true only if the destination IPv4 address belongs to the same network. Otherwise, the MAC destination address has to match the router to be used.**

# ARP – Address Resolution Protocol

Either the destination node MAC address or the router MAC address are required to send an IPv4 packet. ARP is used to provide that information; it runs side by side with IPv4 and manages the so-called **ARP table**, establishing equivalences between IPv4 addresses and local **MAC addresses**.

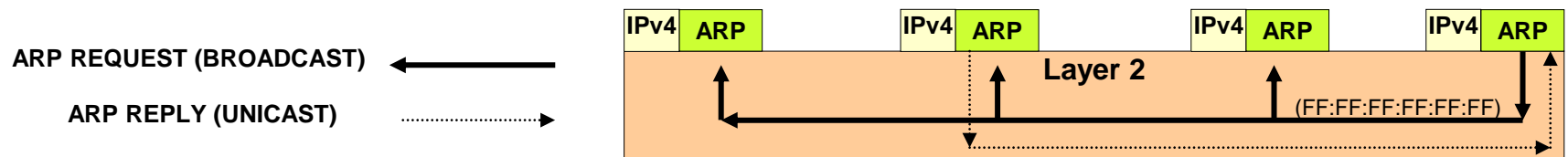
The management of the ARP table is fully dynamic, each line has a lifetime of only a few seconds after that time it's eliminated if not meanwhile refreshed.

ARP TABLE	
IPv4	MAC
192.168.10.5	00:0A:B3:45:34:AB
192.168.10.34	00:0A:B3:50:54:14
192.168.10.240	00:0E:F3:50:00:FA



When sending an IPv4 datagram to a local IPv4 node (it may be the destination node or a router), the IPv4 address is searched in the table, thus, we get the corresponding MAC address to place in the frame header.

If the required IPv4 address is not on the table, then ARP is used to obtain the MAC address and add it to the table. ARP achieves this by sending a layer two broadcast message (ARP request) containing the desired IPv4 address.



# ARP – Address Resolution Protocol

Because every IPv4 node is listening for ARP messages and knows its own IPv4 address, since the **ARP request** is sent to the broadcast address, if the requested IPv4 node exists **on the local network**, then the **ARP request** will be received by the intended node. The node will recognize its own IPv4 address in the ARP request, then it's supposed to send back an **ARP reply** and thus informing the requester about its MAC address.

Upon receiving the ARP reply, the requester adds a new line to its ARP table, this line is valid for a limited time.

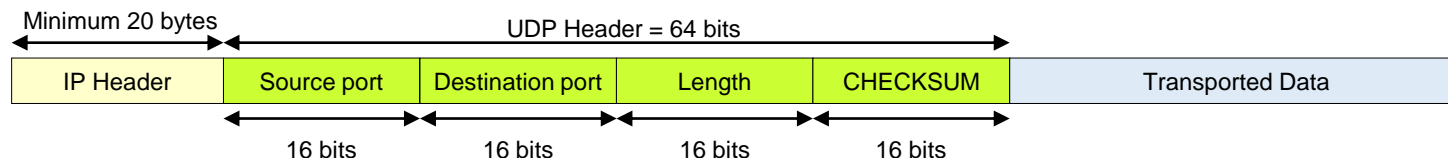
ARP is, generally speaking, insecure, this is because there's no authentication of received ARP replies, this means anyone can forge an ARP reply, and thus insert false information on nodes' ARP tables. Also, ARP is a stateless protocol, it doesn't relate received ARP replies with sent ARP requests, this means any ARP reply is copied to the ARP table, whether or not the node has requested it.

An intruder may simply forge ARP replies and send them to nodes, this is usually called ARP spoofing or ARP cache poisoning.

These attacks to ARP can be used to create Man-In-The-Middle (MITM) attacks deviating data flows through an attacker node.

# UDP (User Datagram Protocol)

IPv4 lacks the minimum features to be used directly by general purpose network applications. A lesser problem is the absence of error detection over transported data, more significant is the absence of any application identification mechanism. The UDP protocol implements both these features.



UDP applications	
UDP	
IPv4	IPv6

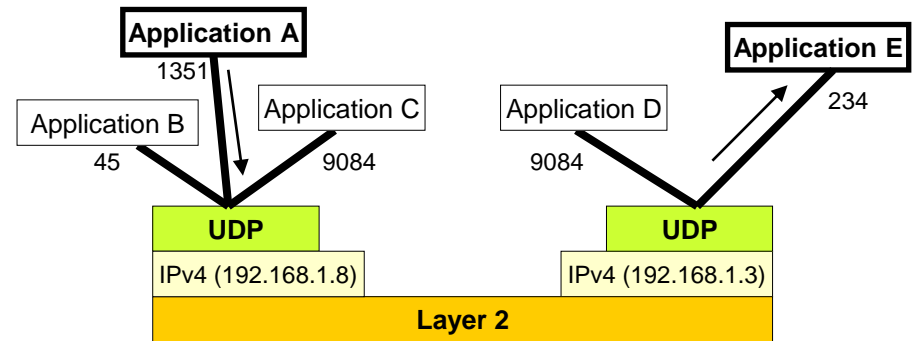
UDP datagrams (image above) are transported inside IPv4 (or IPv6) packets (as payload), the most important added feature is 16-bits port numbers, they are used to identify individual applications within a node. Port numbers make it possible to have several network applications running on the same node without each one's data getting mixed up. Within a node, UDP port numbers must be unique numbers, each assigned to a single application.

**To send data to a UDP application running on a node, the IPv4 node address is not enough, the port number assigned to that application is also required.**

Each UDP datagram carries on its header a source port (the port assigned to the UDP application that sent the UDP datagram) and a destination port (the port assigned to the UDP application that will receive the UDP datagram).

# UDP port numbers

The image represents some UDP applications, each with a unique (within the node) port number assigned.



UDP application A is sending a UDP datagram to application E, to do so, it must specify not only the IPv4 node address application E is running on (192.168.1.3), but also the port number application E is using (destination port = 234).

When application E receives the UDP datagram it will be able to see from which node it came (192.168.1.8), and also, the port being used by application A (source port = 1351), thus it may send a reply to node address 192.168.1.8 to UDP port number 1351, this reply will be delivered to application A only.

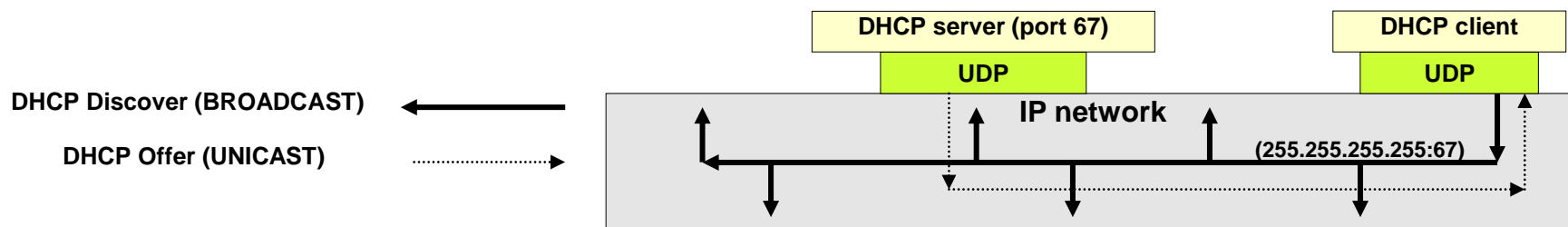
Fixed port numbers (or well-known port numbers) are assigned to specific service types, thus when a client of some service type is started, the user is only required to specify the server's IPv4 node address, this happens because the client application already knows in which port the server application is supposed to be listening on, the default port number for that service type (e.g., 80 for HTTP).



# DHCP (Dynamic Host Configuration Protocol)

DHCP is an evolution of the older BOOTP (Bootstrap Protocol), both these protocols use UDP datagrams to achieve automatic IPv4 nodes configuration. To operate, there must be at least one DHCP/BOOTP server on the network, listening on UDP port number 67, the DHCP/BOOTP port number.

The DHCP client starts by sending a request called **DHCP Discover**, because it's sent to the broadcast address (255.255.255.255) any existing DHCP server will receive it. However, it must be on the same network, packets sent to the broadcast address are not forwarded by routers to other networks.



Once a **DHCP Discover** is received by a DHCP server, it will send back a **DHCP Offer** reply. The client may then select one of the received offers and request configuration data to that specific server. Configuration data provided to the client includes a unique available IPv4 address, the server keeps track of which IPv4 address it has assigned to each client by registering each client's layer two MAC address.

# Dynamic IPv4 range management

Both DHCP and BOOTP provide several other configuration data to clients, for instance, the network mask, the network address, the local router they should use (default gateway) and DNS configuration data.

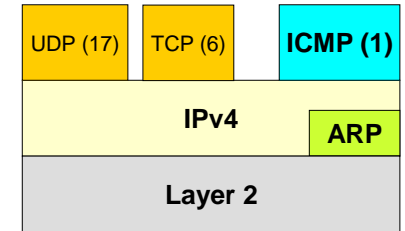
Both DHCP and BOOTP allow statically assigned IPv4 addresses, this means a client with a given MAC address will always have the same IPv4 address.

Nevertheless, they are most useful in managing a range of available IPv4 addresses and assigning them dynamically as new clients emerge. When managing a range of available addresses, DHCP has some advantages over the older BOOTP.

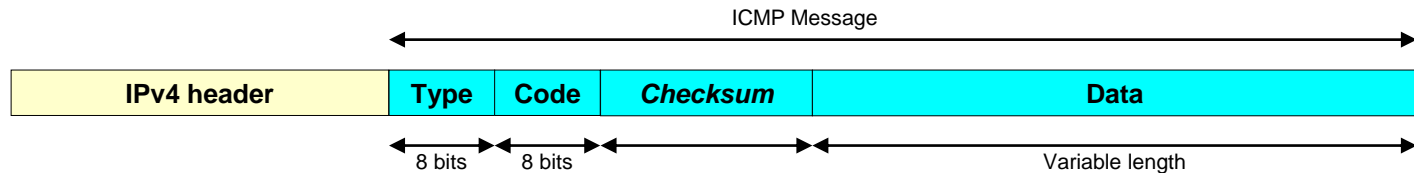
Whereas in BOOTP once an IPv4 address is assigned to a node, that will be forever, DHCP introduces the lease time concept, thus addresses are assigned to nodes only for a period of time. If the client is still alive and using the address it must send a refresh request to the server before the lease expires, otherwise, the server is free to assign that same IPv4 address to another client.

This, of course, makes DHCP more effective on managing a range of available addresses when there's a large number of clients coming and going, like most often happens.

# ICMP (Internet Control Message Protocol)



ICMP is used for testing and error reporting regarding IPv4 operations. ICMP messages are transported by IPv4 packets:

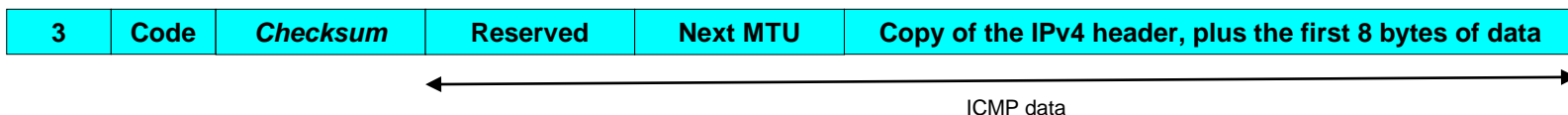


The **Type** field identifies the type of message, for each type of message, there may be several possible values for the **Code** field. The **Checksum** field is used to detect errors in the ICMP message. The **Data** field contains elements dependent on the type of ICMP message.

Type 8 ICMP message is **echo request**, when a node receives this message, is supposed to reply back with a type 0 ICMP message (**echo reply**), they are used to test IPv4 connectivity, namely by the popular **ping** command. For this messages, Code is always zero, Data contains a 16-bits identifier, a 16-bits sequence number and an additional bit patterns defined by the sender. In the echo reply all Data is copied from the echo request. The identifier and sequence number are useful for the requester to associate a received reply with a previously sent request.

# ICMP type 3 - Destination Unreachable

This is an error message, reports that an IPv4 packet has been discarded. It's sent to the node matching the source IPv4 node address of the discarded packet.



Code is used for details about the reason for the packet being discarded, e.g.:

0 – **Network Unreachable** – the packet hasn't reached the destination network, this means there is a routing problem, somewhere along the path one router was unable to determine to where it should send the packet (next-hop), or the next-hop was unreachable.

1 – **Host Unreachable** – the packet has reached the destination network, however, it couldn't be delivered to the destination node.

2 – **Protocol Unreachable** – the packet has reached the IPv4 layer at the destination node, however, the protocol identifier in the IPv4 header refers to a protocol that is not available in the destination node.

3 – **Port Unreachable** – the packet has reached the UDP or TCP layer at the destination node, however, the destination port number is not in use by any application there.

4 – **Fragmentation Needed and DF set** – the packet can't be fragmented (because DF is set), however, along the path it has reached a network with an MTU shorter than the packet size. In this case, the Next MTU field is used to report the MTU value to be used by PMTUD.

# Other significant ICMP error messages

ICMP has a number of different messages to report several types of errors, among others:

**Type 5 (Redirect)** – this usually comes from a router, it's used to inform a node it should update its routing table because it's not sending packets through the best path. This message includes the 32-bits address of the router that should be used, the ICMP data field also carries a copy of the original IP datagram header plus 64-bits of payload.

## **Type 11 (Time Exceeded)**

**Code 0** – informs the TTL field in the IPv4 header has reached zero.

**Code 1** – informs the maximum reassembling time for a fragmented IPv4 packet has expired.

The first 32 bits of the ICMP data field are not used and are followed by a copy of the original IP datagram header plus 64-bits of payload.

Code 0, time exceeded messages, are used by the well known **traceroute** command to generate a succession of errors in the routers along the path, and thus, get to know each router's IP address. This is achieved by sending a succession of IP packets with increasing TTL values starting from one.

# Transmission Control Protocol (TCP)

TCP applications	
TCP	
IPv4	IPv6

TCP presents itself as a **reliable** and **connection oriented** protocol. It provides unprecedented features when compared with previously studied protocols. It's undoubtedly the most widely used on the internet for data transfers.

It's **reliable** because it implements error control, this means that beyond error detection it also automatically corrects all errors. If a piece of data is not received or is received with errors, the sender will retransmit it. The TCP error correction is based on a byte oriented **sliding window protocol**.

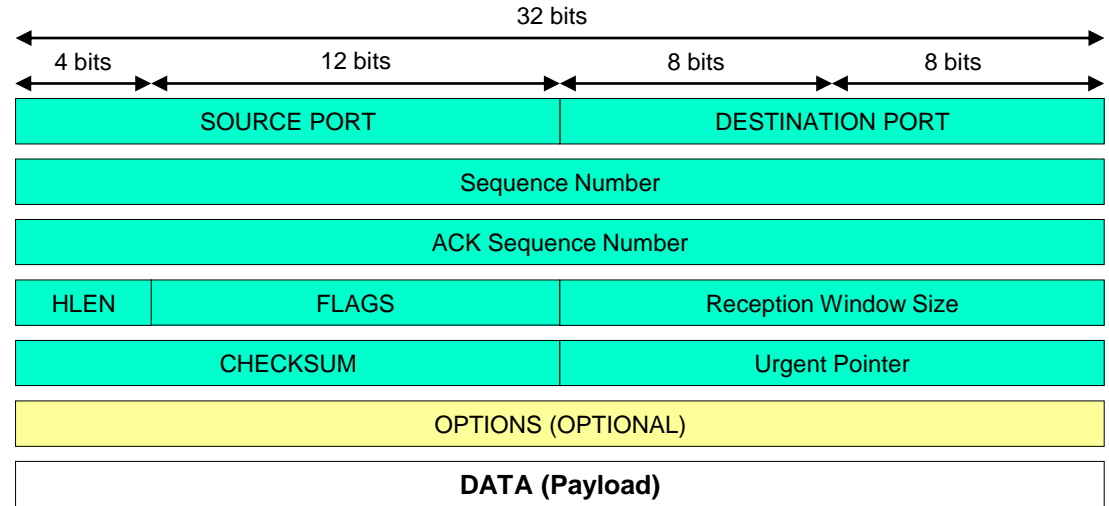
It's **connection oriented** because it establishes dedicated communication channels between applications. Each communication channel (TCP connection) allows the communication only between a pair of applications, no others can interfere. A TCP connection also guarantees data sequence is preserved (bytes are received in the exact same order they are sent) and enforces flow control.

Because it's connection oriented, before any useful data transfer, a TCP connection must be established between two applications. One application takes the initiative of **requesting the connection establishment** to the other application, if the other application **accepts it**, then the TCP connection is created (established). Once established, data can be sent.

# TCP segments

TCP packets are called segments, each segment is transported inside an IPv4 or IPv6 packet, and has a header with several control information:

As with UDP, 16-bits port numbers are used to identify individual application within nodes, yet TCP port numbers are independent of UDP port numbers because they are implemented by different layers.



In a TCP data flow over a TCP connection, each byte has a sequence number. In the segment header, **Sequence Number** is the 32-bits sequence number of the first byte of data present in the segment payload. **ACK Sequence Number** is the 32-bits sequence number of the next byte the emitting node is expecting to receive from the counterpart, this implicitly acknowledges the good reception of all bytes sent before.

HLEN is similar to the IHL field in IPv4, it contains the header length in 32-bits units, it can go from 5 (no options) up to 15.

# TCP segments

The 12-bits FLAGS field is made of 1-bit flags (currently only the nine lower significance bits are used):

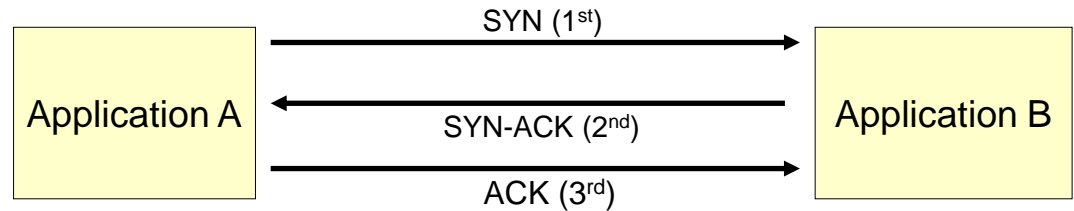
			NS	CWR	ECE	URG	ACK	PSH	RTS	SYN	FIN
--	--	--	----	-----	-----	-----	-----	-----	-----	-----	-----

Some flags of special relevance for now are:

- **SYN** – synchronization request, this is used in connection establishment, only the first segment sent by each application has 1 on this flag, it sets the initial sequence number. When a connection is established, each application sets its own random sequence number and sends it to the counterpart.
- **ACK** – means the **ACK Sequence Number** is significant, it will always be 1, except for the first connection establishment segment with the SYN flag.
- **FIN** – means there is no more data to be sent, and thus, the node informs the counterpart that wants to close the TCP connection.
- **PSH** – push request, request the counterpart to push all buffered data.
- **NS/CWR/ECE** – related to congestion notification and control.



# TCP connection establishment



To establish a TCP connection a three-way handshake is used.

- First, the node taking the initiative (A) sends an SYN request with a randomly set **Sequence Number** ( $N_a$ ) to the other application (B). The ACK flag will be off, in all following segments ACK will always be on.
- If application B is willing to accept, it will reply to A with a SYN-ACK containing its own randomly set **Sequence Number** ( $N_b$ ) and **ACK Sequence Number** containing ( $N_a+1$ ).
- Finally, application A sends to application B an ACK with **Sequence Number** ( $N_a+1$ ) and **ACK Sequence Number** containing ( $N_b+1$ ).

From now on, both applications are synchronized, and they know each other's sequence number, useful data transfer can now begin, and sequence numbers will now on refer to data bytes sent.

# TCP connection closing

Either node can request the connection close, this is achieved by sending a segment with the FIN flag to the counterpart, the counterpart may then reply with a FIN-ACK, meaning it also wants to close the connection. However, the counterpart may also reply with ACK only, by doing so, the connection will be half-open, it may be totally closed latter if the counterpart sends a FIN request.

## TCP Out-Of-Band (TCP OOB)

TCP allows the sending of out-of-band data, this means bypassing the normal data byte flow sequence.

It's achieved by sending a segment with the **URG** (urgent) flag. It's the only case when the **Urgent Pointer** field is used, if URG flag is on, the receiver should immediately process data on the segment up to the byte offset (within the segment data) indicated in **Urgent Pointer** field.

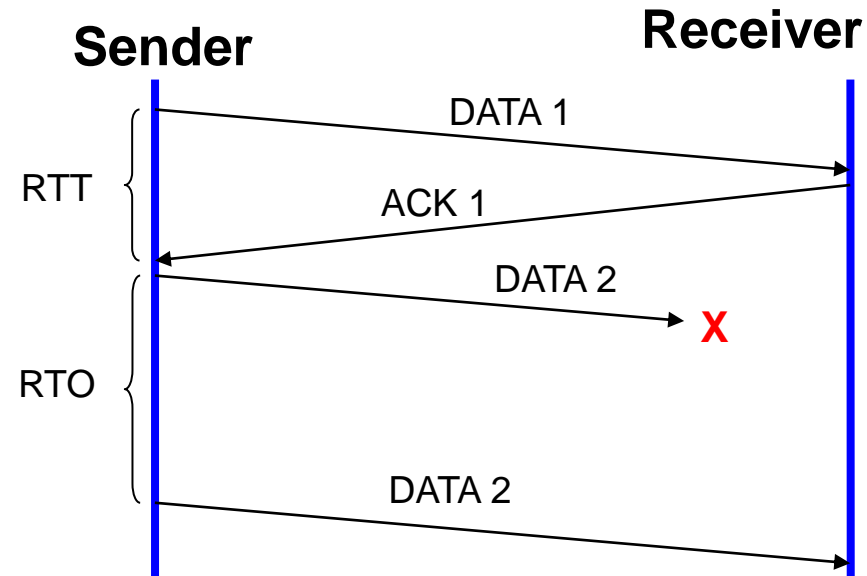
This immediate processing ignores data that was previously received, stored in the receiver's buffer and not yet processed, in this sense out-of-band data overtakes the normal data flow.

# TCP - Retransmission Timeout (RTO)

When a TCP node sends a segment of data with a sequence number it expects to receive the corresponding ACK sequence number within a period of time called RTO, otherwise assumes the segment was lost and must be retransmitted.

The issue here is that the appropriate value for the RTO may be very different depending on network delays.

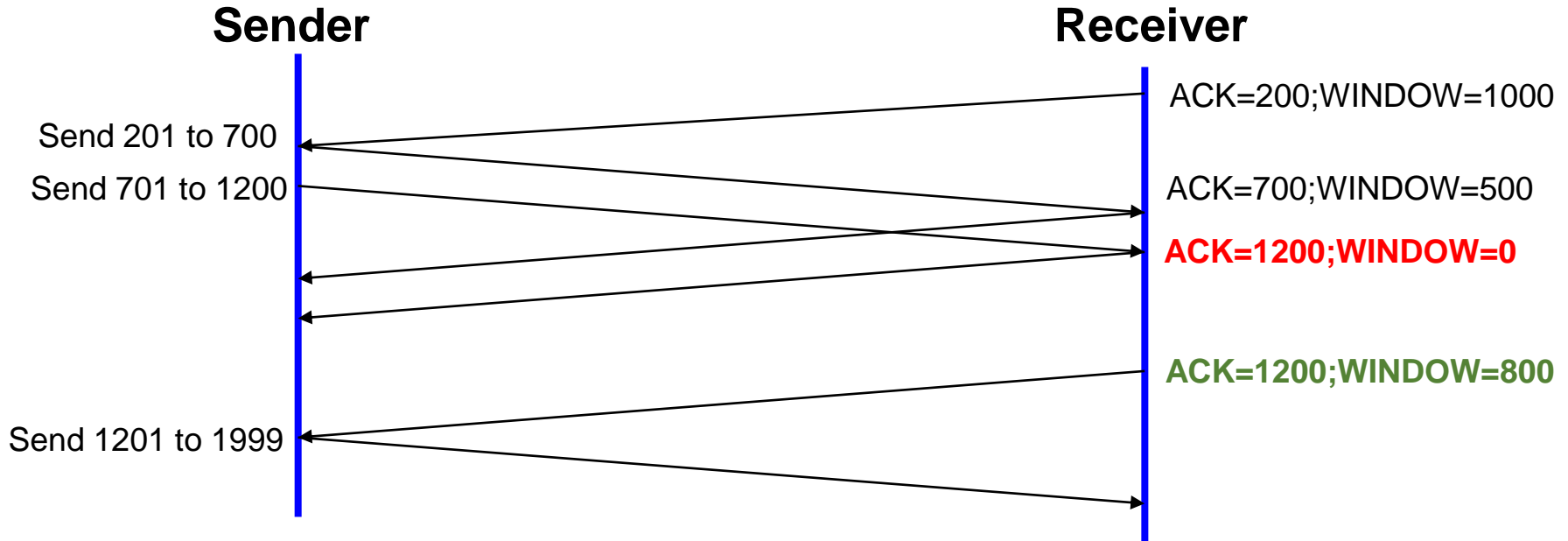
This is solved in TCP by constantly measuring the RTT (Round-Trip Time) on the connection. RTT is the time it takes from the instant a segment is sent until the corresponding ACK arrives.



When a connection is established, TCP sets the initial RTO to 3 seconds, however, as ACK are received, RTT data becomes available and new RTO values are calculated. For the first ACK received,  $RTO = 3 \times RTT$  will be set, but as further RTT values are collected, RTO value will be adjusted taking into account all available RTT values. The minimum RTO value for TCP is, nevertheless, one second.

# TCP – Flow Control

TCP implements flow control by allowing each node to specify in the **Reception Window Size** 16-bits field, exactly how many more bytes it's ready to receive.



On the image above, the receiver initially reports a 1000 bytes window size, thus the sender is allowed to send up to more 1000 bytes. After two more 500 bytes segments, the window is exhausted (WINDOW=0), and therefore the sender is forbidden of sending more data. Later the receiver is ready for more 800 bytes of data and informs the sender with a segment containing WINDOW=800.

# IPv4 Multicast

In opposition to a unicast address, a multicast address represents a set of nodes and not a single node. The meaning is all nodes of that set will receive one copy of the packet. IPv4 broadcast is, therefore, a special case of multicast when the set of nodes is all nodes that belong to an IPv4 network.

To work at layer three, broadcast and multicast must be supported by layer two technologies, Ethernet networks support it by giving a special meaning to the less significant bit of the first MAC address byte, it's called the **broadcast/multicast bit**. Any Ethernet frame with this bit 1 on the destination MAC address is always retransmitted to all ports by layer two switches and bridges. This includes the Ethernet broadcast address (FF:FF:FF:FF:FF:FF).

In IPv4 32-bits address space, addresses starting by bits 1110 (224.0.0.0/4) are reserved for IPv4 multicast. They are also known as Class D addresses.

Each IPv4 multicast address represents a set of nodes (multicast group), some are reserved for specific purposes:

224.0.0.1 – All hosts (all local network nodes, broadcast equivalent)

224.0.0.2 – All routers (all local network routers)

# IPv4 Multicast and Ethernet

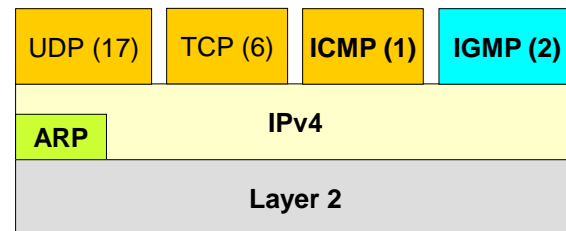
Ethernet addresses ranging from 01:00:5E:00:00:00 to 01:00:5E:7F:FF:FF are reserved for IPv4 multicast, the 23 less significant bits of these Ethernet addresses are a copy from the 23 less significant bits of the IPv4 multicast address. Thus, IPv4 multicast addresses overlapping will occur over Ethernet multicast addresses if only bits values above the 23<sup>rd</sup> bit differ.

For each IPv4 multicast address, there is, however, a single corresponding Ethernet multicast address, for instance when an IPv4 packet is sent to destination 224.0.0.7, this IP packet will be placed inside an Ethernet frame with 01:00:5E:00:00:07 destination address. IEEE 802.11 wireless networks also regard these MAC addresses as multicast.

## Internet Group Management Protocol (IGMP)

IGMP is used to manage IPv4 multicast groups. Unlike broadcast, multicast is not limited to a single IPv4 network.

When a router receives an IPv4 packet with a multicast destination address (multicast group), it must know if there are nodes in other networks that belong to that multicast group, only if so, then the packet is required to be transmitted to those networks, otherwise it's pointless.



# Internet Group Management Protocol (IGMP)

Periodically a router sends an **IGMP Membership Query** message to the 224.0.0.1 address (all hosts) of each network it's connected to.

Each node should then reply with an **IGMP Membership Report** message for each multicast groups it belongs to.

Each **IGMP Membership Report** is sent to the IPv4 multicast address of the corresponding multicast group.

However, for the sake of avoiding redundant traffic, nodes don't reply instantly, they wait a random period of time before doing so. If meanwhile, they receive from another node a Membership Report message for the same group they skip sending. Routers only need to know if there is at least one member of the group in the network, it's not relevant knowing if there is one or several members.

## Layer two IGMP aware switches

Standard layer two switches forward multicast packets to all ports, thus flooding the entire network. There are, however, switches that are capable of listening IGMP messages, and thus, knowing in which network segments are nodes belonging to each group. These switches are then capable of forwarding multicast packets only to segments where members of each group exist.