Computer networking - exam

Nicolaï Pol - Deflandre Guilian

Avril 2019

1 Principles of the Web and HTTP

1.1 Description of a web page

A webpage can be seen as a based HTML files containing references to objects. Each object is addressable by a url. Here we have an example URL : `http://www.someschool.edu:8080/someDept/pic.gif`. It can be divided as :

1. HTTP = name of protocol
2. URL itself
3. File path

1.2 HTTP overview

HTTP is the acronym of hypertext transfer protocol, it's the web's application server protocol. We note the following properties for this protocol.

1. It follows the client-server model.
   (a) Client : browser that requests, receives and displays.
   (b) Server : sends response to requests.
2. HTTP uses TCP connections. This method is used because it keeps the ordering, the non-duplication, the integrity (...) of packets among other reasons. It works like this :
   (a) The client initiates the TCP connection (creates socket) to server.
   (b) The servers accepts the connection.
   (c) HTTP messages (request and responses) are exchanged between the client and the server.
   (d) The TCP connection is closed.
3. HTTP is stateless, the servers maintains no info about past requests.

We notice that there are 2 types of HTTP connections. Persistent and non-persistant HTTP.

1. Non-persistent HTTP
   (a) At most one item is sent at a time, the connection is then closed.
   (b) Download multiple objects requires multiple connections.
   (c) The response time for non-persistent HTTP is equal to 2RTT (round-trip time) + file transmission time (FTT = file size / average throughput of TCP connection).
   (d) Consequences :
      i. Requires 2 RTT/object
      ii. OS overhead for each TCP connection
      iii. Browsers often open parallel connections to fetch referenced objects.
2. Persistent HTTP
   (a) Multiple objects can be send over a single TCP connection.

   (b) Server leaves connection open after sending response
   (c) As little as one RTT for all the referenced objects when requests are pipe-lined.
   (d) Client send request as soon as he encounters a referenced object.

Let's now study the form of the different HTTP request messages. They can be separated in 3 parts.

1. Request line
   (a) Method (GET, POST, HEAD...) followed by white space.
   (b) URL followed by white space.
   (c) Version (HTTP 1.1, 1.0) followed by a carriage return and a line feed.
2. Header lines :
   (a) Header field name followed by " : ".
   (b) Value of the header field followed by a carriage return and a line feed.
   (c) A carriage return followed by a line feed indicate the end of the header block.
3. Entity body.

Let's now study the form of the different HTTP response messages. They can also be separated in 3 parts.

1. Status line protocol :
   (a) Protocol used followed by a white space.
   (b) Status code followed by a white space.
   (c) Status sentence followed by a carriage return and a line feed.
2. Header lines (see above)
3. Data, e.g. requested HTML file.

We note that there are different types of requests, and that these request also differ with the version of the protocol that are used. They are described here under :

1. HTTP/1.0
   (a) GET : data sent to the server with GET is stored in the URL field of the status line of the HTTP request.
   (b) POST : data sent to the server with POST is stored in the request body of the HTTP request.
   (c) HEAD : asks server to leave requested object out of response. It's a POST without the response.
2. HTTP/1.1
   (a) GET, POST, HEAD.
   (b) PUT : uploads file/object in entity body to path specified in URL field (creation or update of an object). The URL tells where to store the file.
   (c) DELETE : delete files/objects specified in URL field.

Following the result of the request, HTTP will give various response to it, and those response are always accompanied by a status code, that appears in the

1. HTTP response status codes (see where they appear in point 7.)
   (a) 200 OK
   (b) 301 Moved Permanently (Object moved, new URL in data)
   (c) 400 Bad Request
   (d) 404 Not Found
   (e) 505 HTTP Version Not Supported

1.3 Cookies

1. Four main components :
   (a) Cookie header line of HTTP response message.
   (b) Cookie header line in HTTP request message.
   (c) Cookie file kept on user's host, managed by user's browser.
   (d) Back-end database at Web site.
2. What cookies can be used for :
   (a) Authorization.
   (b) Shopping carts.
   (c) Recommendations.
   (d) User session state.

1.4 Caches

Their goal is to satisfy the client's need without involving the origin server. See figure 1

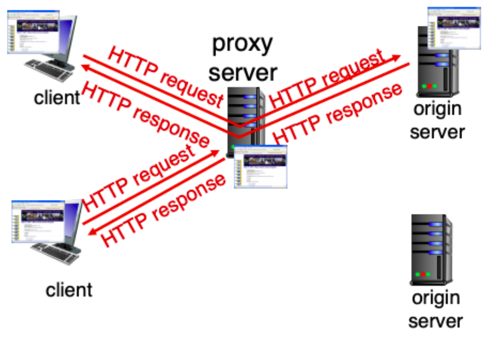


FIGURE 1 – Web Caches

1. Cache acts as both client and server : client for the origin server, and server for the client.
2. Why Web Caching
   (a) Reduce response time for client's request.
   (b) Reduce traffic on institutional network.
   (c) Reduce load on servers.
3. Cache can be used to solve various problems. For example, if the access link rate is too small, it will be fully used (99%). The delay will be quite big. A solution is to increase the access link speed, but that's expensive. So we'll install a local cache.
4. Conditional GET : This is used between the server and the local web cache. The idea is that the server won't send an object if the cache has an up-to-date version of this object. The cache will thus be specify the date of the object in the request to the server.

2 Principles of DNS

2.1 Basic Principles

1. DNS = Domain Name System.
2. We know that people have many different ids (SSN, Name, Passportsequence ,...). Meanwhile, internet hosts and routers have 2 big types of IDs :
   (a) Their IP addresses (32 bits), used for addressing datagrams.
   (b) Their names, used by humans.
3. DNS answers to the question how to map between IP and names and vice-versa.
4. A DNS consists of
   (a) A distributed database, implemented in hierarchy of many name servers.
   (b) An application-layer protocol, to resolve names (address/name translation).

2.2 Service

1. Host-name to IP address translation.
2. Host aliasing
3. Mail server aliasing
4. Load distribution : many IP addresses correspond to one name.

2.3 Centralized DNS

BIG NONO :

1. Single point of failure.
2. Traffic volume.
3. Distant centralized database.
4. Maintenance

2.4 Structure

DNS is a distributed hierarchical database. See figure 2.

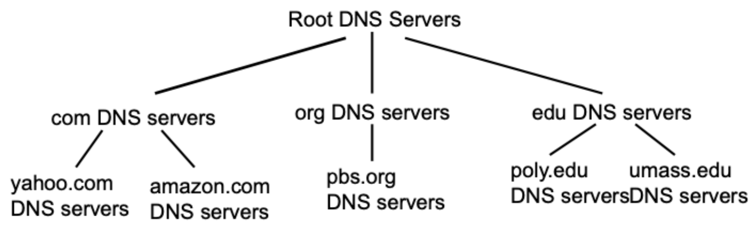


FIGURE 2 – DNS structure

1. Root Name Servers : there are over 13 root name "servers" scattered all over the worlds. These servers are managed by different organizations.
2. TLD servers : the top-level domain servers are responsible for .com, .org, .edu, ... and all top-level country domains, e.g. .be, .fr, .uk, ...
3. Authoritative DNS servers : Organization own DNS server(s), providing authoritative host-name to IP mappings for organization's named hosts.
4. Local DNS name server, also called "default name server " : Does not strictly belong to hierarchy. Each ISP (residential ISP, company, uni,...) has one. When a local host makes a DNS query, it is sent to its local DNS server. This server has a local cache of recent name-to-address translation pairs (may be out of date) and it acts as a proxy, forwarding a query into hierarchy.

2.5 DNS name resolution

There a different methods :

1. Iterated Query : When a query is made, the contacted server replies with the next server to contact : "I don't know this name, but ask this server". Caching in local DNS can help to skip a few steps.
2. Recursive Query : this type of query puts the burden of name resolution on the contacted name server. There is even more caching, since it can occur for every server, ROOT and TLD...

2.6 Caching

1. Once any name servers learns mapping, it caches mapping. Those entries disappear after some time TTL (time-to-live).
2. Cached entries may be out-of-date. If name host changes IP address, this may not be known internet-wide until all TTLs expire.
3. There are update/notify mechanisms (Dynamic DNS).

2.7 DNS records

DNS is a distributed database storing resource records (RR). The RR format is : (name, value, type, ttl). The different types are :

1. A (or AAAA) :
   (a) Name is a hostname.
   (b) Value is IPV4 (or IPV6) address.
2. NS :
   (a) Name of the domain, DNS zone.
   (b) Value of the hostname of authoritative name server for this domain.
3. CNAME :
   (a) Name is alias name for some canonical name.
   (b) Value is canonical name
4. MX :
   (a) Value is name of mailserver associated with the domain name

2.8 Protocol, messages

Query and reply messages both have the message format of figure 3. These messages are usually sent over UDP. Their structure can be described like this :

1. Message header :
   (a) Identification : 16 bit sequence for query. The reply to the query uses the same sequence .
   (b) Flags
      i. Query or reply.
      ii. Recursion desired.
      iii. Recursion available.
      iv. Reply is authoritative.

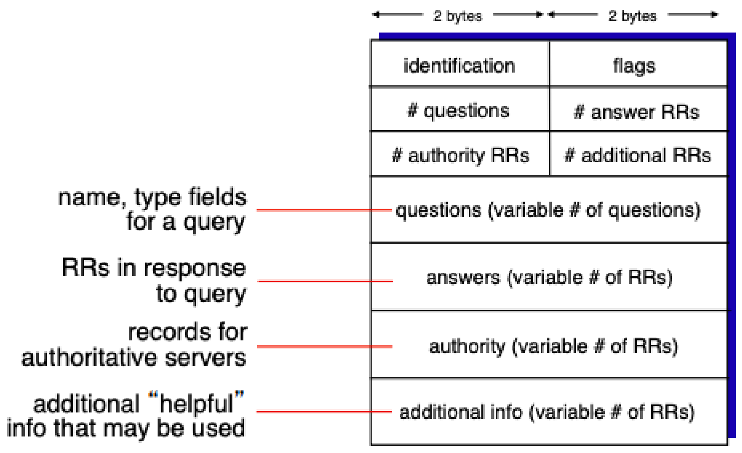

2 bytes
2 bytes
identification
flags
# questions
# answer RRs
# authority RRs
# additional RRs
name, type fields for a query
questions (variable # of questions)
RRs in response to query
answers (variable # of RRs)
records for authoritative servers
authority (variable # of RRs)
additional "helpful" info that may be used
additional info (variable # of RRs)

FIGURE 3 – DNS Message Structure

2.9 Inserting records into DNS

Let's take an example : you want to create a new start up called Network Utopia.

1. First you register the name networkutopia.com at DNS registrar (e.g., Network Solutions).
   (a) To do so, you provide names, IP addresses of authoritative name server (primary and secondary) [1]
   (b) The register then inserts two ressource records into com TLD server.
2. What we did was creating an authoritative server Type A record for www.networkuptopia.com ; and a type NS record for networkutopia.com for each of these two authoritative DNS servers. For the primary authoritative server for networkutopia.com, the registrar would insert the following two ressource records into the DNS system :

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

1. In general, domain names can work with only one name server - the primary DNS. However, practice has shown that a domain name needs to have at least two name servers assigned in order to be available at any time. In case there is a problem with the primary name server, the secondary name server will be able to answer the online request for a particular domain. This back-up requirement has turned into an accepted Internet standard that prevents domain names from going offline. Secondary servers, also know as slave servers, contain read-only copies of the zone file, and they get their info from a primary server in a communication known as a zone transfer.

3 Socket programming (TCP and UDP) ; addressing and (de)multiplexing in the transport layer

3.1 Socket Programming

The goal of socket programming is to learn how to build client/server applications that communicate using sockets. Sockets are analogous to doors :

1. Sending process shoves the message out door
2. Sending process relies on the transport infrastructure on other side of door to deliver message to socket at receiving process.

There are two socket types for two transport devices

1. UDP : unreliable datagram
2. TCP : reliable, byte-stream oriented.

3.1.1 Socket Programming with UDP

1. UPD means that there is no "connection" between the client and the server
   (a) No handshaking before sending data
   (b) Sender explicitly attaches IP destination address and port sequence to each application data unit.
   (c) Receiver extracts sender IP address and port sequence from received data unit.
2. The transmitted data may be lost or received out-of-order.

To sum it up, we can say that UDP provides an unreliable transfer of group of bytes between client and server. On figure 4, we describe the interaction between the client and server sockets. We observe that :

1. Both client and server uses datagram sockets.
2. The destination IP and ports are explicitly attached to the app data unit by the client and the server.
3. We only need 1 socket for 1 port number. That's why you need the address of the client so you can send him the response back.

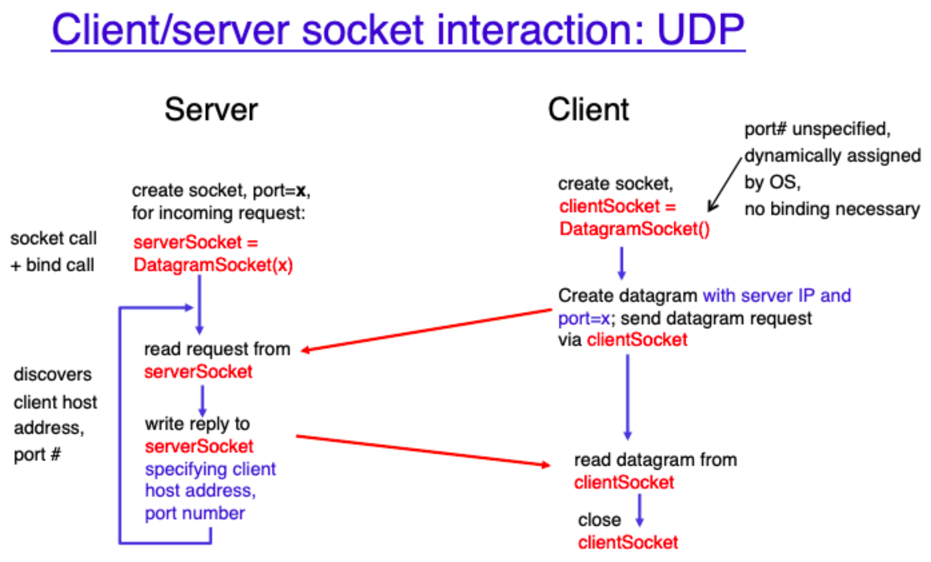


FIGURE 4 – UDP interaction

3.1.2 Socket Programming with TCP

1. Client must contact server.
   (a) The server process must first be running.
   (b) The server must gave created a socket that welcomes client's contact, a welcoming socket.
2. The client contacts the server by :
   (a) Creating a TCP socket specifying the IP address and the port number of the server process
   (b) When socket created, the client TCP will establish a connection to the server TCP.
   (c) When it will be contacted by a client, the server will create a new socket for the server process to communicate with that particular client. This allows the server to communicate with multiple clients. The source IP and source port numbers are used to distinguish the clients.

To sum it up, we can say that TCP provides a reliable, in-order transfer of group of bytes between client and server. On figure 5, we describe the interaction between the client and server sockets. We observe that :

1. There are two types of socket primitives : ServerSocket and Socket.
2. When client knocks on serverSocket's door, server creates `connectionSocket` and completes TCP connection.
3. Destination IP and port are not explicitly attached to the application data unit by client and server.
4. Multiple clients can use a server. In this case, you need 1 welcoming socket for the server, and n sockets for n clients.

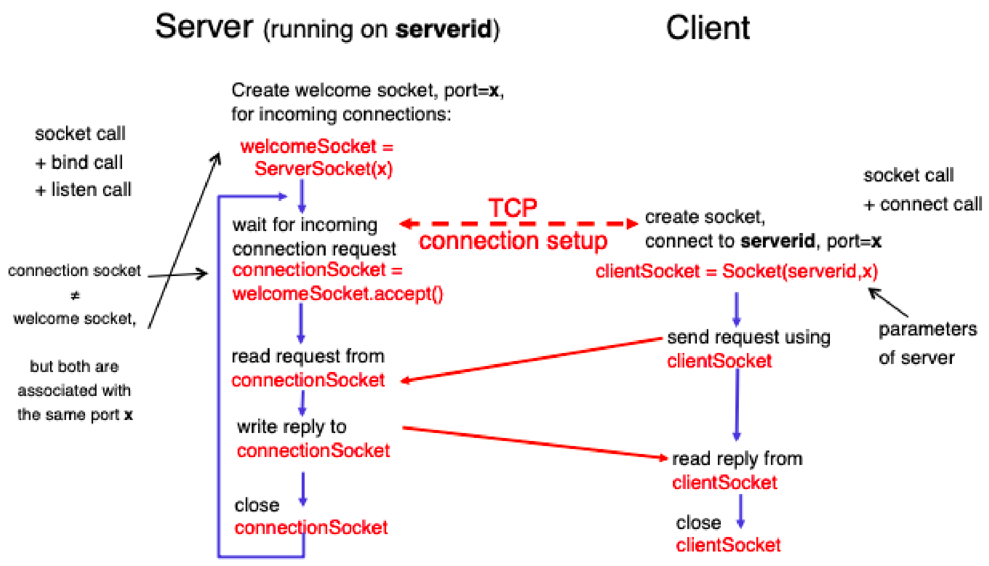

Server (running on serverid)
Client
Create welcome socket, port=x, for incoming connections:
welcomeSocket = ServerSocket(x)
socket call
+ bind call
+ listen call
connection socket ≠ welcome socket,
but both are associated with the same port x
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
create socket, connect to serverid, port=x
clientSocket = Socket(serverid,x)
socket call
+ connect call
parameters of server
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

FIGURE 5 – TCP interaction

3.2 Addressing in the transport layer.

1. To receive messages, process must have identifier. Identifier includes :
   (a) IP address associated with process on host (see Network layer)
   (b) Port number associated with process on host.
2. Host device has (at least) one IP address.
3. On a computer there are $2^{16} = 65536$ ports number. Ports from 0 to 1023 are the well-known ports, used for the most current networks. The ports numbers from 1024 to 49151 correspond to the registered ports assigned by the IANA (**I**nternet **A**ssigned **N**umbers **A**uthority). Finally, the ports from 49152 to 65535 are the dynamics ports used to treat the TCP and UDP connections explained above.

3.3 Multiplexing/Demultiplexing.

Multiplexing takes place on the sender side. Data from multiple sockets are handled and a transport header is added. This header is later used for demultiplexing. Demultiplexing takes place on the receiver side. Header information are used in order to deliver received segments to correct socket.

3.3.1 Demultiplexing main idea.

1. First, the host receives IP datagrams.
   (a) Each datagram has source and destination IP addresses in its header.
   (b) Each datagram carries one transport-layer segment.
   (c) Each segment has source and destination port numbers in its header.
2. The host then uses the IP addresses (in the network header) and the port numbers (in transport header) to direct segment to appropriate socket.

3.3.2 Connectionless (UDP) demultiplexing.

We know that the socket created by the application has an host-local port sequence . We also know that when it creates a datagram to send into the UDP socket, the app must specify the "remote process id", i.e. the destination IP address and the destination port sequence. When UDP will receive the segment from below it will :

1. Check the destination port sequence in the segment.
2. Direct the UDP segment to the socket with that port sequence .

In the end, IP datagrams with same dest. port sequence, but different source IP addresses and or source port numbers will be directed to the same socket as destination.

3.3.3 Connection-oriented (TCP) demultiplexing.

We know that Web servers have different TCP sockets for each connecting client. Those many simultaneous TCP sockets are all associated with the same port sequence. So the destination port sequence is not enough to direct a segment to the appropriate socket. To allow demultiplexing, a TCP socket will thus be identified by a 4-tuple :

1. Source IP address.
2. Source port number.
3. Dest IP address.
4. Dest port number.

The receiver TCP will use all four values to direct segment to appropriate socket. This works also fine with threaded servers.

3.3.4 TCP vs UDP.

	TCP	**UDP**
Acronym	Transmission Control Protocol	User Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to another. This is connection based.	UDP is also a protocol used in message transport or transfer. This is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship.

Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, TFTP, SNMP, RIP, VOIP.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because error recovery is not attempted. It is a "best effort" protocol.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Header size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Some header fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Streaming of data	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.
Weight	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP does not have an option for flow control
Error Checking	TCP does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.	UDP does error checking but simply discards erroneous packets. Error recovery is not attempted.
Fields	1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port	1. Length, 2. Source port, 3. Destination port, 4. Check Sum
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake

Table 1 – Comparaison between UDP and TCP protocols.

4 Reliable data transfer : elementary protocols and their efficiency, error detection (checksum and CRC)

4.1 Principle of a reliable data transfer

We know that the characteristics of an unreliable channel will determine the complexity of the reliable data transfer protocol (rdt). We can see how a rdt look on figure 6.

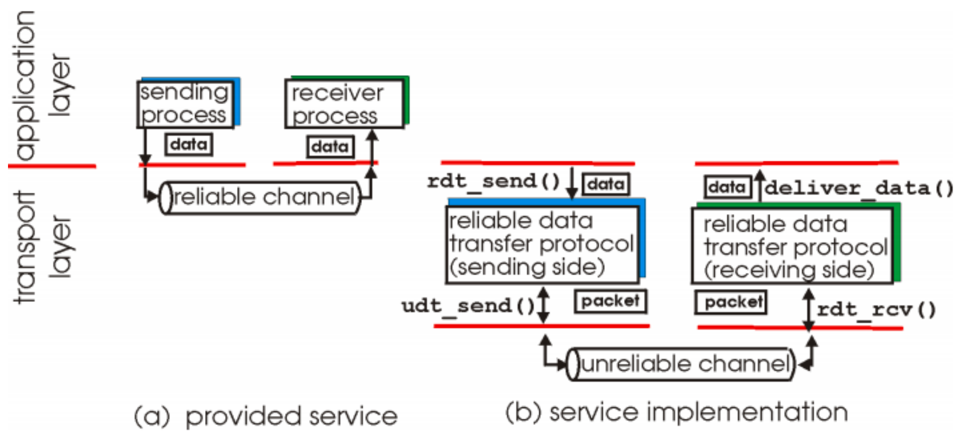


FIGURE 6 – Reliable Data Transfer

This works the following way :

1. `rdt_send()`: is called from above (e.g. by application layer), and it passes the data to deliver to receiver's upper layer.
2. `udt_send()`: is called by RDT, to transfer packet over unreliable channel to receiver.
3. `rdt_rcv()`: is called when packet arrives on rcv-side of channel.
4. `deliver_data()`: is called by RDT to deliver data to upper layer.

4.1.1 RDT 1.0

The process behind finding a rdt is to start with strong assumptions and to eliminate them as we march forward.

1. We assume that the underlying channel is perfectly reliable.
   (a) There are no bit errors.
   (b) There is no loss of packets.
2. We assume that sender and receiver are separate Finite State Machines
   (a) Sender sends data into underlying channel
   (b) Receiver reads data from underlying channel.

The FSM specification is visible in figure 7.

4.1.2 RDT 2.0

We now assume that the underlying channel may flip bits in packet. We'll use a checksum to DETECT the bit errors. The real question though, is how to recover from an error. The idea is the same here as when human have a conversation : we try again ! There's a feedback idea. To do so, we'll use the following technique :

1. acknowledgements (ACKs) : the receiver will explicitly tell the send that the packet received was OK.
2. negative acknowledgements (NAKs) : the receiver will explicitly tell the send that the packet had errors.
3. The sender will retransmit pkt on receipt of NAK.

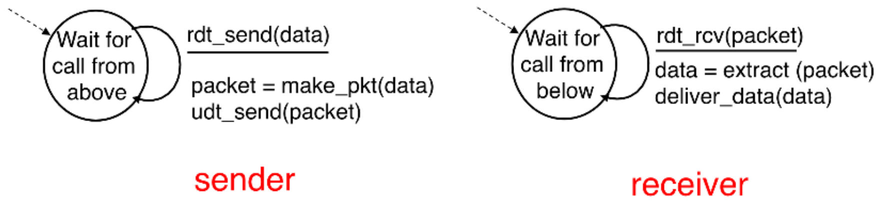


FIGURE 7 – Reliable Data Transfer 1.0

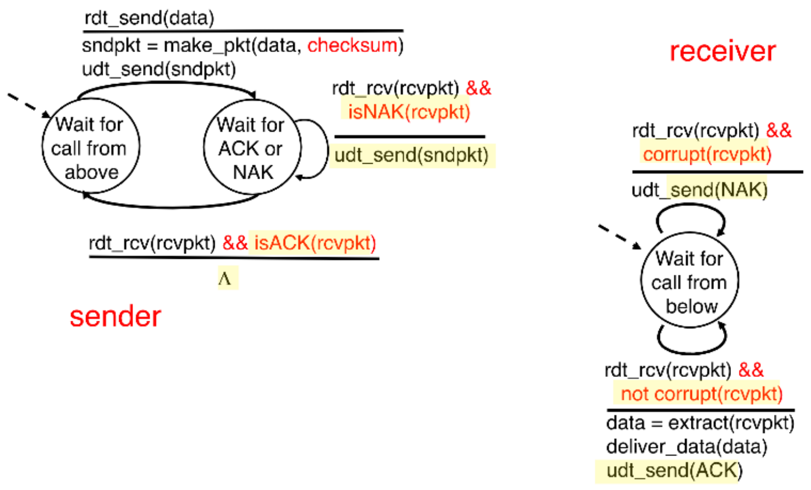


FIGURE 8 – Reliable Data Transfer 2.0

The new mechanisms in rdt 2.0 are error detection and feedback (ACK, NAK) from receiver to sender. The FSM specification is visible in figure 8.

We notice that it works following the "stop and wait" method. However, rdt 2.0 has a fatal flaw. Indeed what happens if ACK/NAK corrupted ?

1. Garbled ACK/NAK is detected by checksum, then discarded.
2. But sender doesn't know what happened at receiver. It can't just retransmit, possible duplicate.

However, there is a solution :

1. The sender retransmits current packet if ACK/NAK garbled.
2. Sender adds sequence number to each packet.
3. The receiver discards (doesn't deliver up) duplicate packet.

This solution is handled in rdt 2.1, whose FSM specification is visible in figure 9 and 10.

We notice that when the receiver gets a duplicate, it still sends an ACK. It's perfecly normal. If the receiver got a duplicate, it's because there was an error with the previous ACK, so we still need to confirm to the sender that we received the data. Now, let's discuss rdt 2.1 :

1. On the sender side :
   (a) sequence number are added to packets.
   (b) Two sequence number (i.e. 0 and 1) will suffice, since we wait between two packets. Either you sent the right one, or the previous one.
   (c) There are twice as many states, since the state must remember whether current packet has sequence sequence of 0 or 1.
2. The receiver must on his side check if the received packet is a duplicate. The state indicates whether 0 or 1 is the expected packet sequence number.

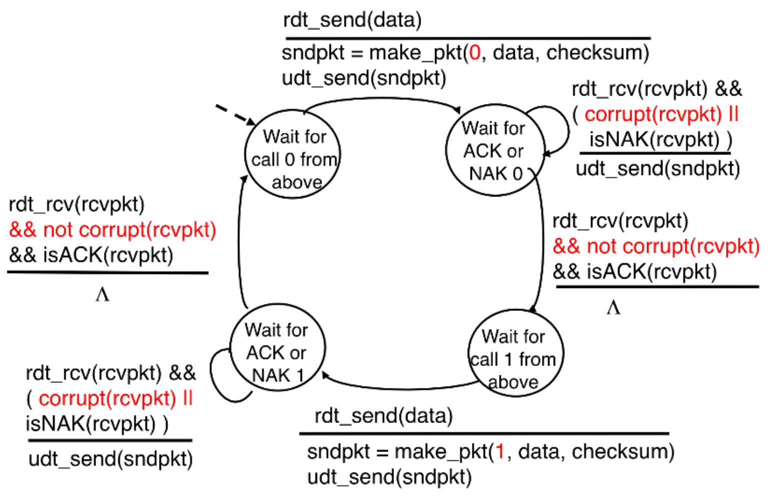

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && isACK(rcvpkt)
Λ
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && isACK(rcvpkt)
Λ
Wait for ACK or NAK 1
Wait for call 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

Figure 9 – Reliable Data Transfer 2.1 (Sender)

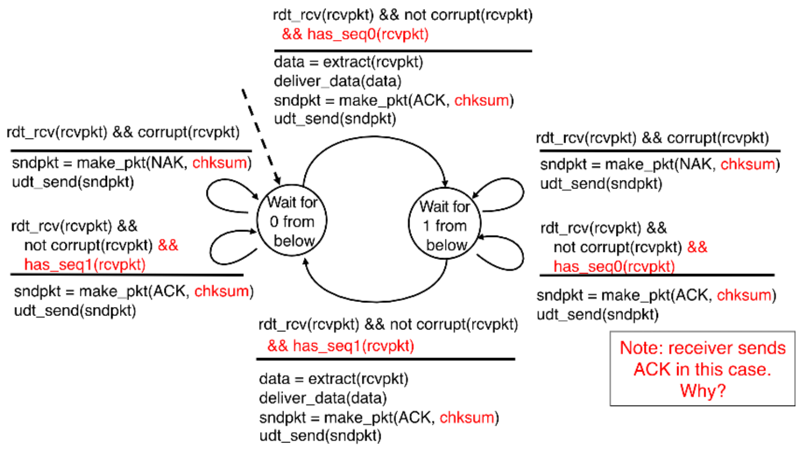

rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq0(rcvpkt)
data = extract(rcvpkt)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq1(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq0(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq1(rcvpkt)
data = extract(rcvpkt)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
Note: receiver sends ACK in this case. Why?

Figure 10 – Reliable Data Transfer 2.1 (Receiver)

4.1.3 RDT 3.0

In this version, the new assumption is that the underlying channel also lose packets (data or ACKs). In this case, the checksum, the sequence number, the ACKs and the retransmission will be helpful, but that won't be enough. The chosen approach is then to use the time. Indeed, the sender will wait a reasonable amount of time for the ACK, after what it will retransmits if no ACK is received in this time. So this update requires a new field which is the countdown timer.

There is no timer on the receiver side, since he doesn't know if he is supposed to receive something or not.

4.1.4 RDT 3.1

Up to now, a timer for packet loss is used, while the NAK's are used for packet errors. But we can simplify this. Indeed, with the timers, no NAKs are needed. They would improve the recovery time, but this is not our concern here. However, there is a flaw in rdt 3.1 which is caused by delayed packets and/or ACKs. On figure 11 we have an example of a problem that might occur. Those race conditions can be unnoticed during the design protocol while they might appear later with a change in the speed of the network.

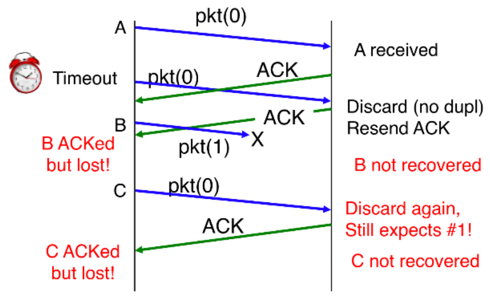


FIGURE 11 – Reliable Data Transfer 3.1 flaw

4.1.5 RDT 3.2

To fix this issue, rdt 3.2 adds the sequence sequence in ACKs. The receiver will thus specify to the sender which packet he is acknowledging. rdt 3.2 in action is described in figure 12. The FSM specification of rdt 3.2 is described in figure 13. This solves most of the problems described, but there's still an issue. In fact, rdt 3.2 is still incorrect if there is packet or ACK reordering. One simple solution would be to choose a timeout time so large that when a packet is re-transmitted, the sender is sure that the previous copy of this packet and its ACK have disappeared from the network. A better solution is to use a much larger sequence number. Let's now look at rdt 3.2's performance.

1. The first packet bit is transmitted a $t = 0$.
2. The last packet bit is transmitted a $t = L/R$, where R is the transmission rate and L is the packet size.
3. The sender, before sending another packet has to wait for RTT more seconds. The time is then of $t = RTT + L/R$.
4. In all this, the U_{sender} (fraction of time where sender is busy sending) is equal to

$$U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{1}{1 + \frac{R*RTT}{L}} \tag{1}$$

   where $(R * RTT)/L$ is the bandwidth-delay product, a.k.a. the number of "in-flight" bits.
5. This isn't great at all. In fact, for a 1Gbps link of 15 second end-to-end propagation delay who transmit packets of size 1KB, each packet will take 30msec to get delivered. This means that the throughput is of 266.7 kbps over a 1Gbps link (0,0266%). We see here that the network protocol widely limits the use of physical resources.

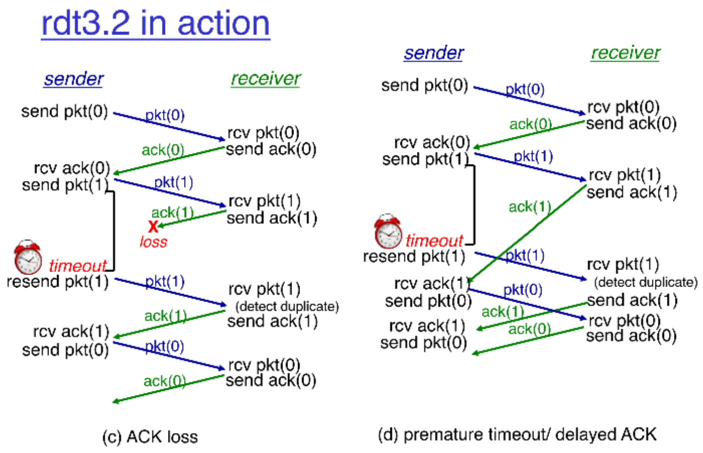


FIGURE 12 – Reliable Data Transfer 3.2

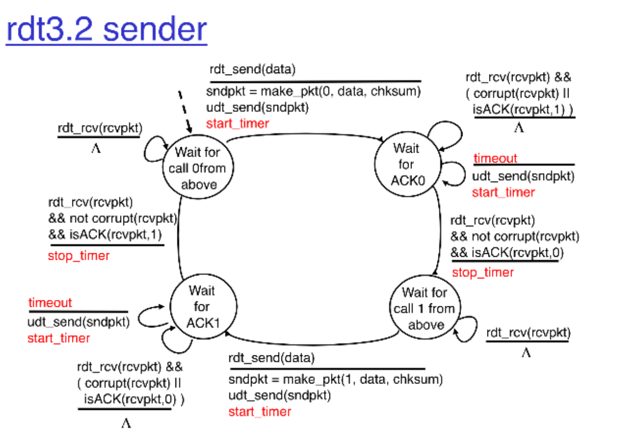

rdt3.2 sender
rdt_send(data)
sndpkt = make_pkt(0, data, chksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0from above
Wait for ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_send(data)
sndpkt = make_pkt(1, data, chksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ

FIGURE 13 – Reliable Data Transfer 3.2 (sender)

4.2 Checksum

We saw that in the header of a UDP segment, there is a field called checksum. The goal of a checksum is to detect errors in transmitted segment. This works the following way :

1. On the sender side :
   (a) The sender treats the segment contents, including header fields as a sequence of 16-bit integers.
   (b) The checksum is addition (one's complement sum) of segment connects.
   (c) The sender puts the checksum value in the UDP checksum field.
2. On the receiver side :
   (a) The receiver computes the checksum of the received segment.
   (b) He checks if the computed checksum is equal to the checksum field value.
      i. NO - error detected
      ii. YES - no error detected

4.3 CRC

An error detector technique used widely in computer network from today is based on **C**yclic **R**edundancy **C**heck (CRC) codes, also knows as polynomial codes, since we can see the bit string to be send as a polynomial whose coefficients are the 0 and 1 values in this string. CRC codes (see fig. 14) operate as follows :

1. We consider d-bit piece of data, D that the sender want to transmit to the receiver.
2. Sender and receiver will agree at first on a $r+1$ bit pattern, knows as generator, which we will denote G. We require that the most significant bit of G is set to 1.
3. For a given piece of data, D, the sender will choose r additionnal bits, R, and append them to D such that the resulting $d+r$ bit pattern is exactly divisible by G using the modulo-2 arithmetic.
4. The process of CRC error checking is then simple, the receiver divides the $d+r$ received bits by G. If the reminder is nonzero, the receiver can conclude that an error occurred, otherwise the data is correct.

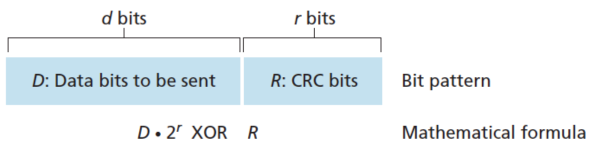
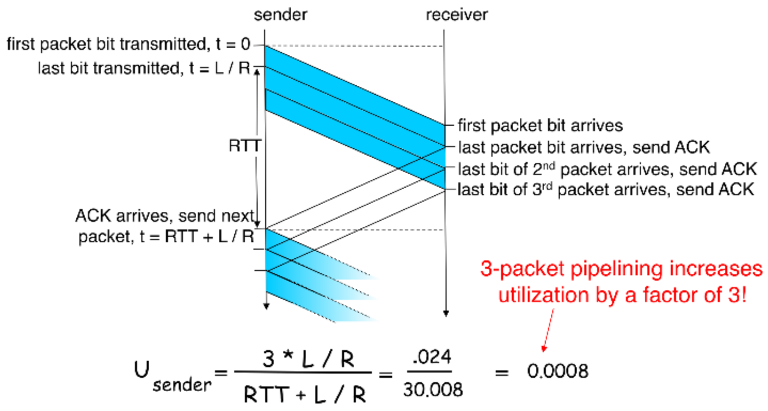


FIGURE 14 – Principle of CRC codes.

5 Reliable data transfer : sliding window protocols and particular TCP mechanisms

We will now talk about another type of protocols. The pipelined protocols. These protocols use parallelisme to improve performances. In pipeling, the sender allows multiple "in-flight" yet-to-be ACKed packets. To do so, the range of sequence numbers must be increased, and buffering is needed for the sender and/or the receiver. On figure 15 we see how pipelined 3 packets can improve the performance.


sender
receiver
first packet bit transmitted, t = 0
last bit transmitted, t = L / R
RTT
first packet bit arrives
last packet bit arrives, send ACK
last bit of 2nd packet arrives, send ACK
last bit of 3rd packet arrives, send ACK
ACK arrives, send next packet, t = RTT + L / R
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

FIGURE 15 – 3-packet pipelining

Anyway, there are 2 generic forms of pipelined protocols :

5.1 Go-Back-N (GBN)

In GBN,

1. Sender can have up to N unacked packets in pipeline
2. Receiver only sends cumulative ACKs, and doesn't ACK a packet if there's a gap.
3. The sender has a timer for the oldest unacked packet. When this timer expires, all the unacked packets are retransmitted.

In short, go-back-n is easy to design, but makes it heavy to retransmit all unacked packets. Let's focus more on it. The sender side :

1. sets a number for the k-bit sequence in the header of the packet.
2. allows up to N (window size, stays constant but window is "moving") consecutive unacked packets to be send. The window is displayed on figure 16.
3. uses only one timer, conceptually for the oldest in-flight packet. When there is a timeout (`timeout(n)`), the sender retransmits packet n and all packets of higher sequence number in window.

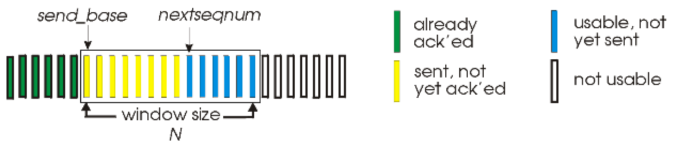


FIGURE 16 – Go-Back-N Window

The FSM of the sender is presented on figure 17.

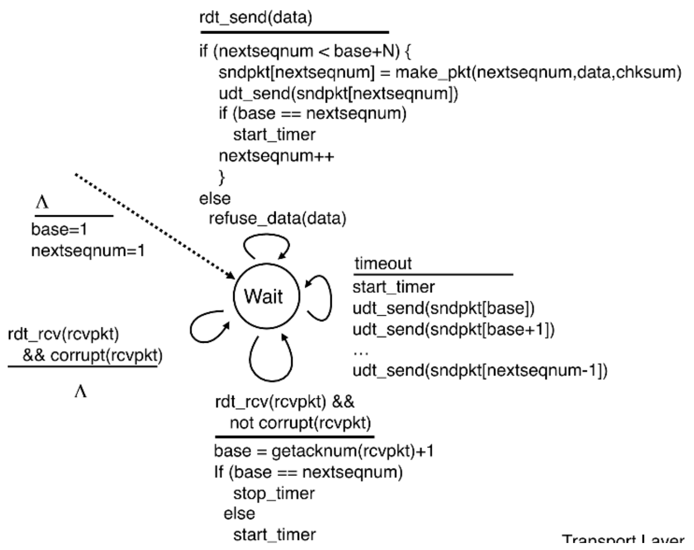


FIGURE 17 – Go-Back-N (sender)

On the other side, the receiver :

1. Only sends ACK. In fact, he always sends ACK for correctly received packets with highest in-order sequence number. He thus only needs to remember the expected sequence number.
2. If he receives an out of order packet, it discards it, and re-ACKs the packet with highest in-order sequence number. It will thus generate duplicate ACKs.

The FSM of the sender is presented on figure 18. On figure 19, we can observe how GBN works.

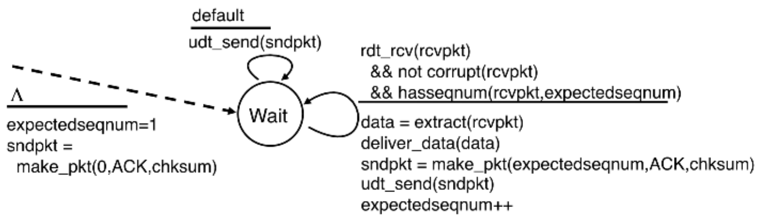


FIGURE 18 – Go-Back-N (receiver)

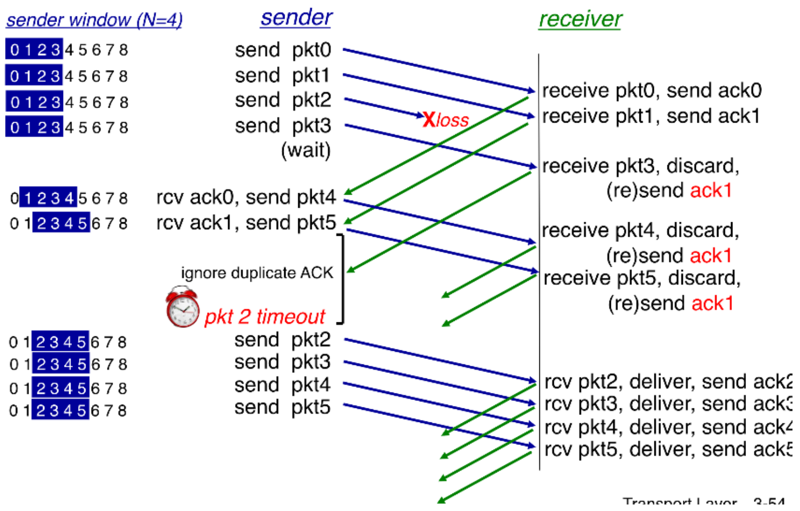


FIGURE 19 – Go-Back-N in action

An important parameter is the maximum window size. In fact N (window size) cannot exceed K (data size), or the receiver will wait for more packet than he shall receive. He thus won't send ACKs for packets received after the first K packets received. We also notice that K and N must be different. In fact, if all the ACKs are lost, the sender will wait for the timeout, then resend the same packets, but receiver won't know that sender did not receive the ACKs so he will consider those "resends" as new packets, and it will thus have duplicates. We then find the condition :

$$window\_size(N) \leq seqsequence\_size(K) - 1$$

This is necessary and sufficient to achieve correctness, when there is no packet reordering in the network. When packet or ACK reordering is possible, the first solution is to choose a timeout so large that when a packet is retransmitted, the sender is sure that the previous copy of this packet and its ACK have disappeared from the network. A better solution is to use a huge sequence space ($K >>$), and keep N much smaller than $K - 1$, and rely on the underlying network to ensure packets and ACKs do not live to long.

5.2 Selective Repeat

In selective repeat,

1. The sender can have up to N unacked packets in pipeline.
2. The receiver sends individual ACKs for each packet.
3. The sender maintains a timer for each unacked packet. When the timer expires, only the unacked packet is retransmitted.

The roles of the sender are :

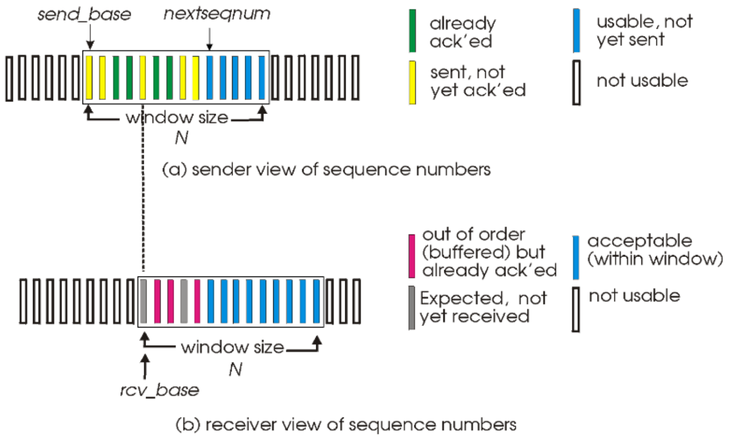
1. With the data from above, if there is a next available sequence number in the window, send the packet.
2. When there is a `timeout(n)`
   (a) Resend packet n.
   (b) Restart `timer(n)`.
3. If it receives an ACK(n) in `[sendbase, sendbase +` $N - 1$`]`
   (a) Mark packet n as received.
   (b) If n is smallest unacked packet, advance window base to next unacked sequence number.

The roles of the receiver are :

1. If it receives a `packet(n)` in `[rcvbase, rcvbase +` $N - 1$`]`
   (a) Send ACK(n)
   (b) If it is out-of-order, put it in the buffer (and then deliver when it is in order).
   (c) If it is in-order, deliver, and advance window to next-not-yet-received packet.
2. If it receives a `packet(n) in [rcvbase -` N`, rcvbase-1]`, it acknowledges it because sender can lag behind receiver, but at most N packets.
3. In other cases, it doesn't do anything and wait for a timeout to occur.

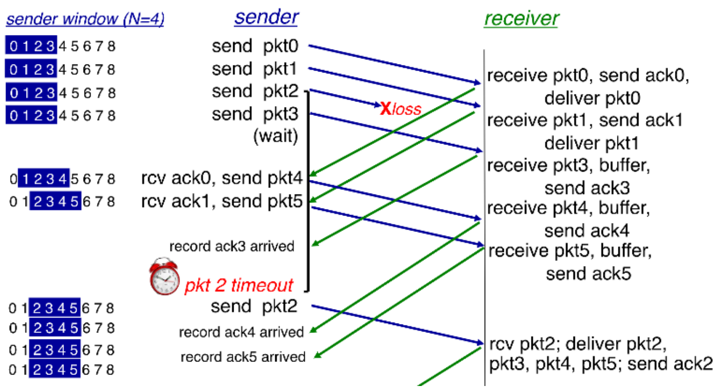
In this case, the sender window has N consecutive sequence numbers and once again limits the sequence numbers of sent and unacked packets. The sender and receiver windows take the form described on figure 20.

We notice that the window is stuck when the lowest packet is not yet acked, but the advantage is that you don't need to resend all packets that have already been acked. The relative positions of the two windows can't be in certain position. Initially, they are both on the same page, then the receiver advances. The sender can't be in front since the packets have to be acked for the sender to move forward. The receiver can't have too much advance neither, since packets must be sent before being acked. The front of the sender can be on the back on receiver though. In fact, this means that every usable packet has been sent, but no acks were received. On figure 21, you can see the selective repeat protocol in action. In this case, when ACK2 finally arrives, the sender will flush all ACKED packet and move on.



send_base
nextseqnum
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
window size N
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable
window size N
rcv_base
(b) receiver view of sequence numbers

FIGURE 20 – Selective Repeat windows

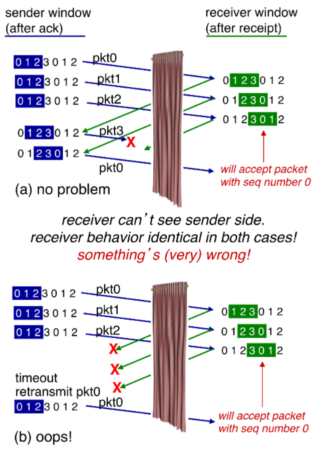


sender window (N=4)
sender
receiver
send pkt0
send pkt1
send pkt2
send pkt3
(wait)
loss
receive pkt0, send ack0, deliver pkt0
receive pkt1, send ack1 deliver pkt1
receive pkt3, buffer, send ack3
receive pkt4, buffer, send ack4
receive pkt5, buffer, send ack5
rcv ack0, send pkt4
rcv ack1, send pkt5
record ack3 arrived
pkt 2 timeout
send pkt2
record ack4 arrived
record ack5 arrived
rcv pkt2; deliver pkt2, pkt3, pkt4, pkt5; send ack2

FIGURE 21 – Selective Repeat protocol in action

Once again, we have to be careful about the size of the window. In fact, for example if sequence number's $=$ 0, 1, 2, 3 and window size, the receiver will see no differences between two scenarios. This case is explained in figure 22. The solution to this issue is to take

$$N \leq K/2$$



sender window (after ack)
receiver window (after receipt)
pkt0
pkt1
pkt2
pkt3
pkt0
will accept packet with seq number 0
(a) no problem
receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!
timeout retransmit pkt0
(b) oops!

FIGURE 22 – Selective Repeat Window Size Flaw

If a packet or ACK reordering is possible, the solutions are the same then for the GBN.

5.3 General Window Size

It is possible to find a general rule for the window size for both GBN and SR. If the sequence space is given by $[0, ..., K-1]$ i.e. modulo K, the maximum sender window size is Ns and the maximum receiver window size is Nr, we say that without packet reordering in network, the protocol is correct if and only if

$$Ns + Nr \leq K$$

.

5.4 TCP

The transport control protocol (TCP) is a reliable data transfert protocol under a unreliable IP service. It is a pipelined protocol but the window size is not fixed, it is handled by congestion and flow control. Congestion control means that the network used to transmit data will be taken into account in the transmission rate choice so that it is not full. Flow control follows the same idea but takes into account the receiver so that he can collect everything it receives. TCP is a connection-oriented protocol, it means that the two sides of the transmissions has to "handshake" before transmission to establish a connection.

TCP, like GBN, uses cumulative ACKs by default but can use SACKs (Selective ACKs) if the option is activated. There are no NAKs, so TCP has to handle duplicates ACK. However, there are no standards on how the out-of-order segments has to be handled, it is up to the developer of the OS to handle them.

There is only one timeout like in GBN but the value of this timeout varies with time according to previous measures. Indeed, it uses the past measures of the RTT to adapt the previous value of the timeout using an exponential weighted moving average which is a technique where the older a measure is the less it is taken into account. To evaluate the timeout value, it adds a safety margin to this computed RTT to handle peaks. This safety margin is also varying over time and computed in the same way. When timeout happens, TCP does not send all unacked packets but just the unacked one with the smallest sequence number like SR protocol.

Thanks to cumulative ACK, some packets can be acked even if the ACK was lost in route. Indeed, the next ACK saying that the previous ones has been received as there is a buffer at the receiver. This particularity can be used to improve efficiency of the protocol by not sending some ACKs. It is called Delayed ACK.

In practice, when the receiver gets a packet, it will wait for the next one before sending anything so that the next ACK will ack both packets at the sender and we remove one message on the network. However, in some cases the ACK response is immediate. For instance, if an out-of-order segments arrives, it will immediately be notified using a duplicate ACK. In the same way, when a gap is filled, an ACK is sent so that the sender can update its list of unacked.

To improve efficiency, another trick is implemented, fast transmitting. The idea is that when the sender receives three duplicate ACKs, it is more than probable that there was a packet loss. So it does not need to wait for the timeout to send back the oldest unacked packet. Figure 23 show the mecanisme of how a fast retransmission is computed.

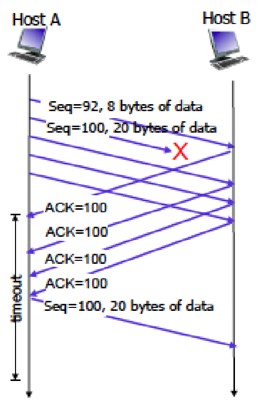


FIGURE 23 – Fast retransmit after sender receipt of triple duplicate ACK.

6 TCP principles : timer, flow control, connection establishment and closure

6.1 Overview of Transmission Control Protocol (TCP)

TCP has the following properties :

1. **Point-to-point :** there's one sender and one receiver.
2. **Reliable, in-order byte stream :** there are no message boundaries.
3. **Pipelined :** TCP congestion and flow control set window size.
4. **Full duplex data :**
   (a) there's a bi-directional data flow in a same connection
   (b) MSS : there's a maximum segment size.
5. **Connection-oriented :** there is a handshake to initialize the exchange.
6. **Flow Controlled :** the sender will not overwhelm the receiver.

The structure of a TCP segment is shown on figure 24. A segment can either deliver a message or deliver an ACK. Here, $K = 2^{32}$ and there is a unique structure for everything. Each bit can be seen as something that can be acked. We notice that there are different flags. Here are their definitions :

1. S/SYN = synchronization flag, first packet to establish the connection.
2. F/FIN = finish, connection finished.
3. A/ACK = acknowledge number is valid.
4. R/RST = reset, used to reset a connection if it didn't work.
5. P/PSH = pass data to upper layer directly.
6. U/URG = mark the segment as an urgent one.

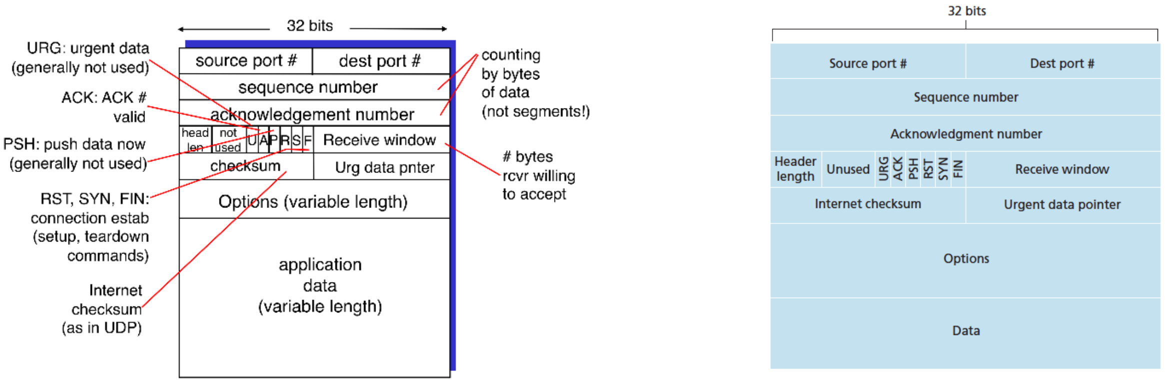


FIGURE 24 – TCP segment structure

TCP also has a particular way of handling sequence number and ACKs. In a TCP connection, all bytes are numbered. The sequence number is the byte number of the first byte of data in the TCP segment. For ACKs, the sequence number is the byte number of the next byte expected from the other side. The ACKs are cumulative, you can also use SACKs (selective ACKs).

6.1.1 TCP round-trip time and timeout

The timeout has to be greater than the RTT, but the problem is that RTT varies. If it is too short, there will be unnecessary retransmissions. Also, the longer the timeout value is, the slower the reaction to a segment lost will be. An important thing is thus to estimate the RTT. To do so, we'll find a SampleRTT, the measured time from segment transmission until ACK receipt. The retransmissions will be ignored. The problem is that this

SampleRTT will vary, and we want a smoother estimation. An average value is thus taken over several recent measurements. Finally, this formula will be used :

$$EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$$

With this formula, the influence of past samples decreases exponentially ($\alpha,\ \alpha(1-\alpha),\ \alpha(1-\alpha)^2$...) fast as we go forward. A typical value for α is 0.125.

With this estimation of RTT, we can now compute the timeout interval. It is given by EstimatedRTT plus a safety margin, given by the deviation of RTT DevRTT. This deviation can be computed as follows :

$$DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$$

In the end, we have :

$$TimeoutInterval = EstimatedRTT + 4 * DevRTT$$

6.2 Flow Control

An important point of TCP connections is the flow control. The idea behind it is that receiver controls the sender, so that the sender won't overflow the receiver's buffer by transmitting too much too fast. Indeed, the receiver will advertise his free buffer space by including receiver window (*rcwd*) in TCP header of receiver-to-sender segments. The sender will then limit the amount of unacked "in flight" data to receiver's receiver window value. This guarantees that the buffer will not overflow.

However, there are ways to improve the performance of this method :

1. Use larger windows
   (a) Precautions have to been taken : TCP throughput cannot exceed $rcwd/RTT$.
   (b) You can only use 16 bits in the TCP header to encode *rcwd*, so max $rcwd_{\max} = 2^{16}$ bytes, thus max TCP throughput is 64 Kbytes per RTT
   (c) However, a TCP option allows to scale the window size by a factor 2^k, with $k \leq 14$, thus leading to a max $rcwb = 2^{30}$ bytes.
2. Nagle algorithm : Reducing overhead by grouping bytes when sending app writes one byte at a time in socket. (see after)
3. Silly window syndrome : Same as nagle, without grouping bytes (see after)

6.2.1 Silly Window Syndrome

Silly Window Syndrome is a problem that arises due to poor implementation of TCP. It degrades the TCP performance and makes the data transmission extremely inefficient. The problem is called so because

- It causes the sender window size to shrink to a silly value ;
- The window size shrinks to such an extent where the data being transmitted is smaller than TCP Header ;

The two major causes of this syndrome are

1. Sender window transmitting one byte of data repeatedly ;
2. Receiver window accepting one byte of data repeatedly ;

Cause-1 : Sender window transmitting one byte of data repeatedly.

Suppose only one byte of data is generated by an application. The poor implementation of TCP leads to transmit this small segment of data. Every time the application generates a byte of data, the window transmits it. This makes the transmission process slow and inefficient. The problem is solved by Nagle's algorithm which work as follow

1. Sender should send only the first byte on receiving one byte data from the application ;

2. Sender should buffer all the rest bytes until the outstanding byte gets acknowledged ;
3. In other words, sender should wait for 1 RTT(Round Trip Time) ;

After receiving the acknowledgement, sender should send the buffered data in one TCP segment. Then, sender should buffer the data again until the previously sent data gets acknowledged.

Cause-2 : Receiver window accepting one byte of data repeatedly.
Suppose the case when the receiver is unable to process all the incoming data. In such a case, the receiver will advertise a small window size. The process continues and the window size becomes smaller and smaller. A stage arrives when it repeatedly advertises window size of 1 byte. This makes receiving process slow and inefficient.The solution to this problem is Clark's solution which works as folow

- Receiver should not send a window update for 1 byte ;
- Receiver should wait until it has a decent amount of space available ;
- Receiver should then advertise that window size to the sender ;

Notes. Nagle's algorithm is turned off for those applications that require data to be immediately send. Nagle's algorithm can introduce delay as it sends only one data segment per round trip. This is useful for instance for Telnet, where we have 41 byte segments containing 1 byte of data. However, there might be issues with Nagle. In fact there could be an additional delays at the end of the connection, if for example the TCP buffer contains slight more than 1 MSS at the end. If he sends a packet with an odd sequence number, the delayed ACK will be active, so the ACK will be send after 200ms, and the algorithm makes the sender wait those 200 msecs. To solve this, we use the relaxed Nagle's algorithm, with this Nagle allows to send a packet smaller than MSS if previous packet was full size, so no problem.

Both Nagle's as well as Clark's algorithm can work together. Both are complementary.

6.3 Connection Management

Before exchanging data, the sender and the receiver will do a handshake. With it, they will agree to establish a connection, and on the connection parameters. To open a connection there must then be a handshake. 2 way handshake ? No, it's too naive and does not work in practice

1. Variable delays
2. Restransmitted messages
3. Can't see other side...
4. Different failures can occur (see figure 25)

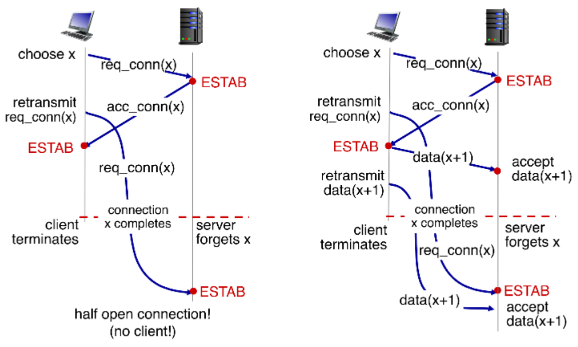


Figure 25 – 2 Way Handshake Failures

We must thus use the three way handshake. This is done like described on figure 26.

The 3 way is more robust than the 2 way. If the requester retransmit a SYN(x) due to premature timeout, and this SYN(x) is receive after the termination of the first exchange, the server will respond with a SYNACK(y', x). Therefore, the server must be careful an choose y' such as it's different from y use in the first exchange but

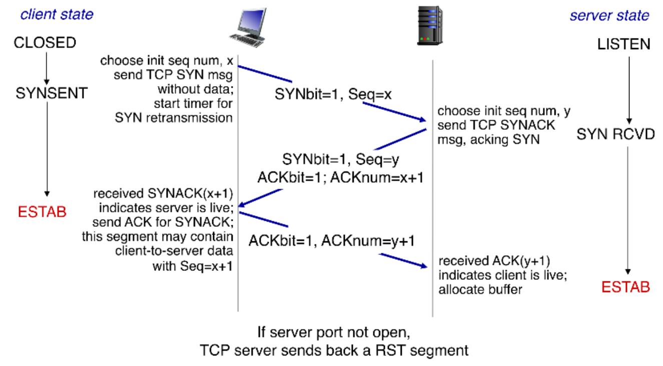

client state
CLOSED
SYNSENT
ESTAB
server state
LISTEN
SYN RCVD
ESTAB
choose init seq num, x send TCP SYN msg without data; start timer for SYN retransmission
SYNbit=1, Seq=x
choose init seq num, y send TCP SYNACK msg, acking SYN
SYNbit=1, Seq=y ACKbit=1; ACKnum=x+1
received SYNACK(x+1) indicates server is live; send ACK for SYNACK; this segment may contain client-to-server data with Seq=x+1
ACKbit=1, ACKnum=y+1
received ACK(y+1) indicates client is live; allocate buffer
If server port not open, TCP server sends back a RST segment

FIGURE 26 – 3 Way Handshake

also such as it's different from all previous recent exchanges that could lead to the same problem (= choosing as less than TCP maximum segment lifetime). In practice, they are picked at random by TCP because too difficult if we have to keep track.

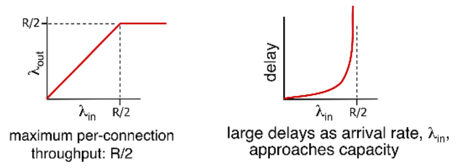
Of course, it is also important to close the connection. They will each close their sending side of connection by sending a TCP segment with FIN bit = 1. They then respond with an ACK. By the way, on receiving FIN, the ACK can be combined with own FIN, and simultaneaous exchanges can also be handled. However, there are flaws. An heart beat mechanism is implemented, but if these messages are lost too, a side might still close while it shouldn't. There's no perfect solution, this can be compared to the two army problem.

7 TCP congestion control and its properties

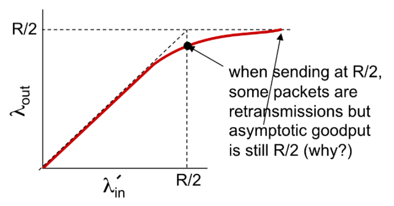
It's important to distinguish congestion control from flow control. We can describe it informally as "Too many sources sending too much data too fast for network to handle". In this case, the receiver cannot detect the congestion, while the sender can, because of packet loss and additional delays. There are different causes and associated costs of congestion.

1. Scenario 1 :
   (a) We have 2 senders and 2 receivers, one router with infinite buffers (unrealistic), an output link capacity of R and no retransmission. The sending rate is of λ_{in}, while the receiving rate is λ_{out}.
   (b) With this setup, the maximum per connection throughput is of $R/2$. Therefore, we will have large delays as the arrival rate approaches capacity (exponential curve).
2. Scenario 2 :
   (a) We have 2 senders and 2 receivers, one router with finite buffers (realistic, with losses). We have $\lambda_{in}, \lambda_{out}$ and $\lambda'_{in} =$ original data + retransmitted data. The retransmission will therefore have an impact on λ_{out}.
   (b) If there's no packet loss (no retransmission), we have $\lambda_{in} = \lambda_{out} = \lambda'_{in}$. This is impossible, it implies that the sender has information about how much data is in the buffer.
   (c) Another case would be to suppose that only lost packets not ACKed are retransmitted. The emission rate is equal to the reception rate, until a certain point. Then, the number of lost packets increases, λ_{out} becomes smaller than λ'_{in}.
   (d) For the third case, we consider that all packets are retransmitted once. The retransmission of packets that are delayed will increase λ'_{in} with a constant λ_{out}. If $\lambda'_{in} = \lambda_{out}$, no retransmission was needed, we'll talk about goodput. In this case, congestion has the following cost : More work and unneeded retransmissions.
3. Scenario 3 :
   (a) Here, we have 4 senders and 4 receivers. There are different possible paths. Retransmission is done where there's a timeout. During the first retransmissions, there's a cascade effect. The overload of transmitted packets adds up to the existing overload and the system just collapses.

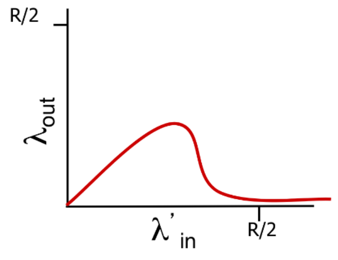
The different scenarios have the following results.



(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.

FIGURE 27 – Plot of λ_{in} by λ_{out} for the 3 scenarios presented above.

There are 2 approaches for congestion control.

1. End-end congestion control :
   (a) No explicit feedback from network.
   (b) Congestion inferred from end-system observed loss, delay.
   (c) This is the approach taken by TCP.
2. Network-assisted congestion control :
   (a) The routers provide feedback to the end-systems.

7.1 TCP congestion control

The goal is to transmit as fast as possible without congesting the network, this is thus decentralized. In fact, each TCP sender sets its own rate, based on implicit feedback :

1. ACKs : segment received, network not congested, so we can increase sending rate.
2. Lost Segment : assume loss due to congested network so decrease sending rate (sometimes TCP infers wrongly a congestion, there are other errors (e.g. bit errors)).

As said earlier, the sender will increase transmission rate until a loss occurs. When the sender increases the transmission, he does so using an additive increase. In fact, the congestion window will be increased by one MSS every RTT until a loss occurs. At that point, multiplicative decrease is used. The *cwnd*[2] will then be cut in half. The rate is never regular, it's the Saw Tooth behaviour. In short, the sender will limit its transmission with :

$$LastByteSent - LastByteACKed \leq cwnd$$

with *cwnd*'s size evolution described earlier. Let's not forget that the sender is also limited by *rwnd*. The limit is actually $\min(cwnd, rwnd)$. Roughly, we can say that the rate is given by

$$rate \approx \frac{cwnd}{RTT} \left[\frac{\text{bytes}}{\text{sec}}\right]$$

An important aspect is the detection and the reaction to losses. This is implemented in TCP Reno (from the city where it was first deploy). It works the following way

1. A loss can be indicated by 3 duplicate ACKs. In that moment, we're in what's so called mild congestion.
   (a) The duplicate ACKs indicate that the network is capable of delivering some segments.
   (b) In this situation, the cwnd is cut in half, the window then grows linearly.
   (c) This can only work if the $cwnd \geq 4$ MSS
2. In other cases, the loss is detected by a timeout. In that case, we're in the case of a severe congestion.
   (a) The *cwnd* is then set to 1 MSS.
   (b) The window then grows exponentially.

Another important concept of TCP congestion is a mechanism called slow-start. It is showcased on figure 28 Explanations :

1. When the connection begins, the transmission rate will increase exponentially until the first loss occurs.
2. Initially, cwnd size = 1
3. The cwnd size is doubled every RTT.
4. This is done by the increment of *cwnd* for every ACK received.

After this slow start technique, the algorithm needs to stabilize, it will go into the congestion avoidance mode. At some point, the exponential increase will switch to a linear one. This happens when *cwnd* gets to half of it's value before timeout. In case of a loss, this value is set to half of the window size before the loss. The linear growth follows the following law

$$cwnd = cwnd + (MSS/cwnd) * MSS$$

for each ACK received.

2. This abbreviation stand for congestion window.

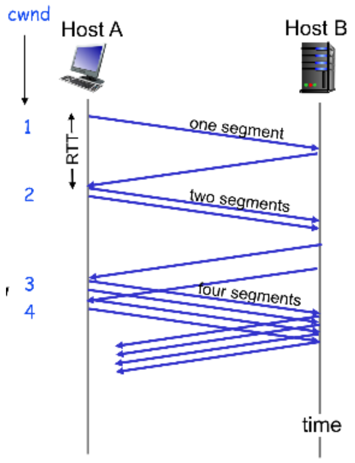


FIGURE 28 – The slow start mechanism

7.2 TCP goodput in steady state

The consequences of this congestion is really important. In fact, this protocol could kill the throughput. If there is congestion in the network and the TCP slows down its rate, it could make the throughput really low.

What is the TCP goodput in steady state. Steady state means that the congestion level is constant. We ignore the slow start. We have the behavior of figure 29. In this behavior, we see a cycle. We want to compute the number of segments sent in a particular segments. It is approximately of 3W/4 * W / 2. We claim that this is the inverse of the packet loss rate. In fact, in a cycle, we sent $3W^2/8$. If we lose one packet loss over all those. So yeah, it's the inverse. The assumption of one packet loss per cycle isn't great but whatever. With this, we can compute the maximum window size. We see that this depends only on one thing, the packet loss rate. The average goodput is given by the average window/RTT. We use the value of the window, and we find

$$\begin{aligned} Avg\_Goodput &= \frac{3W}{4RTT} \\ &= \frac{3\sqrt{\frac{8}{3p}}}{4RTT} \\ &= \frac{6\sqrt{2}}{4\sqrt{3p}RTT} \\ &= \sqrt{3/2}\frac{1}{RTT\sqrt{p}} \end{aligned}$$

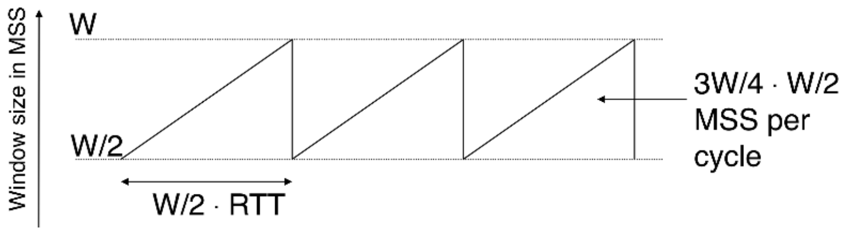


FIGURE 29 – Steady State

What does this tell us. The average goodput in inversely proportional to the RTT. The packet loss rate also kills the throughput, this is also logical. Finally the size of the segment is also very important. Using bigger segments will also increase the goodput. Once again this is logical, because you'll have to send more packets. Of course, this is super simplified, there's no slow start and we're in steady start. But what does this mean ? If you want a big throughput on a long fat pipe, you'll need a very small packet loss rate. This TCP is OK on short distance with fat pipes or long distance on less fat pipes. For this, we need new versions of TCP, used for high-speed.

7.3 TCP fairness

With this, we have a new question : "Is TCP fair ?". Imagine K TCP sessions share the same bottleneck link of bandwidth R. Each should have an average rate of R/K. Let's imagine two connections share a bottleneck, and that they have the same RTT. The sum of their rate has to be equal to R. We thus have a triangle, and we can not go further than the triangle. We can easily find a nice point. We can study the behavior described on figure 30.

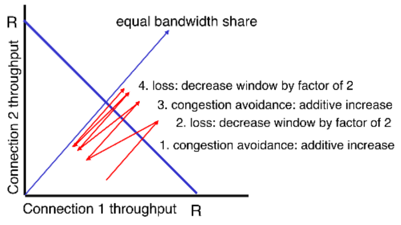

R
equal bandwidth share
Connection 2 throughput
4. loss: decrease window by factor of 2
3. congestion avoidance: additive increase
2. loss: decrease window by factor of 2
1. congestion avoidance: additive increase
Connection 1 throughput
R

FIGURE 30 – TCP Fairness

Now let's study the case, where the RTT are different. Let's take an example with RTT of connection 2 is twice the RTT of connection 1. In this case, connection 1 will increase twice as fast as connection 2. We have the behavior described on figure 31. In this case, we'll have what we call a proportional fairness. The banwidth share will be inversely proportional to the RTT. Indeed the fairness will depend on the RTT of the connections.

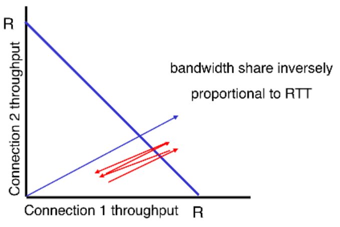


FIGURE 31 – TCP Fairness 2

But in the end, is this fair ? From a user perspective, this looks unfair. But if we take the view of the network, this looks more fair. But in the end, this is even more complicated. Indeed, some apps will use UDP instead of TCP because they don't want to see their rate become smaller.

8 Network layer : Data plane versus control plane ; router architecture ; IP addressing and forwarding

8.1 Data plane vs Control Plane

First we note that there are two key network-layer functions.

1. Forwarding : move a packet from router's input to appropriate router output. Very local, has to been done quite fast. Part of the data plane.
2. Routing : determine route taken by packets from source to destination. It is part of the control plane.

We can then really separate the data plane and the control plane.

1. The data plane :
   (a) Local, per-router function.
   (b) Determines how a datagram arriving on router input port is forwarded to router output port. It is the forwarding function. To do this, you look at the values in the arriving packet header.
2. The control plane :
   (a) Has a network wide logic.
   (b) Determines how the datagram is routed.
   (c) There are 2 control-plane approaches.

   (a) Traditional routing, implemented in routers (see chap 5)
   (b) SDN = software-defined networking, it is implemented in servers.

We can compare the 2 approaches. In a network, you have a certain number of routers that are connected. In each of these routers, we need a table. This is called a forwarding table. On the left of the table, we try to match certain header fields (addresses), and you forward the packet to the right output interface. But these tables have to be computed. In each router, there is a process (daemon) that will talk to the other processes (routers) in the network. This is a distributed interface, the routers have to talk to each other. After this talk, the tables will be created (or updated). Of course, these tables have to match each other. The routers will work in the data and the control plane. This is the 1st approach.

The other approach is more centralized. In this approach, the control plane is removed from the routers. We can look at the control plane as a remote controller, there will be a centralized controller that will compute the tables, and update them. This new technique is a paradigm shift for the big players.

The network layer can offer a lot of different services. In fact, if this layer already implements a lof of services, you don't have to implement them in the upper layer.

8.2 Addressing and Forwarding

The data plane is based on two things : the existence of the forwarding table and the IP protocol. The data plane must therefore implement the IP protocol, which has addressing conventions, gives a certain format to datagrams and has conventions for packet handling.

8.2.1 Addressing

To introduce us to addressing, let's first look at a simple example. This example is showed on figure 32. All the end systems are connected to the routers. To do this, they need an interface (wired, wireless...). We notice that IP addresses are actually associated with each interface and not with the machines. Indeed, today, end systems can have multiple interfaces (very few but still)... But why not use an IP address for each node instead of this ? This is related to the structure of the interfaces. We'll come back to this later. We note that the IPV4 addresses use 32 bits, and a decimal notation. IPV6 addresses use hexadecimal notation.

We also see that some systems are connected in a "cloud". These clouds are called subnets. Let's imagine we isolate the parts of the network. The systems in the subnet can still communicate between each other. This

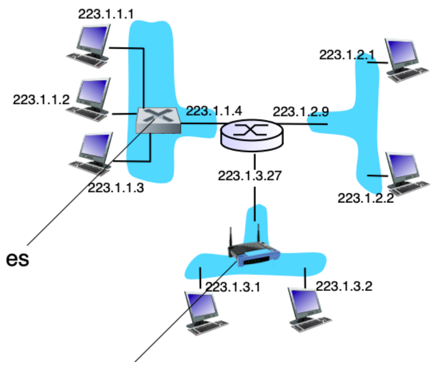


FIGURE 32 – IP Addressing

is useful, since when you want to reach an end-system, you'll have to reach a subnet and not the end-system, it simplifies our problem. Interfaces that are part of a same subnet can have the same structure. They'll have the same prefix, while their suffix will be used to identify the end-system. You can then forward a packet by mostly looking at the prefix of the IP address, the suffix will only be needed locally. In some cases, you only need a really small suffix, (example of subnet defined only by 2 routers). We thus have to find a clever way of choosing the number of bits used for suffix and prefix, so we don't lose some space.

In the past, different classes were used to determine those sizes. If we had a really large network, we would have used class A, so 24 bits would have been used in the suffix. The problem was that many organizations wanted a class A. So it was quickly depleted. Same problem for class B (16-bits suffix). Then people only had access to class C (8-bits suffix), which was too small, so organizations wanted multiple "class C". The problem with this is that it is too static.

Today, CIDR (**C**lassless **I**nter-**D**omain **R**outing) is used. With this technique, the subnet portion of address is of arbitrary length. The address format is therefore : a.b.c.d/x, where x is the number of bits in the subnet portion of address. There are also what well call special IP addresses.

1. The address full of zero is for when you don't know yet your address.
2. You can fill the prefix with 0, and then fill the rest with an host address to reach another host on the network.
3. Full of 1 means you want to broadcast on the local network.
4. Filling the prefix with a subnet and then full of 1 means to broadcast on a distant network.
5. Filling the prefix with 127 will make you loopback.

But how can one get an IP address ? Historically, hard-coding was used and IP address were hard-coded by a system in a file. Today, we employ a plug-and-play approach called DHCP : Dynamic Host Configuration Protocol. So you don't just set an address, you configure the host. See next section for full explanation of this.

8.2.2 Forwarding

We are now studying the data-plane. How do we forward packets ? The difficult part is to forward packet from subnet to subnet. When in a subnet, it's pretty easy. First we can then make the forwarding table by reducing it to the number of subnets instead of having 2^{32} IPv4 addresses. You can have further aggregation with the ISP's. If an ISP acquires another, and a organizations connects trough the acquired ISP, it can keep its address by telling the ISP who will then tell the internet. You have to be careful though, since the router then has a choice, and might not reach the destination by going a certain way. You have to be careful with overlap. The solution is then to use the most specific address, the one with the longest prefix that matches the destination address. You can't implement this naively, or this would be too slow. We note that the matching is often performed in hardware using Ternary Content Addressable Memories (TCAMS). This can retrieve the action in one clock cycle regardless of the table size. Efficient software implementation use tries (digital tree) that can be browsed quite efficiently.

8.3 Architecture of a router

We can see how a router looks on figure 33. We see a certain number of physical ports. All ports are bidirectional even if they are not represented as such on the figure. In the middle, there's a switching fabric. The interface ports are connected to this fabric. When packets come in, they are processed, then they go through the fabric, before being processed again and then sent. This is done at line speed. There's also a routing processor running the routing algorithms, that's the control plane. When the processor is done, the forwarding table is uploaded then copied at the input ports. The forwarding table will thus be copied in every incoming interfaces. Let's look at the differents pieces.

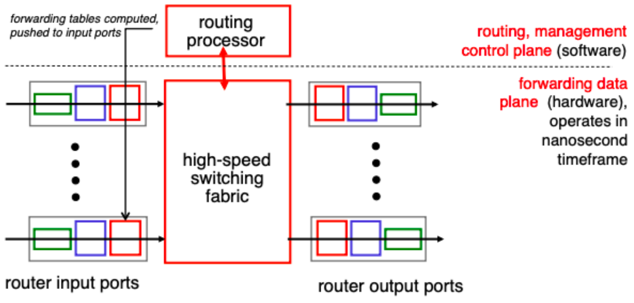

forwarding tables computed, pushed to input ports
routing processor
routing, management control plane (software)
forwarding data plane (hardware), operates in nanosecond timeframe
high-speed switching fabric
router input ports
router output ports

FIGURE 33 – Router Architecture

8.3.1 Input ports

The input ports receives a signal that can be of various forms (photons, waves, light, etc). The first stage is thus bit decoding (this is done by the physical layer). The link layer will help us to isolate packets from each other, it's smarter than the physical layer. It will check that the blocks are correctly received. The processing depends on the protocol in that layer (Ethernet,wifi,...). If it's ok, we push the data (IP packet). We have to forward this packet. The main job of the block is therefore the lookup (longest prefix matching) and then forward the packet. It then pushes this to the right interface. Then we go through the switching fabrics.

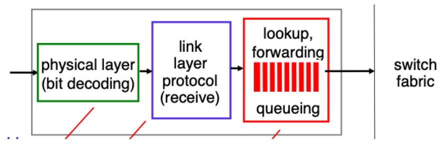


FIGURE 34 – Routers' input ports.

8.3.2 Switching Fabrics

There a 3 types of switching fabrics, each shown on figure 35

1. Memory :
   (a) We use the standard architecture of a computer with switching under direct control of CPU.
   (b) The packet is copied to the system's memory.
   (c) The speed is limited by memory bandwidth, and 2 bus crossings/datagram.
2. Bus :
   (a) The datagram is moved from input to output port memory via a shared bus.
   (b) The switching speed is limited by bus bandwidth.
3. Interconnection Network (crossbar)
   (a) With this, we overcome the bus bandwidth limitations, since we use multiple buses in parallel.

(b) When you write something on the horizontal buses, it will be copied on the vertical ones only where the 'dots' are enabled. With this, you can implement parallelism. The maximum level of parallelism is the number of row/columns you have. If you go over this, you have collision.

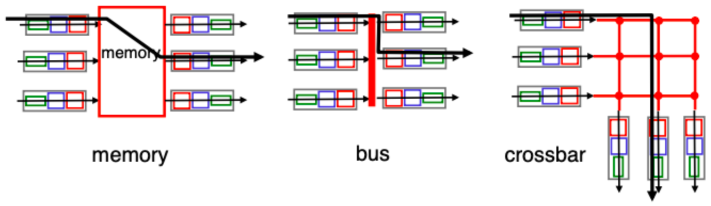


FIGURE 35 – The 3 types of switching fabrics

8.3.3 Input port queuing

The fabric is slower than the input ports combined. Therefore, queuing may occur at input queues. We have to be careful, we can't take packets that want to go to the same output port at the same time. When a packet is blocked, it will block the packets behind him even though they could have though. This is called head-of-the-line blocking. We can see the scenario on figure 36. We can look further in the queue to improve performances (swap packets is okay if there are unlikely to be from the same flow). However, this is and it can introduce another bottleneck. So compromise compromise.

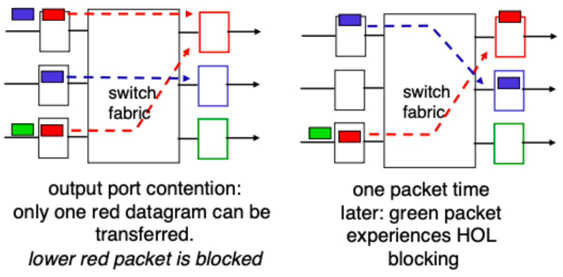

switch fabric
output port contention: only one red datagram can be transferred.
lower red packet is blocked
switch fabric
one packet time later: green packet experiences HOL blocking

FIGURE 36 – Head of line blocking

8.3.4 Output Ports

There is a usually a big queue at output ports. This is the real bottleneck. We have to be careful to not overflow, in fact datagrams can be lost due to congestion. Here the packets are processed as we go down the layers. In the datagram buffer, fragmentation is done if need, and scheduling discipline can be implemented (i.e. choosing among queued datagrams for transmission)

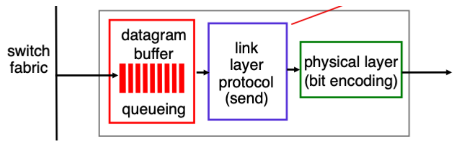


FIGURE 37 – Routers' output ports.

9 Principles of IP : fragmentation, DHCP, NAT, IPv6

9.1 Fragmentation

Fragmentation is needed when a packet is too big to cross a given link. You have to design IP such that it can work with every technology. If the MTU (**M**aximum **T**ransfer **U**nit) is too small, we have to fragment the packet. Each fragmented packet has to become "self sustaining", it has to be able to be forwarded. We note that the packets are reassembled at the end. If there are reassembled before final destination, it would be called transparent. This allows to have less overhead on each link. But in IP, we use non transparent. Maybe you'd have to refragment again, even if we want to avoid that. Also, the fragments might follow differents paths, but there will be at the same destination. If you want to refragment in the middle, that's a problem. Also if you have to wait for the different fragments, you might create congestion by creating a queuing delay. On another note, you have to be careful with fragmentation, since they can be refragmented. This is the technique proposed by IP :

1. An ID is used to identify the packet (valid for enough time) and to put back together data fragments after (re)fragmentation.
2. The **fragflag** is used to tell if it's a part of a fragmentation.
3. The offset is used to tell us where the fragment is supposed to be contained in the original packet. Those offsets are computed by computing "multiples of 8 bytes". The fragflag of last fragment is a copy of fragflag before fragmentation, to make sure this is done correctly.

A question that comes is : is it possible to avoid fragmentation ? So we would have to find the bottleneck of the link. This is called the Path MTU discovery., it works like this :

1. Send an IP packet with the "don't fragment flag" set.
2. The router may be forced to discard the packet.
3. The source is told the failure with an ICMP error message. It tries again with a size equal to the MTU indicated.

This may work but not always :

1. This relies on routers properly returning ICMP error msg. Might not be the case. Routers can get attacked.
2. The ICMP message could be lost.
3. If the router updates, we do the path MTU discovery again.

9.2 DHCP

DHCP (for **D**ynamic **H**ost **C**onfiguration **P**rotocol) follows a leasing approach, you use the address for a certain lease time. In fact, when the computer isn't connected, you don't need this address anymore, so the address can be released. This approach also allows us to fight against the depletion of IP addresses. With this, you can design a smaller subnet and assign the addresses dynamically, and allow mobile users. You can separate this problem into different phases. Let's say that a new client arrives in a subnet, and he needs an address in this network. We assume that there's a DHCP server (there might be several servers) in this subnet. All this exchange is described on figure 38 and here under.

1. First of all, the system that does not know his address probably doesn't know where the DHCP server is and what subnet he's in. So he broadcasts a "DHCP discover" msg. This phase is optional. In fact, if the address expires, we have to renew, but no need to rediscover everything.
2. The DHCP servers replies with a DHCP offer, also sent in broadcast, since the arriving client doesn't have an address yet.
3. The client then has to reply with a DHCP request, telling that he'll take the address. This is necessary, since there might be multiple DHCP servers. In fact, the servers need to know which proposal was taken. Finally, a DHCP ack is sent by the server.

4. All those exchanges are made in Broadcast and using a UDP connection. One can think that the client asking for an IP address could send response to the server using unicast, but the broadcast permit to the possible other proposal servers to be inform about the client's choice. Also here, TCP can't be used connection since to use TCP, you need to establish a connection, and for that you need to know the IP address.

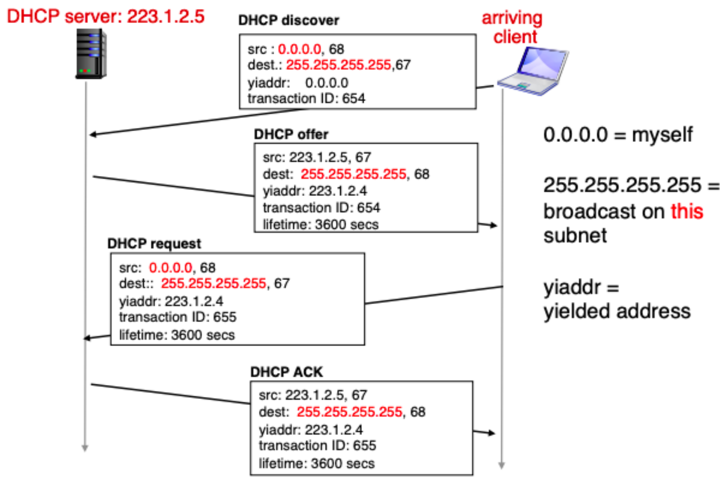


FIGURE 38 – DHCP Exchange

But let's not forget that DHCP is not just used to give an IP address to the client, it will do an entire configuration on the host. In fact, it will give the name and IP address of his default DNS server (maybe multiple proposals). It will also tell you the limitation between the prefix and suffix '/x' (the network mask). Finally, the address of the first-hop router for client is given. It's the "exit door". There are other things too (domain name,...)

We also note that edge routers can have the DHCP server built into them. You can sometimes even enable and disable them. It is fine to do that, can help in situations where router is attached to let's say three small subnets. Makes it way simpler. (see example of slide 17, the DHCP server could be in the router, this would still work perfectly fine and make the network simpler).

The DHCP also has to get his "space", his block of address. A network gets allocated a portion of it's provider IPs address space, so-called Provider Assigned (PA) addresses. An ISP will therefore give a sub-block of it's space. ISP blocks can also be sub-blocks from bigger organizations. This hierarchical addressing allows efficient advertisement of routing information, but lacks flexibility. However, there is an alternative. In this, some organization can have a Provider Independent (PI) range of addresses. They can have multiple providers (and this is called multihoming).

The last question to answer is the allocation of those address. This is done by ICANN (= Internet Corporation for Assigned Names and Numbers). Historically, you asked for an address and you got it. Problem with universities that got allowed big blocks of addresses, and they did not want to give it back. Those issues were solved (somehow).

9.3 Network Address Translation

A big problem is that we still lack addresses, we can't assign blocks this easily. The equipment venders therefore came out with a solution (NAT). This uses the idea that among some of those IP addresses are not rootable. You can therefore create private networks by using only one address in the global internet. This is great, because uses only one IP address, you add security since the private addresses are not easily accessible, you can easily change ISP, and you can change the private addresses without notifying the outside world. But how can you then communicate with the outside ?

In the router, there will be a router called the NAT translation table. The NAT router must :

1. For the outgoing datagrams : replace (source IP address, port sequence) of every outgoing datagram to (NAT IP address, new port sequence).
2. For incoming datagrams : replace (NAT IP address, new port sequence) in destination fields of every incoming datagram to (source IP address, port sequence) stored in NAT.

Why change the port sequence ? Because the computer could have send a port number itself. Here the port number is really important. A new fictitious port number is issued for every connection. We have a 16-bit port number field, which means that there can be 60 000 simultaneous connections with a single LAN-side address. You can also close a door, by checking the destination to protect the network. But you have to be careful not to be to restrictive, or you'll slow down the network.

There's a controversy with NAT : in fact, here, routers intervene in a layer where there shouldn't. By playing with the port numbers, it's the case, and that's not great. It also violates the end-to-end argument that everyone should be able to communicate with everyone. So to solve the address shortage, using IPv6 is a better idea. There's also the traversal problem.

Let's imagine a client outside wants to connect to the server with address 10.0.1. There's only one externally visible NATed address. We can statically configure the NAT that will forward incoming connection requests at given port to the server. A second solution is to allow the NATed host to learn public IP address and to add/remove port mappings. A third solution is to use relaying for p2p systems (skype). Here, a NATed server establishes a connection to relay, and the external client will connect to the relay. The relay will bridge the packets between the two connections.

9.4 IPv6

The main motivation behind this is to have much more addresses. With IPv6, we could also improve the processing in routers, so we don't allow the fragmentation. This change did not induce a change in higher and lower protocol layers. We can see from figure 39 that the IPv6 differ by several points from the IPv4 protocol.

9.4.1 Datagram Structure

We can look at the size of the address in the headers to see the difference. We see that the new space is of 128 bits. With this, there's more than enough space. Also added to the header of IPv4 is a priority field to tell the priority of datagrams (can implement quality of service), a Flow Label to help to identify the datagrams in the same flow, and a Next Header that allows to identify the upper layer protocol (TCP/UDP) for data and to daisy chain several extension headers (to add security). The checksum removed to reduce processing time. The options are now chained by Next Header and are therefore removed.

9.4.2 Addresses

Let's now look at the address. Here, we now use hexadecimal. The 128 bits are thus written as x :x :x :x :x :x :x :x where x is a 16-bit hexadecimal field. If you have a field full of 0, you can replace it with just one 0. Successive fields of 0 are represented as : : but only once in an address.

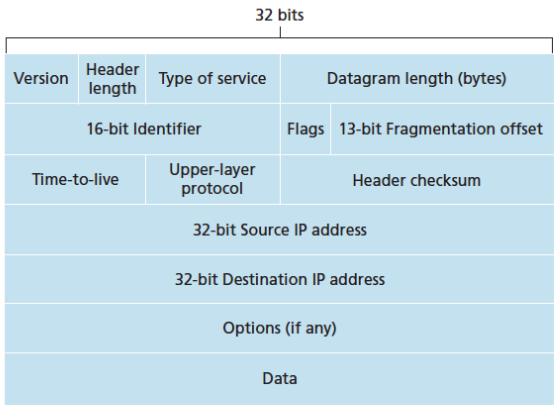
IPv6 still uses hierarchical addresses to allow for aggregation. We still have a "prefix" to identify subnets. The address is cut in 2 parts. The first 64 bits are used to identify a site. The company can then reallocate the 64 other bits the way it wants. The first 64 bits are themselves structured. Usually, 48 bits are already given by ISP, you can then use the remaining 16 to structure how you want. The DHCP can fill the suffixes. But the suffix is so big, we can use other tricked to allocate it. One technique is to use the physical address (layer 2 address or MAC address), unique number in the world. The problem with this, is that the same physical address will always be used wherever you are, so you could be tracked. Another technique is to generate randomly the suffix. With this, no risk of being tracked. But problem that if an attacker uses this, he can change its address really fast so hard to track.

9.4.3 Transition from IPv4 to IPv6

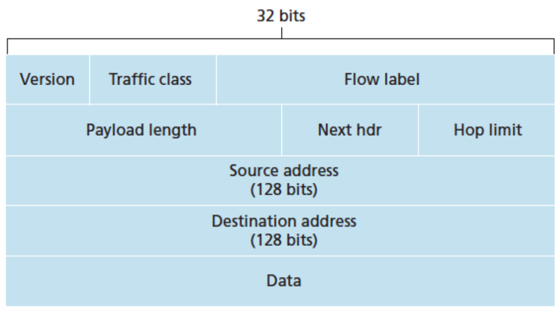
It is impossible to upgrade all the routers simultaneously, no flag days and still that would be just impossible. Also, the companies that handle this have different purposes. Some router vendors are doing this, others not + the differents OS had to decide to implement to IPv6 or not,... All this complicates the transition.

Nowadays, we have dual stacks. Most routers and systems have both. The technique of tunneling is used to connect IPv4 and IPv6. The idea is that when you have an IPv6 packet that can't go through an IPv4 network, we encapsulate the IPv6 in an IPv4 packet. With this, you just have to change a field of the header to say that this is IPV6 instead of TCP or UDP. You apply the encapsulation when you're at the "edge" of the tunnel, and you Decapsulate when you're at the other edge. The source and destination of those IPv4 packets are the routers that cause the problem.

If you look at the adoption rate, we see that it has been very slow (20 years and counting...). Belgium has an adoption rate of 57%.



(a) IPv4 datagram format.



(b) IPv6 datagram format.

FIGURE 39 – IPCM (Internet Protocol Control Messages) format.

	IPv4	IPV6
Address Configuration	Supports Manual and DHCP configuration.	Supports Auto-configuration and renumbering
End-to-end connection integrity	Unachievable	Achievable
Address Space	It can generate $4,29.10^9$ addresses.	It can produce quite a large number of addresses (i.e. $3,4.10^{38}$).
Security features	Security is dependent on application	IPSEC is inbuilt in the IPv6 protocol
Address length	32 bits (4 bytes)	128 bits (16 bytes)
Address Representation	In decimal	In hexadecimal
Fragmentation performed by	Sender and forwarding routers	Only by the sender
Packet flow identification	Not available	Available and uses flow label field in the header
Checksum Field	Available	Not available
Message Transmission Scheme	Broadcasting	Multicasting and Anycasting
Encryption and Authentication	Not Provided	Provided

TABLE 2 – Comparison between IPv4 and IPv6.

10 Routing metrics ; Link-State routing (and OSPF)

10.1 Intro

First, little reminder of network vs control plane to explain what we're about to do (see question 8).

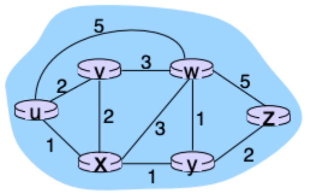
An important protocol used is ICMP (= **I**nternet **C**ontrol **M**essage **P**rotocol). This is implemented just above the IP protocol, and is used for various messages. For instance ICMP can help to discover the tracerout. You send series of UDP segments with differents TTLs. When the n^{th} set of datagrams arrives to the n^{th} router, the router will discard the datagrams and will send the source ICMP messages. Those messages will include the name of the router and IP address. The source can then learn how the network is.

10.2 Graph abstraction and routing metrics

We know that the routing protocol goal is to determine "good" paths (equivalently, routes), from sending hosts to receiving host, through a network of routers. But what is the best mathematical abstraction of this problem. Well quite logically, it is the graph abstraction.

10.2.1 Graph Abstraction

We can see our network as a graph where the nodes are the routers, and the edges are the physical links. Graphically, we have something such as represented at the figure 40.



Graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

FIGURE 40 – Graph Abstraction

With this graph, we can define the costs. The cost c(x,x') if the cost of the link (x,x') (e.g. on our example, c(w,z) = 5). The cost of path $(x_1, x_2, ..., x_p) = c(x_1, x2) + ... + c(x_{p-1}, x_p)$. This tells us something. In this abstraction, we consider that the nodes have no cost. So this makes it simpler, you can always push the cost of the node to the cost of the link. Now we can introduce the question of the link cost path.

10.2.2 Routing metrics

There are many possible ways to define the routing metrics, i.e. to set link costs. These costs can be influences by several parameters, static or dynamics. For examples, the size of a Ethernet cable or the current traffic in this same link. There are various approaches in order to minimize the cost of a travel from an host A to an other host B.

1. A first naive approach is to minimize the number of hops. To do this you can just set all link costs to 1. Doing that will also minimize the (average) link load (and node processing). But this will not necessarily minimize the delay (more hops but faster links) nor congestion (a link could be used too much). This solutions is not optimal because it doesn't take in account the link capacity.
2. We can also decide to change the link costs to optimize the network to some extent. In fact, changing the link costs will surely change the least-cost paths. The cost can be seen as a know that you turn left or right. Increasing the cost (turning the knob right) decreases traffic on this link while reducing it does the opposite. To optimize the costs, you need the TM (traffic matrix). But we want to do it without it.

Let's analyse the first approach : Minimum hop routing to minimise the average link load (the load is equivalent to number of packets/sec, bits/sec that you see on the link, the amount of traffic). We can define our score (that we want to minimize) as

$$Score = Avg\_Link\_Load = \frac{\sum_{i \in links} load_i}{N}$$

In this case, minimizing the average link load is equivalent to minimizing the sum of all the link loads. So we can remove the denominator N from he score. Let's suppose we receive a new flow. This flow will send a certain throughput in our network. This can be represented by its rate R. The flow will follow a path P. With this, we can compute the score increase induced by the addition of this path :

$$Score\_Increase = \sum_{i \in P} R = R * nb\_hops(P).$$

Therefore, minimizing the average link load is equivalent to setting the path P so as to minimize the number of hops. So to achieve this, each link will simply get a static cost = 1. This does not depend on the traffic matrix.

We can now study a second case. Using InvCap (inverse capacity of links) routing to minimize the average link utilisation (the utilisation of a link in the load on the link divided by the capacity of it) once again for any TM. We can compute our score as :

$$Score = Avg\_Link\_Util = \frac{\sum_{i \in links} util_i}{N} = \frac{\sum_{i \in links} \frac{load_i}{capacity_i}}{N}$$

We can once again take the N away. If we again add a new flow, we will this time compute the score increase as :

$$Score\_Increase = \sum_{i \in P} \frac{R}{C_i} = R * \sum_{i \in P} \frac{1}{C_i}$$

Therefore, minimizing the average link utilisation will be equivalent to setting the path P so as to miniminize $\sum_{i \in P} \frac{1}{C_i}$. To achieve this, each link will therefore get the static cost $1/C_i$. This is what InvCap routing means. The link cost is proportional to the inverse of its capacity. And once again, this does not depend on the traffic matrix. This technique means that the cost decreases when the capacity increases, so that looks already way better that minimizing path hops. Of course, these techniques don't mean we have fixed every problem. There are other possible routing metrics.

1. We could minimize the delay.
    (a) But not easy to capture,
    (b) The delay has several components
        i. propagation delay
        ii. transmission delay (depends on the packet size so already not great)
        iii. queuing delay (variable so complicated af)
    (c) Has been attempted but too difficult
2. Basically any summable quantity, where summable = cost of a path is the sum of the costs of all the links composing this path.

10.2.3 Optimality principle

This principle tells us that if a router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route. There is a consequence to this. The set of optimal routes from all sources to a destination form tree rooted at the destination. Similarly, the set of optimal routes from one source to all destinations form a tree rooted at the source.

10.3 Routing algorithms

10.3.1 Classification

The routing algorithms can easily be classified by looking at some of their characteristics. First we notice that a router can have global or decentralized information.

1. Global :
   (a) All routers have a complete topology of the network and link the cost information. You need a description of the graph.
   (b) These algorithms are called the "link state" algorithms.
2. Decentralized :
   (a) The router knows his physically-connected neighbors and its link cost to them. It will never know the full desciption of the graph.
   (b) There is an iterative process of computation (based on dynamic programming) done by exchanging info with the neighbors.
   (c) These algorithms are called the "distance vector" algorithms.

We can also classify them by looking if there are static (route changes slowly over time) or dynamic (routes change more quickly, there is a periodic update done in response to link cost changes). Today, we have a dynamic approach. In the past, we used a static. Static is dangerous if there's a failure, so backup paths had to be computed too. Dynamic will adapt to the real-time situation.

In this section we only study link state routing, see next section for distance vector routing (RIP).

10.4 Link state Routing

10.4.1 General Principles

The first step of these algorithms is what we call 'link state broadcast'. It is used to discover the graph topology. The 2nd step is the least-cost path computation. Every node will compute the least-cost paths to all other nodes. This part uses Dijkstra's algorithm. We then have the forwarding tables for that router.

10.4.2 Link State Packets

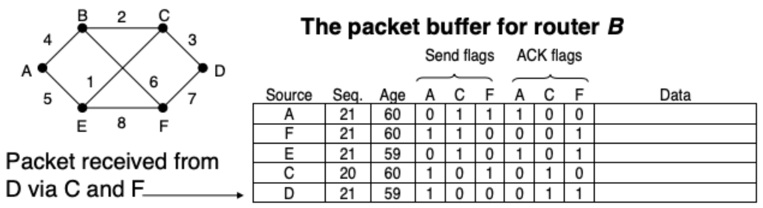
In the first step, we flood the network with link state packets. Each node will create these packets. These packets are composed of :

1. The source node, a sequence number and an age.
2. A distance vector limited to the neighbours. Composed of neighbour's name + distance vector

Those packets are then flooded selectively. Flooding means that a packet that originated at a given node has to reach all the others. But in a network, this won't be done in one step. In fact some packets have to be forwarded. This doesn't seem like a problem but you might use a lot of ressources for nothing. When there are flooded, their are stored in a packet buffer, when we remember the source, the sequence, the age, the send flags (who do you have to propagate the info to), ACK flags (to the ones who sent you the info) and the data from the packet. We also notice that the packets are not forwarded immediately. This is done on purpose. This is done to avoid sending duplicates. We can see this example with the packet D, who is send only to A and not to F on figure 41. The sequence numbers are used to remeber the "freshness" of the information. You'll always take the info with the highest sequence number.

However, there are potential problems with this method. Here is a list of potential issues :

1. What if the sequence number wraps around ? Not really a problem, 32 bits are used, so 132 years would be needed before it happens if a packet was sent every second (in practice, every 10 seconds (for example)).
2. What would happen if a router crashes. It restarts with sequence number 0 until the sequence number would reach the previous value. This is why the age field is decremented by 1 every second and the entry is removed when the age hits 0.
3. What if a sequence number is corrupted ? Same consequence, same solution.


B
2
C
4
3
A
D
1
6
5
7
E
8
F
Packet received from D via C and F
The packet buffer for router B

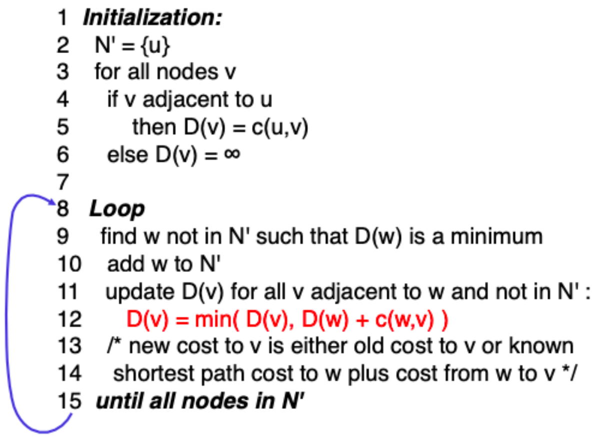
			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

FIGURE 41 – Distributing link state packets (flooding)

10.4.3 Link-State Routing algorithm

One of the most used link-state routing algorithms, and the one we'll study in this case is Dijkstra's algortihm. We have the following notes about the algorithm, which is showcased in figure 42 :

1. Needs the net topology.
2. Computes least-cost paths from one node to all other nodes. It gives the forwarding table for this node.
3. Iterative algorithm : after k iterations, we have the least-cost path to k destinations.
4. Notations :
   (a) $c(x, y)$ = link cost from node x to y, = ∞ if not direct neighbors.
   (b) $D(v)$ = current value of cost of path from source to destination v ; ∞ if not reachable.
   (c) $p(v)$ Predecessor node along from source to v.
   (d) N' : set of nodes whose least-cost path are definitively known.



```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
     D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

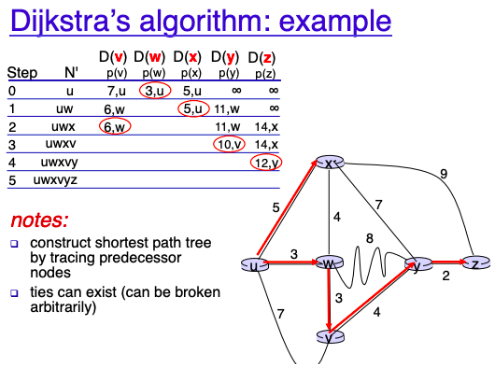
FIGURE 42 – Dijkstra's algorithm

On figure 43 is illustrated an application of the algorithm.

We can now analyze the algorithm and its complexity. We know that there are n nodes. At first look, we could think that the complexity is therefore quadratic. However more efficient implementations exist, and they give us a complexity of $n \log n$. This is much better than n^2, but still not really scalable since this is sub-linear. There is another possible problems with this approach. Indeed, there might be oscillations, when the link costs are traffic dependant. Initially, we don't know anything about traffic so costs are 0. Then given the cost, we find a new routing. With this routing, the traffic is redirected, we will then compute new costs... And so on...

10.5 OSPF

We will now study an example of an existing protocol using link-state routing, the OSPF = Open Shortest Path First. The standard open means that this is publicly available. This protocol uses Link State algorithms.



Dijkstra's algorithm: example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:
- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

FIGURE 43 – Dijkstra's algorithm application

It floods the network with OSPF messages called LSAs (link-state advertisements) that are carried directly over IP. So it uses the data plane. There are a few advanced features for OSPF (in fact, real protocols have to tackle other problems...).

1. A first issue is that a graph is not just composed of point-to-point links. We might have subnets between routers and more... We don't want this much traffic in the subnets. Therefore, a designated router will be selected in the subnets so he is the only one to exchange link-state packets with other routers on the same subnet.
2. We can also add security, by making sure LSAs are authenticated. A device can't just advertise "hello I am a router" to attract info to steal.
3. Multiple same-cost paths are allowed.
4. For each link, multiple cost metrics can be defined for different types of service.
5. When the domain becomes big, Dijkstra become heavy (and the flooding also) so there is a certain structure.

We also note that OSPF is structured following a certain hierarchy. This is represented on figure 44. The domain will consist in a backbone and some areas. Some nodes are part of the backbone (at the edge) and an area. They connect the area and the backbone, they're called area border routers. There is also internal routers (in areas), backbone routers (in the backbone) and a boundary router. On this, the area do not directly communicate with each other. They have to go back to the backbone. This allows us to scale. With this structure, two "instances" of the routing protocol are run. One for the area, and one for the backbone. There will be therefore two distinct forwarding tables. The area border routers will appear as a "proxy".

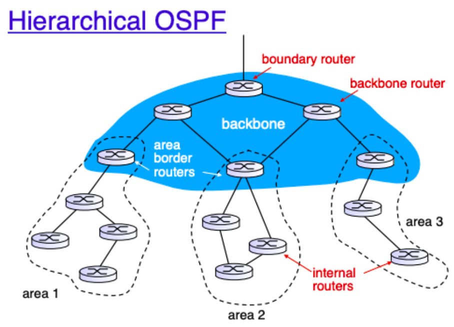


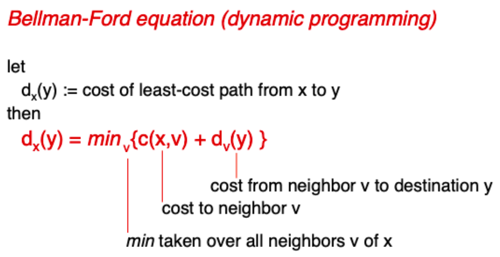
FIGURE 44 – Hierarchical OSPF

11 Distance Vector routing (and RIP)

For a short description of routing, check question 9 and 10.

11.1 Distance Vector Routing

Distance vector routing is based on a equation called the Bellman Ford equation, described on figure 45.



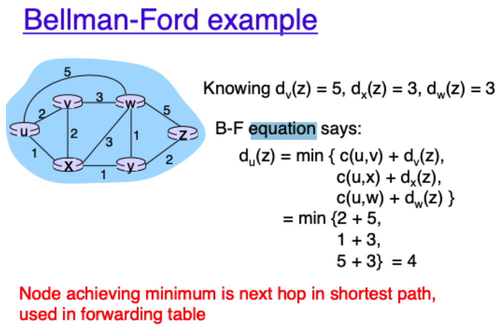
Bellman-Ford equation (dynamic programming)

let
$d_x(y)$:= cost of least-cost path from x to y
then
$d_x(y) = min_v\{c(x,v) + d_v(y)\}$

cost from neighbor v to destination y
cost to neighbor v
min taken over all neighbors v of x

FIGURE 45 – The Bellman-Ford equation

On figure 46 is illustrated an application of the equation.



Bellman-Ford example

Knowing $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$\begin{aligned} d_u(z) &= \min\{ c(u,v) + d_v(z), \\ &\qquad c(u,x) + d_x(z), \\ &\qquad c(u,w) + d_w(z) \} \\ &= \min\{2 + 5, \\ &\qquad 1 + 3, \\ &\qquad 5 + 3\} = 4 \end{aligned}$$

Node achieving minimum is next hop in shortest path, used in forwarding table

FIGURE 46 – Bellman-Ford equation application

The idea is to create a system that iterates and converges to the right solution. This is where we implement the distance vector algorithm. We now have new notations :

1. $D_x(y)$ = estimate of least cost from x to u.
2. The node x :
   (a) Maintains its distance vector $D_x : [D_x(y) : y \in N]$;
   (b) Knows cost to each neighbor $v : c(x, v)$;
   (c) Also maintains its neighbor's distance vectors for each neighbor v, i.e. $D_v : [D_v(y) : y \in N]$;

The key idea behind this algorithm is the following. From time-to-time, each node asynchronously sends its own vector estimate to neighbors. When a node x will receive these estimated DV from his neighbors, he will update its own DV using B-F equations and therefore erasing its previous vector. We thus have :

$$D_x(y) \leftarrow \min_v\{c(x, v) + D_v(y)\} \text{ for each node } y \in N$$

Under minor and natural conditions (in the graph, we should not have a cycle whose cost is negative but in networks, costs are always positive so no problem), the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$.

We can therefore give a few characteristics to this algortihm. It is an iterative and asynchronous algorithm. Each local iteration is caused by a local link cost change, or a DV update message from neighbor. This algorithm is also distributed. In fact, each node notifies its neighbors, and that only when its DV change. We can then say that the algorithm works as described on figure 47.

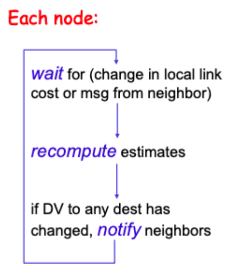

Each node:
wait for (change in local link cost or msg from neighbor)
recompute estimates
if DV to any dest has changed, notify neighbors

FIGURE 47 – Distance Vector Algorithm

An example of the algorithm at work is shown on figure 48. But what happens when we change the link cost ? We assume that the algortihm has converged. The nodes detect a local link cost change. Their will update their routing info by recalculating their distance vector. Since their DV changed, they will notify their neighbor. This works fine if the link cost is diminished. The good news will travel fast, the system will reconverge very quickly. But what happens if it is increased ? In fact, bad news travel slow. We can take the example in figure 49. Here, the distance to A is the number of hops, and we can clearly observe the problem. The nodes don't know the real topology of the network. So for example when C tells us that he has a path to A, B trusts him, and updates his table with the values given by C. He then sends his updated vector to the others. And it's a chain... It will therefore take a long time to stabilize.

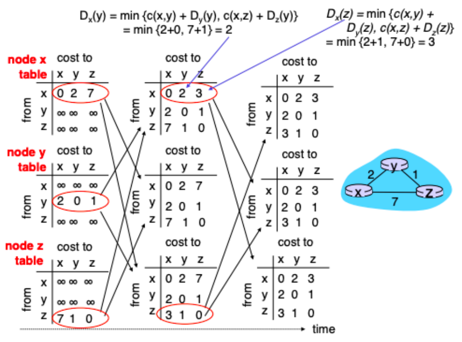
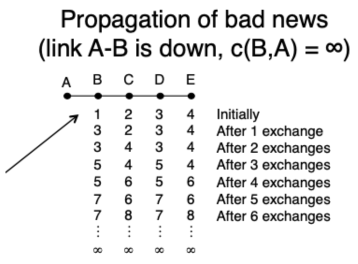

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0, 7+1\} = 2$
$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1, 7+0\} = 3$
node x table
node y table
node z table
cost to
from
time

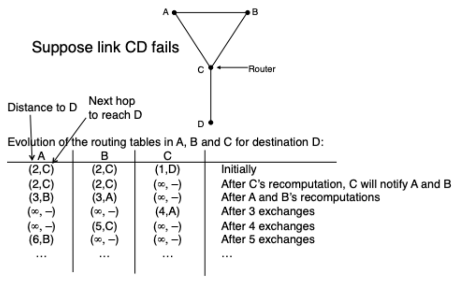
FIGURE 48 – Distance Vector Algorithm application



B	C	D	E	
1	2	3	4	Initially
3	2	3	4	After 1 exchange
3	4	3	4	After 2 exchanges
5	4	5	4	After 3 exchanges
5	6	5	6	After 4 exchanges
7	6	7	6	After 5 exchanges
7	8	7	8	After 6 exchanges
⋮	⋮	⋮	⋮	
∞	∞	∞	∞	

FIGURE 49 – Bad news travel slow

A patch to this problem is called poisoned reverse (also called "split-horizon"). Behind this lies the idea that the network will sometimes "lie" to each other. It works as follows : (see example again) when a node (like C) knows that he uses another node (B) to reach a certain node (A), he will tell the node he uses (B) that his distance to the destination node (A) is infinite. If we apply this to the situation of figure 49, it will be stabilized to the infinity for each node after only 4 iterations. However, poisoned reverse is not a solution for everything. The problem is that to know who you will "lie" to, you need to know which next hop you are using. With the topology presented on figure 50, we see where the problem is. In fact, here (B or A) will not lie to (A or B) so he won't consider the infinity, he will prefer the smaller path (that doesn't really exist anymore). On the next step, it gets worse. A and B now lie to each other. But they don't lie to C anymore. More problems incoming...



Evolution of the routing tables in A, B and C for destination D:

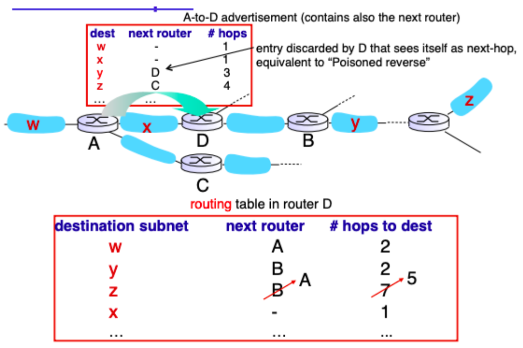
A	B	C	
(2,C)	(2,C)	(1,D)	Initially
(2,C)	(2,C)	(∞, –)	After C's recomputation, C will notify A and B
(3,B)	(3,A)	(∞, –)	After A and B's recomputations
(∞, –)	(∞, –)	(4,A)	After 3 exchanges
(∞, –)	(5,C)	(∞, –)	After 4 exchanges
(6,B)	(∞, –)	(∞, –)	After 5 exchanges
...	...	...	...

FIGURE 50 – Poisoned Reverse is not perfect

The only ultimate resort to this problem is to put a threshold. When we reach this value, we stop.

11.2 RIP

RIP (= routing information protocol) is the oldest routing protocol on the internet. It uses the number of hops as a distance metric with a maximum of 15 hops. Therefore, each link has a cost of 1 (0 means infinity). As we can see, this is made for small networks. The DVs are exchanged with the neighbors every 30 seconds in a response message (also known as an advertisement). Each of this advertisements gives a list of up to 25 (you might need to split the info in a few messages if there is too much of it) destination subnets (and not routers). But reaching a subnet is equivalent to reaching the router that is attached to the end of this subnet. We can see an example of RIP on figure 51. With this scenario, we observe that poisoned reverse is quite easily implemented, since with the "next hop" field, if a router sees himself as a next hop in a routing table coming from another router, he won't update his own table.


A-to-D advertisement (contains also the next router)
entry discarded by D that sees itself as next-hop, equivalent to "Poisoned reverse"
w
x
y
z
A
D
B
C
routing table in router D

dest	next router	# hops
w	-	1
x	-	1
y	D	3
z	C	4
...	...	

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	~~B~~ A	~~7~~ 5
x	-	1
...	...	...

FIGURE 51 – RIP example

RIP is not really used, except on smaller networks. RIP is better with rest in peace than rooting internet protocol.

11.3 Comparison of Distance Vector and Link State

In the end, we prefer to choose link-state protocols. In fact, the problem is that for distance vector, you have no upper bound for the time complexity and the size of the messages and there's also the count to infinity which is a big drawback. To conclude, we can completely oppose these two algorithms, either sending small packets everywhere, or big vectors locally.

	Distance vector routing	**Link state routing**
Algorithm	Bellman ford	Dijsktra
Network view	Topology information from the neighbour point of view	Complete information on the network topology

Best path calculation	Based on the least number of hops	Based on the cost
Updates	Full routing table	Link state updates
Updates frequency	Periodic updates	Triggered updates
CPU and memory	Low utilisation	Intensive
Simplicity	High simplicity	Requires a trained network administrator
Convergence time	Moderate	Fast
Updates	On broadcast	On multicast
Hierarchical structure	No	Yes
Intermediate Nodes	No	Yes

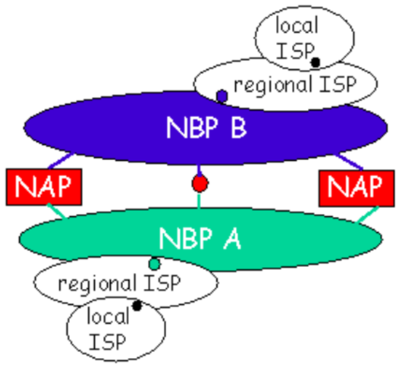
Table 3 – Comparison between distance vector and link state routing.

12 Internet structure, interdomain routing, and BGP

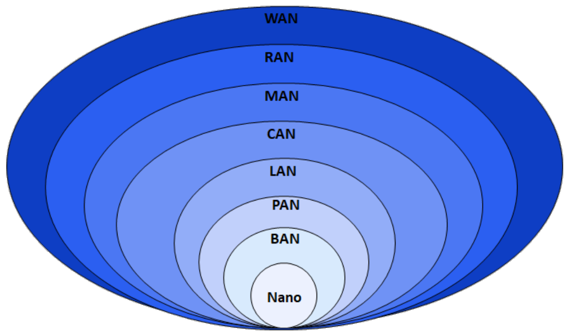
12.1 Internet Structure

The internet has a certain structure. It can be seen as a network of networks. It is created following a hierarchy between what are called the Internet Service Providers (ISPs). These ISPs are interconnected and they are divided in tiers, and they create what is called, the "heart" of the internet.

The 1st tier makes up the backbone of the internet and can be national or international. Those ISPs provide service for millions of subscribers (for example we have AT&T, Verizon, Deutsche Telekom...) and are connected via NAP (network access points). Then we have a bunch of smaller ISPs like national, regional,... And in the end, we have the local ISPs. Those local ISPs are "dead-ends", connected directly to end-systems, but there won't be even smaller ISPs. Figure 52a illustrates this structure.



(a) Internet's structure



(b) Data network classification.

FIGURE 52 – Structure of the current Internet.

However, those ISPs need to be connected, so they are. There are commercial agreements between the different ISPs. Generally, peer ISPs (ISPs from the same tier) are connected "for free". A tier 2 ISP can be client from a tier 1 ISP, which then allows to be connected to the rest of the internet. Of course, the routing of packets takes in account the costs of these links. The shortest link isn't necessarily the most used one.

12.2 Inter-domain Routing

During the course, we saw two intra-domain routing algorithms technique. Link state and distance vector routing. The problem with these techniques is that is not really viable in practice. In fact, we studied everything following a certain idealization of the network, saying that all routers were identical, and the network was "flat". Of course, this isn't true in practice. Plus, there is the scaling problem. There billions of destinations, we can't store all of them in routing tables, plus all this exchange to fill those tables would just swamp the links. We also want an administrative autonomy. Indeed, each network administrator may want to control routing in its own network. To solve this scaling issue, the routers are aggregated into regions knows Autonomous Systems (AS, aka "domains").

In those domains, we have what is called intra-AS routing. This is where we use the protocols that are described in section 10 and 11 (i.e. OSPF and RIP). We note that all routers in AS must run the same protocol, while routers in different ASes can run different routing protocols. At the edge of these ASes are placed gateway routers, that have link(s) to router(s) in other ASes. The routing performed between these routers is called inter-AS routing. This is the routing done among the ASes. We also notice that the gateway routers therefore must apply inter AND intra domain routing. The forwarding tables are configured by both intra and inter-AS routing algorithm. We know that the intra-AS sets entries for internal destinations. The external destinations entries are set by both the inter-AS and the intra-AS (explained later) routing algorithms.

So what's the job of inter-AS routing ? Well, a router has to learn, for a certain destination, how he can reach it, if it is located outside his domain, and then propagate this reachability info to all routers inside of his domain. We can take the example showcased on figure 53.

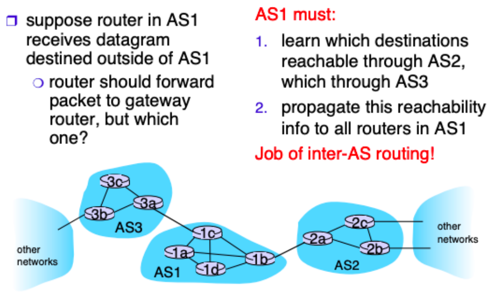

suppose router in AS1 receives datagram destined outside of AS1
router should forward packet to gateway router, but which one?
AS1 must:
1. learn which destinations reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1
Job of inter-AS routing!
3c
3a
3b
AS3
1c
1a
1d
1b
AS1
2a
2c
2b
AS2
other networks
other networks

FIGURE 53 – Inter-AS tasks.

12.3 BGP

BGP (border gateway (=router) protocol) is THE interdomain routing protocol. It can be seen as the "glue that holds the internet together". BGP provides a certain number of basic functionalities, that can be divided in two parts :

1. eBGP (= external BGP) : obtain subnet reachability information from outside (neighboring ASes).
2. iBGP (= internal BGP) : propagate this reachability to all AS-internal routers.

This knowledge is then exploited for the final role of BGP, which is to determine the "good" routes to the other networks. This is where BGP will adopt different strategies. In fact, this part is based on reachability information and policies (political and economical viewpoints). We can see how eBGP and iBGP are connected on figure 54 BGP is also used to advertise your presence to the rest of the internet as a new ISP.

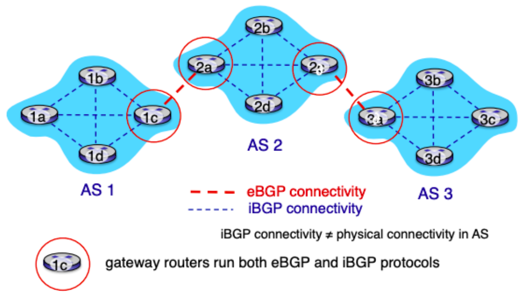

1b
1a
1c
1d
AS 1
2b
2a
2c
2d
AS 2
3b
3a
3c
3d
AS 3
eBGP connectivity
iBGP connectivity
iBGP connectivity ≠ physical connectivity in AS
1c
gateway routers run both eBGP and iBGP protocols

FIGURE 54 – BGP Connections

12.3.1 BGP basics

Let's take the example of figure 54. What happens when two BGP routers (peers) exchange information. Of course, they will exchange information about their own domain, i.e. their own "block" of addresses, so the subnets withint its perimeter can be reached. Of course, he will also share the information he has learned from other domains. He will therefore advertise the paths to other "Blocks". A path will be composed of hops between ASes. Those advertisements can be seen as "promises" that he will forward the data to the right place.

12.3.2 BGP paths

A route is described by the prefix of the address you want to reach, and a certain number of attributes. There are 2 important attributes :

1. AS-PATH : it's a list of ASes (described by a number) through which this prefix advertisement has passed. Therefore, we can say of BGP that it is a path vector protocol (there might be a lot of paths), since the AS-PATH are propogated, whereas in a DV protocol, only the distance is propagated.
2. NEXT-HOP : this is the IP address of the (internal-AS) gateway router leading to the first AS in AS-PATH. (exit point of your own domain)

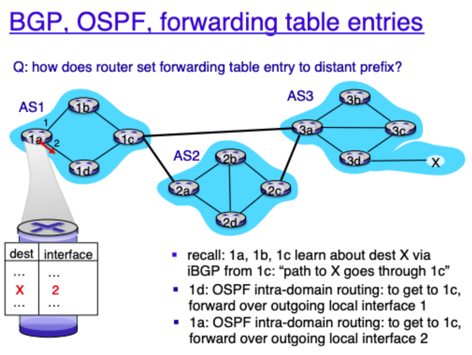
As we said earlier, BPG is policy-based. In fact, the router receiving the route advertisement will use an import policy to accept/decline path. In fact, you might decide not to route through certain ASes (political reasons for example). Then the gateway also has to decide if he advertises the path to other neighboring ASes. He might not do so for economical reasons for example. We also note that gateway routers may learn about multiple paths to destination. With his policy (and its criterias), he will choose 1 path (see later).

12.3.3 BGP messages

To achieve all these exchanges, BGP has a multiple number of different messages. We notice that the communication is done using a semi-permanent TCP connection. This seems strange for a network layer, since TCP is implemented in the transport layer, it's seems like a illogical loop. Well this is not the case, since we're in the control plane so the routing protocols may be running as applications. They then use sockets to communicate with other applications. We use TCP because there is a big amount of information to transmit. Therefore, it's much better to have a permanent connection to exchange this. We also note BGP is incremental. Indeed, instead of repeating the entire path, it will just update by giving new info or withdrawing info. It can also add paths of course. We note that the different messages are OPEN, UPDATE, KEEPALIVE, NOTIFICATION, ROUTE-REFRESH.

12.3.4 BGP forwarding tables

With this, we can now get the form of the forwarding tables. We can look at the example of figure 55. In the end, we see that what's kept in this forwarding table is the interface taken.


BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

dest	interface
...	...
X	2
...	...

Figure 55 – BGP forwarding table

An important point is the route selection. We know that a router may learn that there are more than one route to some destination prefix. It will select route based on the following criterias :

1. Local preference value attribute (1st solution to win money (your own ASes), 2nd solution not to lose money(peer), not to lose too much money (least expensive))
2. Shortest AS-PATH.
3. Closest gateway router (hot potato routing). The idea behind this is to choose the local gateway that has the least intra-domain cost.
4. Additional criteria.

12.3.5 Avoiding loops

One important thing is that we have to avoid loops. To do this, the knowledge of the path is used. In fact, if an AS sees itself in the AS-PATH advertised by its neighbor AS, it discards it, or it would create a loop. This is made possible by the presence of AS-PATH in the advertisements, and it is more powerful than poisoned reversed that is used in Distance Vectors. We can check the example on figure 56

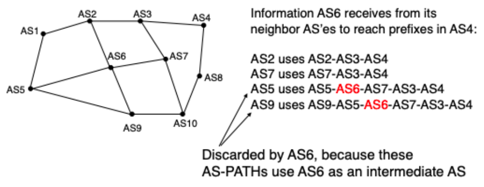

AS1
AS2
AS3
AS4
AS5
AS6
AS7
AS8
AS9
AS10
Information AS6 receives from its neighbor AS'es to reach prefixes in AS4:
AS2 uses AS2-AS3-AS4
AS7 uses AS7-AS3-AS4
AS5 uses AS5-AS6-AS7-AS3-AS4
AS9 uses AS9-AS5-AS6-AS7-AS3-AS4
Discarded by AS6, because these AS-PATHs use AS6 as an intermediate AS

FIGURE 56 – Avoiding Loops

12.3.6 BGP Policies

Let's take the example on figure 57. Here we see that x is dual-homed. He therefore wants to be careful to make sure that B doesn't give him traffic for C via x. To solve this, he will not advertise to B a route to C. We can have a second case. Indeed, here we have that A advertises path AW to B and to C. B advertises path BAW to X. So B will not advertise BAW to C since B will get no revenue for routing $CBAW$ since none of W, C, A are customers.

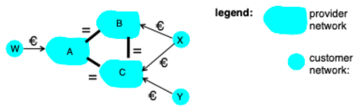


FIGURE 57 – BGP Routing

On figure 58 we have a short summary of what was just explained. We see that you wont advertise a provider to another provider or to your peers, but you have to forwarded it to customers.

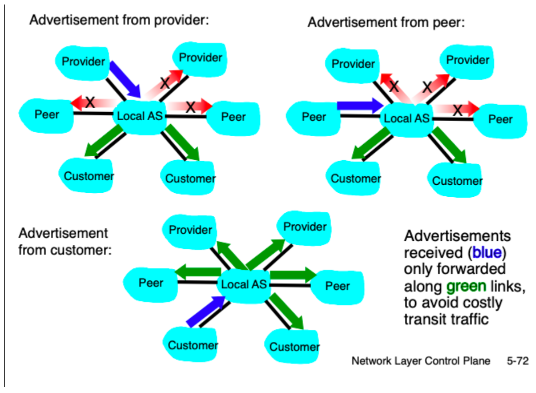

Advertisement from provider:
Advertisement from peer:
Advertisement from customer:
Provider
Peer
Local AS
Customer
X
Advertisements received (blue) only forwarded along green links, to avoid costly transit traffic
Network Layer Control Plane 5-72

FIGURE 58 – BGP advertisements

12.3.7 Why different Intra- and Inter-AS routing

We see that there are big difference between Inter and Intra-AS routing. We see that for inter-AS routing, policies have a big importance while they don't matter for Intra-AS routing. We see that hierarchical routing helps for both Intra and Inter-AS routing, saving table size and reducing the update traffic. Finally, we see that performance-wise ; Intra-AS can focus on this aspect while Inter-AS will usually put policy as a more important factor.

13 Multiple access protocols in LANs : Aloha, CSMA, CSMA/CD, Ethernet

13.1 Multiple access protocols

There are two types of "links"

1. Point-to-point that does not need multiple access protocol as there are only two system that are able to communicate on it.
2. Broadcast, shared wire or medium that need multiple access protocols.

On shared broadcast channel, if there are two or more simultaneous transmission by nodes, interference and/or collision will appears. That is why those channels need a multiple access protocol who will determine how nodes share channel, it can be called Medium Access Control (MAC) protocols. Note that communication about sharing need to use the channel too.

The ideal MAC protocol given a broadcast channel of rate R bps need to respect several conditions.

1. When only one node wants to transmit, it can send at rate R.
2. When M nodes want to transmit, each can send at average rate R/M (fairness).
3. It is fully decentralized, there's no need of a synchronization clock or a special node to coordinate transmissions.
4. It need to be simplest as possible.

As achieving all of those is not easy, several protocols exists but they can be divided into three classes

1. Channel partitioning : divide the channel into smaller "pieces" (time slots, frequencies,...) and allocate those pieces for exclusive use. Time slot partitioning is used in telephony while frequency partitioning is used for television.
2. Random Access : channel is not divided and allow collisions but we have to "recover" from them.
3. "Taking Turns" : nodes take turns to send what they have to but those with more to send can take longer turns.

13.2 Random access protocols

The idea behind those protocols is that when a node has packet to send, it transmit it at full channel data rate R and there are no a priori coordination among nodes. It means that if there are two or more transmitting nodes, data will collide and be lost. Those protocols handle the detection of collisions and how to recover from them.

13.2.1 Slotted Aloha

Let's make first some assumptions in order to define this random access protocol first designed in the network between Hawaii islands. We assume that

- All frames have the same size ;
- Time is divided into equal size slots (time to transmit 1 frame) ;
- Nodes start to transmit only at slot beginning ;
- All nodes are synchronized ;
- If 2 or more nodes transmit in slot, all nodes detect collision ;

The algorithm of the slotted Aloha protocol is given in algorithm 1.

```
when obtains fresh frame
if(no collision)
    node can send new frame in next slot
else
    node retransmits frame in each subsequent slot with probability p until success
```

Algorithm 1 – Aloha transmission decision algorithm

This protocol has several advantages, first a single active node can continuously transmit at full rate of channel. Also, it's highly decentralized : only slots in nodes need to be in synchronized. This is also a very simple protocol to make.

Because there is always defaults when there is advantages, we can list several problems relative to this protocol, namely the fact that there are collisions that leads to wasted slots while nodes may be able to detect collision in less time than the time needed to transmit. Some slots are idle and we need a clock synchronization.

This protocol efficiency (long-run fraction of successful slots when there are many sending nodes N) has the probability of is 37%. Assume that we have N nodes, the probability that given slot is successful slot is the probability that the given node transmit and that all the $N-1$ nodes do not. The probability that a given node transmits is p (instead of 1 in a real slotted Aloha protocol, but here we assume that the probability is p in order to simplify our formulas). Therefore the probability that the remaining nodes do not transmits is given by $(1-p)^{N-1}$. Thus, the probability that a given node has success is $p.(p-1)^{N-1}$ and because there are N node, the probability of found any success node is given by $N.p(p-1)^{N-1}$ (probabilities are independent so we can sum all of them, meaning sum the N transmissions probabilities). In order to found the maximum efficiency of slotted Aloha for N active nodes, we have to find $p*$ that maximize the above expression. By simple derivation and equalization to 0, we found

$$p* = \frac{1}{N}$$

and if we take the limit

$$\lim_{N\to\infty} N.p*.(1-p*)^{N-1} = \lim_{N\to\infty} N.\frac{1}{N}.\left(1-\frac{1}{N}\right)^{N-1} = \frac{1}{e}$$

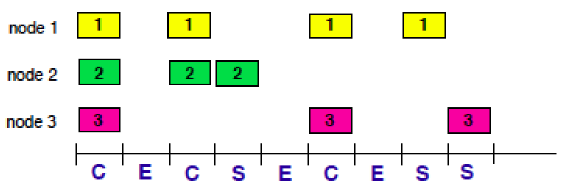
since

$$\lim_{N\to\infty}\left(1-\frac{G}{N}\right)^{N-1} = e^{-G}$$

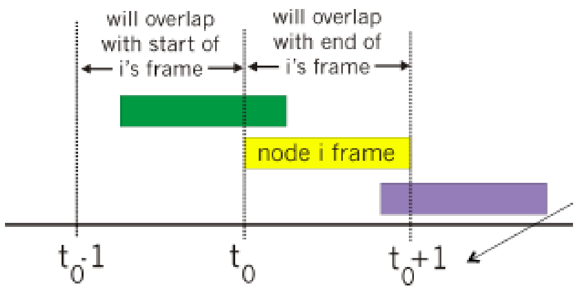
And we found and efficiency of $\frac{1}{e} \approx 0,37$.

13.3 Pure Aloha

In order to simplify the protocol, some tried to remove time slots and allow nodes to transmit whenever they wanted to. It was the birth of the so called Pure ALOHA. However, collision probability increased as frame sent at t_0 collides with other frames sent in $[t_0-1; t_0+1]$ which is twice bigger than the risky interval in slotted ALOHA. Therefore, efficiency gets worse and becomes only around 18%. Note that The method to evaluate efficiency only consider high load. To consider any load, we can speak about efficiency following the average aggregated traffic load per frame time usually, called $G = pN$. Then, for slotted ALOHA, efficiency is Ge^{-G} while pure ALOHA has an efficiency of Ge^{-2G}. It means that those protocols are fine for low load but extremely bad for high load.



(a) Slotted Aloha protocol.


will overlap with start of i's frame
will overlap with end of i's frame
node i frame
t_0-1
t_0
t_0+1

(b) Pure (unslotted) Aloha protocol.

Figure 59 – Aloha random access protocol.

13.4 CSMA

An improvement of this protocol is the **C**arrier **S**ense **M**ultiple **A**ccess protocol. It uses Pure ALOHA but adds a principle of carrier sensing. It means that before sending, node will check whether the link is free or not. Collisions become less frequent but still occur. For example due to propagation delay, two nodes may not hear other's transmission before sending. The algorithm 2 saw how generally a CSMA protocol works.

```
while(true) do
    if(channel is free) then
        with probability p: immediate transmission;
        or
        with probability 1-p: stay idle during at least propagation time (T)
    else
        listen until the channel is freed
```

Algorithm 2 – P-persistent CSMA decision algorithm

When a node sees that channel is busy, it has to wait for it to be free. However, we can imagine multiple possibilities on the moment it will resend the stuck packet. These differences leads to different versions of CSMA.

1. Persistent CSMA : if channel is busy, we listen until it is freed and send directly.
2. Non-persistent CSMA : if channel is busy, we program a new attempt later.
3. P-persistent CSMA : if channel is free, we immediately transmit with a probability p or we stay idle during at least the time needed for propagation with probability $1 - p$. This version introduce useless delays at low loads but increase efficiency at high load. It is a trade-off between efficiency and delay.

Channel efficiency can depend a lot of the value of p. However, this efficiency does not take into account delay which may be more important with a lower p. An improvement of this version is to use a p that depends on the traffic load. Engineering such a system is not so easy. Let's do some maths as an example

- B : channel data rate (bps) ;
- F : frame size (bits) ;
- L : length of the channel (m) ;
- c : propagation speed (m/s) ;
- τ : propagation delay $= \frac{L}{c}$ [s] ;
- T : transmission delay $= \frac{F}{B}$ [s] ;

τ is then the risky period while collision can happen, we may call it contention period. So we are able to define the risk of collision

$$a = \frac{\tau}{T}$$

For $a = 1\%$, $c = 200000 km/s$ we have $F = \pm 5.10^{-7} BL$.

For $B = 10$ Mbps and $L = 2,5$ km, we get a frame size $F = 1562,5$ bytes. This is roughly the Ethernet frame size.

13.5 CSMA/CD

This is a version of CSMA that add a Collision Detection (CD) service. Indeed, with classical CSMA, when there is a collision, the whole time frame is lost, we can just stop the transmission to recover faster. This can leads us to decrease the time lost to less that 2τ. This version of CSMA lead us to a new bottleneck. Indeed, to detect collision, the sender must still be transmitting when the collision propagates back to it.

13.6 Ethernet

The idea behind Ethernet protocol commonly used is implemented in the Network Interface Card (NIC) and follows these steps :

1. NIC receives datagram from network layer and creates a frame.

2. If NIC senses channel idle, it starts the frame transmission. If NIC senses channel busy, it waits until channel is idle, then transmits (persistent CSMA).
3. If NIC transmits entire frame without detecting another transmission, NIC is done with the frame.
4. If NIC detects another transmission while transmitting (collision) aborts and send jam signals.
5. After aborting, NIC starts a binary (exponential) backoff : after m^{th} collision for this frame, NIC chooses K at random from $\{0; 1; 2; ...; 2m - 1\}$. NIC waits $K.512$ bit times and return to step 2. For $m > 10$, it picks from $\{0; 1; 2; ...; 1023\}$.

This corresponds to a p-persistent CSMA/CD with an adaptive p. The bigger the frame size is, the better the efficiency as taking risk once for a big packet is easier than several times for small packets.

14 Link-layer addressing and switching : ARP, self-learning and spanning tree ; differences between hubs, switches and routers

14.1 MAC address

It's well know that every network component has an IP address but we have seen that it's the analogy of the postal post in real life. IP addresses depend of where you are inside the network. In order to never lost track of network material, one marked in red iron every potential technologies who will find a place in the network. These are call MAC (Media Access Control) addresses. Usually, data is broadcast on a link and devices need to detect whether they are the destination or not. To do so, MAC addresses are used. Those addresses are 48-bits long and are specific to a Network Interface Card. It means that any link interface can be identified locally, we usually represent them as 6 hexadecimal 2-digit-numbers.

14.2 ARP

As those addresses are portable we usually need to look for them when we want to transmit to these interfaces. To do so, Address Resolution Protocol (ARP) is used.

In practice, each IP node (host, router) on LAN has a table that allows mapping between IP addresses and MAC addresses. Usually entries of those table also has a TTL so that it can be forgotten if a device is disconnected for example (typically after 20 min). The information is said to be a soft state. At first, when A wants to send a datagram to B, B's MAC address is not in A's ARP table. So, A need to broadcast (destination MAC address is `FF-FF-FF-FF-FF-FF`) an ARP query containing B's IP address to find it. Every nodes receives this query and add A if it has not been discovered yet and B will reply to A with its MAC address in unicast. ARP allows node to be "plug-and-play" as they does not need to be configured to find MAC addresses.

Be careful to the fact that when we speak about nodes and addresses, those are local. So if you want to send a packet away, you first need to send it to the subnet dedicated router and the MAC address you look for is the router's one, not the one of the destination device.

14.3 Switch

At first, the internet network was just series of host connected to the same bus. It was really problematic because every host was relative to the same link so if any trouble was encountered by this link, every host was affected.

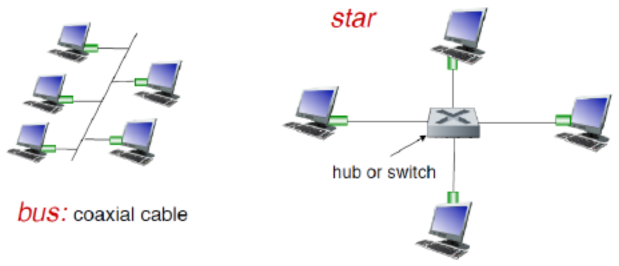


Figure 60 – Internet configuration evolution.

That's why we decided to isolate those connection by using hubs, who are just dumb devices that act as a bus as it broadcast everything it gets but allows interfaces to be disconnected. Then, We think about upgrading those hubs into switches to avoid the broadcasting and establish more privacy. The idea was to selectively forward data (on one single link) like routers. However, it is not like a router at all because it is completely transparent inside the network. It means that hosts or routers connected through a switch cannot detect it. Moreover, the switch is plug-and-play, so you does not have to configure it when you install it.

In order to send data in appropriate locations, they have a NIC at their interfaces, they can buffer data and they have a kind of forwarding table called a switch table. Even if names are similar, be aware that it's not a forwarding table at all, since it just knows which MAC address is relative to which of its interface, remember that there is not any acknowledgement of IP addresses since we are not in a Network protocol. As hosts have a dedicated, direct connection to switch, collisions can only happen when the host sends data in the same time than the switch. If the link is full duplex, collisions can even not happen ! And we know that fewer collisions means better performances.

14.4 Spanning tree

Because switches are dump device, we wouldn't need to configurate them. That's why a switch adopt the self learning policy, it's mean that it take acknowledgement from its past experiences.

Let's assume that an host A wants to send a datagram to B but there is a switch in between. A finds switch's MAC address using ARP and sends the datagram. The switch, seeing a packet coming from A adds an entry with A's MAC address and the incoming interface. Then, as it does not know where to send it, it send the datagram to all of the other interfaces. Let's assume B answer to this datagram and sends one back to A. the switch gets the packet from B and now know from which interface it comes from. It adds it to its switch table and sends the datagram to A in unicast as it already added it before.

When there are multiple switches, this technique also works but only if there are no cycle in the topology. This can be done physically but means that there is no redundancy in case of failure. That's why switches will build a logical spanning tree topology over the real topology. To do so, you have 3 main steps

1. Determine the root switch ;
2. Build tree ;
3. Decide whether non-root ports are (data) forwarding or (data) blocking.

At first, each switch considers itself as the root switch. Then, it receives BPDU (**B**ridge **P**rotocol **D**ata **U**nits) and updates its switch table only if it is "smaller" than what is currently stored so that, it finally knows who's the root and what is the root port, i.e. the port leading to the root. It is similar as a DV routing protocol with the distance vector limited to a unique component/destination : the root switch.

	Hubs	Switches	Routers
Traffic isolation	No	Yes	Yes
Plug & play	Yes	Yes	No
Optimal routing	No	No	Yes

TABLE 4 – Switch, hubs and routers comparison.

14.4.1 Example of spanning tree.

1. The first step consist in to determine the root switch. Switches regularly flood control messages (BPDUs) on all their output ports :

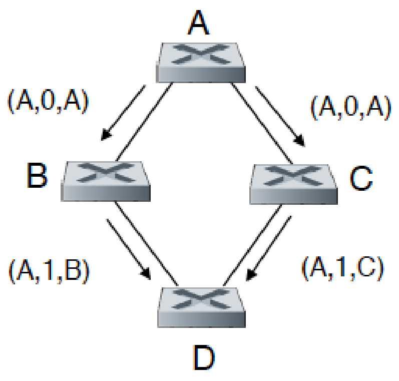
   ```
   BPDU = (root as assumed, distance to root, source switch id)
   ```

   All switches will then soon discover the root id Spanning tree, choosing the node with the smallest switch id in the received messages.
2. A, B, C, D are switch ids : they must be unique. We use the lexicographical order on switch ids to order them : $A < B < C < D$ in this example In practice, a switch id is a pair :

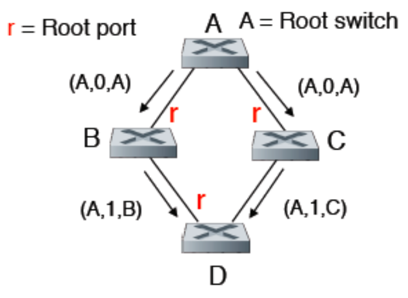
   ```
   (priority level, one MAC address of switch)
   ```

   Switch A (the root) sends BPDU `(A,0,A)` on all its interfaces. B now assumes A is the root switch (since A < B) and forwards (A,1,B) on its interfaces (with an incremented distance). Same for C. D now also assumes A is the root switch (since A < D) (see figure 61a).

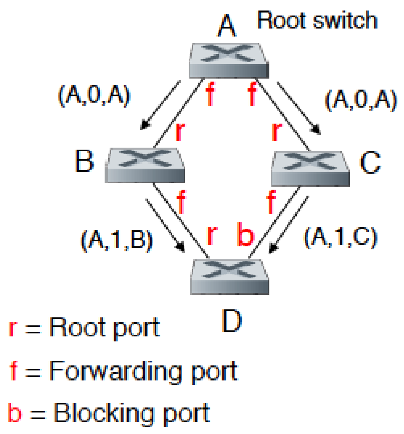
3. Building the tree by continuously receiving BPDUs (possibly on several ports), each switch determines its distance to the root and which port leads to the root by that shortest distance (= root port, r). The root switch has no root port ! In practice, distances are often inversely proportional to link capacities. In figure 61b, D determines its root port (r) by comparing (A,1,B) and (A,1,C) and taking the "smallest", according to a lexicographical ordering relation defined on BPDUs :
```
(root_X, dist_X, source_X) < (root_Y, dist_Y, source_Y)
iff (root_X < root_Y)
or ((root_X = root_Y) and (dist_X < dist_Y))
or ((root_X = root_Y) and (dist_X = dist_Y) and (source_X < source_Y))
```
(when there are multiple links between 2 switches, the port number is used as an additional tie-break)
4. Decide whether non-root ports are (data) forwarding or (data) blocking. A port is "forwarding (f)" on a given segment iff the BPDUs that this switch sends on this segment are "smaller" than those other switches (would) send otherwise the port is "blocking (b)". As show on figure 61c on segment CD, C sends (A,1,C) which is "smaller" than (A,2,D) that D could send on CD. Both C and D reach the same conclusion : C is elected to be the only one allowed to forward frames on segment CD ; C sets its port to forwarding, while D sets its port to blocking.
5. Which leads us to the resulting spanning tree on figure 61d. Some switches may not be part of the tree (when all their ports are blocking, except their root port). A blocking port may become forwarding if another switch or port would fail (leading to no refresh of dominating BPDUs). So, switches have to listen to BPDUs on blocking ports to detect failures



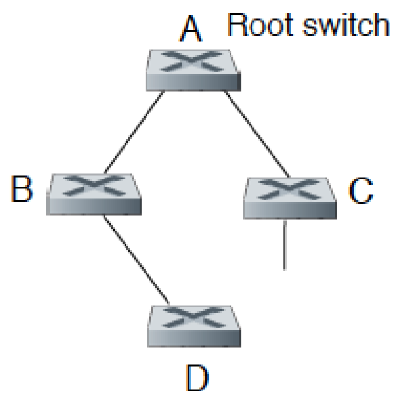
(a) Nodes finding the tree's root.



(b) Determination of the distance to the root.



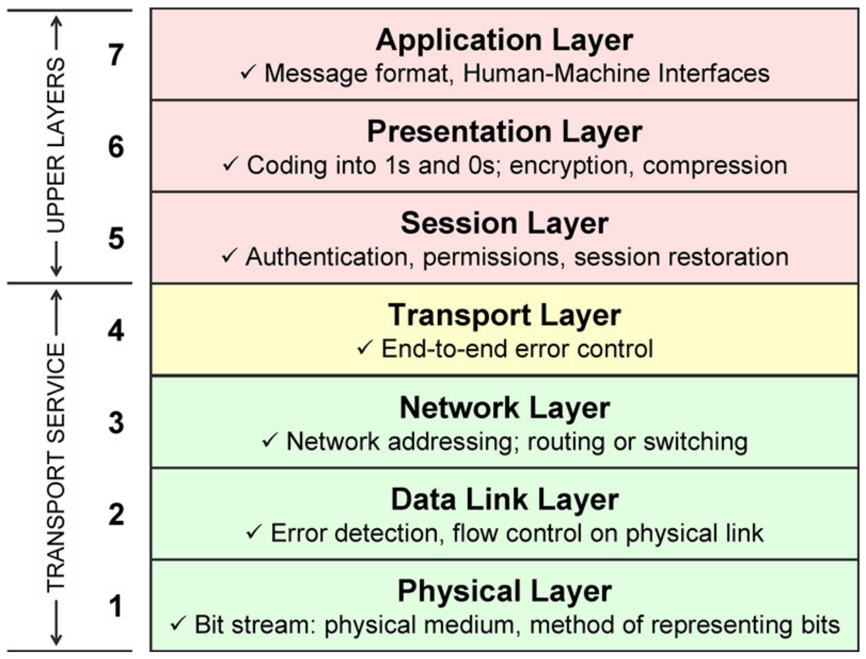
(c) Decide whether non-root ports are forwarding or blocking

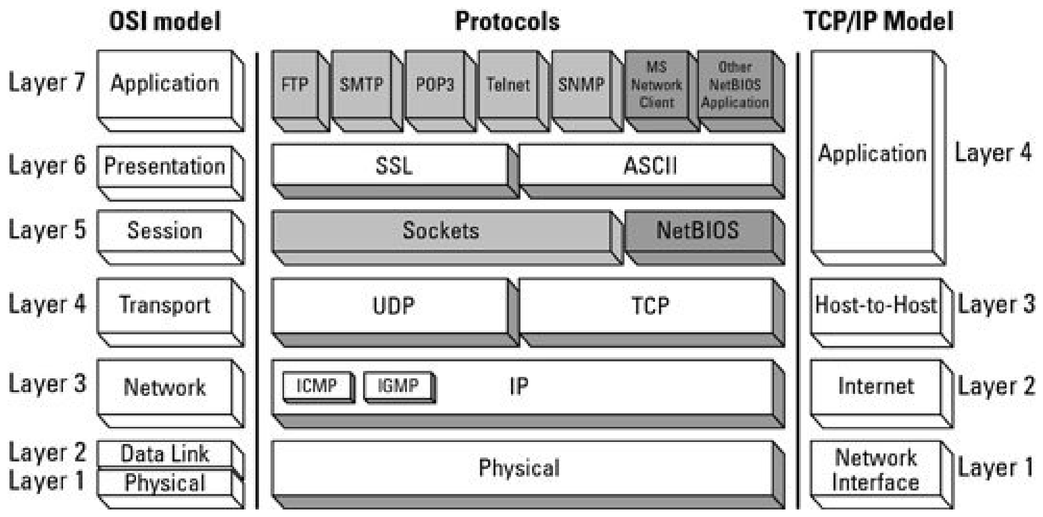


(d) Resulting spanning tree.

FIGURE 61 – Example of a spanning tree protocol execution.

A OSI layer


UPPER LAYERS
TRANSPORT SERVICE
7 Application Layer
✓ Message format, Human-Machine Interfaces
6 Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
5 Session Layer
✓ Authentication, permissions, session restoration
4 Transport Layer
✓ End-to-end error control
3 Network Layer
✓ Network addressing; routing or switching
2 Data Link Layer
✓ Error detection, flow control on physical link
1 Physical Layer
✓ Bit stream: physical medium, method of representing bits


OSI model
Protocols
TCP/IP Model
Layer 7 Application
Layer 6 Presentation
Layer 5 Session
Layer 4 Transport
Layer 3 Network
Layer 2 Data Link
Layer 1 Physical
FTP
SMTP
POP3
Telnet
SNMP
MS Network Client
Other NetBIOS Application
SSL
ASCII
Sockets
NetBIOS
UDP
TCP
ICMP
IGMP
IP
Physical
Application Layer 4
Host-to-Host Layer 3
Internet Layer 2
Network Interface Layer 1