# Summary

## Network Centric Systems (WBCS031-05)

Bjorn Pijnacker

2021–2022 | 1a

# Contents

# Chapter 1

# Introduction & History

## 1.1 Communications Basic

Before electronics were used for long-range communication, we had semaphores. These were towers with big movable wings attached to them that could transmit messages by being seen over a long distance by the next tower, which would then retransmit the message. This was used in the 18th century.

After semaphores the early telegraph was invented (morse code over electric wire). Experiments with this started in 1750 and the first long-range line was completed in 1844. The first multiplexing system was completed in 1874. In 1916, the US had a telegraph network which had to solve the problem of signal coding, the type (and level) of switching, the type of service (connection oriented vs. connectionless), and whether and where to place repeaters and/or routers. The solutions to these issues were later used to pioneer the Internet network. To solve the issue of signal coding, the Baudot telegraph code was invented. This code was the predecessor to modern ASCII.

Another communication technology of the time was telephony. It was invented to remotely listen to music. Later the goals changed to two-way communication. In 1877, A. Bell was granted the first patent for telephone technology. Originally telephones were sold pair-wise with a direct line. Each customer would have a phone for every person they would like to reach. This isn't exactly scalable, so this lead to the created of a control switching station.

> **Telegraph vs. Telephony vs. Television** Telegraphs are connectionless and without dedicated path. Telephones are connection-oriented with a dedicated path between sender and receiver. Television is not one-to-one but uses a broadcasting system, with one sender and many receivers.

## 1.2 The Internet

In 1961, Kleinrock at MIT showed packet switching can increase network performance. In 1962 an idea was proposed for a globally connected network. In 1967, ARPANET concept was published and in 1969 the RFC was established at UCLA. In the years following the initial TCP/IP was presented, to which ARPANET switched in 1983.

The goals for ARPANET included load sharing, messaging, data and program sharing, and remote service. The first two had been done on homogeneous hardware, but ARPANET allowed for doing this on a heterogeneous system as well.

In 1970 many networks beside ARPANET existed. The challenge was now to interconnect these networks, with the biggest question being how to do routing for interconnected networks. Other problems to solve included addressing and flow control. For addressing TCP was implemented first, after which the simpler UDP protocol was implemented. TCP/IP was implemented in ARPANET in 1983. It's well-known IPv4 addressing is still widely used today.

### 1.2.1 The Basics

When looking at network components we differentiate between end systems (ESs) that use or provide data and intermediate systems (ISs) that route data to different end devices. Using these, many different network topologies exist. Which topology is "best" depends on the use case, the transport medium, the distance between nodes in the network, and your definition of "best".

Since computing scientists love their abstraction, The Internet also has a lot of abstraction. We separate The Internet into layers. Each of these layers has some service to be provided, where each uses a protocol to ensure entities interoperate correctly. There is a tradeoff here: overhead vs. complexity. Layers provide services by performing action on its own layer, by using services from layers directly below (or above) it.

> **Connetionless vs. Connection-Oriented** When establishing a protocol we must make the distinction between connectionless networking and connection-oriented networking. With connection-oriented services we have three stages: (1) connect (handshake); (2) transfer data; (3) disconnect. With connectionless service we simply send the data and hope for the best.
>
> Important to note is that connection-oriented does not mean reliability. Also a handshake means more overhead which isn't always desireable. A connection-oriented service can be built on top of a connectionless service.

The abstraction used in the internet is derived from the OSI-ISO model, which has 7 fundamental layers: (1) physical layer; (2) data link layer; (3) network layer; (4) transport layer; (5) session layer; (6) presentation layer; (7) application layer. Intermediate Systems don't need all of these layers. They include a continuous subset of the layers. The above layers have the following functions:

(1) The physical layer is responsible for sending the actual bits;

(2) The data link layer is responsible for reliable data transfer between adjacent stations using frames;

(3) The network layer is responsible for the connection between two end systems. Its concerns include routing a suitable/good path and internetworking between networks;

(4) The transport layer is responsible for the communication from source to destination. We wish to optimize QoS[1] using this;

(5) The session layer can support a session over a longer period;

(6) The presentation layer negotiates the data structure used;

(7) The application layer is responsible for providing the data and what should happen with it.

In contrast to the OSI-ISO model, the TCP/IP reference model (which the Internet uses) has only 5 layers. The presentation, session, and application layer are merged to form the TCP/IP application layer, and the data link layer and physical layer are merged to form the network interface layer.

[1] Quality of Service

# Chapter 2

# Network Layer

## 2.1 Functions in the Network Layer

The goal of the network layer is to move packets from end system to end system. This is done in several hops, potentially between heterogeneous subnetworks. The network layer also has to compensate for differences between ESs during transmission.

The services that the network layer provides are standardized for end systems and independent from the used technology and the number, type, and topology of the subnetworks. This is the abstraction that the network layer provides to ESs.

The primary tasks that the network layer has to perform are virtual circuits and datagram transmissions, routing and internetworking, congestion control, addressing, and QoS. Secondary tasks that the network layer performs include multiplexing network connections, fragmentation and reassembling, error detection (and correction), flow control, and maintaining packet order.

The network layer is required to know subnetwork topology, because any IS must know which other ISs exist for routing to. This includes address/localization of the end system. It also needs to know the network status, for flow control and congestion detection, etc.

## 2.2 Types of Switching

There is four main types of switching: circuit switching, message switching, packet switching, and virtual circuit switching.

### 2.2.1 Circuit Switching

In circuit switching, a connection exists physically for the duration of the conversation. Think of: telephone switching boards. For this type of switching, a connection has to occur before transmission, and establishing such a connection takes time.

### 2.2.2 Message Switching

In message switching all data that is sent is treated as a "message". Each node in the network accepts the message, treats possible errors, stores the package, and forwards it to the next node. This happens node-by-node until the packet arrives at the ES. An example of this is the early telegram service.

A downside of message switching is high memory requirements at nodes, since the message has to be stored in full prior to forwarding, and there is no limit to the size of a message.

### 2.2.3 Packet Switching

Packet switching is what the Internet uses. Packets are of limited size, and a dynamic path is created from source to destination; there is no connection phase. A problem is packet switching is the possibility of congestion, along with which come many issues that we discuss later.

### 2.2.4 Virtual Circuit Switching

Virtual circuit switching exists on top of packet switching, except there is a connection-setup phase. State information in nodes can then be used to make it easier to ensure QoS for this specific connection.

In practice, virtual circuit switching is often used for voice transmissions over networks.

## 2.3 Services: Concepts

### 2.3.1 Connection-oriented communication

Connection-oriented communication can use state for error free communication and is usually duplex. It is more favorable for real-time communications and for telephone and telecommunication companies.

When transferring data using connection-oriented services, there a three stages to a connection: connection (building a common state), data transfer, and disconnection. The existence of a connection state also allows for relatively complex protocols to be employed.

### 2.3.2 Connectionless communication

Connectionless is by design unreliable. There is hardly any error control; this is left to the higher layers. Communication is simplex. It is more favorable for simple data communication. It is used by the IP protocol.

Compared to connection-oriented communication, connectionless communication uses relatively simple protocols.

## 2.4 Routing — Overview

Routing is the task of moving some message from ES $A$ to ES $B$, where both $A$ and $B$ are part of some larger network. To do this, we need to find a path through the network from $A$ to $B$. The task of routing (a.k.a. pathfinding) belongs to the network layer.

A routing algorithm determines a route for a packet to take. Routers (ISs) run these routing algorithms and forwards packets to the right nodes. A router does the task of routing to fill a routing table, which is then used in the process of forwarding packets.

Different networks have different routing algorithms. The Internet uses several "simultaneously". Any single (or in the case of hierarchical networking, lowest level) network uses the same routing algorithm in all ISs, because routing wouldn't work otherwise.

> **Routing vs. Forwarding** Routing and forwarding are different processes that both happen inside of routers. The routing process is that of populating the routing table, that is, taking decisions on which routes to take to which known destinations. Forwarding is the process of forwarding a packet to a neighboring IS using the information in the routing table. Routing is a continuous process, while forwarding happens whenever a packet arrives.

The three main different routing methods are unicast routing, multicast routing, and broadcast routing. Unicast routing is one-to-one, where a message is sent from a sender to a single receiver. Multicast routing sends packets from a single sender to a specific set of receivers (1:n). Broadcast routing sends packets from a sender to *everyone*.

## 2.5 Congestion Control

### 2.5.1 Basics

Congestion occurs in the case of too much traffic, where the IS service rate is too slow of there is a limited capacity on its outgoing lines. Congestion tends to amplify itself, where more congestion occurs due to congestion further in the line. This happens, for example, when an IS drops a packet due to congestion and it has to be retransmitted. This uses additional bandwidth close to the sender.

### 2.5.2 Avoidance

#### Traffic shaping

Congestion is often caused by bursts. If we smoothen the traffic (at cost of a delay), ISs will be able to keep up more easily. To do this we need to negotiate the traffic contract beforehand. The traffic is shaped by the end system.

**Traffic shaping by leaky bucket** In the leaky bucket model we view the network buffer as a leaky bucket. The end system writes to the bucket and it slowly drips out by a hole in the bottom. This allows for smoothening of the traffic stream. If the buffer reaches maximum capacity the sender may either block, or drop packets.

**Traffic shaping by token bucket** The token bucket permits a certain amount of data to flow for a certain amount of time, by controlling it with a limited number of tokens. Tokens are periodically added to a bucket. For each packet received, we remove one token from the bucket. This allows for limited bursts of a certain size, but if the bucket is empty we have to wait to send more data.

#### Reservation

Reservation uses the concept of virtual circuits. It reserves the amount of needed resources in the ISs when setting up the connection, and chooses an alternative path or refuses connection if the requested resources are not available. When congestion exists, the actual network layer may reroute the routing when needed, without the virtual circuit noticing this.

### 2.5.3 Reaction and Correction

#### Packet dropping

One way to correct for congestion is to drop packets. This means congestion won't get worse *at this location*. Retransmitted packets waste bandwidth, where a packet has to be sent $x$ times before it is accepted, where

$$x = \frac{1}{1-p}$$

with $p$ being the probability of the packet being dropped.

Packet dropping is optimized by dropping packets early in their travels, because this reduces the amount of bandwidth wasted by retransmitting.

#### Choke packets

Traffic can also be reduced by signaling to the source to slow down sending. A way to do this is by sending CHOKE packets. When a IS detects high utilization on an incoming line, it may send a CHOKE packet. When this is received by the source, it will adapt its sending rate.

# Chapter 3

# Routing

## 3.1 Foundations of Routing

Routing is the task of finding the "best" path to go from one system to another. The routing process updates the routing table, which is then used by the forwarding process which we talked about in chapter 2.

Desireable properties of a routing algorithm is correctness, simplicity, robustness, stability, fairness, and optimality. There's often a trade-off between multiple of these properties, namely robust vs. stable and fairness vs. optimality. Optimization criteria we use in practice is most often the least number of hops per packet[1].

There is two classes of routing protocols: adaptive and non-adaptive. Non-adaptive algorithms don't use network state to influence the routing; they're static. Adaptive algorithms do use the network state to influence routing: they make decisions based on the network state to improve routing and decrease congestion and other possible issues. Within adaptive routing algorithms there is the three types local, neighboring, and global. This tells us which other ISs any IS will look at to influence their routing decisions.

> **Principle of Optimality** For all subpaths $N'$ on an optimal path $N$ to $S$, the path $N'$ to $S$ is also optimal.

## 3.2 Some algorithms

### 3.2.1 Non-adaptive algorithms

#### Non-adaptive shortest path routing

If the topology of a network is fully static, all routing tables can be manually set once by the system administrator. The routing should occur using the shortest path pairs, obtained using—

[1] This is different for the BGP routing protocol which we get to later

for example—Dijkstra's shortest path algorithm.

#### Non-adaptive flow-based routing

We use the topology and the average capacity per edge. The average delay per edge is calculated according to

$$T_i = \frac{1}{\text{edge capacity} - \text{mean edge utilization}} = \frac{1}{\mu C_i - \lambda_i}$$

where

$$\mu C_i, \lambda_i = {}^{\text{packets}}/_{\text{sec}}$$

The information on utilization is usually given by another algorithm. We know the utilization between any pair of nodes in the network. Using this data we can calculate how much data we can send over each link and what delay there is for each line. We use this to assign a weight to each line using

$$\text{Weight}_{AB} = \frac{\lambda_{AB}}{\sum \lambda_i}$$

#### Non-adaptive flooding

With flooding, each IS forwards a packet to all outgoing lines there are. This generates a potentially "infinite" amount of packets, which isn't desired behavior. To combat this, packets get a hop counter which is decremented after every IS it passed through (hop). When this counter reaches zero an IS simply drops the packet.

Another—arguably better—way to combat this is to use a sequence counter, where any IS appends its ID to the sequence. If an IS receives a packet that it's already seen before, the packet is dropped. This avoids loops.

We also have selective flooding, where a router only forwards the packet in one "direction".

Flooding is very heavy on overhead, so isn't practical in most applications. It is however very robust, so it's good for small-network military use.

### 3.2.2 Adaptive algorithms

We subdivide the adaptive algorithms into centralized algorithms, isolated algorithms, and distributed algorithms. Centralized algorithms have a single system that is in charge of the routing, isolated algorithms have each IS making their own decision, and distributed algorithms have ISs communicate with each other to make decisions.

#### Adaptive centralized routing

In adaptive centralized routing, decisions are made using the current network state. Each router transmits its state to a centralized routing control center (RCC). The RCC now has complete network information and can do perfect reasoning to determine the routes from/to any

pair of ISs. This information is then distributed to the ISs from the RCC so they can do the forwarding.

Adaptive centralized routing has quite low robustness. It also has quite some overhead, since recalculations are necessary at least once a minute, but often even more frequently.

#### Adaptive isolated routing

In adaptive isolated routing, every IS makes decisions based on information they gather themselves. There is only limited adaptive decision-making possible by using this routing algorithm.

#### Adaptive distributed — distance-vector routing

Adaptive distributed routing uses a distance-vector based routing algorithm, a.k.a. Bellman-Ford algorithm or the Ford-Fulkerson algorithm. In this algorithm, the router maintains a table stating the best known distance to destinations and which outgoing line to use (NEXTHOP). It updates its routing table using this information, and it distributes its data to other (nearby) routers.

The distance-vector algorithm has the following steps:

(1) **Initialization**

- The function $c(x, v)$ gives the cost for direct link between $x$ and $v$ if $x$ and $v$ are neighbors, $\infty$ otherwise.
- $D^x(y)$ gives the estimate of the least cost from $x$ to $y$. Node $x$ maintains distance vector $D^x = \{D^x(y) \mid y \in N\}$, that is, the list of estimated distances from itself to all other nodes in the network $N$.
- Node $x$ gets the distance vectors from its neighbors. For each neighbor $v$, $x$ gets $D^v = \{D^v(y) \mid y \in N\}$.

(2) **Update received** or **link cost to neighbor changed**

- If the link to neighbor $v$ is changes by $d$ then $D^x(v) = D^x(v) + d$.
- If update $D^v(y)$ is received from $v$, then $D^x(y) = \min(c(x, v) + D^v(y)$ for all nodes $y \in N)$, that is, $D^x(y)$ is updated if the cost to the destination in $D^v(y)$ plus the cost to get to $v$ from $x$ is lower than the cost currently registered in $D^x(y)$.

(3) Update neighbors $v$ of $x$ using $D^x(y)$.

An issue with distance-vector routing is that of the "count-to-infinity" property. If we have long paths with many hops, information distribution is very slow. This makes is such that when a link is removed the derived path costs are wrong. If we have some path $\{A, B, C\}$ and link $AB$ goes down, $C$ will still report a link to $A$ since $C$ doesn't know it goes through $AB$. Node $B$ reports it's not connected to $A$ anymore, while $C$ reports a connection to $A$, which $B$ will see

as a possible route for it to take to $A$. Node $B$ will keep reporting a link of $C + 1$ to $A$ and this will cause both nodes to start counting to infinity together, all while not actually routing to $A$. Potential solutions to this are the split horizon algorithm and the poisoned reverse algorithm.

**Poisoned reverse algorithm** The idea behind the poisoned reverse algorithm is that when a node $Z$ routes through $Y$ to get to $X$, then $Z$ advertises to $Y$ that its distance to $X$ is $\infty$.

This works in the situation described above, but in a bigger network the count-to-infinity problem may still occur when there is more nodes to consider.

**Split horizon** The idea behind split horizon is to not advertise to routers that sent an update.

#### Adaptive distributed — Link-State routing

Link-state routing uses the basic principle of measuring the distance to directly adjacent ISs, distributes this information, and calculates routes using it. There is five steps in its procedure:

(1) Determine the address of adjacent systems;

(2) Measure the distance (delay, ...) to each of its neighbors;

(3) Construct a packet with the link state information;

(4) Distribute the information to all ISs/routers;

(5) Calculate the shortest route to all other ISs/routers based on the obtained information.

## 3.3 UNICAST enhancements

### 3.3.1 Hierarchical routing

A flat network topology does not scale very well: routing tables grow to unmanageable sizes and algorithms do not converge in a reasonable time. The solution to this is hierarchical networking and autonomous systems (ASs).

An autonomous system is a subnetwork that connects to a higher-level network. When routing, the AS is seen as a single node, and routing is based on this. The Internet has about 70000 ASs which represent the subnetworks that The Internet routes between worldwide. Hierarchical routing has a downside, which is that routing may be somewhat suboptimal compared to static flat topologies.

### 3.3.2 Multipath routing

Routing can be optimized by allowing multiple paths between ISs to exist, instead of just routing the most optimal one.

# Chapter 4

# Internet Protocol

## 4.1 Packet Fragmentation

Sometimes, during routing, fragments need to be split up, since there is a max size a packet can be. These packets are reassembled at the ES where it is destined to go. We have transparent and non-transparent fragmentation. In IP we use non-transparent fragmentation. In transparent fragmentation any router can reassemble the packets and send it in full; in non-transparent fragmentation, only the ES can reassemble the packet in full.

Each fragment gets its own IP header; only the first one gets the TCP header. The packets are reconstructed before the TCP header is of importance. All packets short from the last one are multiples of 8 (octet-)bytes in size.

## 4.2 Internet Protocol (IPv4)

IPv4 uses a dotted decimal notation of a 4-byte address. IPv4 packets have a header and data. The header consists of the following fields:

- Version: 4, 5, or 6
- IHL: Header length
- Type of Service: Sets priority for forwarding. In practice, most routers ignore this. We can set bits to prefer delay, throughput, reliability, or cost
- Total length: Max 65535 bytes
- Identification: Identifies the datagram this fragment belongs to
- Flags: 2 bits: DF = don't fragment; MF = more fragments (means this isn't the last fragment in the datagram)
- Fragment offset: Fragment position in the datagram

- Time-to-live: Life-cycle; max 255. Decremented every hop
- Protocol: The type of higher-level protocol used (protocol numbers)
- Header-checksum
- Source-address
- Dest-address
- Options: Routing, testing, and debugging options
- Padding: Fill header to 32-bit multiple size
- Data: Payload in 8-bit multiple length

**Internet Control Message Protocol (ICMP)** If an error occurs, the ICMP protocol is used to notify the sender. It includes the first 32 bits of the IPv4 data field in its header. The error is reported all the way up the network layers so the application can decide what to do.

ICMP is encapsulated in an IP datagram with protocol number 1. In the end, we have an IPv4 packet containing an ICMP packet with another IPv4 packet as its payload. ICMP headers include a type, code, and checksum for reporting purposes.

## 4.3 Internet Protocol version 6 (IPv6)

The main motivation for moving from IPv4 to IPv6 is the increase of address space. While IPv4 has a 32-bit address space, IPv6 has a 128-bit address space. IPv6 is written in hexadecimal, using 8 pairs of 2 bytes. In IPv6, zeros can be concatenated into a single zero: We can shorten `FE00:0000:0000:0000:0012:0000:0000:0043` to `FE00::0012:0:0:0043`.

The goals of IPv6 also include a simplification of the header, increasing security, supporting real-time traffic, support for multicast, and proper coexistence with IPv4.

IPv6 replaces multicast, unicast, and broadcast with "anycast". This allows sending data to a group, where IP will find the geographically closest member of the group which then sends it to the other members.

### 4.3.1 Headers

IPv6 headers are version; priority; flow label; payload length; next header; hop limit; source address; dest address. A flow label can be shared among fragments so every packet takes the same route to the destination. The "next header" field takes the place of the IPv4 protocol header. IPv6 doesn't have a checksum header. It trusts in a higher level layer to provide this kind of error detection. IPv6 has a minimum packet size of 1280 bytes.

The priority header gives some information about the content of the packet. This is used for the purpose of flow control. This works together with the flow label header field.

IPv6 supports so-called jumbograms, which don't have a payload size limit. This means we do not need fragmentation on the network layer anymore, and higher layers can take care of this.

### 4.3.2 IPv4 → IPv6 transition

All the changes from IPv4 to IPv6 greatly simplified the work of a router, mainly by removing checksums and router-level fragmentation.

To operate IPv6 and IPv4 on a mixed-router network we carry IPv6 packets in an IPv4 packet as the payload that is unpacked if a router supports IPv6.

## 4.4 Internet Address and Routing

Addressing today is done by the Internet Cooperation of Assigned Numbers and Names (ICANN).

We differentiate between classful routing and classless routing. In practice, we only use the latter.

### 4.4.1 Classful addressing

In classful addressing, we have 5 kinds of addresses. The type of address is determined by the prefix. See Figure 4.1. Depending on the class, a different number of global networks and host-addresses within the subnetwork is possible. Which class to choose depends on the application: a big business needs more host addresses, while a city may need a lot of networks with a small number of hosts per network.

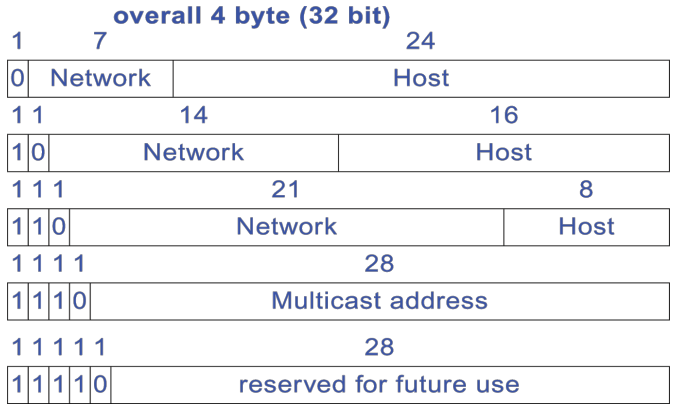


Figure 4.1: Classful Addressing

Internet subnets allow for structured network growth. Having several networks is preferred so there is enough space for, for example, a type B address. A few bits from the host section are used to indicate the subnet. Subnet masks are used to indicate where the network/subnet/host section of the address is.

### 4.4.2 Classless addressing

Nowadays, Classless InterDomain Routing (CIDR) is used. The CIDR principle allows for allocating IP-addresses in variable-sized blocks without regard to classes. A CIDR address includes the standard 32-bit IP address and a subnet mask: `194.24.8.0/22` indicates that the first 22 bits are part of the network identification, and the last 10 bits are for host identification.

If a packet comes to a router and we want to know where it should go, we search the adjacent routers with a larger mask that includes our host; this routing gets us close to the host by moving to a more specific subnet.

## 4.5 Address Resolution

Internet addresses have to be mapped to physical addresses for networking to be possible. The Address Resolution Protocol (ARP) is used for this. ARP broadcasts to everyone "Who does this IP belong to?" and waits for the holder of the IP to respond with its physical address, which are stored in the router cache. See also: Reverse Adress Respolution Protocol (RARP).

Nowadays we mostly use DHCP to assign IP addresses. A client broadcasts a DHCP DISCOVER packet and the router responds with a DHCP OFFER packet, including an IP address that is offered to the client. Addresses assigned in this way have a lease so that they expire after some time. This makes sure an IP address is added to the pool of available ones when a device disconnects.

# Chapter 5

# IP Routing

## 5.1 Autonomous Systems and Hierarchical Routing

### 5.1.1 Autonomous Systems

Routers (ISs) are grouped into Autonomous Systems (ASs). Within an AS, all routers run the same routing algorithm and know each other. The algorithms that run inside ASs are interior gateway protocols. This routing algorithm can be freely chosen by the network administrator.

Some routers in ASs are gateway routers that connect to other gateway routers in other ASs. These routers also run an exterior gateway protocol, for inter-AS routing. This needs to be standardized among *all* routers; otherwise networks can't properly interoperate.

In networking, each AS has a unique number that is used to identify it. Every AS must also know a route to each other AS/network. This reachability information is distributed by the exterior gateway protocol.

ASs solve the problems of scaling and administrative authority, since using ASs allows for proper hierarchical routing to an arbitrary number of levels, and within an AS the owner has full say about how routing is done, as long as the gateways support the exterior gateway protocol.

### 5.1.2 IP Routing: Internal and External Routing

In internal routing, both the source and destination ES are in the same subnetwork/AS. This allows for direct routing from source to destination using the internal routing protocol. When source and destination ES are on different subnetworks, we have indirect—or external—routing.

### 5.1.3 Initial Gateway-to-Gateway Protocol (GGP)

#### Original Implementation

In the original implementation of GGP, each AS was connected to a big backbone using a gateway router. All these gateway routers communicated through the backbone. Having a huge backbone like this results in very poor scalability and causes a lot of overhead.

#### Solution: Hidden Networks

Have an AS with a core gateway to the backbone, and allow for more gateway routers to connect to this AS. The first gateway router then has the reachability information for all the ASs that fall under it. This causes a more tree-like structure which is far more scalable in terms of backbone size.

## 5.2 Real World: Interior Gateway Protocol

### 5.2.1 Routing Information Protocol (RIP)

RIP was developed as part of the Berkeley UNIX in 1988. It is a distance-vector routing algorithm. It periodically sends a list containing estimated distances to all its neighbors. The distance is represented as number of hops between 0 and 15, where 15 is regarded as $\infty$. This is done every 30 seconds. After 180 seconds of no packet from a known neighbor, its distance is set to $\infty$ as the connection was most likely lost.

### 5.2.2 Open Shortest Path First (OSPF)

OSPF is a link-state protocol. Each IS measures its "distance" to the adjacent ISs and distributes this information. This is then used to calculate the optimal route from IS to IS.

Advanced features in OSPF compared to RIP are: reliable transport of OSPF message in TCP connections; authentication of all OSPF messages; support for multiple cost metrics; integrated uni- and multicast support.

## 5.3 Real World: Exterior Gateway Protocols

### 5.3.1 Border Gateway Protocol (BGP)

The Border Gateway Protocol is the protocol that The Internet uses as its exterior gateway protocol. BGP is a distance-vector protocol, where an IS periodically sends its neighbors a list containing the estimated distance from itself to *all known* destinations. Along with this it also sends a preferred path, so certain routing decisions can be made w.r.t. factors other than just shortest distance.

# Chapter 6

# Transport Layer

## 6.1 Transport Service

The transport layer enable inter-process communication. It is referred to as an end-to-end protocol as it interconnects applications. Common requirements for the transport service include: guarantee of transmission; order preservation; client/server pattern support; efficient communication; arbitrarily large messages; sender/receiver synchronization; flow control.

Most transport protocols offer either connectionless service (e.g. UDP) or connection-oriented service (e.g. TCP). On the transport layer, addressing is done by manner of ports. Processes can bind to a port and use it for communication. This way an ES can be addressed using an IP address and within the ES the application process can be addressed using a port.

A process knows about a port either using well-known ports for certain applications (22 for SSH, etc.) or by use of a mapping of service name to IP-address and port.

## 6.2 User Datagram Protocol (UDP)

UDP is a connectionless protocol. It does not offer reliable communication: this is the application's responsibility. It only gives minimal additional services compared to IP. It has finer addressing, namely ports and it has error detection—not correction—of bit failures.

UDP messages have a small header, including a source and destination port, length, checksum, and data padded to 32-bit multiple.

## 6.3 TCP Service

The TCP service provides a communication channel with failure-free transmission. It is order preserving with full reliability, flow control, and congestion control. TCP build on the IP protocol. It "fixes" the IP issues that are packet length limits, unreliable communication, and no preserved order.

The TCP service has a byte stream abstraction that the sender writes bytes to and the receiver reads bytes from. TCP instances exchange segments between them. TCP segments are a bit more complication than UDP ones. A segments has the following headers: source port; destination port; sequence number (for ordering); acknowledgement; advertised window; flags; UgrPtr; checksum; header length.

Flags contain control information. This may be SYN or FIN for connection setup, ACK if the acknowledgement field is used, URG is set if the segment compromises urgent data, PUSH if the sender called push, and RESET if the connection is to be aborted. The UrgPtr identifies urgent data if the URG flag is set.

### 6.3.1 TCP Data Transmission

In data transmission, TCP needs reliable ordered communication. This means there has to be treatment of flipped bits, loss of segments, duplication of segments, and reordered segments.

In TCP bit errors are fixed by adding a checksum to every segment. This allows detecting bit failures with high probability. Upon detection of a bit failure, the segment is simply dropped. A very simple procedure to do this is parity bits.

Loss of segments is fixed by positive acknowledgement and retransmitting. There is three steps: (1) TCP sender starts a timer when sending segment $S$; (2) TCP receiver acknowledges $S$ with ACK; (3) upon timeout—before ACK is received by the sender—the sender retransmits $S$. Using this, as long as $P(\text{successful transmission}) > 0$ then eventually we succeed.

Duplicates of segments may occur using the above method. The way to solve this is by sending sequence numbers in acknowledgements. This tells the sender the number of successfully received bytes, so it knows what to resend and so it knows an ACK belongs to the packet it should. If ACK is sent after every message, a message $S$ may cause multiple ACK packets leading to a wrong number of ACK packets at the sender.

With the above, we can optimize using cumulative acknowledgements. Using this, one ACK can acknowledge multiple packets by adding the sequence number of the last continuously received packet. This increases robustness against loss of ACKs.

### 6.3.2 TCP State & Flow Control

The TCP send buffer comprises of data that has been sent but not acknowledged by the receiver or written by the application but not yet sent. The TCP receive buffer comprises of data that is received but not yet read by the application. The sending buffer has three pointers: LastByteAcked, which is the last byte to be acknowledged, LastByteWritten, which is the last byte that the application wrote to the buffer, and LastByteSent, which is the last byte that was sent. The receive buffer has the pointers LastByteRead, NextByteExpected, and LastByteRecvd.

The receiving buffer is limited in size. Flow control avoids overflow of this buffer. Every ACK packet contains AdvertisedWindow, which is MaxRcvBuffer $-$ (NextByteExpected $-$ LastByteRead $-$ 1), that is, the size of the available buffer at the receiver. It also includes

the next byte expected. The sender receives these values and computes the size of the available buffer at the receiver using EffectiveWindow $=$ AdvertisedWindow $-$ (LastByteSent $-$ LastByteAcked). The sender blocks sending if EffectiveWindow $= 0$.

### 6.3.3 Establishing a Connection

The first stage of TCP transfer is establishing a connection. We first consider a two-way-handshake, then a three-way-handshake establishing.

#### Two-way-handshake

Two control messages are sent: ConnectRequest ($CR(X)$) and ConnectConfirmation ($CC(X)$). These carry $X$, which is a reference of the connection. After establishing the connection duplicate segments of the same connection carry $X$ too and can be safely dropped.

Using a two-way-handshake we cannot detect duplicate control messages. This may lead to undefined state between sender and receiver, which is bad. This is solved by using locked references.

> **Locked References** When at time $t$ a reference $X$ is admitted, all duplicates that comprise reference $X$ are extinct, that is, $X$ isn't reused within the lifetime of packets that used $X$ in history. A locked reference may be at most once admitted within a time interval $T$, where $T$ depends on the protocol's maximal message lifetime.

Our problem is still not completely solved by this, so we consider the three-way-handshake next.

#### Three-way-handshake

We send a $CR(X)$ from sender to receiver. The receiver then sends $CC(Y, X)$, where $Y$ is a different locked reference from $X$. The sender then returns $AK(X, Y)$. Segments are only accepted after reception of $AK(X, Y)$.

### 6.3.4 Congestion Control

TCP supports congestion control: the sending rate is adapted depending on the expected capacity of the connection. It is designed such that the bandwidth is fairly shared between multiple connections. We maintain a congestion windows which is the maximum number of unacknowledged bytes in transmission. It is increased if capacity limit isn't reached, and reduced when retransmission of segments shows to be necessary.

## 6.4 QUIC Service

QUIC UDP is a service motivated by making the Web faster. It enhances the transport layer but also takes over part of the application layer. It is a reliable multiplexed transport protocol atop

of UDP; optimized for HTTP/2. QUIC is still a work in progress, but has the following design goals compared to TCP+TLS+HTTP/2:

- lower connection establishment latency;
- flexible congestion control;
- multiplexing without head-of-line blocking;
- authenticated header and authenticated and encrypted payload;
- forward error correction;
- better flow control;
- connection migration.

# Chapter 7

# Distributed Programming

## 7.1 Communication — Basic Concepts

In concurrent programming we can distribute sections of the program to different processes. We communicate between these processes using a communication step. These processes may be on different networked machines. This can be done using unicast, broadcast, and multicast, which we have seen before.

## 7.2 Message Passing: Basic Operations

When message passing we send something to a specific destination from one process and receive something from a specific source from the receiver. We have blocking and non-blocking IO. Blocking IO blocks the target process as long as the sent message hasn't been received or as long as their has been no received message. Non-blocking IO writes to a buffer and then continues the process regardless of whether the sending/receiving is done yet. When using non-blocking IO on the sender and/or receiver, we have asynchronous message passing. When using blocking IO on both sides we have synchronous message passing.

## 7.3 Remote Procedure Call

A remote procedure call (RPC) allows for processes to call specific procedures in other address spaces on the same or remote machines. The calling process is blocked until a return value is received. No message passing or IO is visible to the programmer: this is fully abstracted away.

### 7.3.1 Remote Procedure Call Architecture

Both the server and client have a program, a stub, and a communications component. The program is that issuing the RPC or processing it; the stub is what translates the data from

the program into something ready for communication, and vice versa; the communications component does the communication.

### 7.3.2 Remote Procedure Call Interface Definition Language (IDL)

In general, and RPC mechanism must work in heterogeneous environments. This has us need a programming language independent IDL and language binds for the corresponding programming language. This IDL defines a set of procedures and their signatures and an it defines an identifier scheme to uniquely identify procedures.

An IDL compiler generates the client and server stubs.

### 7.3.3 Remote Procedure Call Binding

Binding is the task of selecting a server implementing the called remote procedure and binding the client to the server. The client-stub performs this binding during runtime. This is done—for example—using a directory service, where the client asks the directory service where to find the procedure-implementing server and the directory service responds with what to bind to.

### 7.3.4 Remote Procedure Call Failure Semantics

The ultimate goal for RPC systems is achieving distribution transparency to the greatest extent possible. This allows for simple programming and distribution-independent programs.

There is a set of semantics that is defined in case of failures. These are “maybe”, “at-least-once”, “at-most-once”, and “exactly-once”. This may guarantee some aspect of the execution when failure occurs.

*See the slides for how to implement these.*

## 7.4 Distributed Objects and Remote Method Invocation

RMI expands on RPC by allowing object-based technology to interact with RPC, where objects (or references to objects) are sent along in the request. We distinguish between local and remote objects, where local objects are passed by-value and remote objects are passed by-reference. Sadly, this allows only for limited distribution transparency for the programmer.

# Chapter 8

# Naming

## 8.1 Names and Addresses

Addresses contain an entity's location or access point. A name identifies the entity, independent of its location. Names are a prerequisite for distribution transparency.

If we know a name, we can use a name resolution to obtain the address. The name resolution has a database mapping names to addresses.

## 8.2 Organization of Name Systems — in General

We have two types of names: flat names and structured names. Flat names are good for machines, rather than humans, since they're not easy to understand. Structured names are hierarchically structured which makes them better for human-readability.

Name systems maintain name-to-address bindings. In its simplest form, the binding is implemented by a table of $(\text{name}, \text{address})$ pairs. Typically this is partitioned and/or (partly) replicated. Organization has a strong impact on efficiency, scalability, and availability.

## 8.3 Organization of Name Systems — Flat Naming

A simple solution is broadcast: a request including name of entity is broadcast to each node. Each nodes checks whether it hosts that entity and nodes that host that entity reply with the entity's address. Examples of this are DHCP and ARP.

DHCP works in four steps:

(1) DHCP discover is sent by the arriving client to the DHCP server;

(2) DHCP offer is sent from the server to the client, with a IP address that is offered to the client;

(3) The client sends DHCP request to the DHCP server which indicates to the server that the client would like to use the offered address;

(4) The server then sends DHCP ACK to the client. After this the client is reachable.

Flat naming can be done using a Distributed Hash Table, where entities are represented as key-value pairs. A hash function then maps these keys to the identifier space. Using this we can find which node is responsible for resolution of which entity (not which node hosts the entity directly). An example of this is the "Chord"-system. See the slides for this.

## 8.4 Organization of Name Systems — Structured Naming

Structured names are easier to remember. Examples of structured naming systems are DNS, Unix file names, and URLS. In hierarchical namespaces, a leaf node represents a named entity; a directory node has a number of outgoing edges and we have a root note. In theory, multiple root nodes are possible.

In DNS, a name space is partitioned into disjoint zones. Each zone is entirely managed by a name server, though zones may be replicated on multiple names servers. High level name servers have high availability and high performance. Low level name servers have local relevance of naming information.

Resolution can be iterative or recursive. In iterative, the client keeps asking more specific nameservers for info; in recursive resolution the name servers ask the more-specific name servers themselves. Doing recursive resolution has a higher performance demand on each name server but caching becomes more effective and communication *may* be reduced.