Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

Computer Engineering 2

Roger Wattenhofer

wattenhofer@ethz.ch

Spring 2016

Contents

Introduction

What are Operating Systems?

Computers come in all shapes and sizes: servers, laptops, tablets, smartphones, smartwatches, all the way down to that tiny microcontroller in a washing machine. People buy a computer because (i) it gives them access to the Internet, (ii) it provides storage, and probably also because (iii) it computes. While having network access seems to be vital, advanced storage and computing capabilities more and more move to designated servers ("the cloud"). In this lecture, we learn how computers provide networking, storage, and computation by means of an operating system.

We start out with networking, and discuss the internet protocol, addressing, routing, transport layer protocols, flows, some representative application layer protocols, and how to implement these with sockets. We also discuss the link and physical layer, Markov chains and PageRank, and selected topics in security. Regarding storage, we talk about the memory hierarchy, file systems, caching, efficient data structures such as hashing, and data base principles. Concerning computation, we discuss the virtualization of the processing units with processes and threads. We focus on concurrency and examine scheduling, locking, synchronization, mutual exclusion, deadlocks, and consistency.

Course Overview

The lecture will use various teaching paradigms. The majority of the lecture will be based on blackboard discussions, supported by a script. Where appropriate we will also use slides or demonstrations. A few lectures will be flipped classroom style. The lecture will feature weekly paper exercises.

However, some of the course material is best learned in front of an actual computer. In addition to the lecture we offer exciting hands-on exercises in a lab environment.

Have fun!

Chapter 1

Network Layer

How is data navigated between the billions of computers of the internet?

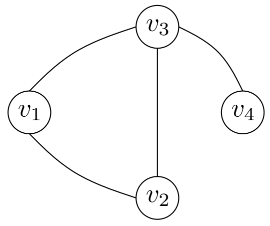
1.1 Graphs

A graph is an abstract model for communication networks, and many other types of networks.

Definition 1.1 (Graph)**.** *A graph G is a pair (V, E), where V is a set of* ***nodes*** *and $E \subseteq V \times V$ is a set of* ***edges*** *between the nodes. The number of nodes is denoted by n and the number of edges by m.*

Remarks:

- In the Internet, there are many types of nodes: Computers or smartphones that communicate over an infrastructure that consists of routers or switches. These nodes are connected by wired or wireless edges.
- A typical way to store a graph is an *adjacency matrix*. An adjacency matrix is a binary $n \times n$ square matrix with a 1-entry in location (i, j) if and only if nodes i and j are connected by an edge.



	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	1	0	1	0
v_3	1	1	0	1
v_4	0	0	1	0

Figure 1.2: A graph $G = (V, E)$ with node set $V = \{v_1, v_2, v_3, v_4\}$ and edge set $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}\}$, and the adjacency matrix of G.

Remarks:

- We say that node u is *incident* to edge $\{u, v\} \in E$ and we say that nodes u and v are *adjacent*. The neighborhood $N(v)$ of node v is the set of nodes adjacent to v, i.e., $N(v) = \{u \in V \mid \{u, v\} \in E\}$. The size $|N(v)| = \delta(v)$ of the neighborhood of node v is referred to as the *degree* of v. In Figure 1.2, $\delta(v_1) = 2$.

- A *directed* graph $G = (V, E)$ is a graph, where each edge has a direction, i.e., we distinguish between edges (u, v) and (v, u). If all edges of a graph are undirected, then the graph is called *undirected*. The matrix representation of an undirected graph is always symmetric, whereas an asymmetric matrix can also encode undirected edges.

Definition 1.3 (Weighted Graph)**.** *A* ***weighted*** *graph $G = (V, E, \omega)$ is a graph, where $\omega : E \to \mathbb{R}$ assigns a weight $\omega(e)$ for each edge $e \in E$. The weight of graph G is $\omega(G) = \sum_{e \in E} \omega(e)$.*

Remarks:

- To capture weighted graphs, the adjacency matrix representation can be extended by replacing the 1-entries by the weights of the corresponding edges.

- If the graph is *sparse*, i.e., has few edges, the matrix representation can take a lot of unnecessary space. Such a sparse matrix can be stored as an *adjacency list*. In an adjacency list, every element corresponds to an edge of the graph identified by its endpoints.

Definition 1.4 (Path)**.** *Let $G = (V, E)$ be a graph. A* ***path*** *between nodes v_1 and v_k is a sequence of nodes $(v_1, v_2, \ldots, v_k)$, where $\{v_i, v_{i+1}\} \in E$ for all $1 \le i < k$. The* ***length*** *of the path is $k - 1$.*

Definition 1.5 (Connected Graph)**.** *A graph $G = (V, E)$ is* ***connected*** *if there exists a path between any two nodes $u, v \in V$.*

Definition 1.6 (Cycle)**.** *Let $G = (V, E)$ be a graph. A* ***cycle*** *is a sequence of nodes $(v_1, v_2, \ldots, v_k, v_1)$ such that $\{v_k, v_1\} \in E$, $\{v_i, v_{i+1}\} \in E$ for all $1 \le i < k$, and no node appears twice.*

Definition 1.7 (Tree)**.** *A* ***tree*** *is a connected graph that contains no cycles.*

Remarks:

- Trees are connected graphs with $n - 1$ edges. By removing any edge in a tree, the tree becomes disconnected.

- A *rooted* tree is a tree with a special root node r. Every other node v has a *parent*, that is the node adjacent to v and closer to r.

- Next, we will discuss ways to construct *spanning trees*. A spanning tree is a tree that connects all nodes in a graph.

1.2 Spanning Trees

Definition 1.8 (Subgraph)**.** *Let $G = (V, E)$ be a graph. A* ***subgraph*** *$G' = (V', E')$ of G is a graph such that $V' \subseteq V$ and $E' \subseteq E$.*

Definition 1.9 (Spanning tree)**.** *Given a graph $G = (V, E)$, a* ***spanning tree*** *$T = (V, E')$ is a subgraph of G that is a tree.*

Definition 1.10 (MST)**.** *Given a weighted graph $G = (V, E, \omega)$, a minimum spanning tree (MST) T is a spanning tree that minimizes the total weight $\omega(T)$.*

Remarks:

- In the beginning of the 20th century, at the time of electrification, engineers were faced with the problem of designing an efficient network of power lines. In particular, a Moravian (Czech) academic called Otakar Borůvka defined this problem abstractly, and observed that the power grid should have the following properties: (1) it should connect all the nodes; (2) building lines is expensive, thus redundant edges (edges which can be removed without disconnecting the power grid), should be avoided; (3) the cost should be minimized. In other words, he defined the Moravian Spanning Tree (MST) problem.

Algorithm 1.11 MST Algorithm

```
1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let $T$ be a tree just consisting of the single node $u \in S$, no edges
4: while $S \neq V$ do
5:    Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6:    Add node $w$ to $S$
7:    Add edge $e$ to $T$
8: end while
```

Lemma 1.12. *Algorithm 1.11 outputs a minimum spanning tree.*

Proof. In every iteration of the while loop, a new edge is added. Since the new edge connects the current tree to a new unseen node, it does not produce a cycle, and the output is indeed a tree. Since nodes are added until $S = V$, the tree is a spanning tree.

Now for minimum weight: To simplify the proof, let us assume that no two edges have the same weight. Assume (for the sake of contradiction) that our tree T is not of minimum weight, and the true minimum spanning tree T^* has weight $\omega(T^*) < \omega(T)$. Let e be the first edge added to T that is not in T^*. Edge e is the cheapest edge that connects a node in set S with a node in $V \setminus S$. Since T^* is not using e, it must use another edge e^* to connect the nodes in S with the nodes in $V \setminus S$, with $\omega(e^*) > \omega(e)$ Replacing e^* with e in T^* improves $\omega(T^*)$ by $\omega(e^*) - \omega(e) > 0$, a contradiction to the assumption that $\omega(T^*)$ was minimum. □

Theorem 1.13. *The time complexity of Algorithm 1.11 is $\mathcal{O}(m \log n)$.*

Proof. We use a heap data structure to memorize the edges which are eligible, i.e., edges which connect nodes in S with nodes in $V \setminus S$. Whenever a node v is added to set S, we add all edges adjacent to v to the heap. Adding an edge e to the heap costs time $\mathcal{O}(\log m)$, as there are at most m edges in the heap. Removing the minimum-weight edge e from the heap also costs time $\mathcal{O}(\log m)$. Testing whether the current minimum-weight edge e is still eligible (one of its adjacent nodes still in $V \setminus S$) costs constant time. As such, every edge e enters and leaves the heap at most once, with m edges this gives a total cost of $\mathcal{O}(m \log m)$. With $m < n^2$, we have $\log m < 2 \log n$, and a total runtime of $\mathcal{O}(m \log n)$. □

Remarks:

- The term $\mathcal{O}()$ used in Theorem 1.13 is called "big O" and is often used in math. Roughly speaking, $\mathcal{O}(f)$ means "in the order of function f, ignoring constant factors and smaller additive terms". More formally, for two functions f and g, it holds that $f \in \mathcal{O}(g)$ if there are constants x_0 and c so that $|f(x)| \leq c|g(x)|$ for all $x \geq x_0$. For an elaborate discussion on the big O notation we refer to other introductory math classes, or Wikipedia.

1.3 Shortest Path

Definition 1.14 (Shortest path)**.** *Let $G = (V, E, \omega)$ be a weighted graph. The shortest path between nodes $u \in V$ and $v \in V$ corresponds to the path P between u and v of minimum total weight $\omega(P)$.*

Definition 1.15 (Distance)**.** *Let $G = (V, E, \omega)$ be a weighted graph and let P be the shortest path between nodes $u \in V$ and $v \in V$. The **distance** $d(u, v)$ between u and v is $\omega(P)$.*

Remarks:

- In the unweighted case, the distance corresponds to the length of the shortest path.
- A shortest path tree (SPT) is a spanning tree T, rooted at node r, of graph $G = (V, E, \omega)$, where the distance from any node $v \in V$ to r in T equals to the distance $d(r, v)$ in G.

Lemma 1.17. *Algorithm 1.16 computes the shortest path between node r and every other node v.*

Proof. By induction, we assume that every node v in set S with $d_v \leq d$ has the correct distance d_v and parent p_v. This is true initially, with only root r in set S. In every iteration of the while loop, a new node w is added to set S. The new node w we add to S has the minimum distance $d_w = d_v + \omega(e)$ to the root. As such, node w is the nearest node that is directly reachable from set S. The nodes not directly reachable from S cannot be closer than w because any (shortest) path must go through some node reachable from S. So node w can be added to S, with v being the correct parent, and d_w being the correct distance. Since nodes are added until $S = V$, every node will be included. □

Algorithm 1.16 SPT Algorithm

```
1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: while $S \neq V$ do
6:     Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7:     Set $p_w = v$
8:     Set $d_w = d_v + \omega(e)$
9:     $S = S \cup \{w\}$
10: end while
```

Theorem 1.18. *Algorithm 1.16 runs in time $\mathcal{O}(m \log n)$.*

Proof. The proof is similar to the proof of Theorem 1.13. We use a heap data structure to memorize the edges which are eligible, i.e., which nodes in S connect with which nodes in $V \setminus S$. Whenever a node w is added to set S, we add all edges adjacent to w to the heap. Adding an edge e to the heap costs time $\mathcal{O}(\log m)$, as there are at most m edges in the heap. Removing the minimum-weight edge e from the heap also costs time $\mathcal{O}(\log m)$. Testing whether the current minimum-weight edge e is still eligible (one of its adjacent nodes still in $V \setminus S$) costs constant time. As such, every edge e enters and leaves the heap at most once, with m edges this gives a total cost of $\mathcal{O}(m \log m)$. With $m < n^2$, we have $\log m < 2 \log n$, and a total runtime of $\mathcal{O}(m \log n)$. □

Remarks:

- Runtime can be improved to $\mathcal{O}(m + n \log n)$ by using a so-called Fibonacci heap.

1.4 Addressing

To allow unambiguous node to node communication, every node requires a unique *address*.

Definition 1.19 (Address)**.** *Every node in a graph has an address.*

Remarks:

- In the graph model, the nodes can be addressed by their unique names $v_1, v_2, \ldots, v_n$. What about the Internet?

Protocol 1.20 (IPv4)**.** *Every node in the network is assigned a unique* 32*-bit label. For readability, the* 32 *bits are grouped into* 4 *chunks of* 8 *bits, i.e., the addresses range between* 0.0.0.0 *and* 255.255.255.255*.*

Remarks:

- Some IPv4 addresses have a special meaning. The IPv4 address 127.0.0.1, for example, is reserved for the localhost, i.e., an address for a computer back to itself.
- A *prefix* of k bits of an IPv4 address corresponds to the first k bits of the address. An address *block* is the set of addresses that share a prefix. The set of (private) addresses sharing the first 12 bits between 172.16.0.0 and 172.31.255.255 is denoted by 172.16.0.0/12. In the early days of the Internet, IPv4 addresses were assigned in huge blocks, e.g., MIT owns the address block 18.0.0.0/8. This waste of addresses (as well as the need for hiearchical addressing) resulted in the need for more IP addresses, giving rise to the IPv6 protocol.
- Soon many devices will be Internet capable, including specialized heart-rate pacemakers. There are estimates that there will be over 25 billion devices connected to the Internet by 2020. These devices have to be reachable by an address.

Protocol 1.21 (IPv6)**.** *Every node in the network is assigned a unique* 128*-bit label. In the IPv6 notation, the address is represented as* 8 *chunks of* 16 *bits separated by colons, where each chunk is written as* 4 *hexadecimal digits.*

Remarks:

- Since IPv6 has a huge number of addresses, nodes often have more than one address.
- To simplify the IPv6 notation, the standard is to leave out leading zeros in every chunk. Furthermore, a consecutive section of zeros can be replaced by a double colon. However, the double colon notation can only be used once in an address to preserve unambiguity. Therefore, the IPv6 address 6666:0db8:0000:0000:ff00:0000:0042:8329 can be written as 6666:db8::ff00:0:42:8329.
- Every IPv4 address is included in the IPv6 domain as $::ffff:abcd:efgh$, where *ab.cd.ef.gh* is the (hexadecimal) IPv4 address.
- IP addresses are not really user friendly. When browsing the Internet, a site is accessed by another address, the *hostname*, e.g., **www.netflix.com**. The Domain Name System (DNS) is a service that translates the hostname into an IP address. We will learn about it in Chapter 3.
- There are many more examples of real world addressing. For example, every land line phone has a unique address, the phone number.

1.5 Packets

In the Internet, the communication is based on *packets*.

Definition 1.22 (Packet)**.** *Every network packet contains a* ***header*** *and a* ***payload****. The payload of a packet corresponds to the actual data of the packet. The header contains information for delivering the payload.*

Remarks:

- The size of an IPv4 packet is theoretically limited to $65,535$ bytes and thus, to send a lot of information, the nodes need to send a lot of packets.
- The IPv4 header contains at least 160 bits of information. The header contains, e.g., the source and the destination IPv4 address. One field of 4 bits is reserved for the version number, e.g., for IPv4, this field contains the binary value 0100. Furthermore, an IPv4 header contains a checksum (for error checking the contents of the header), and a number of possible *options* that can be used, e.g., for debugging. Due to the variable amount of options, the header contains a field for the length (in bytes) of the header, up to 480 bits. There is also a field for the total length (in bytes) of the packet.
- Some nodes or edges can handle larger packets than others and sometimes, a packet has to be split into smaller *fragments* before forwarding it. The header contains an identification number of 16 bits for identifying a group of fragments plus more fields for ordering of the fragments, and whether or not there are more fragments to come. Fragmenting is unpopular, so in practice, packets are usually smaller than $1,500$ bytes.
- To prevent packets from traveling in the Internet indefinitely (due to routing misconfigurations), the header contains an 8 bit time-to-live (TTL) field, which specifies how many hops a packet is allowed to travel before it is dropped. Every node decrements the TTL by one and drops the packet if $\text{TTL} = 0$.
- Some of the IPv4 header fields are rarely used and, perhaps surprisingly, the IPv6 header contains less fields than the IPv4 header. For example, the IPv6 header does not contain a checksum. In total, the IPv6 header has 320 bits, most of these for source and destination address.
- In the literature, a packet is sometimes referred to as a *datagram*.

1.6 Routing

The task of a *routing* protocol is to decide along which path(s) a packet travels from its source to its destination.

Definition 1.23 (Routing)**.** *Every node v has a* ***routing table*** *that maps every destination address to a neighbor of v.*

Remarks:

- The process of an intermediate node receiving a packet and sending it to the next node along the path to the destination is referred to as *forwarding.* To perform forwarding, every node has to know to which neighbor to forward each packet.

Routing table of v_1	
Destination	Next node
v_1	deliver
v_2	v_2
v_3	v_3
v_4	v_3

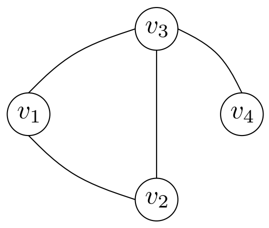


Table 1.24: A simplified routing table.

Remarks:

- Since there are millions of devices connected to the Internet, it is infeasible for every node to store an entry in the routing table for every destination. Towards this end, close-by nodes often have similar addresses, i.e., they share a common prefix. Thanks to this "hierarchical" addressing, routing table sizes remain manageable, as often enough a single entry for all addresses with the same prefix is sufficient. Nodes just match a destination address to longest matching prefix in their routing table – this is known as longest prefix matching.

- Assigning addresses that minimize the maximum routing table storage needed per node is an interesting question. In the literature, this problem is known as the *compact routing* problem.

Algorithm 1.25 Link-State (LS) Routing Algorithm.

```
1: Given a weighted graph $G = (V, E, \omega)$
2: Learn $\omega(e)$ for every edge $e \in E$
3: Compute shortest paths to between all nodes, e.g., by using Algorithm 1.16
```

Remarks:

- The nodes eventually discover changes in the network topology and update their routing information.

- Link-State routing is used in practice, for example, by the Open Shortest Path First (OSPF) protocol. On top of Algorithm 1.25, OSPF offers advanced features such as routing along multiple paths to increase performance.

- A drawback of LS routing is that the nodes need to know the whole topology of the network and therefore, LS routing is not feasible on a larger scale. However, autonomous systems in the Internet are relatively small and therefore, LS algorithm is useful for routing within autonomous systems.

Definition 1.26 (Autonomous System)**.** *Let $G = (V, E)$ be a graph. An autonomous system (AS) is a collection of nodes owned by one company, i.e., a subgraph $G' = (V', E')$ of G, where every $u \in V'$ has the knowledge of the whole topology of G'.*

Remarks:

- Every autonomous system is identified by a unique autonomous system number (ASN), which helps to unify multiple blocks of addresses (that do not share the same prefix) given to the same AS. The ASN is used, e.g., by the Border Gateway Protocol (BGP) to advertise connections between distinct autonomous systems.
- The number of autonomous systems in the Internet is relatively small, below $100,000$.

Algorithm 1.27 Distance-Vector (DV) Routing Algorithm.

```
1: Given a weighted graph $G = (V, E, \omega)$ and a node $u \in V$
2: Initialize a distance estimate $D(u \to v) = \omega(\{u, v\})$ for all neighbors $N(u)$
   and $D(u \to w) = \infty$ for all other nodes
3: Send distance vector $\mathcal{D}(u) = \{D(u \to v) \mid v \in N(u)\}$ to all neighbors $N(u)$
4: while true do
5:   Upon receiving a distance vector $\mathcal{D}(v)$ from a neighbor $v$, update the
     distance estimate to all destinations accordingly
6:   if $D(u \to w)$ changed for any $w$ then
7:     Send the updated distance vector $\mathcal{D}(u)$ to all neighbors
8:   end if
9: end while
```

Remarks:

- Overhead in the packet size can be reduced by only sending the updated entries in the distance vector in Line 7.
- DV routing is distributed, i.e., the nodes do not need to acquire the knowledge of the whole network topology to perform routing.
- On the negative side, the DV algorithm has troubles if (the weights of) the network links are subject to change. Imagine a network with nodes u, v, w and edges with weights $\omega(\{u, v\}) = \omega(\{v, w\}) = 1$. Eventually Algorithm 1.27 will compute $D(u \to w) = 2$ and $D(v \to w) = 1$. Now, we set a new weight $\omega(\{v, w\}) = 100$ and let v detect the weight change. According to Algorithm 1.27, node v still thinks that there is a path to from u and w that has a cost of 2, since u reported that $D(u \to w) = 2$. Thus, v tells u that its cost of the path to w is now

3. Once u learns this new cost, it will similarly update its estimate to 4 and so on. This is known as the *count-to-infinity* problem.

- Distance-Vector routing is used for example by the Routing Information Protocol (RIP). In RIP, the cost to the destination if defined simply by the number of hops, i.e., the underlying graph is unweighted. In the RIP protocol, the distance vectors are exchanged between nodes approximately every 30 seconds. Furthermore, RIP limits the maximum cost of a path to 15, i.e., the protocol can only be executed in a graph where the maximum distance between nodes is 15.

Definition 1.28 (Intra-Domain and Inter-Domain)**.** *The division of the Internet into autonomous systems allows for different routing protocols. Routing within an autonomous system is known as* ***intra-domain*** *routing, often link-state routing algorithms are used. Routing between different autonomous systems is known as* ***inter-domain*** *routing, and the routing protocol is the Border Gateway Protocol (BGP).*

Algorithm 1.29 Border Gateway Protocol (BGP)

```
1: Basically, BGP is a DV Routing Protocol, see Algorithm 1.27
2: BGP nodes send out annoucements about every 30 seconds
3: BGP nodes send reachability information: every node announces which address blocks (prefixes) it can reach
4: Instead of just distance, nodes announce the whole AS path to each prefix
5: The network is not weighted, an edge between two AS nodes costs 1. If an AS does not want other nodes to route through it, the AS will prepend its number multiple times to make the path longer, and hence less attractive
```

Remarks:

- Since nodes announce whole AS paths, BGP is also called a Path Vector Routing Protocol.
- Sending whole paths solves the count-to-infinity and other nasty problems. In our previous count-to-infinity example, node v_2 immediately sees that the path of node v_1 goes through v_2 itself, so it is not a viable alternative.
- BGP allows outbound policies: If a node absolutely does not want to attract traffic to a certain prefix from a neighbor, it simply does not announce the prefix to this neighbor.
- BGP also allows inbound policies: If a node does not want to route through a neighbor, the node just ignores the announcement of that neighbor.
- Good policies allow ASs to make routing decisions. In particular large provider ASs will only route traffic through smaller customer ASs if necessary.
- Now to some downright nasty hacks.

1.7 Tunnels & NATs

Definition 1.30 (Tunnel)**.** *The payload of a packet is a complete packet, with header and payload. In other words, we have two headers.*

Remarks:

- A good application for a tunnel is a virtual private network (VPN), as defined by IPsec. Some node u wants to send a packet to node w as if it was sent by node v. So node u prepares a packet header with source v and destination w, and tunnels that packet in a packet header with source u and destination v. When the first header arrives at v, node v will forward the payload (which is itself a packet with a correct header) to w. VPNs can be used to access a company network (v, w) from nodes outside the company network (u). VPNs can be used to access web services with country restrictions.
- Tunnels are used to sneak through firewalls. A firewall checks whether a packet header is correct, i.e. source and destination addresses make sense. If a firewall is so simplistic, one may simply tunnel an "interesting" packet inside an unsuspicious packet.
- Tunnels are also used to translate back and forth between IPv4 and IPv6 protocols. If some nodes on a path only understand IPv4, an IPv6 packet can be tunneled through these nodes by prepending an IPv4 header.
- Another application of tunnels is security. One can send unencrypted traffic over a network in an encrypted tunnel.
- Tunnels are also used in Multiprotocol Label Switching (MPLS): Many internet service providers (ISPs) implement virtual circuit style routing by prepending an MPLS header, to get more control which paths packets are taking.
- What are virtual circuits? Packets are not the only approach to network communication. In the plain old telephone service, a *circuit* is established on a path for the duration of the call and disconnected afterwards. Simulation of a circuit with packets is called a *virtual circuit.*

Definition 1.31 (Network Address Translation, NAT)**.** *A node systematically exchanges the header of packets in order to be able to route to nodes with private addresses.*

Remarks:

- The address blocks 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16 are reserved for private networks. In other words, addresses of these blocks are not unique as promised in Definition 1.20, but many nodes may have the same address. Nodes outside the private network cannot route to such a private address.

- In IPv6, every edge assigns a private link-local address to the two nodes adjacent to the edge.
- Because of the shortage of IPv4 addresses, ISPs do not want to give many IPv4 addresses to their customers, often each customer gets exactly one IPv4 address. Instead, in a home or a small business, all machines but the entry node (router) only get private addresses.
- We have a client node with a private address in a network with a router, and a server. While the client can easily send a packet with a search query to the server, how does the server send back its answer? When the client packet arrives at the router, the router will switch the client's private IPv4 address with its own router IPv4 address, and forward that to the server. When the server's answer comes back to the router, it will switch back the destination address to the client's address and forward the packet to the client.
- How does the router know which answer belongs to which client, if several clients communication with the server at roughly the same time? For instance by using port numbers, a concept that is not in the network layer (and thus beyond this chapter).

1.8 Beyond IP

The network layer features other protocols, beyond the internet protocol IP. The routing protocol RIP is such a protocol, but also the Internet Control Message Protocol (ICMP). ICMP is used by network devices, like routers, to send error messages indicating that a requested service is not available or that a host could not be reached. ICMP starts with the regular IPv4 header with IP protocol number 1, and then appends a short ICMP header. ICMP is used by some diagnostic tools like ping or traceroute, which tell you whether a node is online, and what the route to a node is.

Chapter Notes

Graph theory and networks are very central topics and have been studied even before the time of modern computers. For example, Euler studied traversal problems (among other things) already in the 18th century. Prim's algorithm (Algorithm 1.11) dates back to 1957 [9] and Dijkstra invented his famous algorithm (Algorithm 1.16) in 1959 [2]. The algorithm by Dijkstra can also be used to compute the shortest paths between *all* pairs of nodes, i.e., it solves the all pairs shortest path (APSP) problem, instead of just a fixed pair. Not much later than Dijkstra, the Floyd-Warshall algorithm [3, 12], that can also deal with negative edge weights, was invented. This algorithm, however, has a slightly worse running time of $\mathcal{O}(n^3)$.

Due to the vastness of the modern internet, there are various textbooks covering details of computer networks [4, 11]. More details of the IPv4 [8] and IPv6 [1] can be found from the RFC specifications and similarly for the RIP protocol [5], OSPF protocol [7], the BGP protocol [10] and the Domain Name System [6].

This chapter was written in collaboration with Jara Uitto.

Bibliography

[1] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). Technical report, RFC Editor, 1998.

[2] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[3] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[4] Jim Kurose and Keith Ross. *Computer Networking: A Top-down Approach Featuring the Internet.* Addison Wesley, 2005.

[5] G. Malkin. RIP Version 2. Technical report, RFC Editor, 1994.

[6] P. Mockapetris. Domain Names - Implementation and Specification. Technical report, RFC Editor, 1987.

[7] J. Moy. OSPF Version 2. Technical report, RFC Editor, 1991.

[8] Jon Postel. Internet Protocol. Technical report, RFC Editor, 1981.

[9] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[10] Y. Rekhter, T. Li., and S. Hares. A Border Gateway Protocol 4 (BGP-4). Technical report, RFC Editor, 2006.

[11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks.* Prentice Hall, 2011.

[12] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

Chapter 2

Transport Layer

How does the internet decide at what quality you can watch a video?

2.1 Flows

Since data flows are inherently directed, we will only consider (weighted) directed graphs in this chapter. Moreover, in contrast to Chapter 1, the weights will not indicate the latency of edges, but the bandwidth capacity.

Definition 2.1 (Flow, Rate)**.** *Let s, t be two nodes. A **flow** from **source** s to **destination** t (also called an s-t-**flow**) is a function $F : E \to \mathbb{R}_{\geq 0}$ such that the following hold:*

$F(e) \leq c(e)$	*for all $e \in E$*	*(capacity constraints)*
$\sum_{e \in \textit{in}(v)} F(e) = \sum_{e \in \textit{out}(v)} F(e)$	*for all $v \in V \setminus \{s, t\}$*	*(flow conservation)*

*We call $F(e)$ the **rate** of F on edge e and the net flow leaving s ($\sum_{e \in \textit{out}(s)} F(e) - \sum_{e \in \textit{in}(s)} F(e)$) the **rate** of F, also denoted by F.*

Remarks:

- By `in`(v) resp. `out`(v) we denote the set of all incoming resp. outgoing edges at node v.
- You may wonder what happens if there is not only one flow in the graph, but if there are multiple source-destination pairs. Welcome to the world of multi-commodity flows!

Definition 2.2 (Multi-Commodity Flow)**.** *A **multi-commodity flow** $\mathcal{F} = (F_1, ..., F_k)$ is a collection of s_i-t_i-flows F_i such that for each edge $e \in E$ the sum of the flows' rates on e does not exceed the capacity of e, i.e.,*

$$\sum_{i=1}^{k} F_i(e) \leq c(e) \qquad \textit{for all } e \in E.$$

Remarks:

- A commodity is simply a source-destination (or sender-receiver) pair.
- As a multi-commodity flow consists of single-commodity flows, all F_i must satisfy flow conservation. Note that the additional condition regarding the sum of the flows on an edge already implies that the capacity constraints are satisfied.
- Can we transfer single-commodity flow techniques and results directly to the multi-commodity world? A first hint that things get a bit more difficult is given by the max-flow min-cut theorem: It turns out that for multi-commodity flows, the size of the maximum flow does no longer equal the size of the minimum cut in general.
- What about augmenting paths, as used in the famous Ford-Fulkerson algorithm? If we are given a graph with a multi-commodity flow, can we use augmenting paths in order to increase the flow for some commodity (s_i, t_i)? Figure 2.3 shows that augmenting paths and multi-commodity flows do not go well together. What can we do instead? A technique that solves many different multi-commodity flow problems is linear programming.

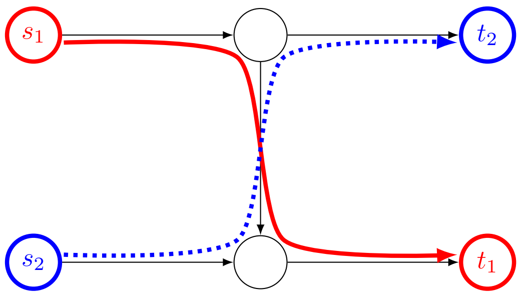


Figure 2.3: Given the depicted graph with a flow from s_1 to t_1, there is an augmenting path from s_2 to t_2 in the corresponding residual graph. If we now add a flow to the graph according to the augmenting path, then the flows starting in s_1 and s_2 will end up at the wrong destinations!

2.2 Linear Programming

Linear programming is a tool that is applicable for a wide range of optimization problems. In an optimization problem, one wants to maximize (or minimize) some function under certain restrictions, e.g., maximize the value of the term xy given the restriction $x + y \leq 5$. In order to be suitable for being solved by linear programming, the restricting inequalities and the function have to be linear (hence, the name).

Remarks:

- Let's have a look at an example of a linear program. Imagine you want to throw a party. How much booze should you buy? You can buy beer for a liter price of 1, and self-made cocktails where the ingredients for a liter will cost you 3. Your fridge has a capacity of 30 liters, but for each liter of cocktail you only need half a liter of fridge space. You figure that 50 liters in total should be enough for your friends. Here's the linear program for your problem:

Minimize $f(\mathbf{x}) = x_1 + 3x_2$
subject to

1. $x_1 + x_2 \geq 50$
2. $x_1 + \frac{1}{2}x_2 \leq 30$
3. $x_1 \geq 0$
4. $x_2 \geq 0$

Figure 2.4: Linear program for throwing a party

Remarks:

- How is a linear program defined in general?

Definition 2.5 (Linear Program, LP)**.** *A* ***linear program (LP)*** *consists of a set of m inequalities*

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & \leq & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \leq & b_m \end{array}$$

and a linear function

$$f(\mathbf{x}) = c_1x_1 + c_2x_2 + \cdots + c_nx_n \ .$$

The a_{ji}, b_i and c_i are given real-valued parameters and a vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ is a solution to the linear program if $x_i \geq 0$ for all $1 \leq i \leq n$ and $\mathbf{x}$ maximizes $f(\mathbf{x})$.

Remarks:

- If a linear program is specified as in the above definition, then we say that it is given in *canonical form.* There is also a short hand notation

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$$

  where A is the matrix with entries a_{ji} and $\mathbf{b}$ and $\mathbf{c}$ the vectors given by the b_i and c_i, respectively.

- In general, if you have the problem of maximizing or minimizing a linear function under constraints that are linear (in)equalities, there is a way to formulate it in canonical form. For instance, a constraint of the form $a_1x_1 = b_1$ can be rewritten as a combination of $a_1x_1 \leq b_1$ and $a_1x_1 \geq b_1$ which itself can be rewritten as $-a_1x_1 \leq -b_1$. Also, minimizing a linear function with coefficients $c_1, \ldots, c_n$ is the same as maximizing a linear function with coefficients $-c_1, \ldots, -c_n$.

- Now we know how to transform a linear problem into a linear program, but how do we solve LPs? Geometrically, an LP basically corresponds to an n-dimensional convex polytope and the hyperplanes bounding the polytope are given by the restricting inequalities. In order to solve an LP, one has to find a point on the polytope that maximizes our objective function $f(\mathbf{x})$. It is known that there is always a vertex of the polytope where the maximum is attained. One popular method for finding such a vertex and thus solving the LP is the simplex algorithm.

Algorithm 2.6 Simplex Algorithm

```
1: choose a vertex $\mathbf{x}$ of the polytope
2: while there is a neighboring vertex $\mathbf{y}$ such that $f(\mathbf{y}) > f(\mathbf{x})$ do
3:    $\mathbf{x} := \mathbf{y}$
4: end while
5: return $\mathbf{x}$
```

Remarks:

- There are other methods for solving LPs, such as interior point methods, where a solution is approached through the interior of the polytope. While the simplex algorithm performs well in practice, there are instances where its runtime is not polynomial in n. For some interior point methods it has been proved that the runtime is polynomial.

- In our party example, the solution of the LP uses fractional amounts of beer and cocktail ingredients. Sometimes fractional solutions are not possible and we need an integer solution. Solving integer linear programs is usually NP-hard.

- LPs can solve flow problems. For simplicity, we only present the LP for maximizing a single-commodity s-t-flow. The multi-commodity case is similar, with the number of inequalities growing roughly linearly with the number of commodities.

Maximize $f(\mathbf{x}) = \sum_{e \in \texttt{out}(s)} x_e$
subject to

1. $x_e \geq 0$ for all $e \in E$
2. $x_e \leq c(e)$ for all $e \in E$
3. $\sum_{e \in \texttt{in}(v)} x_e = \sum_{e \in \texttt{out}(v)} x_e$ for all $v \in V \setminus \{s, t\}$
4. $\sum_{e \in \texttt{in}(s)} x_e = 0$

Figure 2.7: LP for maximizing a single-commodity s-t-flow

Remarks:

- For each edge e, x_e is a variable indicating the amount of flow on e. As our goal is to find a maximum s-t-flow, we want to maximize the function $f(\mathbf{x})$ describing the amount of flow exiting s. The first constraint ensures that the amount of flow is non-negative on each edge, and the second guarantees that no edge capacities are violated. The third enforces flow conservation. The fourth is required because we do not want any part of the flow leaving s to return to s.

- So far, a flow was allowed to split up at vertices, resulting in a branched flow. In practice, we often want each flow to follow just a path.

Definition 2.8 (Unsplittable Flow)**.** *An s-t-flow F is called* ***unsplittable*** *if the edges $e \in E$ with $F(e) > 0$ form a path from s to t. If we do not impose this path restriction on a flow, it is called* ***splittable****.*

Remarks:

- The notion of an unsplittable flow also extends to multi-commodity flows. Unfortunately, we cannot use a simple LP for maximizing an unsplittable multi-commodity flow, as the additional constraint cannot be expressed by linear inequalities.

- Maximizing an unsplittable multi-commodity flow is NP-hard, but various algorithms solve the problem approximately.

2.3 Fairness

Definition 2.9. *The* ***demand*** *$d_i \in \mathbb{R}_{\geq 0}$ of a flow F_i is the rate at which F_i wants to transmit. The actual flow rate is always at most as large as the demand, i.e., $F_i \leq d_i$.*

Remarks:

- Due to the capacity restrictions in our network and the presence of other flows, the rate of a flow may be considerably smaller than its demand.
- For convenience we will assume in the following that all considered flows are unsplittable and that, for each flow, we are given a designated path this flow will follow.
- A fundamental problem of managing data flows in a network is how to allocate the bandwidth of a link whose capacity is not sufficient for simultaneously accomodating all flows (at full demand) which are to be routed along this link. On one hand, it may seem reasonable to allocate the available resources in a way that throughput is maximized. On the other hand, if throughput is maximized, some flows may starve. A certain fairness is desirable.

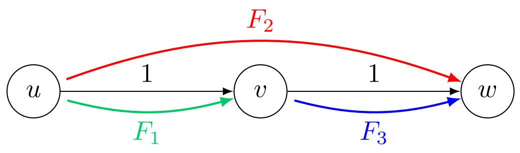


Figure 2.10: We have three flows, all with demand 1.

Remarks:

- What is a fair bandwidth allocation in Figure 2.10? Throughput is maximized if flow F_2 is ignored and F_1 and F_3 are allocated a bandwidth of 1. A fairer allocation that still takes the throughput into account is to allocate a bandwidth of $2/3$ to F_1 and F_3 each and of $1/3$ to F_2. There is an argument for allocating F_2 only half of the bandwidth of F_1 and F_3 since it uses twice as many edges. If we ignore throughput completely, then allocating a bandwidth of $1/2$ to each flow is simple and fair. How can we formalize this intuitive concept of fairness?

Definition 2.11 (Max-Min-Fairness). *A bandwidth allocation is called **max-min-fair** if increasing the allocation of a flow would necessarily decrease the allocation of a smaller or equal-sized flow.*

Remarks:

- There is only one max-min-fair allocation for a given set of flows in a network. It can be found by Algorithm 2.12.

Algorithm 2.12 Max-Min-Fair Allocation

```
1: Given a graph $G$, a set $\mathcal{F} = \{F_1, \ldots, F_k\}$ of flows with initial rate 0 on all
   edges, paths $p_1, \ldots, p_k$ along which the respective flows are to be routed and
   demands $d_1, \ldots, d_k$
2: while $\mathcal{F} \neq \emptyset$ do
3:   repeat
4:     increase rate of all flows in $\mathcal{F}$ evenly, but at most up to the respective
       demands
5:   until there is an edge $e \in E$ such that $\sum_{i:e \in p_i} F_i = c(e)$
6:   for all such edges $e$ do
7:     for all $i$ such that $e \in p_i$ do
8:       $\mathcal{F} := \mathcal{F} \setminus \{F_i\}$
9:     end for
10:    $E := E \setminus \{e\}$
11:  end for
12: end while
```

Remarks:

- Small networks indeed adopt centralized approaches for finding good allocations, e.g., using *Software Defined Networking* (*SDN*) or *Multiprotocol Label Switching* (*MPLS*). However, for large networks with quickly changing data flows, such as the internet, calculating and maintaining a good allocation in a centralized way is difficult. Even more so, who should do it?! There is no central authority for bandwidth allocation. We need a distributed way of avoiding congestion. How can we achieve this?

- One such congestion control mechanism commonly used is the AIMD algorithm, where AIMD stands for additive increase/multiplicative decrease. When using AIMD, the rate of any flow continuously changes as follows: As long as no congestion is reported, each flow repeatedly increases its rate additively. When congestion occurs on some edge, the affected flows decrease their rates by a multiplicative factor. The function describing the rate of a flow thus roughly follows a sawtooth behavior.

- To be precise, congestion occurs in a node (router), and not on an edge. When the router's buffers are full while data packets come in, those packets are dropped and packet loss occurs. Such a packet loss is used as indicator that the affected flow has to perform a multiplicative decrease.

- If the bandwidth allocation is performed according to AIMD, then it roughly converges to a max-min-fair allocation (roughly because according to AIMD the allocation never reaches a stable state). Consider what happens if a router used by two flows becomes congested: If both flows drop packets, then their rates are multiplied by the same factor, e.g. 1/2, and the absolute difference between the two rates decreases. The subsequent additive increase does not change this difference and when the next congestion occurs on this link, the rates converge again.

- It is possible that only one flow drops a packet during congestion, but this only improves the convergence rate as the probability of packet loss is larger for the larger flow.
- AIMD is used for congestion avoidance in an omnipresent distributed transport protocol called TCP.

2.4 UDP

As multiple applications running on the same computer want to use a network at the same time, it is necessary to distinguish between those applications (and their respective data flows). This distinction is provided by *ports*.

Definition 2.13 (Port)**.** *A* ***port*** *is a numeric identifier used in transport protocols to identify which application sent the packet and which application should receive it on the destination computer.*

Definition 2.14 (Client-Server Model)**.** *In the client-server model, the sender is called* ***client*** *and the receiver* ***server****. The client is regarded as a consumer of the services offered by the server.*

Remarks:

- When communicating with a server, a client transmits its port so that the server knows where to reply, if needed.
- There exists a multitude of protocols used when communicating between applications, with various tradeoffs in terms of latency, security and consistency. The most common ones are UDP and TCP.

Protocol 2.15 (UDP)**.** *The* ***user datagram protocol (UDP)*** *is a no-frills transport protocol that allows an application to send packets from client to server.*

Remarks:

- In Chapter 1 you learned that IP packets consist of header and payload. In the transport layer (when using UDP) the IP payload is divided further into the UDP header and the actual data.
- In the UDP header, the source and destination ports are specified along with a checksum and a packet length.
- UDP does not include any protection against packet loss.
- Furthermore, UDP does not guarantee any order on the delivery of packets.

- Dealing with these issues is delegated to the client application. However, UDP also has very little overhead in terms of packet size and latency, hence it is commonly used in scenarios where the application requires low overhead, e.g., real-time applications.

2.5 TCP

Definition 2.16 (Connection)**.** *A **connection** is a bidirectional long-term relationship established between a client and a server in order to transmit data reliably.*

Protocol 2.17 (TCP)**.** *The **transmission control protocol (TCP)** is a connection-oriented transport protocol guaranteeing that lost packets are being retransmitted and that packets are delivered in the same order they are sent.*

Remarks:

- Like UDP, TCP introduces the notion of ports to address a specific application on a computer. In addition to UDP, the header also includes a sequence number, an acknowledgement number, a window size, and a number of binary flags.
- In TCP, the partitioning of data into packets is abstracted into a continuous data stream from sender to receiver. For applications exchanging data it is not visible where the actual packets begin and end.
- In the literature, the TCP packets are also called *segments*.
- While UDP simply sends packets, TCP establishes a connection between source and destination before starting to send packets containing the actual data to be transmitted.

Definition 2.18 (Acknowledgement)**.** *An **acknowledgement (ACK)** is the confirmation that a sent packet has actually been received. The ACK is sent from the receiver of the packet to the sender.*

Remarks:

- In TCP, each data byte is specified by a sequence number. The sequence number of a packet is the number of the first data byte in the packet. Upon receiving a packet, the receiver sends back a packet where the acknowledgement number is set to the number of the last data byte of the received packet plus 1, i.e., the sequence number of the first byte of the packet it expects to receive next. By sending this acknowledgement packet, the receiver confirms to have received all data up to the specified byte. The acknowledgement packet may be void of any actual data.
- In addition, TCP often also supports non-cumulative acknowledgements known as selective ACKs (SACKs).

Protocol 2.19 (Establishing a Connection)**.**

- *The client sends a SYN (synchronize) packet to the server.*
- *The server acknowledges the packet by sending back a SYN/ACK packet.*
- *The client acknowledges the reception of the SYN/ACK packet by sending an ACK packet itself.*

Remarks:

- Terminating a connection can be done by a similar process where the SYN packets are replaced by FIN packets.
- A packet is specified as a SYN, FIN, or ACK packet by setting the respective binary flag in the header.
- The sequence number x of the first SYN packet is not simply set to 0 (for security reasons), but to some arbitrary number. Based on this number the subsequent data is numbered (bytewise). The rules explained above for the used sequence and acknowledgement numbers also apply for establishing the connection. Consequently, the server's SYN/ACK packet has acknowledgement number $x + 1$ and the client's ACK packet, containing also the first actual data, has sequence number $x + 1$.

Definition 2.20 (Flow Control)**.** *Avoiding congestion on the recipient's side which occurs, e.g., because the recipient is a device processing relatively slowly, is called* ***flow control****.*

Remarks:

- For flow control, the server uses the window size field in the header to specify how many packets it can receive before its buffer is full. The client accordingly adjustes its rate so that no more packets are in flight than specified by the window size.

Definition 2.21 (Round-Trip Time)**.** *The time it takes a packet to travel from sender to receiver and back is called* ***round-trip time (RTT)****.*

Definition 2.22 (Congestion Control)**.** *Avoiding congestion on a link (or, more precisely, in the router transmitting over the link) in the network is called* ***congestion control****.*

Remarks:

- Congestion control is exercised by implementing a congestion window. The actual window size used for determining the rate of a flow is the minimum of the size of the congestion window and the window size specified in the header of packets received by the client.
- Basically, the congestion window implements AIMD. Congestion in the network causes packet loss which is then reported to the affected sender, who decreases the congestion window by a factor of 1/2. Afterwards, the congestion window is increased by (the maximum size of) one packet per RTT, resulting in a linear increase.

- In TCP, recognition of dropped packets on the sender side is implemented by *timeouts*, i.e., if a packet is not acknowledged in some time frame it is considered as lost. Thus, some time elapses between a congested router dropping a packet and the affected flow decreasing its rate which in turn causes other flows to suffer packet loss in the congested router since the congestion is not remedied immediately.

- How long a sender should wait for the acknowledgement of a sent packet depends on the RTT. For determining the waiting time, a variable called *smoothed RTT* (*SRTT*), set initially to the RTT of the first acknowledged packet, is used. The new SRTT is the weighted ("smoothed") mean of the last SRTT and the RTT of the last acknowledged packet.

- Over time, various heuristics have been incorporated into TCP to improve performance, e.g., the slow-start algorithm which governs the initial growth of the size of the congestion window. According to slow-start, whenever a packet is acknowledged, the window size is increased by one packet. Thus, the initially small window grows exponentially in size until a certain threshold is reached upon which the additive increase part of AIMD starts. Thereby, the time where the network is not used close to full capacity is reduced.

- TCP relies on the goodwill of the senders as this is where the adjustment of the flow rates takes place. You may tweak your local version of the TCP protocol in order to obtain more bandwidth for yourself, e.g., by simply ignoring the multiplicative decrease.

Chapter Notes

As Leighton and Rao show in [4], for multi-commodity flows, the size of the maximum flow does not equal the size of the minimum cut in general. The NP-hardness of maximizing an unsplittable multi-commodity flow can be inferred from [1].

Two of the first researchers who formulated applied problems from logistics/economics as linear programs were Kantorovich and Koopmans who later received the Nobel Prize in economics for their contributions. The simplex algorithm was developed by Dantzig in 1947. In 1979, Khachiyan showed that linear programs can be solved in polynomial time. In 1984, Karmarkar developed an interior point algorithm that not only had a polynomial-time runtime, but was also practically feasible.

TCP was developed by Cerf and Kahn, based on their work [2]. Analysis of the AIMD algorithm can be found in [3].

This chapter was written in collaboration with Sebastian Brandt.

Bibliography

[1] Georg Baier, Ekkehard Köhler, and Martin Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

[2] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974.

[3] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks*, 17:1–14, 1989.

[4] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999.

Chapter 3

Application Layer

If you ever write internet code, almost surely it will be application layer code.

3.1 HTTP

Today a large majority of all data is transferred using the HTTP protocol, with a share of over 80%. HTTP is used by websites such as Facebook, Wikipedia, or Google, but also by popular content streaming services such as Netflix or Youtube.

Definition 3.1 (URL). *A Uniform Resource Locator (URL) is a string that uniquely identifies a resource on a server. In its most common form it consists of a* ***scheme*** *detailing the protocol to use, the* ***host identifier*** *on which the resource is located and a* ***path*** *detailing the location on the server.*

Remarks:

- The scheme in this section will either be *http* for unencrypted HTTP or *https* for HTTP over an encrypted connection. If the scheme is http, then by default the client will contact the server on port 80. For https the client uses port 443.
- HTTPS and HTTP only differ in the underlying transport protocol. While HTTP directly communicates over a TCP connection, HTTPS further encrypts its traffic by inserting an encryption layer between HTTP and the TCP connection. Please refer to Section 3.6 for more details on layers and Chapter 12 for the encryption. for details about encryption.
- The host identifier may either be the server's IP address or a domain name, which would then be translated to the IP address using a DNS lookup. Refer to Section 3.3 for details about DNS.
- There are a number of optional fields that may also be part of a URL. These are *username* and *password* for simple authentication, a *port* if a non-default port is used, a *query*-string to pass arguments to a resource, and a *fragment* describing which part of the resource is of interest: `scheme://user:password@host:port/path?query#fragment`.

Protocol 3.2 (HTTP). *The Hypertext Transport Protocol (HTTP) is a network protocol built on top of TCP, used by a client to interact (retrieve, store, etc.) with resources on a server.*

Remarks:

- HTTP has simple request-response semantics, i.e., the client issues a request and the server responds. The requests and responses are in a human readable format.
- HTTP is a client-server protocol, in which the client requests an operation, called *method*, on the resource and the server returns a response. The most common method for HTTP requests is `GET` which is used to retrieve a resource from the server. Other methods include `HEAD`, used to retrieve metadata about a resource from the server, `POST` and `PUT`, commonly used to update an existing resource or create a new resource on the server, and `DELETE`, used to remove a resource from the server.
- Both request and response comprise a header and a payload separated by an empty line.
- The request header contains a request line, consisting of a method, a URL specifying which resource it should be applied to, and an HTTP version. In addition the header may include a number of request options, i.e., `Key: Value` pairs with additional information about the client and the request, each on a new line. The request header is terminated by two successive newline characters: `\n\n`.
- The `PUT` and `POST` methods include a *payload*, i.e., some data that is transferred along with the request. Semantically this corresponds to the content of the resource that the client is attempting to create or update. If a payload is included in the request, a `Content-Length` header option specifies the length of the payload in bytes, and the payload is appended after the two newline characters.
- The server response header consists of a response line containing a protocol version, a numeric return code and a human readable response code. Additionally it may include response options, in the same format as the request options. Options usually include the resource `Content-Type` and `Content-Length`. The response header is terminated by two successive newline characters and is followed by the payload, i.e., content, of the resource.
- The numeric return codes returned by the server are 3-digit numbers organized into groups according to their first digit:
  - *1xx: Informational* Not used, but reserved for future use
  - *2xx: Success* The action was successfully received, understood, and accepted, e.g., 200, indicating a successful request returning the requested resource.

  - *3xx: Redirection* Further action must be taken in order to complete the request
  - *4xx: Client Error* The request contains bad syntax or cannot be fulfilled, e.g., 404, indicating that a non-existing resource was requested.
  - *5xx: Server Error* The server failed to fulfill an apparently valid request

- A resource may be any content that can be serialized and returned in response to a request. This includes textual as well as media contents.

- Most resources on the internet are HTML pages and associated media resources, such as images and videos. A *browser* on the user's computer is used to render the HTML pages and associated media into a webpage and display it to the user. However, the pervasive deployment of HTTP for user content has resulted in HTTP becoming the primary transport mechanism for other resources as well. Standards such as Representational State Transfer (REST), XML-RPC, Webservices (SOAP), and JSON-RPC are nowadays used to enable application to application communication on top of HTTP.

- The server may respond to a request with a continuous stream of data. Conceptually streaming is no different from a download, the only difference is that the server does not announce the size of the returned response in the response options and it does not close the connection.

- HTTP is a stateless protocol.

Definition 3.3 (Stateless Protocol)**.** *A protocol is stateless if each request is treated independently. In other words, the communication consists of independent request-response pairs. A stateless protocol relieves the server from storing information about its clients.*

Remarks:

- Stateless protocols are simpler but limited in applicability. For example, authenticating a user is tricky in a stateless protocol.

- To allow stateful applications on top of HTTP, a `Cookies`-option has been introduced by browsers. This option can be used to send small amounts of data (session identifier, authentication token, . . .) from the client to the server with every request, allowing the server to recognize the user, storing state about the user.

Protocol 3.4 (HTTP 1.0)**.** *HTTP 1.0 is the first version of HTTP.*

```
GET /index.html HTTP/1.0       HTTP/1.0 200 OK
User-Agent: Mozilla/5.0        Content-Type: text/html; charset=UTF-8
Accept: text/html              Content-Length: 312

                               <html>...
```

Figure 3.5: Simple HTTP 1.0 interaction retrieving the *index.html* resource on the server. The client request on the left is terminated by an empty line. On the right side, the response header and content are sent back by the server.

Remarks:

- HTTP 1.0 was only intended to transfer hypertext and some embedded resources such as images.
- In HTTP 1.0 a client opens a new connection for each request, which is closed once the response is delivered. Due to the increasing number of resources required to render a webpage, this results in a large number of (slow start) TCP connections between client and server.

Protocol 3.6 (HTTP 1.1)**.** *HTTP 1.1 is the second released version of HTTP, which includes a number of incremental changes such as request pipelining, protocol upgrade capabilities, range request, better support for caching and compression.*

Remarks:

- Pipelining uses persistent connections that are reused for multiple requests before being closed. The client opens a connection once and issues multiple requests on that connection. A browser may still open multiple concurrent connections, but the overall number of connections is reduced, and the overhead of establishing the connection and the TCP slow start is amortized over multiple requests.
- If supported by the server, range requests allow the client to specify a byte-range that is to be returned in the request options. Without range requests it is not possible to resume an interrupted download.
- Protocol updates are used to switch to completely different protocols over the same connection. The server responds to the current request and then the protocol is switched to the protocol indicated in the request options.
- One example of a protocol upgrade allows are WebSockets. For security reasons, scripts running in the browser are not allowed to open TCP connections to servers. Protocol upgrades allow an existing HTTP connection to be used as a WebSocket, allowing bidirectional communication between a client script and the server, including server-side pushes of messages.

Protocol 3.7 (HTTP/2)**.** *HTTP/2 or HTTP 2.0 is the latest major release of HTTP. It includes a number of changes that are not backward compatible.*

Remarks:

- While in previous versions of HTTP headers are human readable, HTTP/2 uses a binary format to reduce the size of the request and the response, resulting in lower latency and a smaller protocol overhead.

- HTTP/2 introduces multiplexing of logical streams onto a single connection. Each request-response pair forms a logical stream. The streams are split into frames that may be interleaved arbitrarily and sent over a single shared connection. This allows a single connection between client and server to be used for an arbitrary number of concurrent requests, thus eliminating the need to open multiple connections to the same server to load resources concurrently. Multiplexing furthermore enables dynamic prioritization of requests and server-side pushes, e.g., a server could push additional resources for a webpage without waiting for the client to request them.

3.2 HTML

Definition 3.8 (HTML)**.** *Hypertext Markup Language (HTML) is a markup language used to annotate plain text with hyperlinks and rendering instructions, enabling the creation of* ***structured documents****.*

Definition 3.9 (Element)**.** *An HTML element is a part of a document delimited by a pair of opening and closing* ***tags****. Tags themselves are delimited by angle brackets (`<` and `>`).*

Remarks:

- Figure 3.2 displays a simple HTML document. HTML elements are used both as rendering directives, e.g., the above `<strong>` tag will result in bold text, and semantic markup, e.g., `<head>` delimits a region containing the metadata of the document.

- Elements can be nested arbitrarily, e.g., a bold text as part of an italic text. The resulting structure is a rooted tree of elements in which child elements span parts of a parent element. Notice that this implies that tags must be closed in the inverse order they have been opened.

- The content of a tag is defined as the text between the opening and closing tags. The content may also be empty, i.e., the opening tag is directly followed by a closing tag. Empty elements may use the `<tag/>` shorthand instead of `<tag></tag>`. An example of an empty tag is the linebreak `<br/>` which forces a line break and does not have any content itself.

- Elements may have attributes that specify the behavior of the element. For example a hyperlink has a *content*, i.e., the clickable text, and a *location* which the browser should visit when clicked. Such a link has the following markup: `<a href='location'>content</a>`

```
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p><strong>Hello</strong> world!</p>
  </body>
</html>
```

Figure 3.10: Simple HTML document with header and a formatted body.

- Most browsers have a basic interpretation of how an element is to be rendered. This behavior can be arbitrarily modified by applying *styles* to the elements.

Definition 3.11 (Cascading Style Sheets)**.** *Cascading style sheets (CSS) can be used to override the rendering defaults included in a browser.*

Remarks:

- A cascading style sheet consists of a set of *rules*. Each rule has a *selector*, which specifies which elements the rule should be applied to, and a *declaration*, specifying how the rendering should be changed from the browser's defaults.
- Simple selectors could for example match all elements with a given element, e.g., if the text in a `<strong>` element should also be italic.
- Selectors may make use of the tree structure of the document, e.g., a rule may match all `<strong>` elements whose parent is a `<p>` using `p > strong` as a selector.
- The declaration is a block of key-value pairs, each specifying one aspect of the rendering.
- The cascading style sheet is either inlined into the HTML document as a `<style>` element in the `<head>`, or it is stored in a separate file and bound to the document using a `<link>` element in the head.

```
p > strong {
  color: red;
}
```

Figure 3.12: A cascading style sheet with one rule matching all strong-elements that are direct children of a p-element. The declaration specifies that the text in all matched elements should be red.

3.3 DNS

When visiting a website, do you enter an IP address like 195.176.255.237 or do you prefer using a name like www.google.com? In the latter case you are using DNS.

Protocol 3.13 (DNS)**.** *The domain name system (DNS) maps human readable domain names to the IP addresses of servers which are serving requests for those domains.*

Remarks:

- Translating a domain name to IP addresses is called *resolving* the domain name.
- Besides mapping domain names to IP addresses, DNS also provides the inverse mapping from IP addresses to domain names.

Definition 3.14 (Nameserver)**.** *A server in the domain name system is called a nameserver. It offers the domain name resolution service to its clients.*

Remarks:

- DNS is a request-response based protocol. The client sends a request packet to the nameserver, which looks up the matching record and returns it as a response to the client. The protocol is not human readable and is composed of two single packets which hold all fields for both request and response.
- Despite being part of the core functionality of the internet, DNS is implemented in the application layer, as a protocol on top of UDP.
- DNS is implemented as a distributed database in order to guard against single points of failure, to distribute traffic over a large number of servers, and to enable local caching for faster domain name resolution.
- The distributed database stores *resource records* (RRs). Resource records may have a multitude of types. Common types are `A` and `AAAA` for domain name to IPv4 and IPv6 mappings respectively, `CNAME` for canonical names, `NS` to delegate the domain name to another nameserver, and `MX` to locate the mailserver responsible for the queried domain.
- There is no single server holding all the database's records, instead each nameserver only knows a subset of domain names.

Definition 3.15 (Authoritative Nameserver)**.** *A server that is responsible for a domain name is called the domain's authoritative nameserver. The authoritative nameserver is the server in which the mapping can be configured by the domain name owner. It is the server that should be contacted to ultimately resolve the domain name.*

Remarks:

- In order to resolve the domain name of a server we first need to locate the authoritative nameserver for that domain name. This chicken-and-egg problem is solved by introducing a hierarchy of nameservers that delegate the resolution of domain names until the authoritative nameserver is found.
- The hierarchy results in a tree structure with well known static nameservers at the root, which then delegate the resolution to their descendants.

Definition 3.16 (Root Nameserver). *A root nameserver is a nameserver that serves as the reliable point of contact when resolving a domain name. A list of root nameserver IPs is included with the operating system of all nameservers. Root nameservers are contacted if no better nameserver is known.*

Remarks:

- There are a total of thirteen root nameservers in the world, operated by a multitude of organizations. Having the root nameservers operated by a diverse group of organizations should ensure that the system is resilient against failures.
- Some root nameservers share the same IP address. IP anycast routing assures that the closest server is reached.
- Upon receiving a request, a nameserver checks whether it has the required information to respond to the request. Should the nameserver not have the necessary information, it will delegate the request to another nameserver, or drop the request if it does not know a better nameserver. The delegating server returns a `NS` response containing the next nameserver the client should contact.
- The first level below the root servers are the regional nameservers responsible for the top level domains (TLDs). Regional nameservers then delegate to a number of registrars which cater to the end users. Further levels are introduced to divide the load and the responsibility of a domain to a number of nameservers.
- If we were to always perform resolution by contacting the root nameservers, they would become bottlenecks. For this reason clients and nameservers cache the responses and return these if a matching request is received. To this end each response contains a time-to-live (TTL) field which specifies how long, in seconds, the response may be cached. The higher levels in the hierarchy specify a high TTL, since they have very few changes over time. Lower levels, where the majority of changes are performed specify lower TTL values.
- It is desirable for some nameservers to act as a local cache, e.g., an ISP nameserver may directly serve its customers to reduce latency and reduce the number of requests. Instead of delegating the request to another nameserver, these nameservers perform the resolution on

behalf of the client. This mechanism is called *recursive resolution* and allows the nameserver to cache the response and later serve similar requests locally.

3.4 Mail

Definition 3.17 (Mailserver)**.** *A mailserver is a server that routes mail messages from the sender to the recipient and stores incoming mail messages for its users.*

Remarks:

- A mail address is composed of a username and a domain, separated by an `@`. Mail destined to users of the same domain is delivered to the same mailserver. The mailserver that is responsible for handling a domain's mail is specified by an `MX` record returned by the nameserver of that domain. Mail on a mailserver awaiting to be retrieved by the user is said to be *spooled*.
- When sending a mail, the sending user will look up the responsible mailserver for the recipient's domain through DNS and contacts it to deliver the mail. The delivery from the user to the receiving mailserver is done using *SMTP*.
- Most modern clients may be configured to use a single mailserver for outgoing mail. When sending a mail the client contacts its outgoing mailserver instead of the destination mail server. The outgoing mailserver then delivers the mail on behalf of the client, allowing a number of advanced features such as sender authentication and automatically reattempting delivery should the destination mailserver be unreachable. In this scenario SMTP is used in both interactions, from the client to the outgoing mailserver and the outgoing mailserver to the destination mailserver.

Protocol 3.18 (SMTP)**.** *The simple mail transfer protocol (SMTP) is a protocol used to deliver mails from a client to a mailserver.*

Remarks:

- SMTP is an interactive human readable protocol over TCP connections. A mailserver listens to port 25 for unencrypted incoming connections, while port 465 is commonly used for encrypted incoming connections. Similar to HTTPS the encryption is implemented by adding an SSL layer inbetween the TCP layer and the SMTP layer. An interactive succession of requests and responses is called a *session*.
- The client issues requests in the form of text commands, while the server responds with a numerical response code and a textual response.
- Common commands include `HELO` to initiate a session, `RCPT TO` to specify the mail recipient, `MAIL FROM` to specify the mail sender and `DATA` to specify the mail content.

- With the exception of `DATA`, the commands are terminated by a newline. The `DATA` command is followed by the content of the mail, hence it allows multiple lines to follow, and is terminated by a `\n.\n`, i.e., a punctuation mark on an otherwise empty line.
- Similar to HTTP, the response codes are grouped into function groups:
  - *2xx: Success* The command issued by the client succeeded.
  - *3xx: Start mail input* In response to a `DATA` command, the client may send the mail body.
  - *4xx: Client error* The client issued an invalid command.
  - *5xx: Server error* The server failed to act on the command.
- Initially SMTP did not include any form of authentication, since it is solely used for mail delivery, i.e., it is not possible to retrieve someone else's mails over SMTP. The destination mailservers often still accept mails from unauthenticated sessions, however outgoing mailservers today require the user-agent to authenticate to reduce the risk of denial-of-service attacks and spam.
- Multipurpose Internet Mail Extensions (MIME) is a standard that describes how the content of a mail may be encoded. MIME enables HTML formatted messages, attachments, and avoids some problems with plaintext mails.

Protocol 3.19 (POP)**.** *The post office protocol (POP) is a protocol that enables a client to retrieve and manipulate spooled mail messages on a mailserver.*

Remarks:

- The first version of POP was introduced in 1984. Later versions introduced incremental changes. The latest version, POP3, includes an extension mechanism and a flexible authentication mechanism.
- POP3 is the retrieval counterpart of SMTP. It is therefore not surprising that the protocols are very similar with text based commands and text based responses. However, unlike SMTP, the responses do not contain a numeric response code, instead responses are prefixed with `+OK` if the command succeeded and failure causes the connection to be closed.
- Common commands include `USER` and `PASS` for authentication, `STAT` to retrieve overall statistics of the mailbox, `LIST` to retrieve a list of messages including the message ID and its length, `RETR` to retrieve a message and `DELE` to delete a message.
- Similar to the `DATA` command in SMTP, some responses may return multiple lines, in which case they are terminated using a punctuation mark on an otherwise empty line.
- Using SMTP and POP3, the mailserver is used for temporary storage of mails until they are retrieved. If a user uses multiple computers, mails retrieved on one client will not be synchronized with the other client.

- Modern alternatives to POP3 include the Internet Message Access Protocol (IMAP) and web-based mail clients.

Protocol 3.20 (IMAP). *The Internet Message Access Protocol (IMAP) is a protocol used by mail clients to access mail messages from a mailserver over a TCP connection. IMAP was designed with the goal of permitting complete management of a mailbox by multiple mail clients, therefore, clients generally leave messages on the server until the user explicitly deletes them.*

Remarks:

- An IMAP server typically listens on port number 143. IMAP over SSL (IMAPS) is assigned port number 993.
- In contrast to POP, IMAP focuses on organizing the mail directly on the mailserver as opposed to downloading them and organizing them locally. For this purpose IMAP introduces the concept of folders into which mail can be organized and retrieved from. Keeping the mail on the mailserver furthermore allows multiple synchronized clients.
- IMAP is a human readable protocol that follows a similar format as POP. IMAP prefixes the last line of a request-response pair with an alphanumeric unique *tag*. The tag is generated by the client and sent with the request, and the server will reply with the same tag in its response. This uniquely identifies the last line of a response and hence reduces the ambiguity of having some responses span multiple lines.

3.5 Socket API

Definition 3.21 (Socket). *A socket is the principal abstraction for network communication exposed in programming languages. A socket exposes the necessary information and methods to a user application, to exchange data between clients and servers.*

Remarks:

- A socket is endpoint of a connection between two applications. Data sent through a socket on one end is received by the socket on the other end.
- In order to establish a connection, the server must have a socket listening for incoming connections, and the client creates an outgoing socket connecting to the listening server socket.
- TCP and UDP are the most common transport layer protocols used when communicating between applications, however a multitude of other protocols exist with various tradeoffs in terms of latency, security and consistency.

```
 1 import java.io.*;
 2 import java.net.*;
 3
 4 public class HelloWorldClient {
 5     public static void main(String[] args) throws
           IOException {
 6         if (args.length != 3) {
 7             System.err.println(
 8                 "Usage:␣java␣HelloWorldClient␣<name>␣<host
                       >␣<port␣number>");
 9             System.exit(1);
10         }
11         String name = args[0];
12         String host = args[1];
13         int port = Integer.parseInt(args[2]);
14
15         Socket sock = new Socket(host, port);
16         PrintWriter out = new PrintWriter(
17             sock.getOutputStream(), true);
18         BufferedReader in = new BufferedReader(
19             new InputStreamReader(sock.getInputStream()));
20
21         out.print(name + "\n");
22         out.flush();
23         System.out.println(in.readLine());
24     }
25 }
```

Listing 3.22: Simple greeting client in Java.

Remarks:

- Our example client (Listing 3.22) takes three arguments: a name, a host and a port. The client connects to the specified host and port (line 15), sends the provided name (line 21), and reads the server's response (line 22). Both the request and response are terminated by a newline character, which allows the use of the readLine method to retrieve the contents.

- Java buffers most of its I/O operations. This includes network communication. After calling the print method of the PrintWriter on line 21, the written data is buffered as more data may be added. Calling flush instructs the underlying socket implementation to construct a TCP packet and send the data that was buffered so far. Manually flushing is not always necessary and happens automatically if the size of the buffered data exceeds the maximum packet size or a timeout is triggered. Splitting data into individual packets requires that the data is reassembled on the receiving side.

```java
import java.net.*;
import java.io.*;

public class HelloWorldServer {
    public static void main(String[] args) throws
        IOException {
        int portNumber = 31337;

        ServerSocket serverSocket = new ServerSocket(
            portNumber);
        Socket clientSocket = serverSocket.accept();

        PrintWriter out = new PrintWriter(
            clientSocket.getOutputStream(),
            true);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(
                clientSocket.getInputStream()));

        String name = in.readLine().trim();
        out.print("Hello␣" + name + "\n");
        out.close();
        in.close();
    }
}
```

Listing 3.23: Simple greeting server in Java.

Remarks:

- The server in Listing 3.23 listens for an incoming connection on port 31337. Once a connection is established, the server reads a line, i.e., the name of the client terminated by a newline character, responds with a greeting terminated by a newline, and closes the connection.
- Our program only accepts a single connection and then terminates. Real servers continually accept incoming connections and hand off the actual interaction to a thread.
- Notice that the server does not flush the buffered data, but the close method on the PrintWriter also causes the buffered data to be flushed.

3.6 Protocol Layers

The structure of network protocols (Chapter 1), transport protocols (Chapter 2) and application protocols (Chapter 3) is similar. Let's formalize this similarity by introducing a layered architecture for network communication.

Definition 3.24 (Internet Protocol Suite)**.** *The internet protocol suite is a computer networking model, and set of communications protocols, used in the internet as well as similar computer networks. It divides the responsibility of* *individual protocols into* ***layers*** *that expose an abstract interface to higher up protocols.*

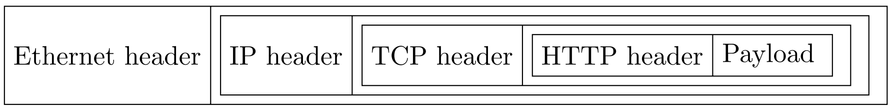


Figure 3.25: An example of nested layers for an HTTP request. Each layer has its own header and a payload consisting of the next higher layer.

Remarks:

- The internet protocol suite defines the following layers:
  - Application layer
  - Transport layer
  - Network layer
  - Link layer
- Layers are stacked so that higher layer protocols do not need to deal with the implementation details of lower layers, instead they may rely on the abstract interface exposed by those layers. Protocols in a layer should be designed in such a way that they are interchangeable. For example the format of an IP packet is independent of the physical medium it is transferred on, e.g., wire or air.
- The layering is implemented by repeatedly nesting packets of a layer in the next lower layer. Each layer is composed of a header and a payload. The payload simply contains the next higher layer packet. Figure 3.25 shows an example of how a link layer packet, Ethernet in this case, looks like.
- Over time the network has consolidated to just a few protocols that are in use, and the predominant network layer protocol is IP.

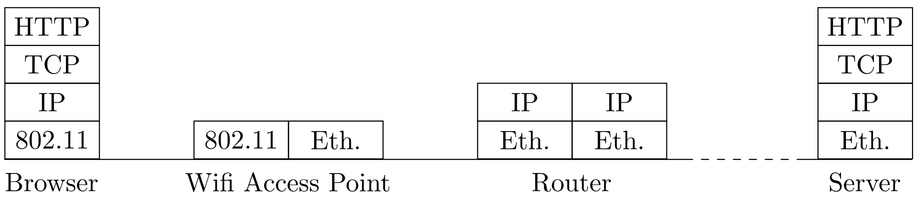


Figure 3.26: The browser issues an HTTP request over Wifi (802.11 link layer protocol) which is received by an access point and forwarded over Ethernet to a router. The router reads the IP header to identify the next hop and forwards the packet accordingly. Only the server needs to parse the entire stack of layers to get the original HTTP request.

Remarks:

- Separating the protocols into layers also allows keeping routing devices simple: a network switch does not need to understand the application protocol in order to correctly route a packet.
- Figure 3.26 shows an example of how packets can be routed without parsing the entirety of the packets.
- It may not always be possible to uniquely assign a protocol to a layer. Lower layer protocols may sometimes be tunnelled by higher layer protocols. In Section 1.7 we have seen that IP packets can be tunnelled through a VPN connection with IPsec by wrapping IP packets in IPsec packets. If we were to establish a TCP connection over IP tunnelled through IPsec, then the IP layer may be categorized as the transport layer or the network layer depending on whether we consider the point of view of the TCP connection or the IPsec connection.

Chapter Notes

Application layer protocols are specified in Requests for Comments (RFCs) which are the canonical specification of internet protocols. This allows a number of manufacturers to implement these protocols and guarantee compatibility.

Mail is one of the earliest applications, it even predates DNS. Mail relies on a number of protocols, some of which we have discussed in this chapter. SMTP was introduced in 1982 with RFC 821 [6], and initially required users to log into the nameserver where the mails were stored in a file structure to read and send mails. POP was introduced in 1984 with RFC [7], allowing users to retrieve mail on a mailserver and read it locally, while still using SMTP to send outgoing mails.

DNS was introduced in 1983 in RFC 883 [5]. It initially did not include any authentication of the responses, meaning that it was trivial for an intermediary nameserver to impersonate authoritative nameservers and redirect users to the wrong server. Domain Name System Security Extensions (DNSSEC) published in RFC 2065 [3] introduces authentication through digital signatures authenticating responses, and a public key infrastructure to authenticate the domain name owners.

HTTP 1.0 was released in 1996 in RFC 1945 [2] and was quickly superseded in 1997 by version 1.1 in RFC 2068 [4]. In 2009 Google announced a new transport protocol called speedy (SPDY) between its Chrome browser and its own services. After successful deployment of SPDY by Google it was picked up by the IETF and evolved into HTTP/2, which will eventually supersede both HTTP/1 and SPDY. HTTP/2 was finalized and published in 2015 as RFC 7540 [1].

This chapter was written in collaboration with Christian Decker.

Bibliography

[1] M. Belshe, R. Peon, and M. Thomson. RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2), 2015.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol – HTTP/1.0, 1996.

[3] D. Eastlake and C. Kaufman. RFC 2065: Domain Name System Security Extensions, 1997.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. RFC 2068: Hypertext Transfer Protocol – HTTP/1.1, 1997.

[5] P. Mockapetris. RFC 883: Domain Names - Implementation and Specification, 1983.

[6] Jonathan B. Postel. RFC 821: Simple Mail Transfer Protocol, 1982.

[7] J. K. Reynolds. RFC 918: Post Office Protocol, 1984.

Chapter 6

Dictionaries

You manage a library and want to be able to quickly tell whether you carry a given book or not. We need the capability to *insert*, *delete*, and *search* books.

Definition 6.1 (Dictionary)**.** *A **dictionary** is a data structure that manages a set of **objects**. Each object is uniquely identified by its **key**. The relevant operations are*

- *search: find an object with a given key*
- *insert: put an object into the set*
- *delete: remove an object from the set*

Remarks:

- There are alternative names for dictionary, e.g. key-value store, associative array, map, or just set.
- If the dictionary only offers `search`, it is called *static*; if it also offers `insert` and `delete`, it is *dynamic*.
- For our purposes, we will ignore that we actually have a set of objects, each of which is identified by a unique key, and just talk about the set of keys. With regard to the library example, books are globally uniquely identified by a key called `ISBN`. Whenever we say we insert/delete/search a key, we can just drag the key's object along via encapsulation.
- The classic data structure for dictionaries is a binary search tree.

6.1 Search Trees

Definition 6.2 (Binary search tree)**.** *A **binary search tree** is a rooted tree (Definition 1.7), where each node stores a key. Additionally, each node may have a pointer to a left and/or a right child node. For all nodes, if existing, the left child stores a smaller key, and the right stores a larger key.*

Algorithm 6.3 Search Tree Search

Input : key k, root r of search tree
Output: k if it is in the tree, else $\perp$

```
1: If $r$ contains key $k$, return $k$
2: If $k$ is smaller than the key of $r$, set $r$ to left child and recurse
3: If $k$ is larger than the key of $r$, set $r$ to right child and recurse
4: Return $\perp$
```

Remarks:

- The symbol $\perp$ ("bottom") signifies `null` or `undefined`.
- The cost of searching in a binary search tree is proportional to the *depth* of the key, which is the distance (Definition 1.15) between the node with the key and the root.
- There are search trees called *splay trees* that keep frequently searched keys close to the root. There may be keys with linear depth in a splay tree, but on average the cost of a search is logarithmic in the number of keys.
- Using *balanced search trees*, we can maintain a dictionary with worst-case logarithmic depth for all keys, and thus worst-case logarithmic cost per insert/delete/search operation.
- Is there a way to build a dictionary with less than logarithmic cost?

6.2 Hashing

Definition 6.4 (Universe, Key Set, Hash Table, Buckets)**.** *We consider a **universe** U containing all possible keys. We want to maintain a subset of this universe, the **key set** $N \subseteq U$ with $|N| =: n$, where $|N| \ll |U|$. We will use a **hash table** M, i.e. an array M with **buckets** $M[0], M[1], \dots, M[m-1]$.*

Remarks:

- The standard library of almost every widely used programming language provides hash tables, sometimes by another name. In C++, they are called `unordered_map`, in Python `dict`, in Java `HashMap`.
- The translation from virtual memory to physical memory uses a piece of hardware called *translation lookaside buffer* (TLB), which is a hardware implementation of a hash table. It has a fixed size and acts like a cache for frequently looked up virtual addresses.
- Compilers make use of hash tables to manage the symbol table.

Definition 6.5 (Hash Function)**.** *Given a universe U and a hash table M, a **hash function** is a function $h : U \to M$. Given some key $k \in U$, we call $h(k)$ the **hash** of k.*

Remarks:

- A hash function should be efficiently computable, e.g. $h(k) = k \bmod m$ for a key $k \in \mathbb{N}$.
- If we use `ISBN` mod m as our library hash function, can we insert/delete/search books in constant time?!
- But what if two keys k and l have $h(k) = h(l)$?

Definition 6.6 (Collision). *Given a hash function $h : U \to M$, two distinct keys $k, l \in U$ produce a **collision** if $h(k) = h(l)$.*

Remarks:

- There are competing objectives we want to optimize for with regard to the size of a hash table. On the one hand, we want to make the hash table small since we want to save memory. On the other hand, small tables will have more collisions. How many collisions will we get for a given n and m?

Theorem 6.7 (Birthday Problem). *If we throw a fair m-sided dice $n \leq m$ times, let D be the event that all throws show different numbers. Then D satisfies*

$$\Pr[D] \leq \exp\left(-\frac{n(n-1)}{2m}\right).$$

Proof. We have that

$$\begin{aligned}\Pr[D] &= \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-(n-1)}{m} = \prod_{i=0}^{n-1} \frac{m-i}{m} \\ &= \prod_{i=0}^{n-1} \left(1 - \frac{i}{m}\right) = \exp\left(\sum_{i=0}^{n-1} \ln\left(1 - \frac{i}{m}\right)\right)\end{aligned}$$

We can use $\ln(1+x) \leq x$ for all $x > -1$:

$$\Pr[D] = \exp\left(\sum_{i=0}^{n-1} \ln\left(1 - \frac{i}{m}\right)\right) \leq \exp\left(\sum_{i=0}^{n-1} -\frac{i}{m}\right) = \exp\left(-\frac{n(n-1)}{2m}\right)$$

□

Remarks:

- Theorem 6.7 is called the "birthday problem" since traditionally, people use birthdays for illustration: In order to have a chance of at least 50% that two people in a group share a birthday, we only need 23 people.
- If we insert more than roughly $n \approx \sqrt{m}$ keys into a hash table, the probability of a collision approaches 1 quickly. In other words, unless we are willing to use at least $m \approx n^2$ space for our hash table, we will need a good strategy for resolving collisions.

- Theorem 6.7 assumes totally random hash functions — for non-random distributions of hashes, we might have more collisions. In particular, if we fix a hash function, then we can always end up with a key set N that suffers from many collisions. E.g., if many books have an `ISBN` that ends in 000, then `ISBN` mod 1,000 is a terrible hash function.
- Maybe we can use modulo, but with a different m? In particular, we might apply a simple function to the `ISBN` first to introduce some randomness, then use a moderately large prime number for m since primes are less likely to cause collisions?
- However, for any hash function there are bad key sets.
- On the other hand, for every key set there are good hash functions! How do we efficiently pick a good hash function, i.e. one that is likely to distribute hashes evenly?

Definition 6.8 (Universal Hashing)**.** *Let $\mathcal{H} \subseteq \{h : U \to M\}$ be a family of hash functions from U to M. $\mathcal{H}$ is called* ***universal*** *if for any two distinct keys $k, l \in U$, if we choose $h \in \mathcal{H}$ uniformly, then the probability of a collision is $\Pr[h(k) = h(l)] = \frac{1}{m}$.*

Remarks:

- In other words: if we choose a hash function from a universal family, we can expect the hashes to be distributed well, regardless of the key set.
- We cannot just pick a random function from U to M because there are $|M|^{|U|}$ many, so we need $|U| \log |M|$ bits to encode such a random function. That is even more bits than keys in our huge universe U.

Theorem 6.9 (Universal Hashing)**.** *Let m be prime and $r \in \mathbb{N}$. For $U = [b]^{r+1}$ where $[b] = \{0, \ldots, b-1\}$ and $M = [m]$ with $b \leq m$ and $a = (a_0, \ldots, a_r) \in [m]^{r+1}$, define*

$$h_a(k_0, \ldots, k_r) = \sum_{i=0}^{r} a_i \cdot k_i \mod m.$$

Then $\mathcal{H} := \{h_a : a \in [m]^{r+1}\}$ is a universal family of hash functions.

Proof. Let $(k_0, \ldots, k_r) = k \neq l = (l_0, \ldots, l_r) \in U$. Since k and l are different, there must be a smallest index $0 \leq j \leq r$ such that $k_j \neq l_j$. For a given $a \in [m]^{r+1}$, consider

$$\begin{aligned} h_a(k) = h_a(l) \Leftrightarrow \quad & \sum_{i=0}^{r} a_i \cdot k_i \equiv \sum_{i=0}^{r} a_i \cdot l_i && \mod m \\ \Leftrightarrow \quad & a_j(k_j - l_j) \equiv \sum_{i \neq j} a_i \cdot (l_i - k_i) && \mod m \\ \Leftrightarrow \quad & a_j \equiv (k_j - l_j)^{-1} \cdot \sum_{i \neq j} a_i \cdot (l_i - k_i) && \mod m \end{aligned}$$

where $(k_j - l_j)^{-1}$ exists because $k_j - l_j \neq 0$. There are m^r choices for all a_i with $i \neq j$; for each of those choices, there is a unique a_j for the hashes to be equal since $[m]$ is a field, and in fields linear equations have unique solutions. Since there are a total of m^{r+1} choices for $(a_0, \ldots, a_r)$, this gives us a probability of $\frac{m^r}{m^{r+1}} = \frac{1}{m}$ for the hashes to be equal. $\square$

Remarks:

- Theorem 6.9 gives us a general method for constructing universal hash functions in an efficient manner. We simply choose a prime number m and uniformly at random some factors $a_0, \ldots, a_r$. Thus, we can represent our hash function as the tuple $(m, a_0, \ldots, a_r)$.
- In practice, hash tables perform really well, and if we detect that we had bad luck in choosing our hash function, we just choose a new one and rebuild our table with the new function — this is called *rehashing.*
- In Java, creating an `int` as the hash of an `Object` is the job of the JVM. In OpenJDK for example, the first time `hashCode()` is called on an `Object`, the JVM creates a random number as its hash and stores it with the `Object`.
- Hash functions are usually defined on classes, not by the hashing structures themselves. For classes in the Java standard library that have fields (e.g., `String`s have a `char[]` as a field), `hashCode()` is implemented such that the hash is derived from the fields that are considered when deciding whether one instance `equals()` another. This is called the contract between `hashcode()` and `equals()`: if two instances of the same class are equal, then they have to have the same hash. On the other hand, two objects with the same hash need not be equal.
- In Theorem 6.9 we assume that $U = [m]^{r+1}$. In applications, we often want to find hashes for keys that are not numbers, and keys of different "sizes", e.g. `String`s of different lengths.
- The Java standard library uses a fixed version of a weaker form of this type of hashing for `String`. Instead of choosing $(a_0, \ldots, a_r) \in [m]^{r+1}$, Java fixes a value $a_0 \in$ `int` and uses $(a_0^0, a_0^1, .., a_0^r)$ instead, where r is the number of characters in the `String`. In Java, $a_0 = 31$ was chosen since it produced comparatively few collisions on English language test data. Also, this hash function can be represented as a single value a_0, regardless of how long the strings we want to hash are, and it will also work to manage `String`s with different lengths in the same hash table.

6.3 Static Hashing

How can we state the tradeoff between space and collisions more precisely?

Definition 6.10 (Number of Collisions). *Given a hash function $h : U \to M$ and a key set $N \subseteq U$, define the number of collisions that h produces on N as*

$$C(h, N) := |\{\{k, l\} \subseteq N : k \neq l, h(k) = h(l)\}|.$$

Lemma 6.11 (Space vs. Collisions)**.** *If we uniformly sample a hash function from a universal family, given an upper bound b on the number of collisions, we only need to sample a constant number of times in expectation to find a hash function h_b that satisfies $C(h_b, N) < b$ in $m = \lceil \frac{n(n-1)}{b} \rceil$ space for a fixed key set N of size n.*

Proof. There are $\binom{n}{2}$ pairs of distinct keys in N, and each of those produces a collision with probability at most $1/m$ since h is chosen from a universal family. Together, using the linearity of expectation we get

$$\mathbb{E}[C(h, N)] \leq \binom{n}{2} \cdot \frac{1}{m} = \frac{n(n-1)}{2m}.$$

The Markov inequality states that for any random variable X that only takes on non-negative integer values, we have $\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$. Hence,

$$\Pr[C(h, N) \geq 2 \cdot \mathbb{E}[C(h, N)]] \leq \frac{1}{2}$$

and so

$$\Pr[C(h, N) < 2 \cdot \mathbb{E}[C(h, N)]] \geq \frac{1}{2}$$

If we choose m such that $2 \cdot \mathbb{E}[C(h, N)] \leq b$, then we only need to sample 2 hash functions in expectation. Solving for m, we get

$$2 \cdot \mathbb{E}[C(h, N)] \leq b \Leftrightarrow \frac{n(n-1)}{m} \leq b \Leftrightarrow \frac{n(n-1)}{b} \leq m.$$

□

Remarks:

- According to Lemma 6.11, if we want no collisions, we set $b = 1$ and choose $m = \lceil \frac{n(n-1)}{1} \rceil = n(n-1)$.
- Similarly, if we can tolerate n collisions, we find that a hash table of size $m = n - 1$ suffices.
- Algorithm 6.12 defines perfect static hashing, which applies the result of Lemma 6.11.

Algorithm 6.12 Perfect Static Hashing

```
Input    : fixed set of keys $N$
Output   : Primary hash table $M$ and secondary hash tables $M_i$
Function: $N_i := \{k \in N : h(k) = i\}$
Function: $n_i := |N_i|$
 1: $M$ := hash table with $n$ buckets
 2: repeat
 3:    $h$ := hash function $N \to M$
 4: until $C(h, N) < n$
 5: for $i \in M$ do
 6:    $M_i$ := hash table with $2\binom{n_i}{2} = n_i(n_i - 1)$ buckets
 7:    repeat
 8:       $h_i$ := hash function $N_i \to M_i$
 9:    until $C(h_i, N_i) < 1$
10: end for
11: return $(M, h, (M_i)_{i\in[m]}, (h_i)_{i\in[m]})$
```

Remarks:

- In a first stage (Lines 1 to 4), we find a hash function h with at most n collisions in linear space according to Lemma 6.11.
- In a second stage (Lines 5 to 10), we find a hash function h_i per bucket i without collisions by using an amount of space that is quadratic in the number of keys in the bucket n_i as per Lemma 6.11.

Theorem 6.13 (Perfect Static Hashing)**.** *When Algorithm 6.12 returns, the size of M and all M_i together is less than $3n$.*

Proof. Due to Line 1, the size of M is exactly n.

The number of collisions produced by the keys in bucket i is $\binom{n_i}{2}$ since any two of them produce one. We know that $2\binom{n_i}{2} = n_i(n_i - 1)$. As two keys in different buckets cannot produce a collision, we can sum the number of collisions per bucket over all buckets to get the number of all collisions, and so

$$\sum_{i=0}^{m-1} n_i(n_i - 1) = \sum_{i=0}^{m-1} 2\binom{n_i}{2} = 2\sum_{i=0}^{m-1}\binom{n_i}{2} = 2C(h, N) < 2n.$$

We used that $C(h, N) < n$ due to Line 4. Because of the choice of the size of M_i in Line 6, all buckets M_i together use less than $2n$ space. In total, M and all M_i together have a size of less than $n + 2n = 3n$. □

Remarks:

- We now have a hashing algorithm that can be built in linear space and expected linear time, and offers worst-case constant time search for a static set N.
- But what about a dynamic dictionary?

6.4 Collisions

Definition 6.14 (Hashing with Chaining)**.** *In* ***hashing with chaining****, every bucket $M[i]$ stores a pointer to a secondary data structure that manages all keys k with $h(k) = i$. Insertion, search, and deletion of k are all relegated to those data structures. In the simplest implementation, we can use linked lists.*

Remarks:

- Algorithm 6.12 is an instance of hashing with chaining with the M_i being the secondary data structures managing the buckets.
- The Java standard library uses hashing with chaining to resolve collisions.
- From Java 7 to Java 8, the standard library changed from `HashMap` always using a linked list for a bucket to using a linked list as long as the bucket contains less than a certain number of keys, and building a search tree once the bucket reaches that number.
- More concretely: `HashMap` applies its own hash function to the hash supplied by the keys (remember, each class defines `hashCode()`, either by overriding it or by inheriting it from `Object`) to determine each key's bucket. For the ordering within the trees, there are two possibilities: the class implements `Comparable` or it does not.
- If the class of the keys implements `Comparable`, then the natural ordering of the keys is used.
- If the keys are not `Comparable`, then the tree uses the values returned by `System.identityHashCode(Object x)` to order keys; this method returns the same value that the default implementation of `Object.hashCode()` returns. This means that if your class is not `Comparable` and does not override `hashCode()`, then `System.identityHashCode(Object x)` is equal for all keys within a given tree; this makes the trees degenerate to lists.

Definition 6.15 (Load Factor)**.** *The fraction $\frac{n}{m} =: \alpha$ is called the* ***load factor*** *of the hash table.*

Remarks:

- The performance of all three operations (insert/delete/search) depends on the load factor for all collision resolution strategies discussed in this section.
- Hashing with chaining allows for a load factor $\alpha > 1$ since the size of the table is the number of secondary data structures; performance deteriorates with growing α.
- If we use linked lists as secondary structures and use a hash function chosen from a universal family, the cost for an unsuccessful search is $1 + \alpha$ in expectation, while that for a successul search is roughly $1 + \frac{\alpha}{2}$ in expectation.

- If we use one of the strategies of this section and α grows too large, we should rehash with a bigger m in order to maintain expected constant time cost. In the Java standard library, if a hash table surpasses a load factor of 0.75, it is rehashed into a hash table with twice the size of the old one.

Definition 6.16 (Hashing with Probing)**.** *Algorithm 6.17 defines how to search for a key in* ***hashing with probing****. Line 5 is a* ***successful search****, and Lines 7 and 11 are the two cases of an* ***unsuccessful search****. We call the sequence* $(h_i(k) \mod m)_{i\geq 0}$ *the* ***probing sequence*** *of* k*, and each step of the iteration a* ***probe****.*

Algorithm 6.17 Hashing with Probing: Search

Input : key k to search for
Output : key k if found, else $\perp$
Function: parametrized hash function h_i

```
1:  $i := 0$
2:  while $i < m$ do
3:     $j := h_i(k) \mod m$
4:     if $M[j] = k$ then
5:        return $M[j]$
6:     else if $M[j] = \perp$ then
7:        return $\perp$
8:     end if
9:     $i := i + 1$
10: end while
11: return $\perp$
```

Remarks:

- To insert a key, we adapt Algorithm 6.17: with an unsuccessful search in Line 7 we insert in the empty bucket. Therefore, the cost of an insert is roughly the cost of an unsuccessful search. An unsuccessful search in Line 11 triggers a rehash.

- Table 6.18 describes three different types of hashing with probing, each together with the approximate time that a successful or unsuccessful search takes in expectation. More generally, linear probing uses some linear function $h_i(k) = h(k)+ci$ for some $c \neq 0$, and quadratic probing uses some quadratic function $h_i(k) = h(k) + ci + di^2$ with $d \neq 0$. As long as we guarantee that $h_i(k)$ is integer for all $i \in [m]$, the constants c and d can be rational.

Type	$h_i(k)$	$\approx$ cost successful	$\approx$ cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$	$\frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha}\right)$

Table 6.18: Different types of hashing with probing together with the expected number of probes per search. α is the load factor of the table, and for hashing with probing, it has to satisfy $\alpha < 1$ since we cannot store more keys in the table than it has buckets. Each of h, h_1, h_2 is a hash function drawn from a universal family.

Remarks:

- The main reason for the differences in access times is *clustering*.
- Linear probing suffers from *primary clustering*: from some point on, the probing sequences of any two keys will become identical.
- Quadratic probing does not suffer from primary clustering, but it is subject to *secondary clustering*: if two keys have the same hash, then their probing sequences will still be identical.
- The form of quadratic probing defined in Table 6.18 has one additional issue: the probing sequence of a key does not necessarily cover the whole table. Assume the size of the table is $m = 7$ and $h(k) = 0$, then the probing sequence of k is $(0, 1, 4, 2, 2, 4, 1)$ — buckets $3, 5, 6$ do not appear.
- Double hashing does not suffer from either version of clustering. One can show that if the hash functions h_1 and h_2 used in double hashing are independently drawn from a universal family, then double hashing performs as well as an idealized hash function that assigns hashes uniformly at random.

6.5 Worst Case Guarantees

So far, the cost of all operations has been given in expected time cost. There are algorithms that allow us to do better and give us worst case guarantees on some of the operations. Two widely known possibilities to achieve this are called *dynamic perfect hashing* and *cuckoo hashing*.

Remarks:

- To adapt perfect static hashing to a dynamic setting where we can also handle inserts and deletions, all we have to do is choose the size of M_i twice as large as in Algorithm 6.12, and rehash appropriately: Whenever $C(h_i, N_i) > 0$ for some bucket i, we rehash that bucket until there are no collisions. Once some bucket reaches $n_i^2 \approx |M_i|$ due

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter `limit` specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \to M_1, h_2 : U \to M_2$; chosen independently and uniformly at random from universal families

```
1: if $M_1[h_1(k)] = k$ or $M_2[h_2(k)] = k$ then
2:   return
3: end if
4: $t := 1$
5: while $t \leq$ limit do
6:   swap $k$ with $M_1[h_1(k)]$
7:   if $k = \perp$ then
8:     return
9:   end if
10:  swap $k$ with $M_2[h_2(k)]$
11:  if $k = \perp$ then
12:    return
13:  end if
14:  $t := t + 1$
15: end while
16: rehash()
17: CuckooHashingInsert($k$, limit)
```

to insertions, we rehash the entire table. This leaves us with expected constant time insert and delete, and worst case constant time search. To keep the table linear-sized, we rehash everything after every m updates (inserts or deletes).

- Another option is *cuckoo hashing*, which is described in Algorithm 6.19. The idea behind cuckoo hashing is to use the *"power of two choices"*, which can be roughly described as: if you can choose between two resources and use the one that is less busy, you gain efficiency.
- The counter `limit` used in Algorithm 6.19 has to be chosen carefully to guarantee the expected insert cost is constant. Specifically, one can show that we get this guranatee if we choose `limit` $\approx \log m$.
- Search and delete only need to check $M_1[h_1(k)]$ and $M_2[h_2(k)]$ to figure out whether a given key k is in the table, and so those operations are worst case constant time.
- Cuckoo hashing gets its name from cuckoo birds: they lay their eggs into the nests of other birds, and once the cuckoo chicks hatch, they push the other eggs/chicks out of the nest.

Chapter Notes

Dictionaries based on search trees are useful for providing additional operations such as nearest neighbor queries or range queries, where we want to find all keys in a certain range. Binary search trees were first published by three independent groups in 1960 and 1962 (for references, see Knuth [9]). The first instance of a self-balancing search tree that guarantees logarithmic cost for insert/search/delete is the AVL-tree, named so after its inventors Adelson-Velski and Landis [1]. For multidimensional keys, e.g. geometric data or images, there are specialized tree structures such as kd-trees [2] or BK-trees [3].

Hashing has a long history and was initially used and validated based on empirical results. One of the first publications was Peterson's 1957 article [11] where he defined an idealized version of probing and empirically analyzed linear probing. Universal hashing was introduced two decades later by Carter and Wegman in 1979 [4]. Perfect static hashing was invented in 1984 by Fredman et al. [7] and is sometimes also referred to as FKS hashing after its inventors. Its dynamization by Dietzfelbinger et al. took another decade until 1994 [6]. A comprehensive study on perfect hashing by Czech at all was compiled in 1997 [5]. Cuckoo hashing is a comparatively recent algorithm; it was introduced by Pagh and Rodler in 2001 [10].

There have been a number of other developments regarding hashing since the late 1970s; for an overview, see Knuth [9], in particular the section on History at the end of chapter 6.4. For a neat visualization of hashing with probing, see [8] online.

The power of two choices paradigm has found widespread application and analysis in load balancing scenarios. It was initially studied from the perspective of a balls-into-bins game where we want to minimize the maximum number of balls in any bin, and to do this we can pick two random bins and put the next ball into the least full of the two bins. Richa et al. [12] compiled an excellent survey on the earliest sources and numerous applications of this paradigm.

This chapter was written in collaboration with Georg Bachmeier.

Bibliography

[1] M Adelson-Velskii and Evgenii Mikhailovich Landis. An Algorithm for the Organization of Information. *Doklady Akademii Nauk USSR*, 146(2):263–266, 1962.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[3] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, 1973.

[4] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[5] Zbigniew J. Czech, George Havas, and Bohdan S. Majewski. Perfect hashing. *Theoretical Computer Science*, 182(1 - 2):1 – 143, 1997.

[6] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.

[7] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[8] David Galles. Closed hashing. `https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html`. Accessed: 2016-04-05.

[9] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[10] Rasmus Pagh and Flemming Friche Rodler. *Algorithms — ESA 2001: 9th Annual European Symposium Århus, Denmark, August 28–31, 2001 Proceedings*, chapter Cuckoo Hashing, pages 121–133. Springer Berlin Heidelberg, 2001.

[11] W. W. Peterson. Addressing for random-access storage. *IBM J. Res. Dev.*, 1(2):130–146, 1957.

[12] Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

Chapter 7

Databases

What is the movie with the largest cast? How many directors have directed more than ten movies? The internet movie database (`www.imdb.com`) contains the answer to such questions, but writing a new program that evaluates the data in a specific way for every such question is laborious. Relational databases can store large amounts of structured data and answer possibly complex questions about it.

7.1 Relational Databases

Definition 7.1 (Table, Row, Column, Database)**.** *A **table** consists of **rows**, so that each row (data record) contains the same **fields**, i.e., kinds of entries. When the rows of a table are written line by line, the fields form the **columns** of the table. Each column is referred to by a descriptive name, and is associated with the type of the respective field, e.g., integer, floating point, string, or a date. A **database** is a collection of tables.*

Remarks:

- In the database context, tables are also called *relations*, because the entries in each row are related to each other, namely by belonging to the same row.

movies

title	director	year
12 Angry Men	Sidney Lumet	1957
Raiders of the Lost Ark	Steven Spielberg	1981
War of the Worlds	Steven Spielberg	2005
Manos: The Hands of Fate	Harold P. Warren	1966
⋮		

Figure 7.2: A database containing a single table called "movies" storing the title, director, and year of release for each movie.

Remarks:

- Databases as we study them are accessed using the so-called *structured query language* (SQL). Thus they are referred to as SQL or *relational* databases.
- There are also databases for storing other data, e.g., key-value pairs (Chapter 6), graphs, or whole documents. Such databases are sometimes called *NoSQL* databases.
- Some people pronounce SQL letter-by-letter, while others prefer to say “sequel”, which stems from a predecessor with that name.
- MySQL and PostgreSQL are two popular open source SQL databases.
- MongoDB, CouchDB and Redis are popular open source NoSQL databases.
- Like http servers, SQL databases typically run as a daemon process on some server. Client applications connect to the server and authenticate themselves via username and password.
- Multiple users accessing the same database may result in concurrency issues. Some form of concurrency control is necessary!
- Other databases are tailored to single-user processing. They relieve developers from the burden of implementing efficient data structures for relational data. SQLite is one such example, and is used, e.g., in Firefox, Chrome, Android, Adobe Lightroom, and Windows 10.
- How can we store the data from Figure 7.2 using a SQL database?

7.2 SQL Basics

Remarks:

- All SQL statements end with a semicolon. The SQL language is case insensitive, but by convention keywords are often typed in upper case.
- The SQL specification is over 600 pages long. To add insult to injury there are lots of vendor specific “SQL dialects”, i.e., modifications and extensions.
- However, the basic set of commands for creating, manipulating, and querying tables are largely the same across database implementations. The same is true for the basic data types.

Definition 7.3 (SQL Data Types)**.** *SQL defines the following types of columns.*

- CHARACTER(m) *and* CHARACTER VARYING(m) *for fixed and variable length strings of (maximum) length m,*
- BIT(m) *and* BIT VARYING(m) *for fixed and variable length bit strings of (maximum) length m,*
- NUMERIC, DECIMAL, INTEGER*, and* SMALLINT *for fixed point and integer numbers,*

- FLOAT, REAL*, and* DOUBLE PRECISION *for floating point numbers,*
- DATE, TIME*, and* TIMESTAMP *for points in time, and lastly*
- INTERVAL *for ranges of time.*

Remarks:

- The range of each type includes the special value NULL. Note that NULL is different from the string 'NULL', the empty string, and from the number 0 (zero). NULL indicates that the row has *no value* for the corresponding field.
- For the CHARACTER VARYING type, some database systems support strings of arbitrary length.
- Many databases implement more types, e.g., geographic coordinates, IP addresses, geometric objects, or large integers.

Listing 7.4 Creating the database moviedb containing a table movies.

```
CREATE DATABASE moviedb;
USE moviedb;
CREATE TABLE movies (
    title CHARACTER VARYING(200) NOT NULL,
    director CHARACTER VARYING(200) DEFAULT 'Steven Spielberg',
    year INTEGER
);
```

CREATE DATABASE *database-name*;
Additional parameters allow to set database-specific options, e.g., user-based permissions, or default character sets for text strings. How a database is opened depends on the implementation. Listing 7.4 shows how to do it in MySQL.

CREATE TABLE *table-name* (*field-name type, field-name type, . . .*);
To enforce that all rows have a value for a particular field, one can add NOT NULL to the type when creating the table. Fields have a default value, which is NULL if not specified by adding DEFAULT *value* to the type description.

Remarks:

- There are also GUI and web-based client applications (that execute locally or on an http-server, respectively) and offer access to the database in a more intuitive manner than the classic command line tools. Examples for web-based interfaces are phpPgAdmin and phpMyAdmin for PostgreSQL and MySQL, respectively.
- Such tools are especially helpful for creating the databases and tables. They also feature importing data from various formats, e.g., CSV files, instead of using SQL statements to populate the tables.

Listing 7.5 Populating the movies table with data.

```
INSERT INTO movies
    (title, director, year) VALUES
    ('12 Angry Men', 'Sidney Lumet', 1957),
    ('Raiders of the Lost Ark', DEFAULT, 1981),
    ('War of the Worlds', DEFAULT, 2005),
    ⋮
    ('Manos: The Hand of Fate', 'Harold P. Warren', 1966)
;
```

INSERT INTO *table-name* (*field-name*, . . .) VALUES (*value*, . . .);
Values must be listed in the same order as the corresponding field names. When a field name (and thus its value) is omitted the field's default value is assumed. When the list of field names is omitted the field's values must be listed in the same order that was used when creating the table. To insert more than one row in one statement, multiple rows may be separated by a comma.

Listing 7.6 Querying the movies table.

```
SELECT * FROM movies;
SELECT * FROM movies WHERE director = 'Steven Spielberg';
SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;
SELECT * FROM movies WHERE title IS NULL OR director IS NULL;
SELECT title, director FROM movies WHERE title LIKE '%the%';
```

SELECT *field-name*, . . . FROM *table-name* WHERE *condition*;
Lists all specified fields of all rows in the table that fulfill the condition. The special field * lists all fields. The WHERE condition may be omitted to list the whole table. A condition can include comparisons ($<, >, =, <>$) between fields constants. The special value NULL can be tested with IS NULL. Conditions can be joined using parenthesis and logic operators like AND, OR, and NOT. Strings can be matched with patterns using ***field-name* LIKE *pattern***. In the pattern, an underscore (_) matches a single character, whereas % matches arbitrarily many.

Listing 7.7 Aggregation with SQL.

```
SELECT MIN(year) FROM movies;
SELECT AVG(year) FROM movies WHERE director='Sidney Lumet';
SELECT COUNT(*) FROM movies;
SELECT COUNT(DISTINCT director) FROM movies;
```

SELECT *aggregate*, . . . ;
Functions for aggregation include AVG to compute the average of a certain field, MIN and MAX for the minimum and maximum value, SUM for the sum of a field, and COUNT to count the number of occurrences. In an aggregation, the keyword DISTINCT indicates that only distinct values should be considered.

Remarks:

- Query 12 in Listing 7.7 returns the number of entries in the table, whereas query 13 returns the number of different movie directors.

Listing 7.8 Grouping and sorting.

```
14: SELECT director, COUNT(title) FROM movies GROUP BY director;
15: SELECT director, COUNT(title) FROM movies GROUP BY director
    HAVING COUNT(title)>10;
16: SELECT year, director, COUNT(title) FROM movies
      GROUP BY director, year
      ORDER BY year DESC, director ASC;
```

SELECT *field-name|aggregate*, ... GROUP BY *field-name,...*;
Aggregations may be partitioned using the group-by clause. Similar to before, the query result can only include aggregates and fields by which the result is partitioned.

Since WHERE clauses are applied before GROUP BY the result of aggregations cannot appear in them. When the result should be conditioned on the result of an aggregation, a HAVING clause can be used.

Remarks:

- Query 14 in Listing 7.8 reports how many movies of each director are in the database, and query 16 breaks the same down by year.

SELECT ... ORDER BY *field-name,...*;
After each field-name, the keyword ASC or DESC can be used to determine ascending or descending sorting order, respectively.

Listing 7.9 Updating and removing rows.

```
17: UPDATE movies SET title = 'Star Wars Episode IV: A New Hope'
      WHERE title = 'Star Wars';
18: DELETE FROM movies WHERE title = ' ';
```

UPDATE *table* SET *field-name = value*,... WHERE *condition*;
Updates the specified fields in all rows fulfilling the condition.

DELETE FROM *table-name* WHERE *condition*;
Removes all rows fulfilling the condition from the table.

7.3 Modeling

The way our example table from Figure 7.2 is designed results in lots of duplicate data—the director's name is stored anew for each row, and two directors with the same name cannot be distinguished. The situation worsens when we want to store the cast of each movie. Clearly the way we modeled our data can be improved. *Entity-Relationship* (ER) diagrams are a tool to find good representations for data.

Definition 7.10 (ER Diagram)**.** *Rectangles denote **entities** (tables), and diamonds with edges to entities indicate **relations** between those entities. On such an edge, the number* 1 *or the letter n denotes whether the corresponding entity takes part once or arbitrarily many times in the relation. Entities and relations can have **attributes** (columns) with a name, drawn as ellipses. Italicised attributes are **key attributes** which must be unique for each such entity.*

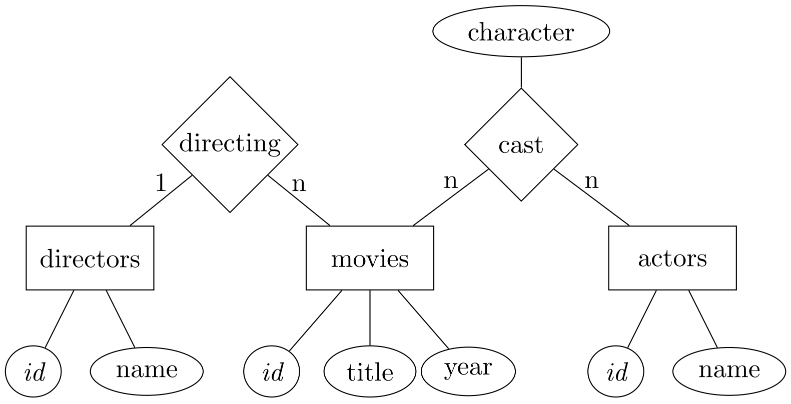


Figure 7.11: Model for a movie database. Movies and directors are in a 1-to-n relation: Each movie is directed by 1 director, and a director may work on many movies. Movies and actors are in a n-to-n relation, which has an additional attribute: An actor may appear in many movies, and each appearance is associated with a character in that movie, played by that actor.

Remarks:

- It is standard practice to assign a so-called *key attribute*, often named *id*, to every entity.
- What do ER diagrams have to do with SQL? Primarily, ER diagrams are for conceptually modeling the kind of data and relations one wishes to store. They can be translated into databases, but not in a unique way.
- A close relative of the ER diagram is the Unified Modeling Language (UML). UML is used to represent the tables of a database (or classes of object oriented software) accurately, with detailed information, e.g. fields.
- Each entity corresponds to a table with the corresponding attributes as columns. An n-to-n relation is represented by a table with columns for each attribute, and a column for the key attribute of each entity in the relation.

actor

id	name
1	Harrison Ford
2	Tom Cruise
⋮	⋮

cast

actor_id	character	movie_id
1	Indy	2
2	Ray Ferrier	3
	⋮	

Figure 7.12: The actor table and a table capturing the cast relation.

Remarks:

- The same scheme can be used for 1-to-1 and 1-to-n relations. However, one may also include the relation in the table storing the entity on the 1-side.

directors

id	name
1	Sidney Lumet
2	Steven Spielberg
3	Harold P. Warren
	⋮

movies

id	title	year	director_id
1	12 Angry Men	1957	1
2	Raiders of the Lost Ark	1981	2
3	War of the Worlds	2005	2
4	Manos: The Hands of Fate	1966	3
	⋮		

Figure 7.13: The movie and director tables using the new database layout. The director table simply maps ids to director names. Since the directing relationship is 1-to-n, it can be represented by adding a column to the movies table that stores the director for each movie.

Remarks:

- Similarly, a 1-to-1 relation can be turned into an attribute of one of the entities.
- Tables dedicated to capturing relations are often called *join* tables.

7.4 Joins

How can we access the data, which is now scattered across multiple tables?

Listing 7.14 A query that returns the table presented in Figure 7.13.

```
SELECT movie.title, director.name AS director, movie.year FROM movie
    INNER JOIN director ON movie.director_id = director.id;
```

SELECT . . .
FROM *left-table* INNER JOIN *right-table* ON *condition*;
Returns all rows that can be formed from a row in the left-table and a row in the right-table that satisfy the specified condition.

Remarks:

- In a query, one can create aliases for field and table names using the AS keyword, see Listing 7.14.
- A row in one of the tables that does not have a matching row (that satisfies the condition) in the other table will not appear in the result. For example, a director with id 5 would not appear, since there are no movies that reference that director_id.
- An INNER JOIN where the condition is TRUE returns the carthesian product of both tables. This special case can also be obtained with the CROSS JOIN clause.
- OUTER JOINs include also unmatched rows.

SELECT . . .
FROM *left-table* LEFT|RIGHT|FULL OUTER JOIN *right-table* ON *condition*;
Returns all rows from the inner join. In addition, a LEFT or RIGHT OUTER JOIN also returns all rows from the left or right table that have no matching row on the opposite table, respectively. The fields in unmatched rows that cannot be filled from the other table are filled with NULL values. A FULL OUTER JOIN returns both of the above.

Remarks:

- A RIGHT OUTER JOIN lists the movies that have a director and include all "directors" that have not directed any movie. A LEFT OUTER JOIN includes the movies with no director instead.
- The result of a JOIN clause can be ordered, fields can be aggregated and grouped, and conditions can be added using WHERE clauses.
- We can combine joins and aggregations to answer our initial question:

Listing 7.15 Finding the 10 movies with the largest cast.

```
SELECT movie.title, COUNT(*) AS cast_size
FROM cast RIGHT OUTER JOIN movie ON cast.movie_id = movie.id
GROUP BY movie.id ORDER BY cast_size DESC LIMIT 10;
```

Remarks:

- The query from Listing 7.15 uses a LIMIT clause to return only the ten first entries of the sorted results.
- We used a RIGHT OUTER JOIN to make sure also movies without a cast are taken into account.
- Queries may use more than one JOIN clause.

Listing 7.16 Finding all movies that Harrison Ford did not appear in.

```
SELECT movie.title
FROM actor INNER JOIN cast
ON cast.actor_id = actor.id AND actor.name = 'Harrison Ford'
RIGHT OUTER JOIN movie ON cast.movie_id = movie.id
WHERE cast.actor_id IS NULL;
```

Remarks:

- The conditions for the first join in Listing 7.16 ensure that only movies with Harrison Ford are taken into account for the second OUTER JOIN. That second join in turn delivers all movies that cannot be matched, yielding a NULL entry for the actor_id for movies without Harrison Ford.
- To ensure that every row in the cast table contains a value one can specify that the fields are NOT NULL when creating the table.

7.5 Keys & Constraints

What is stopping us from inserting a row in the cast table that contains an actor_id or a movie_id that does not exist? Or from creating a director with a duplicate id?

Definition 7.17 (Key)**.** *In a table, a column (or set of columns) is a* ***unique key*** *if the corresponding values uniquely identify the rows within the table. The* ***primary key*** *of a table is a designated unique key. A* ***foreign key*** *is a column (or set of columns) that references the primary key of another table.*

Remarks:

- SQL databases can automatically enforce these constraints. For example, a row containing a foreign key can only be inserted if it references an existing primary key. Vice versa, a row may only be removed if its primary key is not referenced by any foreign key.

Listing 7.18 Adding constraints to the database.

```
ALTER TABLE movies ADD CONSTRAINT UNIQUE (actor_id, character, movie_id);
ALTER TABLE director ADD PRIMARY KEY id;
ALTER TABLE movies
  ADD FOREIGN KEY (director_id) REFERENCES director;
```

ALTER TABLE *table*
ADD CONSTRAINT UNIQUE (*field-name*,. . .);
The values held by the specified fields must be unique among all rows.

ALTER TABLE *table* ADD PRIMARY KEY (*field-name*,. . .);
Sets the specified fields as the primary key for the table. Doing so also ensures that no duplicate entry is present when inserting or updating data.

ALTER TABLE *left-table* ADD FOREIGN KEY (*field-name,...*) REFERENCES right-table;
Ensures that the values in the specified fields in the left table are the primary key of a row in the right table.

Remarks:

- Constraints for new tables can also be set using CREATE TABLE.
- Other ALTER TABLE queries add different constraints (e.g., checking that an integer field contains only certain values), remove constraints, and change the name, type or default value of fields.
- To ensure that checking constraints and searching for data is fast, databases rely on *index* data structures.

7.6 Indexing

Definition 7.19 (Index)**.** *In the database context, an* ***index*** *is a data structure that speeds up searching for rows with specific values.*

Remarks:

- Without an index data structure, rows with a specific value can only be found by scanning through the whole table.
- In Chapter 6 you learned that hash tables can retrieve the row associated with a key in expected constant time. Many databases implement hash tables as one possible index data structure.

Listing 7.20 Adding a hash table index to our database.

```
CREATE INDEX directorid ON director USING HASH (id);
```

Remarks:

- The director associated with a movie is now found quickly when performing a join.
- Some database implementations automatically create index data structures to speed up queries that involve frequently used fields.
- Index data structures have a name—"directorid" in Listing 7.20. This is for referencing it later, e.g., if one decides to delete the index.
- Hash tables scatter the data across the storage (volatile or persistent), and it is likely that every access incurs overhead. Many database queries require scanning through ranges of the data sequentially. For example, when searching the movies from 2000–2005. Thus, accessing supposedly closeby rows requires accessing items at many different places.

- B+ trees are a data structure designed to minimize the amount of I/O operations for both searching and scanning.

Listing 7.21 Adding a B+ tree index to our database.

```
CREATE INDEX movieyear ON movies USING BTREE (year);
```

Definition 7.22 (B+ Tree)**.** *A **B+ Tree** of order b is a rooted search tree mapping **keys** to **rows**. In a B+ Tree, every non-leaf node has between $\lfloor b/2 \rfloor$ and b children, whereas leaf nodes have between $\lfloor (b-1)/2 \rfloor$ and $b-1$ children. A non-leaf node v with i children contains exactly $i-1$ keys, in sorted order. The keys contained in the sub-tree rooted at v's i^{th} child are between the $i-1^{st}$ and i^{th} key contained in node v.*

*B+ trees are **balanced**, i.e., all leaf nodes are at the same depth. Leaf nodes contain all keys inserted into the tree, and the child pointer corresponding to key k is used to point to the row associated with k. The unused child pointer of a leaf w is used to point to w's next sibling.*

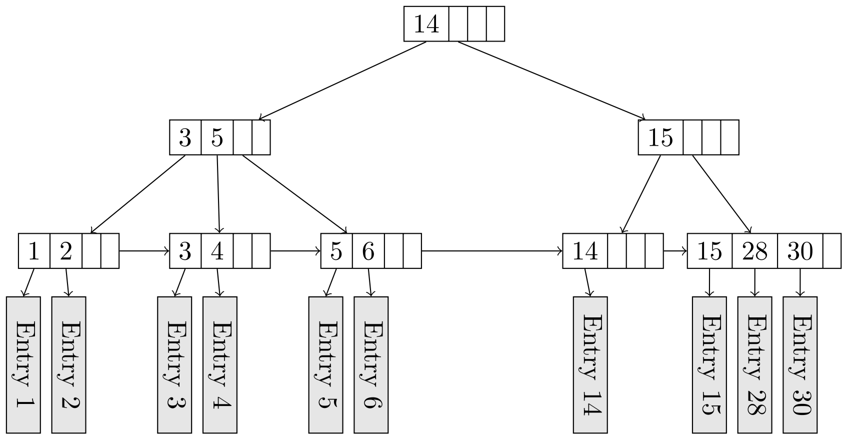


Figure 7.23: Example B+ tree of order $b = 4$.

Remarks:

- The root node is a special case—it may have as little as 1 child if it is a leaf itself, or 2 children if it is an inner node.
- The order b is sometimes called *branching factor*. To reduce the number of necessary I/O operations, b is chosen so that all data necessary to store a node is the size of (at least) one block on the disk/one cache line.
- Finding the row for some key k in a B+ tree works similar to a binary search tree.
- When inserting a key k it may happen that the leaf v that should contain k is already full. In that case v, and possibly predecessors of v that contain too many keys, need to be split.

Algorithm 7.24 B+SplitUp(k, r)

```
1: Given a B+ tree, a key $k$, and a node $v$
2: Create a new node $v'$
3: Distribute $k$ and the keys in $v$ among $v$ and $v'$ s.t. $v'$ gets the larger keys
   and both nodes are half filled
4: Let $k'$ be the smallest key in $v'$
5: Let $p$ be $v$'s parent
6: if $p$ is full then
7:    SplitUp($p, k'$)
8: end if
9: Insert $k$ with child $v$ at node $p$
```

Remarks:

- If the root node is split into two nodes v, v', then a new root r containing key k and v and v' as children is created, and the recursion stops.
- Inserting a key k is now performed by first making room using B+SplitUp if necessary, and then inserting k at the leaf.

Algorithm 7.25 B+Insert(k, r)

```
1: Given a B+ tree, a key $k$, and a row $r$
2: Perform a search for $k$ to find the leaf $v$ at which $k$ must be inserted
3: if $v$ contains $b - 1$ keys then
4:    B+SplitUp($v, k$)
5:    Replace child of key $k$ with row $r$ in node $v$
6: else
7:    Insert key $k$ with row $r$ into node $v$
8: end if
```

Remarks:

- Vice versa, when deleting a key, nodes with too few keys need to be filled up or removed from the tree.

Algorithm 7.26 B+MergeUp(v)

```
1: Given a node $v$ containing less than $(b-1)/2$ keys
2: Let $l$ and $r$ be the left and right sibling of $v$
3: if $l$ contains more than $(b-1)/2$ keys then
4:    Move largest key $x$ from $v$'s left sibling to $v$
5:    Update key in parent corresponding to $v$ to $x$
6: else if $r$ has more than $(b-1)/2$ keys then
7:    Move smallest key $x$ from $r$ to $v$
8:    Update key in parent corresponding to $r$ to $x$
9: else
10:   Merge all keys of $v$ and one of $v$'s siblings (use the node that is further to the left to store the keys)
11:   Remove the now empty node and its corrpesponding key from $v$'s parent
12:   if $v$'s parent contains less than $(b-1)/2$ keys then
13:      B+MergeUp($p$)
14:   end if
15: end if
```

Remarks:

- If v does not have a left or right sibling, the corresponding **if**-statement is ignored.
- The properties of B+ trees ensure that every node has at least one sibling. Thus, the merge operation (Lines 10–14) always has two nodes to work with.
- If no keys can be "borrowed" from a sibling, the merge may propagate until the last two children of the root node are merged into one node. In that case the root node is replaced by the merged node, decreasing the height of the tree by 1.

Algorithm 7.27 B+Delete(k)

```
1: Given a B+ tree and a key $k$
2: Perform a search for $k$ to find the leaf $v$ containing $k$
3: Remove $k$ from $v$
4: if $v$ contains less than $(b-1)/2$ keys then
5:    B+MergeUp($v$)
6: end if
```

Remarks:

- The height of a B+ tree is changed only when inserting a new or removing an old root node. Therefore, all leaf nodes are always at the same depth, thus ensuring the *balanced* property.
- A B+ tree containing n keys has height at most $O(\log_b n)$.
- It may happen that many nodes contain as little as $b/2$ keys, wasting memory and I/O operations. B* trees ensure that nodes contain at least $2/3b$ keys by cleverly "trading" entries with neighboring nodes when they contain too many or too few keys.

7.7 Transactions

Definition 7.28 (Transaction)**.** *A database* ***transaction*** *is a sequence of statements that is executed atomically.*

Remarks:

- Why would we need transactions? Consider a bank managing customer's accounts using a database system. Alice wants to calculate the liquid assets, and Bob wants to make a money transfer:

Listing 7.29 Concurrency issues in databases.

```
Alice's statement:
SELECT SUM(balance) FROM accounts;
Bob's statements:
UPDATE accounts SET balance=balance−100 WHERE customer='Bob';
UPDATE accounts SET balance=balance+100 WHERE customer='Jim';
```

Remarks:

- Assuming that the database uses multiple threads or processes to process queries, Alice's query may be CHF 100 short.
- To execute the queries atomically, both Alice and Bob can use transactions.

BEGIN TRANSACTION; *statement₁*; . . . ; END TRANSACTION;
Executes the statements atomically.

Remarks:

- One way to implement transactions is to keep track of all fields read from and written to (the read- and write-set, respectively). Then, before a transaction ends, the database system checks whether another transaction wrote to any value in the read-set. If the read-set is unchanged, the write-set can be applied atomically, e.g., by using a global lock.

- SQL offers different so-called isolation levels. The isolation level defines when writes of one transaction become visible to others. The above technique implements the *repeatable reads* level, ensuring that read values were committed before and are not written by another transaction.

- Consider some transaction A that selects all years between 1999 and 2004. What happens if another transaction B concurrently inserts an entry for the year 2000? In the *repeatable reads* isolation level, A may not see B's data if B's insert is scheduled *after* A read all other entries for the year 2000, and A would still be allowed to finish. *Repeatable reads* do not ensure atomicity ...

- The highest isolation level is called *serializable.* This level ensures that the transactions behave "as if they were executed in some sequential order", possibly at the cost of low concurrency.

7.8 Programming with Databases

How do you write an application that relies on a SQL database to store data? Should you construct the necessary SQL statements by manipulating strings, send them to the SQL server, and then parse the result?

Remarks:

- Writing such a SQL client is one possibility, but this is error-prone: The compiler used for the application will not be able to detect errors made in the SQL statements. Moreover, the declarative SQL most likely does not mix well with the programming language chosen for the application.

- One way to mitigate these issues in object oriented programming languages is object/relational mapping.

Definition 7.30 (Object/Relational Mapping)**.** ***Object/Relational Mapping (ORM)*** *is a design pattern used in object oriented programming to store objects in and retrieve them from relational (SQL) databases.*

Remarks:

- In the simplest case, an ORM simply maps a class to a table. An object then corresponds to a row, and the object's attributes correspond to the row's fields.

- The ORM takes care of storing and retrieving object in the database and performs type conversions where necessary. It provides object oriented abstractions for database queries involving WHERE and other clauses. ORMs also remove boilerplate code, i.e., setting up the SQL connection, error handling, data conversion, etc.

- This way no—or only very little—SQL code "leaks" into the object-oriented program.

- Popular ORMs include SQLAlchemy for Python, ActiveRecord for Ruby, Hibernate for Java, and the Entity Framework for .NET.

Listing 7.31 Using Hibernate for Java to change the personal information of an existing director.

```
Director director = session.load(Director.class,new Long(3464377));
// director: id = 3464377, name = "Larry Wachowski", gender = "m"
director.setName('Lana Wachowski');
director.setGender('f');
commit();
```

Remarks:

- The ORM needs to know how it should translate between objects and rows. For that, many ORM implementations allow to specify the database layout using object oriented methods.
- Many ORM mappers also support creating the database using the object oriented specification. This ensures that the database and what the ORM expects are kept in sync.
- What if you need to add or remove a column without deleting and re-creating the database? There are so-called migration tools that facilitate this process.
- Some concepts from object oriented programming are difficult to model with database concepts, and vice versa. The problems arising from combining these two paradigms are called the *Object-relational impedance mismatch*.

Chapter Notes

In 1970, Edgar F. Codd proposed the relationad database model [5] while working at IBM research. Later in the 70s, another group at IBM developed SQL's predecessor SEQUEL (Structured English QUEry Language) [3]. After being renamed SQL due to trademark issues, it was standardized by the ISO in 1987 and later revised [7]. Other companies started developing relational databases, and nowadays there are many SQL databases implementing different feature sets to choose from.

Around the same time, ER diagrams were conceived as a modeling tool [2, 4]. The Unified Modeling Language (UML), first standardized by the ISO in 1995 [8] and revised in 2012, also includes diagrams that model databases.

B Trees were invented in 1970 [1] for use in file systems. Many variants were studied, among them B* Trees [9], in which at most 1/3 of the memory is unused instead of 1/2 for B Trees. People soon realized that (also for file systems) scanning subsequent rows is an important operation. B+ Trees require at most one I/O operation to find the next element, cf. [9, 6].

Techniques from database systems can also be found in other areas of computer science. Transactions as a parallel programming model have been adotped for other programming languages under the term *transactional memory*. Ideas developed to ensure that database transactions appear atomic w.r.t. writing data to disk were adopted by general purpose file systems under the name *journaling*.

This chapter was written in collaboration with Jochen Seidel.

Bibliography

[1] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, 1970.

[2] A. P. G. Brown. Modelling a real world system and designing a schema to represent it. In *IFIP TC-2 Special Working Conference on Data Base Description*, 1975.

[3] Donald D. Chamberlin and Raymond F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '74. ACM, 1974.

[4] Peter Pin-Shan Chen. The entity-relationship model&mdash;toward a unified view of data. *ACM Trans. Database Syst.*, 1976.

[5] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 1970.

[6] Douglas Comer. The ubiquitous B-Tree. *ACM Comput. Surv.*, 1979.

[7] International Organization for Standardization. Information technology – Database languages – SQL – part 1: Framework (SQL/Framework), 2011. ISO/IEC 9075-1.

[8] International Organization for Standardization. Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure, 2012. ISO/IEC 19505-1.

[9] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison Wesley Longman Publishing Co., Inc., 1973.

Chapter 10

Link Layer

How are packets exchanged between two neighboring nodes?

10.1 Addressing

Definition 10.1 (Link Layer)**.** *The **link layer** deals with the transmission of packets between two neighboring nodes, i.e., **single-hop**.*

Remarks:

- The link layer is the bottom layer of the internet protocol suite.
- The concepts of the link layer can be grouped into two parts: the *medium access control* (MAC) dictates access patterns to the underlying wired or wireless medium. The *physical layer* (PHY) specifies the encoding of the data stream on the medium. Some layering models such as the Open Systems Interconnection (OSI) model treat these two parts as separate layers, and some models are even more detailed and split up both parts into multiple layers each.

Definition 10.2 (MAC Addresses)**.** *To identify nodes below the network layer, **MAC addresses** are used. A MAC address consists of 6 bytes and is typically formatted as 6 2-digit hexadecimal numbers separated by hyphens or colons, e.g., `00:21:cc:63:e8:5f`.*

Remarks:

- Every network interface device is assigned a worldwide unique address by the manufacturer. However, many devices also support overriding this address through software.
- On the link layer only MAC addresses are valid as source and destination addresses for packets – IP addresses are a concept introduced above the link layer and can hence not be used on the link layer.

Definition 10.3 (Link Layer Packets)**.** ***Link layer packets**, also called **frames**, have additional fields to mark the exact frame in time they occupy during transmission: a synchronization header and in some protocols also a synchronization* *footer are added, containing predefined bit sequences any listener can recognize as the start (or the end) of a packet.*

Sync. Header	MAC Dest.	MAC Source	Length	Packet Payload	Checksum	Sync. Footer

Figure 10.4: Typical link layer packet.

Remarks:

- When an IP packet is transmitted, it makes up the payload of a link layer packet.
- Some protocols operate directly on the link layer, i.e., they do not send IP packets and address nodes directly by their MAC addresses. We will see an example of such a protocol below (Protocol 10.7).
- There are also devices operating strictly below the network layer. The most prominent among them is the *switch*.

Definition 10.5 (Switches)**.** *A **switch** is a central network node with the task of mediating traffic between its neighbors. Unlike routers, switches are unaware of IP addresses and operate on the link layer only.*

Remarks:

- Without the need for routing, subnets or port forwarding tables, switch hardware can be a simpler and cheaper alternative to routers for connecting a set of nodes locally.
- In its most basic form, a switch simply copies any incoming packet to all other connected neighbors without any inspection or modification of the packet. A switch does not have a MAC address.
- However, typically, a switch also keeps track of what source MAC addresses were received on each of its physical ports. If a packet arrives with a known destination MAC address, the switch can forward the packet to that port only.
- A port of a switch does not necessarily have to be connected to an IP aware node – it is also possible to chain switches. This means, a switch might need to internally assign several MAC addresses to a single port.
- In the wireless setting, as every packet is broadcasted by the very nature of the medium, the concept of switches is superfluous. However, devices called *repeaters* may extend the reach of a wireless network by rebroadcasting any received packets. As repeaters are not even aware of MAC addresses, they only operate on the physical layer (PHY).

Definition 10.6 (Broadcast MAC Address)**.** *The MAC address* `ff:ff:ff:ff:ff:ff` *is the designated **broadcast address** on the link layer. When used as a packet's destination address, any node hearing the packet will process it.*

Protocol 10.7 (ARP)**.** *The **Address Resolution Protocol** is used to find out the MAC address belonging to a given IPv4 address.*

Algorithm 10.8 ARP lookup for an IP address a

```
1: Send a query containing $a$ to ff:ff:ff:ff:ff:ff (broadcast)
2: if there is a node with IP address $a$ then
3:     That node responds with its MAC address
4: else
5:     After some timeout, conclude that $a$ does not exist or is not reachable
6: end if
```

Protocol 10.9 (NDP)**.** *The **Neighbor Discovery Protocol** offers the functionality of ARP for IPv6. It also includes additional features, such as the detection of duplicate addresses.*

Remarks:

- Address resolution is the main way to obtain destination MAC addresses – the routing done on the network layer merely outputs IP addresses.
- Caching is used, hence ARP/NDP look-ups are only made for new IP addresses.
- ARP/NDP packets are not IP packets – they are their own kind of packet.
- Nowadays often all traffic of nodes at the "edge" of the network (such as personal computers and smartphones) is routed over a *gateway* router. As the gateway is the only direct neighbor of edge nodes, they never contact any MAC address apart from the gateway. However, when connecting nodes through a switch, e.g., at LAN parties, ARP/NDP are vital to make newly plugged in nodes reachable.

Definition 10.10 (Global Broadcast Address, IPv4)**.** *The IP address `255.255.255.255` is the designated **global broadcast address** on the network layer for IPv4. When used as a packet's destination address, any node hearing the packet will process it.*

Remarks:

- A router receiving a packet with a broadcast destination will echo the packet to all connected devices.
- Certain routers will drop broadcast packets, for example, routers belonging to an ISP – broadcasting a packet to everybody on the Internet is not a reasonable operation.

Protocol 10.11 (DHCP)**.** *The **Dynamic Host Configuration Protocol** is used to automatically assign unused IP addresses to newly connecting network participants. To do so, one node in the network runs a designated DHCP server.*

Algorithm 10.12 Acquiring an unused IP address using DHCP

```
1: DHCP client sends a request with its MAC address to 255.255.255.255,
   using source address 0.0.0.0
2: DHCP server decides on an unused IP address $a$ and marks $a$ as “reserved”
   and replies with the offer for $a$
3: DHCP client notes the IP address of the DHCP server and replies, this time
   directly, that it accepts $a$
4: DHCP server marks $a$ as in use and replies with a final confirmation
5: DHCP client receives the confirmation and uses $a$ as its IP from then on
```

Remarks:

- DHCP is strictly speaking an application layer protocol as it builds upon UDP.
- As broadcast IP addresses cannot resolve to MAC addresses, the broadcast MAC address `ff:ff:ff:ff:ff:ff` is used.
- The DHCP server may base its choice of the offered IP address on the joining device's MAC address, and assign a returning device its previous IP address.
- In addition to unused IP addresses, the DHCP server often also distributes other configuration data such as its subnet mask (the block of local addresses, e.g., 192.168.0.0/24), the gateway node's address and the preferred DNS server's address.
- If no DHCP server is present, unique IP addresses as well as the network configuration have to be set manually for every participant.
- There are two separate versions of DHCP, for IPv4 addresses and IPv6 addresses respectively, fulfilling the same purpose. IPv6 defines several specialized broadcast addresses; for DHCP the address `ff02::1:2` is used.

10.2 Wireless Phenomena

Let us briefly discuss some of the differences between wired and wireless communication.

As radio transmissions are electromagnetic waves, their propagation is reminiscent of other waves we experience in everyday life such as light and sound. For example, phenomena such as shadowing, reflection and even diffraction are observable in radio waves.

Definition 10.13 (Half-Duplex, Full-Duplex)**.** ***Half-duplex*** *devices are* ***not*** *able to both send and receive at the same time.* ***Full-duplex*** *devices can send and receive simultaneously.*

Remarks:

- Wireless devices are typically *half-duplex*, which means they cannot receive anything while sending.
- In contrast, wired communication is usually *full-duplex*, i.e., both ends of the cable may send and receive at the same time.
- The most prominent special property of the wireless medium is that by nature any transmission is a broadcast, i.e., any wireless receiver physically within range will receive a sent message, not just the intended recipient. "Within range" means that the signal is sufficiently stronger than the ambient electromagnetic noise as well as interfering signals. This can be modeled by the *signal-to-interference-plus-noise ratio.*

Definition 10.14 (SINR)**.** *The **signal-to-interference-plus-noise ratio** (SINR) is a model for the quality of a received signal. It is defined as:*

$$SINR = \frac{S}{I+N} \overset{!}{>} \beta$$

- *S: the strength of the signal to be received*
- *I: the sum of the interference caused by other transmissions*
- *N: the ambient noise*
- *β: the **SINR threshold** which needs to be cleared for successful signal reception.*

Remarks:

- This formula may be evaluated at each receiver separately to determine whether it can correctly decode the signal.
- The SINR threshold β depends on hardware and encoding.
- Physics dictate that in vacuum an electromagnetic signal's strength diminishes quadratically with distance traveled. When permeating other materials such as air or concrete walls the signal is weakened even more quickly. This effect is called *fading.*
- There exist detailed models predicting the effect of not only fading but also wave propagation phenomena such as shadowing and reflection, but these are beyond the scope of this lecture.
- Due to the different travel times of the signal over different paths, the received signal may be the sum of several components delayed by different amounts. This is called *multipath.* For example, the received signal may consist of the direct line-of-sight component of the sent signal plus a component with a longer travel time reflected off a wall.
- By using nodes with multiple antennas, multipath can be exploited to transmit and decode multiple signals at once, increasing throughput. Such schemes are the foundation of the field of *MIMO transmissions* (multiple-input, multiple-output).

- In general, it is desirable to use lower transmit powers when possible, as this reduces power consumption as well as interference caused to other nearby wireless links. However, standards designed for throughput, such as Wireless LAN, often rather prefer to use the highest available transmit power to maximize the achieved SINR, as this allows employing more efficient encodings.

10.3 Medium Access Control (MAC)

Definition 10.15 (Multiple Access)**.** ***Multiple access*** *describes a setting in which multiple devices use a shared medium to communicate. It also describes the problem of avoiding deterioration of service caused by the collisions of transmissions in such a setting.*

Remarks:

- Collision mainly concerns wireless networks nowadays. In the past, sometimes bus network structures were used, i.e., every node was connected to the same bus cable, exhibiting similar problems for wired networks.

Definition 10.16 (Medium Division)**.** *By subdividing the medium into separate domains, in each of which only one device may send at a time, collisions can be prevented from occurring in the first place. Such subdivisions may be done in several ways:*

- ***Space division:*** *segment the area of operation such that fading prevents any two potential senders' signals from colliding. Examples: AM/FM radio, GSM.*

- ***Time division:*** *segment time into time slots, in each of which only one device may send as designated by some kind of schedule. Examples: Bluetooth, GSM.*

- ***Frequency division:*** *segment the available frequency spectrum into multiple frequencies bands that can be used in parallel. However, note that usually a device cannot listen on multiple frequencies simultaneously. Examples: Wireless LAN, Bluetooth.*

- ***Code division:*** *stretch the signal and xor it with a pseudorandom bit sequence unique to each sender. Knowing the pseudorandom bit sequences, a receiver can then distinguish simultaneously arriving superimposed signals. Examples: GPS, UMTS/3G.*

Remarks:

- Typically, multiple kinds of division are combined to reach a desired level of sender separation. Bluetooth, for example, makes heavy use of temporal and frequency division through the use of its strict scheduling and frequency hopping.

Definition 10.17 (Carrier Sensing)**.** ***Carrier sensing*** *or* ***clear channel assessment*** *(CCA) is a technique to prevent collisions from occurring by listening to the medium (the "carrier") for a short while before sending, such that one might pick up on an already ongoing transmission.*

Remarks:

- If no other transmission is detected, sending is performed immediately. If another transmission is detected, sending is postponed as it is assumed a collision would occur wiping out both packets. Before the next sending attempt carrier sensing is performed again.

Definition 10.18 (Hidden Terminal Problem)**.** *Due to the fading nature of the wireless medium one may not always hear the other senders during carrier sensing, even though at the intended recipient the signals of the senders would collide. This is referred to as the* ***hidden terminal problem****.*

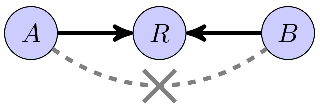


Figure 10.19: The hidden terminal problem: senders A and B can reach the recipient R, but they cannot hear each other. Hence carrier sensing cannot avoid collisions.

Definition 10.20 (Exposed Terminal Problem)**.** *The* ***exposed terminal problem*** *is the opposite of the hidden terminal problem: two close senders trying to send to different recipients may sense each others' signals and avoid sending simultaneously even though each receiver would be able to receive its signal perfectly well.*

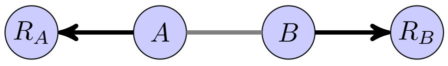


Figure 10.21: The exposed terminal problem: senders A and B could send to their respective recipients R_A and R_B simultaneously, but believe it would cause collisions due to carrier sensing.

Protocol 10.22 (RTS/CTS)**.** ***Request To Send / Clear To Send*** *is a packet exchange proposed as a solution to the hidden and exposed terminal problems.*

Remarks:

- RTS/CTS solves the hidden terminal problem as a receiving node's CTS will allow exactly one of its neighbors to send.

- The exposed terminal problem is also solved as long as the CTS messages do not interfere with other ongoing transmissions. For instance, assuming the setup from Figure 10.21, if B was already transmitting, A may not be able to hear a CTS message from R_A.

Algorithm 10.23 RTS/CTS

```
1: Before sending, the sender sends out a short RTS packet
2: If the intended recipient hears the RTS packet, it answers with a short CTS
   packet
3: If the sender receives the CTS packet, it begins transmission, otherwise it
   assumes it is not clear to send and tries again later
4: Other nodes hearing the CTS abstain from sending for some time since they
   know one of their neighbors is about to receive a packet from somewhere
   else
```

Definition 10.24 (Collision Response)**.** *The counterpart to collision avoidance is the approach of detecting collisions and responding to them after the fact.*

Remarks:

- Collisions are usually detected by immediately following up every successfully received packet with an acknowledgment (ACK) packet back to the sender. If the sender does not receive the ACK it will assume its packet got lost and try again.
- Collisions can also be detected as they occur, if the devices support simultaneous sending and receiving (i.e., are *full-duplex*) and the medium guarantees for multiple senders to hear each other (common for wired bus networks, but usually does not apply to wireless networks).
- Even though carrier sensing may prevent collisions from actually destroying packets and thus reducing the network throughput, the response is usually similar to reacting to a collision after it occurred: wait for some amount of time and then retry.
- Making a good choice for the amount of time to postpone the sending is not trivial.

Definition 10.25 (Backoff Time, Backoff Strategy)**.** *The time waited before retrying an unsuccessful transmission is called the* ***backoff time****. Ways to choose a backoff time are called* ***backoff strategies****.*

Remarks:

- Using a fixed duration as backoff time is not advisable: If two conflicting senders employ the same backoff strategy, their sending attempts would keep conflicting. Thus, feasible backoff strategies require a random component.
- Thought experiment: n nodes all try to send at the same time towards a single receiver. All transmissions start at the start of a time slot and have exactly the length of the time slot. How would a strategy maximize the probability of exactly one node sending at a time?

Protocol 10.26 (Slotted Aloha)**.** *In every time slot, every node transmits with probability* $1/n$*.*

Theorem 10.27. *Using Protocol 10.26 allows one node to transmit alone after expected time e.*

Proof. The probability for success, i.e., that the number of transmitting nodes X is exactly 1, is

$$\Pr[X = 1] = n \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \approx \frac{1}{e},$$

where the last approximation is a result from Theorem 10.28 for sufficiently large n. Hence, if we repeat this process e times, we can expect one success. □

Theorem 10.28. *We have*

$$e^t\left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t$$

for all $n \in \mathbb{N}, |t| \leq n$. Note that

$$\lim_{n\to\infty}\left(1 + \frac{t}{n}\right)^n = e^t.$$

Remarks:

- The origin of the name is the ALOHAnet protocol which was developed at the University of Hawaii to wirelessly connect the islands.
- Protocol 10.26 also works in an unslotted time model, with a factor 2 penalty, i.e., the probability for a successful transmission will drop from $\frac{1}{e}$ to $\frac{1}{2e}$. Essentially, each slot is divided into t small time slots with $t \to \infty$ and the nodes start a new t-slot long transmission with probability $\frac{1}{2nt}$.
- Protocol 10.26 requires knowledge of the number of senders n. What if we don't know n?

Algorithm 10.29 Random exponential backoff

```
1: $i \leftarrow 0$
2: Attempt sending
3: **while** sending unsuccessful **do**
4:     $i \leftarrow i + 1$
5:     Pick a value from the interval $[0, c^i]$ uniformly at random and wait that
       many time units
6:     Attempt sending again
7: **end while**
```

Remarks:

- c is some constant, often 2.
- We've already seen random exponential backoff in Chapter 4 as a way to deal with lock contention without queues. In the current scenario, queues are not an option as there is no shared memory.
- Growing the range of values $[0, c^i]$ after every failed transmission attempt allows the system to adapt dynamically to the number of senders, as each sender spreads out its transmissions more when the number of senders is large, but still does not waste too much time when the number of senders is small.
- Both Aloha and random exponential backoff waste slots, in which more or fewer than one sender send. If it is possible to coordinate a schedule implicitly or explicitly, the frequency of successful transmissions can be improved significantly.

Definition 10.30 (Duty Cycling)**.** *Nodes in a network may agree on periods of time in which no messages are exchanged. During these periods the nodes may remain in a low-power sleep mode to conserve energy. This called **duty cycling**.*

Remarks:

- Duty cycling is especially interesting to mobile devices without a constant power supply. As wireless devices consume a significant amount of energy both when transmitting and when only listening, shutting down the wireless hardware when it is not needed has become a priority.
- As wireless communication requires both the sender and the receiver to be awake at the same time to be successful, such shutting down needs to be carefully coordinated as not to carelessly lose packets.
- In networks coordinated by a central access point, the most straightforward way is to have the access point synchronize all participants and declare some wake-up schedule. Whenever a scheduled wake-up is reached, nodes power on to exchange messages. As soon as a node knows it won't need to participate in any more traffic until the next wake-up it can go to sleep.

10.4 Physical Layer (PHY)

Once we decide when to send, when not to send, and whom to address, all that remains of the link layer are several physical considerations. The link layer packet format used for almost all the wired connections of the Internet today is Ethernet v2. Wireless protocols such as Wireless LAN (following the IEEE 802.11 standard) and Bluetooth define their own packet formats. Figure 10.31 shows some examples of some common packet types.

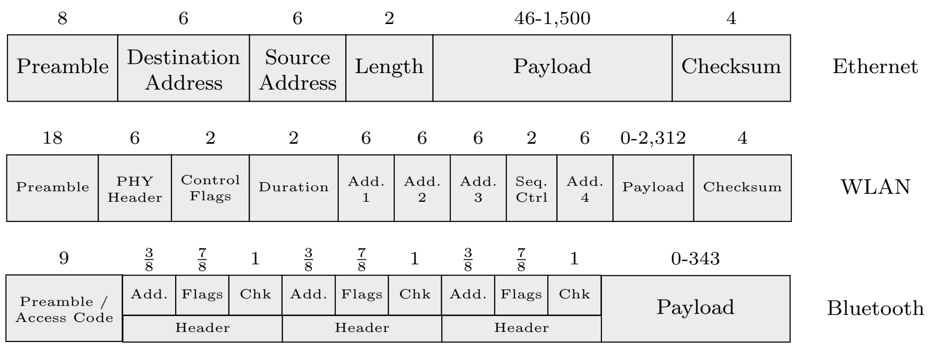


Figure 10.31: The complete physical representations of typical Ethernet, Wireless LAN and Bluetooth packets. The number above each field corresponds to its length in bytes.

Remarks:

- Wireless LAN uses several address fields, to express packets being forwarded by a base station or a repeater.
- The Bluetooth header is repeated three times and only contains one 3-bit address, as more is not required in Bluetooth's network topologies of at most 8 nodes, in which all packets are either sent or received by the network's master node.
- All these formats have preambles and checksums in common. These two features we will discuss in the remainder of this section.

Definition 10.32 (Syncword/Preamble)**.** *To establish clearly detectable packet boundaries, i.e., when exactly a packet begins and when it ends, physical layer implementations typically specify a fixed sequence of bits or bytes to be transmitted at the start of every packet called **syncword** or **preamble**.*

Remarks:

- To specify the end of the packet, either the length of the packet is encoded in the packet's header or another packet end sequence (a "footer") is attached to the end of the packet.
- It may be of interest to make the syncword unique, i.e., not let it appear as part of packet's body. This way, participants freshly joining or just waking up can be certain a new packet started when they hear the syncword.

Definition 10.33 (Bit/Byte Stuffing)**.** ***Bit stuffing*** *and* ***byte stuffing*** *are techniques representing bit resp. byte sequences, which should be unique to the packet boundaries (such as syncwords), within a packet's body in a way such that these sequences do not occur in the packet's body.*

Remarks:

- Naturally, whatever technique is employed should be reversible: the receiver should be able to restore the original content of the packet's body.
- For simplicity's sake, for the remainder of this section consider the case of a critical byte sequence consisting of only some byte `X`. The results generalize to arbitrary critical bit and byte sequences.
- First, consider the naive approach: We cannot simply replace every occurrence of `X` in the body with another byte `Y`, as we would not be able to distinguish these `Y`s from bytes which were originally `Y`.

Definition 10.34 (Escape Sequences)**.** *Given some critical byte `X`, we choose a byte `Y` $\neq$ `X` as* ***escape byte*** *and use it to define two* ***escape sequences*** *consisting of two bytes each, say, `YA` and `YB` (`A` $\neq$ `X`, `B` $\neq$ `X`, `A` $\neq$ `B`). The sender replaces every `Y` in the original body with `YA` and every `X` with `YB`. The receiver in turn performs the substitution in reverse.*

Remarks:

- This scheme is correct: the encoded body does not contain any `X`, and decoding will always yield the original body.
- The sequence `Y`z in the encoded body is undefined for values of $z \notin \{\texttt{A}, \texttt{B}\}$.
- The general concept of escape sequences is also frequently used in software. For example, to encode a quotation mark we use a backslash as escape character, e.g., `"Herman \"Babe\" Ruth"`. In web addresses `%` is used to escape the bytes of "illegal" characters, e.g., `%20` for spaces and `%E2%98%83` for the unicode snowman.
- The main disadvantage of this simple scheme is that it may cause the packet's body to become a lot longer than it originally was – up to twice as long!

Definition 10.35 (Consistent Overhead Byte Stuffing)**.** *Treat the original body as a sequence of byte strings $s_0, s_1, \ldots, s_n$ separated by the forbidden `X` byte, then alternatingly send the length of a string and the string itself: $|s_0|, s_0, |s_1|, s_1, \ldots, |s_n|, s_n$. The receiver can then reconstruct the original body by joining the strings back together with `X`s in between.*

Remarks:

- If there are multiple subsequent `X` in the original body, s_i may be an empty string.
- The encoded body is always exactly 1 byte longer than the original, no matter how often `X` occurred.
- We need to avoid using `X` in a length value. This can be accomplished adding 1 to all length values $\geq$ `X`.

Definition 10.36 (Checksums)**.** *Another common feature is the inclusion of a checksum over the whole packet, including header and payload.*

Remarks:

- Computing a checksum typically entails xoring together all input bits several times following certain patterns to obtain a checksum of 1-4 bytes.
- Checksum algorithms usually require only a single pass over the data and are simple to enough to allow performing computation and checking of these checksums in hardware.
- Checksums on the link layer serve multiple purposes. For one, it would be a shame if a lossy wireless link corrupted a packet which traveled across the globe – resending it from its source node would be a waste. Further, link layers are interested in also having checksums for non-data packets (such as those for connecting and disconnecting, synchronizing schedules or RTS/CTS).
- Higher layers in the network stack may employ additional checksums, such that they may be used on unreliable link layers.
- IPv6, as opposed to IPv4, no longer includes a checksum and expects the underlying link layer to employ reliable error detection.
- There also exist *error correcting codes* which allow not only detecting but also correcting a certain amount of bit errors. In practice, they are used only in certain Wireless LAN versions; usually, it is assumed that most packets are transmitted either completely without errors or damaged beyond repair.

Definition 10.37 (MTU)**.** *Every link layer implementation specifies a **maximum transmission unit**, the maximum link layer payload size this link layer supports.*

Remarks:

- For Ethernet this value is 1,500 bytes, for Wireless LAN it usually is 2,312 bytes, and for Bluetooth it usually is 672 bytes, using a higher transmission rate for the payload.
- It is the network layer's responsibility to ensure it creates no packets larger than the MTU.
- IPv4 and IPv6 support fragmenting oversized transport layer packets into several network layer packets, using fields in the IP header to indicate the number of fragments (Definition 1.22).
- Since Ethernet and Wireless LAN packets are common, an MTU of 1,500 bytes has become commonplace in many applications and frameworks.
- So how do we actually transmit a packet?

Definition 10.38 (Line Coding)**.** ***Line coding*** *is a physical encoding representing a data bit stream as a series of values from* $\{-1, 0, +1\}$*. When transmitting, each value is to be held for 1 time unit on the line before moving on to the next value in the series.*

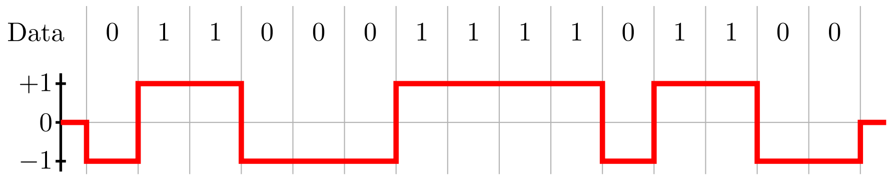


Figure 10.39: Simple line coding.

Remarks:

- Figure 10.39 shows the simplest kind of line coding: mapping every '0' bit to -1 and every '1' bit to $+1$.
- Such simple codings have two disadvantages when there is a long string of equal bits: 1) It is hard to verify that a signal is still being sent during these periods. 2) If the clocks of sender and receiver are not running at exactly the same rate, the receiver may count a different number of consecutively equal bits than what the sender intended to send.
- One workaround is to have nodes agree on a maximum number of permitted equal bits in a row. This requires encoding the data in a way that the resulting data bit stream exhibits the desired behavior.
- Another disadvantage of this coding is that there may be an undesirable bias towards $+1$ or -1, i.e., the mean value may not be 0.

Definition 10.40 (Manchester Coding)**.** ***Manchester coding*** *is a kind of line coding, in which every bit is represented by two values: '0' bits by first* -1 *then* $+1$*, and '1' bits by first* $+1$ *then* -1*.*

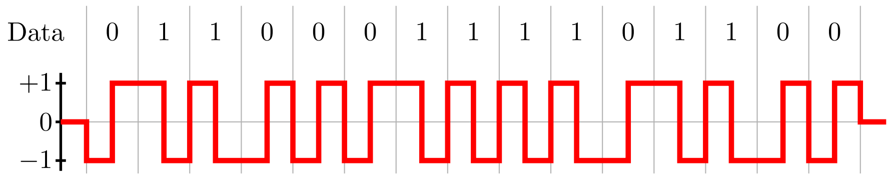


Figure 10.41: Manchester coding.

Remarks:

- Manchester coding solves the aforementioned problems with long runs of the same bit. In particular, the receiver may use the ongoing signal's edges to keep its clock in sync. It also exhibits no bias towards towards $+1$ or -1.

Definition 10.42 (Modulation)**.** *Expressing data bits as changes in the properties of a regular periodic waveform, the* ***carrier signal****, is called* ***modulation****.*

Definition 10.43 (Amplitude Modulation, AM)**.** ***Amplitude modulation*** *is a modulation which expresses data by varying the carrier signal's amplitude.*

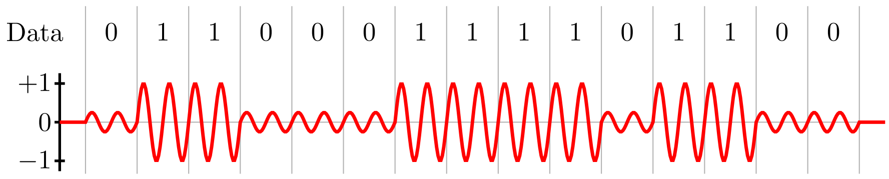


Figure 10.44: Amplitude modulation.

Remarks:

- The reception of amplitude modulated signals suffers greatly from noise, shadowing and signal transposition. For example, if the signal is reflected from a surface to reach a location behind a corner, the signal's power is decreased, which means the received amplitude value also decreases.

Definition 10.45 (Frequency Modulation, FM)**.** ***Frequency modulation*** *is a modulation which expresses data by varying the carrier signal's frequency.*

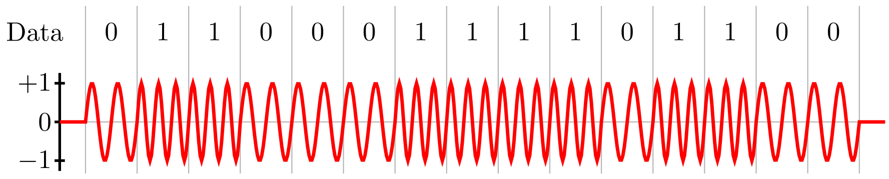


Figure 10.46: Frequency modulation.

Remarks:

- The frequency is usually varied only by small amounts, staying within a narrow frequency band.
- As opposed to amplitude modulated signals, frequency modulated signals are very robust to noise, which is one of the main reasons for the popularity of FM.

Definition 10.47 (Phase Modulation, PM)**.** ***Phase modulation*** *is a modulation which expresses data by varying the carrier signal's phase.*

Definition 10.49 (Symbols)**.** *Multiple data bits may be grouped into* ***symbols*** *before being encoded. This allows making use of the ability to represent more than 2 states at a time in the coding.*

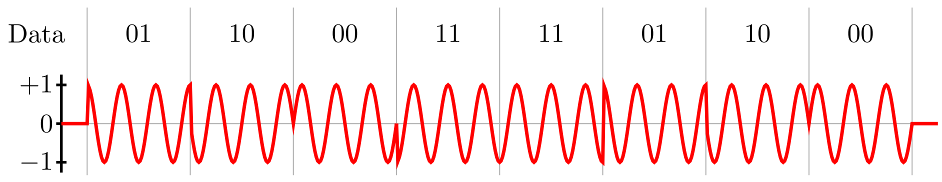


Figure 10.48: Phase modulation with 2-bit symbols.

Remarks:

- In reality, signals must be narrow-band, so "jumps" as they occur in Figure 10.48 must be avoided.
- The three modulation schemes presented above are often combined to express more different values with a single symbol.
- Encodings and modulations much more involved than the ones presented here have been designed optimized for parameters such as throughput or feasible SINR thresholds.
- A modulation encoding more data bits within the same time and frequency band raises the SINR threshold required for successful reception and vice versa.

Chapter Notes

This chapter was written in collaboration with Michael König.

Chapter 11

Markov Chains & PageRank

Let us try to predict the weather! How long until it is rainy the next time? What about the weather in ten days? What is the local "climate", i.e., the "average" weather?

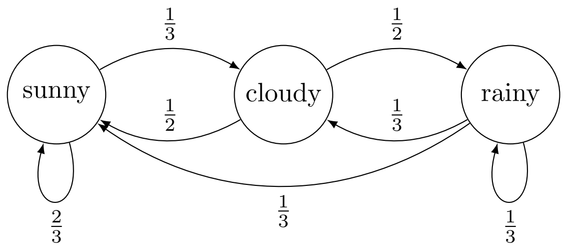


Figure 11.1: According to a self-proclaimed weather expert, the above graph models the weather in Zürich. On any given day, the weather is either *sunny*, *cloudy*, or *rainy*. The probability to have a cloudy day after a sunny day is $\frac{1}{3}$. In the context of Markov chains the nodes, in this case *sunny*, *rainy*, and *cloudy*, are called the *states* of the Markov chain.

Remarks:

- Figure 11.1 above is an example of a *Markov chain*—see the next section for a formal definition.
- If the weather is currently sunny, the predictions for the next few days according to the model from Figure 11.1 are:

Day	sunny	cloudy	rainy
0	1	0	0
1	$\frac{2}{3}$	$\frac{1}{3}$	0
2	0.611	0.222	0.167
3	0.574	0.259	0.167
4	0.568	0.247	0.185
⋮	⋮	⋮	⋮

11.1 Markov Chains

Markov chains are a tool for studying *stochastic processes* that evolve over time.

Definition 11.2 (Markov Chain)**.** *Let S be a finite or countably infinite set of* ***states****. A* ***(discrete time) Markov chain*** *is a sequence of random variables $X_0, X_1, X_2, \ldots \in S$ that satisfies the Markov property (see below).*

Definition 11.3 (Markov Property)**.** *A sequence (X_t) of random variables has the* ***Markov property*** *if for all t, the probability distribution for X_{t+1} depends only on X_t, but not on $X_{t-1}, \ldots, X_0$. More formally, for all $t \in \mathbb{N}_{>0}$ and $s_0, \ldots, s_{t+1} \in S$ it holds that $\Pr[X_{t+1} = s_{t+1} \mid X_0 = s_0, X_1 = s_1, \ldots, X_t = s_t] = \Pr[X_{t+1} = s_{t+1} \mid X_t = s_t]$.*

Remarks:

- A sequence of random variables is also called a *discrete time stochastic process*. Processes that satisfy the Markov property are also called *memoryless*.
- The probability distribution of X_0 does not depend on a previous state (since there is none). It is called the *initial distribution*, and we denote it by the vector $q_0 = (q_{0,s})_{s \in S}$ with the entries $\Pr[X_0 = s]$ for every state $s \in S$. If the first day is sunny, the initial distribution is $q_0 = (1, 0, 0)$.

Definition 11.4 (Time Homogeneous Markov Chains)**.** *A Markov chain is* ***time homogeneous*** *if $\Pr[X_{t+1} = s_{t+1} \mid X_t = s_t]$ is independent of t, and in that case $p_{i,j} = \Pr[X_{t+1} = i \mid X_t = j]$ is well defined.*

Remarks:

- We will only consider time homogeneous Markov chains.
- Markov chains are often modeled using *directed* graphs, as in Figure 11.1. The states are represented as nodes, and an edge from state i to state j is weighted with probability $p_{i,j}$.
- Just like directed graphs, Markov chains can be written in matrix form (using the adjacency matrix). In this context, the matrix is called the *transition matrix*, and we denote it by P. For the example from Figure 11.1, the transition matrix is:

			to	
		sunny	cloudy	rainy
from	sunny	2/3	1/3	0
	cloudy	1/2	0	1/2
	rainy	1/3	1/3	1/3

- Let $q_t = (q_{t,i})_{i \in S}$ be the probability distribution on S for time t, i.e., $q_{t,i} = \Pr[X_t = i]$. The probability to be in state j at time $t + 1$ is $q_{t+1,j} = \sum_{i \in S} \Pr[X_t = i] \cdot \Pr[X_{t+1} = j \mid X_t = i] = \sum_{i \in S} q_{t,i} \cdot p_{i,j}$. This can be written as the vector-matrix-multiplication $q_{t+1} = q_t \cdot P$.

- The state distribution at time t is $q_t = q_0 \cdot P^t$. We denote by $p_{i,j}^{(t)}$ the entry at position i, j in P^t, i.e., the probability of reaching j from i in t steps.
- Another interpretation of Markov chains is that of a *random walk*.

Definition 11.5 (Random Walk)**.** *Let $G = (V, E)$ be a directed graph, and let $\omega : E \to [0,1]$ be a weight function so that $\sum_{v:(u,v)\in E} \omega(u,v) = 1$ for all nodes u. Let $u \in V$ be the* ***starting node****. A* ***weighted random walk on*** *G* ***starting at*** *u is the following discrete Markov chain in discrete time. Beginning with $X_0 = u$, in every step t, the node X_{t+1} is chosen according to the weights $\omega(X_t, v)$, where v are the neighbors of X_t. If G is undirected and unweighted, then X_{t+1} is chosen uniformly at random among X_t's neighbors and the random walk is called* ***simple****.*

Remarks:

- Random walks are a special case of Markov chains where the initial distribution is a single state. In Section 11.4 we will study simple random walks.
- If it is sunny today, how long will it stay sunny?

Definition 11.6 (Sojourn Time)**.** *The* ***sojourn time*** *T_i of state i is the time the process stays in state i.*

Remarks:

- It holds that $\Pr[T_i = k] = p_{i,i}^{k-1} \cdot (1 - p_{i,i})$, i.e., T_i is *geometrically distributed*. For example $\mathbb{E}[T_{\text{sunny}}] = 2$.
- The sojourn time T_i does not depend on the time the process has spent in state i already (memoryless property). The geometric distribution is the only discrete distribution that is memoryless.
- If it is currently sunny, how long does it take until we see the first rainy day?

Definition 11.7 (Hitting Time & Arrival Probability)**.** *Let i and j be two states. The* ***hitting time*** *$T_{i,j}$ is the random variable counting the number of steps until visiting j the first time when starting from state i, i.e., the value of $T_{i,j}$ is the smallest integer $t \geq 1$ for which $X_t = j$ under the condition that $X_0 = i$. The* ***expected hitting time*** *from i to j is the expected value $h_{i,j} = \mathbb{E}[T_{i,j}]$. The* ***arrival probability*** *from i to j is the probability $f_{i,j} = \Pr[T_{i,j} < \infty]$.*

Remarks:

- The time $c_{i,j} = h_{i,j} + h_{j,i}$ is referred to as the *commute time* between i and j.
- By definition, $h_{i,j}$ is the sum $\sum_{i=1}^{\infty} i \cdot p_{i,j}^{(t)}$. The following lemma states that the expected hitting time can also be computed by solving a system of linear equations.

Lemma 11.8. *If if $h_{i,j}$ exists for all $i, j \in S$, then the expected hitting times are*

$$h_{i,j} = 1 + \sum_{k \neq j} p_{i,k} h_{k,j} \,.$$

Proof. Plugging in the definition of $h_{i,j}$ and applying the law of total probability we get that

$$h_{i,j} = \mathbb{E}[T_{i,j}] = \sum_{k \in S} \mathbb{E}[T_{i,j} \mid X_1 = k] \cdot p_{i,k} \,.$$

Taking the j^{th} term out, we obtain

$$\begin{aligned} h_{i,j} &= \mathbb{E}[T_{i,j} \mid X_1 = j] \cdot p_{i,j} + \sum_{k \neq j} \mathbb{E}[T_{i,j} \mid X_1 = k] \cdot p_{i,k} \\ &= 1 \cdot p_{i,j} + \sum_{k \neq j} (1 + \mathbb{E}[T_{k,j}]) \cdot p_{i,k} \,. \end{aligned}$$

Since $p_{i,j}$ together with all the values $p_{i,k}$ sum up to 1, we can simplify to

$$h_{i,j} = 1 + \sum_{k \neq j} \mathbb{E}[T_{k,j}] \cdot p_{i,k} = 1 + \sum_{k \neq j} p_{i,k} h_{k,j} \,. \qquad \square$$

Remarks:

- On a sunny day it takes in expectation 8 days until it starts raining.
- Lemma 11.9 for the arrival probabilities can be established similarly to Lemma 11.8.

Lemma 11.9. *For all $i, j \in S$, the arrival probability is*

$$f_{i,j} = p_{i,j} + \sum_{k \neq j} p_{i,k} f_{k,j} \,.$$

11.2 Stationary Distribution & Ergodicity

What is the "climate" in Zürich? Often one is particularly interested in the long term behavior of Markov chains and random walks. The mathematical notion that captures a Markov chain's long term behavior is the *stationary distribution*, which we will introduce and study in the following.

Remarks:

- The entries in P^t contain the probability of entering a certain weather condition (state). What happens for large values of t? The matrix seems to converge!

$$P^3 \approx \begin{pmatrix} 0.574 & 0.259 & 0.167 \\ 0.556 & 0.222 & 0.222 \\ 0.537 & 0.259 & 0.204 \end{pmatrix} \quad P^{10} \approx \begin{pmatrix} 0.563 & 0.250 & 0.187 \\ 0.562 & 0.250 & 0.187 \\ 0.562 & 0.250 & 0.188 \end{pmatrix}$$

- No matter what the initial weather q_0 is, the product $q_0 \cdot P^t$ seems to approach $\tilde{q} \approx (0.563, 0.250, 0.188)$ as t grows. Moreover, if we multiply the vector $\tilde{q}$ with P we *almost* get $\tilde{q}$ again. In other words, $\tilde{q}$ is almost an eigenvector of P with eigenvalue 1.

Definition 11.10 (Stationary Distribution)**.** *A distribution π over the states is called* ***stationary distribution*** *of the Markov chain with transition matrix P if $\pi = \pi \cdot P$.*

Remarks:

- Our weather Markov chain converges towards $\pi = (9/16, 4/16, 3/16)$, which is an eigenvector of P with eigenvalue 1. We conclude that in the long run, 9 out of 16 days are sunny in Zürich. The weather model appears to be not as accurate as the weather expert led us to believe ...

- Consider the sequence $q_i = q_{i-1} \cdot P$, where q_0 is the initial distribution. In general, this sequence does not necessarily converge as t grows. However, if it does converge to some distribution π, then it must hold that $\pi = \pi \cdot P$.

Lemma 11.11. *Every Markov chain has a left eigenvector with eigenvalue* 1*.*

Proof. Let P be the transition matrix of a Markov chain, and denote by $e = (1, \dots, 1)^\top$ the all-ones vector. Because in P the entries in each row sum up to 1 (P is row stochastic), it holds that $Pe = e$. Denoting by I the identity matrix, it follows that $(P - I)e = 0$. In other words, e is an eigenvector with eigenvalue 0 for $(P-I)$, which implies that $(P-I)$ is singular, i.e., not invertible. Thus, also $(P - I)^\top$ is singular, and it follows that there is a vector $\pi \neq 0$ so that $0 = (P - I)^\top \pi = P^\top \pi - I\pi$. Transposing and rearranging we obtain that $\pi^\top P = \pi^\top$, as desired. $\square$

Remarks:

- Using Brouwer's fixed point theorem one can show that there is also a left eigenvector π that corresponds to a probability distribution.

- The stationary distribution is not necessarily unique, see Figure 11.12. The issue is that some states are not reachable from all other states.

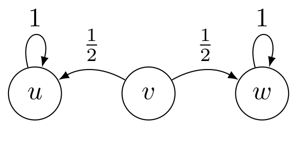


Figure 11.12: This Markov chain has infinitely many stationary distributions, for example $\pi_0 = (1, 0, 0)$, $\pi_1 = (0, 0, 1)$, and $\pi_{0.8} = (0.2, 0, 0.8)$. The states u and w are called *absorbing* states, since they are never left once they are entered.

Definition 11.13 (Irreducible Markov Chains)**.** *A Markov chain is* ***irreducible*** *if all states are reachable from all other states. That is, if for all $i, j \in S$ there is some $t \in \mathbb{N}$, such that $p_{i,j}^{(t)} > 0$.*

Lemma 11.14. *In an irreducible Markov chain it holds that $h_{i,j} < \infty$ for all states i, j.*

Proof. Fix some state j, and observe that due to Definition 11.13 for every $s \in S$, there is some t_s so that $p_{s,j}^{(t_s)} > 0$. Denote by $t = \max\{t_s \mid s \in S\}$ the largest such value. State j can be reached from every state in at most t steps. We partition the random walk into *trials* of t successive steps. Within each trial, state j is reached with probability at least $p = \min\{p_{s,j}^{t_s} \mid s \in S\}$. The number of trials until the random walk reaches j is thus upper bounded by a geometric distribution with parameter p. It follows that at most $1/p$ trials are necessary to reach j, and we conclude that $h_{i,j} \le t/p$ for any i. □

Remarks:

- Similarly, it follows that $f_{i,j} = 1$ for all states i, j if the Markov chain is irreducible.

Lemma 11.15. *Every finite irreducible Markov chain has a unique stationary distribution π. The distribution is $\pi_j = \frac{1}{h_{j,j}}$ for all $j \in S$.*

Proof. Denote by P the transition matrix of an irreducible Markov chain. Let $\pi \neq 0$ be a left eigenvector of P with eigenvalue 1 as promised by Lemma 11.11. Denote further by $h_{i,j}$ the expected hitting times guaranteed by Lemma 11.14.

We first consider the case that $\sum_i \pi_i \neq 0$ and w.l.o.g. assume that $\sum_i \pi_i = 1$. Due to Lemma 11.8 it holds that for any $j \in S$,

$$\pi_i h_{i,j} = \pi_i \left(1 + \sum_{k \neq j} p_{i,k} h_{k,j}\right) \text{ for all } i \in S\,.$$

Since $\sum_i \pi_i = 1$, summing up those equations over all i yields

$$\begin{aligned}\pi_j h_{j,j} + \sum_{i \neq j} \pi_i h_{i,j} &= 1 + \sum_i \pi_i \sum_{k \neq j} p_{i,k} h_{k,j} \\ &= 1 + \sum_{k \neq j} h_{k,j} \sum_i \pi_i p_{i,k}\,,\end{aligned}$$

by switching the summation on the right hand side. Since π is an eigenvector with eigenvalue 1, it holds that $\sum_i \pi_i p_{i,k} = \pi_k$, and thus the equation becomes

$$\pi_j h_{j,j} + \sum_{i \neq j} \pi_i h_{i,j} = 1 + \sum_{k \neq j} h_{k,j} \pi_k\,.$$

Noting that all $h_{j,j} > 1$ we conclude that $\pi_j = 1/h_{j,j}$, as desired. In the remaining case where $\sum_i \pi_i = 0$, the equation turns into

$$\pi_j h_{j,j} + \sum_{i \neq j} \pi_i h_{i,j} = \sum_{k \neq j} h_{k,j} \pi_k\,,$$

yielding that $\pi_j = 0$ for all j. This contradicts that π is an eigenvector. □

Remarks:

- Irreducible Markov chains with an infinite number of states do not necessarily have a stationary distribution.
- Depending on the choice of the initial distribution, even an irreducible Markov chain does not necessarily converge towards its stationary distribution, see Figure 11.16.

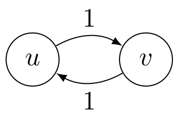


Figure 11.16: This Markov chain is irreducible, and has the unique stationary distribution $\pi = (0.5, 0.5)$. In this particular chain, each state can only be reached every other step, or in other words, both states have *period* 2. Therefore, the initial distribution is attained in every second step, and only $q_0 = \pi$ "converges" towards the stationary distribution.

Definition 11.17 (Aperiodic Markov Chains)**.** *The* ***period*** *of a state $j \in S$ is the largest $\xi \in \mathbb{N}$ such that*

$$\{n \in \mathbb{N} \mid p_{j,j}^{(n)} > 0\} \subseteq \{i \cdot \xi \mid i \in \mathbb{N}\}$$

A state with period $\xi = 1$ is called ***aperiodic****, and the Markov chain is* ***aperiodic*** *if all its states are.*

Remarks:

- One can show that if the Markov chain is irreducible, then all states have the same period.
- A state j with $p_{j,j} > 0$ is trivially aperiodic.
- If $p_{j,j} = 0$, then one can check whether state j is aperiodic by testing, as illustrated in Figure 11.18, if the following holds: Does j lie on two directed cycles of lengths k and l (counting the edges in the chain) so that k and l are relatively prime, i.e., have a greatest common divisor of 1? Or, using the k^{th} and l^{th} powers of P, are there relatively prime k and l such that both $p_{j,j}^{(k)}$ and $p_{j,j}^{(l)} > 0$?

Definition 11.19 (Ergodic Markov Chains)**.** *If a finite Markov chain is irreducible and aperiodic, then it is called* ***ergodic****.*

Theorem 11.20. *If a Markov chain is ergodic it holds that*

$$\lim_{t \to \infty} q_t = \pi,$$

where π is the unique stationary distribution of the chain.

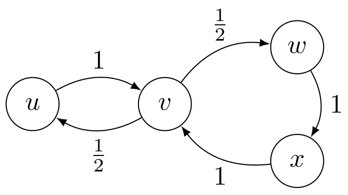


Figure 11.18: Starting at state v there is a cycle $v \to u \to v$ using 2 edges, and a cycle $v \to w \to x \to v$ using 3 edges. Because 2 and 3 are relatively prime, the state v is aperiodic.

Remarks:

- The theorem holds regardless of the initial distribution.
- The stationary distribution of ergodic Markov chains can thus be approximated efficiently, namely by successively multiplying a vector with a matrix instead of computing the powers of a matrix.

11.3 PageRank Algorithm

Google's PageRank algorithm is based on a Markov chain obtained from a variant of a random walk.

Remarks:

- Google provides search results that match the user's search terms. Under the hood Google maintains a ranking among websites to make sure "better" or "more important" websites appear early in the search results. Instead of solving the whole problem at once, this ranking is first established globally (independent of the search terms), and only later websites matching the search query are sorted according to some rank. In this section we focus on the ranking part.
- The first step to ranking websites is to crawl the web graph, i.e., a directed graph in which the nodes are websites, and an edge (u, v) indicates that website u contains a hyperlink to website v.

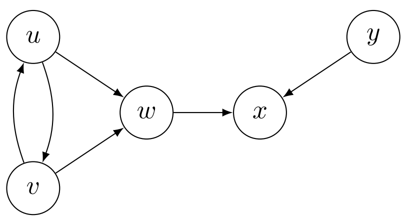


Figure 11.21: An example of a web graph with 5 websites. Website x does not link to any other website, i.e., x is a *sink*.

- A naïve approach is to rank the sites by the number of incoming hyperlinks. In the example from Figure 11.21 this yields the same rank for websites w and x. One could, however, argue that the link from w to x means that x is more important than w.

- Google's idea is to model a *random surfer* who follows hyperlinks in the web graph, i.e., performs a random walk. After sufficiently many steps, the websites can be ranked by how many times they were visited. The intuition is that websites are visited more often if they are linked by many other sites, which should be a good measure of how important a website is.
- Since the walk is directed, the random surfer can get stuck in sinks (nodes with no outgoing edges). To fix this issue, a random website is chosen for the next step whenever the random surfer reaches a sink.
- Let us denote the *random surfer matrix* describing this random walk by W.
- Simulating the random walk described by W to find a stationary distribution is not feasible: There are over 1 billion websites—meaning that a lot of steps have to be simulated to get a good estimation of the stationary distribution. Using our knowledge about Markov chains we can simulate many random walks at once by repeatedly multiplying some initial distribution q_0 with W.
- There is no guarantee that this process converges to a stationary distribution. We know that this can be fixed by making the Markov chain ergodic.
- One way to make a Markov chain ergodic is to insert an edge between every two nodes.

Definition 11.22 (Google Matrix)**.** *Let W be a random surfer matrix, and let $\alpha \in (0,1)$ be a constant. Denote further by R the matrix in which all entries are $1/n$. The following matrix M is called the **Google Matrix**:*

$$M = \alpha \cdot W + (1-\alpha) \cdot R\,.$$

Remarks:

- The intuition behind R is that in every step, with probability $1-\alpha$, the random surfer "gets bored" by the current website and surfs to a new random site.
- While the R-component in M ensures that the Markov chain converges, it also changes the stationary distribution. To ensure the impact is not too large, α should be chosen close to 1. A typical value for α is 0.85.
- The rate at which the process converges depends on the magnitude of M's second largest eigenvalue. One can show that for M the second largest eigenvalue is at most α, and that the error decreases by a factor of α in each step.

- In the example from Figure 11.21, the page ranks are

Website	Rank
x	0.384615
w	0.230769
u	0.153846
v	0.153846
y	0.0769231

- This initial version of the PageRank algorithm worked well at the time it was invented. However, it can be (and has been) fooled. Consider the following example.

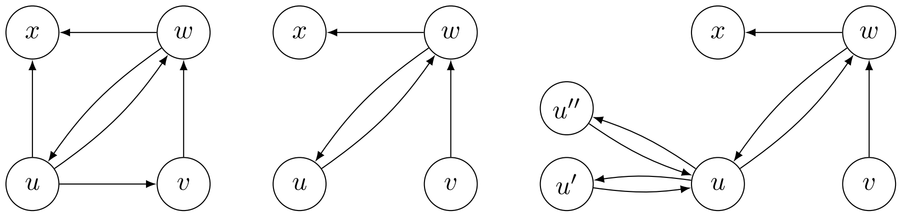


Figure 11.23: Website u wants to improve its PageRank, which is ≈ 0.23 in the initial setting on the left. First, all outgoing links to websites that do not link back are removed. The PageRank improves to ≈ 0.27. In a *Sybil attack* (right) the owner of u creates fake websites u' and u'' whose purpose is to exchange links with u. Moreover, the new websites increase the probability to visit u after a sink. Now, website u is the highest ranked site in the network with a rank of ≈ 0.41.

- Attacks where a single party pretends to be more than one individual are called *Sybil Attacks*.

- It is unknown how exactly Google ranks websites today, and specifically how the engineers at Google mitigate the effects of attacks.

- A different kind of attack on Google is *Google bombing*. This attack relies on the fact that the search terms for which a website v is considered relevant also take the *anchor text* of hyperlinks to website v into account. If, for instance, many websites link to `http://www.ethz.ch` using the anchor text "Smartest People Alive", then a search query for smart people might end up presenting ETH's website.

11.4 Simple Random Walks

In this section, all random walks are considered to be simple. This means that the edges are undirected, and the node for the next step is chosen uniformly at random among the current node's neighbors.

Lemma 11.24. *Let G be a graph with m edges. The stationary distribution π of any random walk on G is*

$$\pi_u = \frac{\delta(u)}{2m} .$$

Proof. Let π be as above, and consider some arbitrary node $u \in V$. It holds that

$$\pi_u = \sum_{v \,\in\, N(u)} \pi_v \cdot p_{v,u} = \sum_{v \,\in\, N(u)} \frac{\delta(v)}{2m} \cdot \frac{1}{\delta(v)} = \frac{\delta(u)}{2m} ,$$

i.e., the distribution is stationary. Since the Markov chain underlying the random walk is irreducible, it is also unique. □

Remarks:

- It follows from Lemma 11.15 that for a random walk, $h_{u,u}$ is $2m/\delta(u)$.
- The *cover time* $\operatorname{cov}(v)$ is the expected number of steps until all nodes in G were visited at least once.
- One could use the following Markov chain to compute the cover time of a random walk on the graph $G = (V, E)$. The set of states is $\{(v, I) \mid v \in V \text{ and } I \in 2^V\}$, where v denotes the "current state" and I denotes the visited states. The probabilities $p_{(v,I),(w,I\cup\{w\})}$ is 0 if either $I = V$ or $\{u, v\} \notin E$, and $1/\delta(v)$ if $\{u, v\} \in E$. Then, the cover time is $\operatorname{cov}(v) = \sum_{w\in V} h_{(v,\{v\}),(w,V)}$.

Lemma 11.25. *Let $G = (V, E)$ be a graph with n nodes and m edges. It holds that* $\operatorname{cov}(s) < 4m(n-1)$ *for any starting node $s \in V$.*

Proof. Let $\{u, v\} \in E$ be an edge. It holds that

$$\frac{2m}{d_u} = h_{u,u} = \frac{1}{d_u} \sum_{w\in N(u)} (h_{w,u} + 1) ,$$

and thus it must be true that $h_{u,v} < 2m$. Next, observe that it is possible to traverse all nodes in G by using no more than $2n-2$ edges, e.g., by traversing a spanning tree rooted at s. Since $h_{u,v} < 2m$ holds for every edge $\{u, v\}$ used in the traversal, it follows that $\operatorname{cov}(s) < (2n-2)\cdot 2m = 4m(n-1)$, as desired. □

Remarks:

- Consider the resistor network obtained from G by replacing every edge with a 1Ω resistor. Let u and v be two nodes in the resistor network and apply a current of 1V to them. It can be shown that $c_{u,v} = 2m \cdot R(u, v)$, where $R(u, v)$ denotes the effective resistance between u and v.
- Foster's Theorem states that for every connected graph $G = (V, E)$ with n nodes,

  $$\sum_{(u,v) \,\in\, E} R(u, v) = n - 1 ,$$

  i.e., that adding/removing an edge in G reduces/increases the effective resistance, respectively.

Chapter Notes

Historic background on the development of Markov chains can be found in [1]. The short version is that in a 1902 paper [9], the theologist Pavel Alekseevich Nekrasov, in his effort to establish free will on a mathematical basis, (falsely) postulated that independence of events is necessary for the law of large numbers. Markov, being an atheist and considering Nekrasov's reasoning an "abuse of mathematics", set out to prove him wrong.

In 1906, Markov published his first findings on chains of pairwise dependent random variables [7]. This work already includes a variant of Theorem 11.20, thus disproving Nekrasov's claim. Markov also studied the notion of irreducibility [8], proving that for irreducible Markov chains 1 is a single eigenvalue and the largest by magnitude. Today, Markov's ideas are widely applied in, e.g., physics, chemistry, and economics.

Markov chains are the basis for queueing theory, an important transport layer concept. Another application in computer science is the PageRank algorithm [10]. The bound on the Google matrix' second eigenvalue is from [5]. Sybil attacks were originally studied in the context of peer to peer systems [3], and PageRank's sensitivity to such attacks was investigated in [2].

The connection from random walks to resistor networks is investigated in depth in [4]. By associating a word with each state, random walks can be used to generate random text [11]. More than 120 "scientific" papers were generated using such methods [6] and later withdrawn by the publishers.

This chapter was written in collaboration with Jochen Seidel.

Bibliography

[1] Gely P. Basharin, Amy N. Langville, and Valeriy A. Naumov. The life and work of a.a. markov. *Linear Algebra and its Applications*, 386:3 – 26, 2004.

[2] Alice Cheng and Eric Friedman. Manipulability of pagerank under sybil strategies, 2006.

[3] John R. Douceur. The sybil attack. In *Peer-to-Peer Systems*, volume 2429 of *LNCS*, pages 251–260. Springer Berlin Heidelberg, 2002.

[4] Peter G. Doyle and J. Laurie Snell. Random walks and electric networks. `http://math.dartmouth.edu/~doyle/docs/walks/walks.pdf`, July 2006. Originally published 1984. Website accessed Sep. 23, 2015.

[5] Taher Haveliwala and Sepandar Kamvar. The second eigenvalue of the google matrix. Technical Report 2003-20, Stanford InfoLab, 2003.

[6] Cyril Labbé and Dominique Labbé. Duplicate and fake publications in the scientific literature: How many SCIgen papers in computer science? *Scientometrics*, 94(1):379–396, 2013.

[7] Andrey Andreyevich Markov. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 2-ya seriya 15 (94):135–156, 1906. (Extension of the law of large numbers to random variables dependent on each other).

[8] Andrey Andreyevich Markov. Rasprostranenie predel'nyh teorem ischisleniya veroyatnostej na summu velichin svyazannyh v cep'. *Zapiski Akademii Nauk po Fiziko-matematicheskomu otdeleniyu*, VIII seriya 25 (3), 1908. (Extension of the limit theorems of probability theory to a sum of variables connected in a chain).

[9] Pavel Alekseevich Nekrasov. The philosophy and logic of science of mass phenomena in human activity, Moscow 1902. In Russian.

[10] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[11] Jeremy Stribling, Max Krohn, and Dan Aguayo. SCIgen - an automatic CS paper generator. `https://pdos.csail.mit.edu/archive/scigen/`, 2005. Website accessed Sep. 23, 2015.

Chapter 12

Security

Every day people order and pay online – but is it secure?

12.1 Transport Layer Security

Protocol 12.1 (Transport Layer Security, TLS)**.** *TLS is a network protocol in which a client and a server exchange information in order to communicate in a secure way. Common features include a key exchange protocol (Section 12.2), the authentication of the server to the client (12.3), a bulk encryption algorithm (12.4), and a message authentication algorithm (12.5).*

Remarks:

- TLS is the successor of Secure Sockets Layer (SSL). However, sometimes in practice the term SSL includes (the newer) TLS as well.
- HTTPS (Hypertext Transfer Protocol Secure) is not a protocol on its own, but rather denotes the usage of HTTP via TLS or SSL.
- SSH (Secure Shell), even though close in name to SSL, is something different: It is a protocol to allow a client to remotely access a server, e.g., for a command-line interface.

12.2 Key Exchange

How to agree on a common secret key in public, if you never met before?

Definition 12.2 (Primitive Root)**.** *Let $p \in \mathbb{N}$ be a prime. $g \in \mathbb{N}$ is a primitive root of p if the following holds: For every $h \in \mathbb{N}$, with $1 \leq h < p$, there is a $k \in \mathbb{N}$ s.t. $g^k = h \mod p$.*

Algorithm 12.3 Diffie-Hellman Key Exchange

Input: Publicly known prime p and a primitive root g of p.
Result: Alice and Bob agree on a common secret key.

```
1: Alice picks $k_A$, with $1 \leq k_A \leq p-2$ and sends $g^{k_A} \mod p$ to Bob
2: Bob picks $k_B$, with $1 \leq k_B \leq p-2$ and sends $g^{k_B} \mod p$ to Alice
3: Alice calculates $\left(g^{k_B}\right)^{k_A} \mod p = g^{k_B k_A} \mod p$
4: Bob calculates $\left(g^{k_A}\right)^{k_B} \mod p = g^{k_A k_B} \mod p$
5: Alice & Bob have a common secret key $g^{k_A k_B} \mod p = g^{k_B k_A} \mod p$
```

Remarks:

- In crypto, it's always Alice and Bob, with a possible attacker Eve.
- Also, we will use k for keys, m for messages, p for primes, g for primitive roots, and c for ciphertext (encrypted messages). Generally speaking, an encryption algorithm encrypts a plain message m by applying a key k, resulting in ciphertext c.
- Small (not so secure) example for prime $p = 5$ and primitive root $g = 2$: $2^1 = 2 \mod 5$, $2^2 = 4 \mod 5$, $2^3 = 3 \mod 5$, $2^4 = 1 \mod 5$. One more primitive root for $p = 5$ exists. There are sophisticated methods to quickly find primitive roots, but they are beyond the material covered in this lecture.
- Algorithm 12.3 with $p = 5$ and $g = 2$: Alice picks $k_A = 2$ with $2^2 = 4 \mod 5$, and Bob picks $k_B = 3$ with $2^3 = 3 \mod 5$. Thus, Bob receives 4 and Alice receives 3. Then, Bob calculates $4^3 = 4 \mod 5$, and Alice calculates $3^2 = 4 \mod 5$. Hence, Alice and Bob have agreed on the common secret key of 4.
- How secure is Algorithm 12.3?

Definition 12.4 (Discrete Logarithm Problem)**.** *Let $p \in \mathbb{N}$ be a prime, and let $g, a \in \mathbb{N}$ with $1 \leq g, a < p$. The discrete logarithm problem is defined as finding an $x \in \mathbb{N}$ with $g^x = a \mod p$.*

Remarks:

- Intuitively, the best approach to calculate the common secret key of Algorithm 12.3 from the publicly known p, g, g^{k_A}, g^{k_B} is to solve the discrete logarithm problem. This is also the best known attack.
- However, for some classes of primes there are better attacks, which is why one often resorts to so-called safe primes p, where $p' = (p-1)/2$ is also a prime.
- How to find big enough primes though? Deterministic methods are still too slow in practice. Thus, let's go probabilistic with the following primality test.

Algorithm 12.5 Probabilistic Primality Testing

Input: An odd number $p \in \mathbb{N}$.
Result: Is p a prime?

```
1: Let $j, r \in \mathbb{N}$ and $j$ odd with $p - 1 = 2^r j$
2: Select $x \in \mathbb{N}$ uniformly at random, $1 \leq x < p$
3: Set $x_0 = x^j \mod p$
4: if $x_0 = 1$ or $x_0 = p - 1$ then
5:     Output "$p$ is probably prime" and stop
6: end if
7: for $i = 1, \ldots r - 1$ do
8:     Set $x_i = x_{i-1}^2 \mod p$
9:     if $x_i = p - 1$ then
10:        Output "$p$ is probably prime" and stop
11:    end if
12: end for
13: Output "$p$ is not prime"
```

Lemma 12.6. *Algorithm 12.5 is correct with probability* 75% *if it outputs "p is probably prime", and* 100% *correct if it outputs "p is not prime".*

Corollary 12.7. *The runtime of Algorithm 12.5 is* $O(r) \in O(\log p)$

Remarks:

- The proof for the probabilistic correctness of the primality test in Algorithm 12.5 goes beyond the material covered in this lecture.
- Algorithm 12.5 is a Monte Carlo algorithm as its (fast) runtime is deterministic, but the output can be wrong with bounded probability. However, running the algorithm again on the same p, but with different x, produces an independent result, allowing to bound the error probability by $\frac{1}{4^r}$ in r runs.
- A simple method to find big primes is thus as follows: Pick a big random number p, with p being odd. Run Algorithm 12.5 until p is prime with the desired probability of $1 - \varepsilon$. If p is not prime, pick another p. According to the prime number theorem, the average distance between two primes of size at most n is just $\ln n$, i.e., there is a good chance to find a big prime.

Definition 12.8 (Man in the Middle Attack)**.** *A man in the middle attack is defined as an attacker Eve deciphering or changing the messages between Alice and Bob, while Alice and Bob believe they are communicating directly with each other.*

Theorem 12.9. *The Diffie-Hellman Key Exchange from Algorithm 12.3 is vulnerable to a man in the middle attack.*

Proof. Assume that Eve can intercept and relay all messages between Alice and Bob. That alone does not make it a man in the middle attack, Eve needs to be able to decipher or change messages without Alice or Bob noticing. However, Eve can emulate Alice's and Bob's behavior to each other, by picking her own k'_A, k'_B, and then agreeing on common keys $g^{k_A k'_B}$, $g^{k_B k'_A}$ with Alice and Bob, respectively. Thus, Eve can relay all messages between Alice and Bob while deciphering and (possibly) changing them, while Alice and Bob believe they are securely communicating with each other. □

Remarks:

- It is a bit like concurrently playing chess with two grandmasters: If you play white and black respectively, you can essentially let them play against each other by relaying their moves.
- How do we fix this? One idea is to personally meet in private first, exchange a common secret key $k_{A,B}$, and then use this key for secure communication. Now a man in the middle cannot change the key.

Definition 12.10 (Forward Secrecy)**.** *A sequence of secured communication rounds has the property of forward secrecy, if discovering the secret key(s) of a single communication round does not reveal the content of past communication rounds.*

Remarks:

- So Alice and Bob cannot use the same secret key multiple times.

Algorithm 12.11 Diffie-Hellman Key Exchange with Forward Secrecy

Input: Alice's and Bob's common secret key $k_{A,B}$.
Result: A Diffie-Hellman key exchange not vulnerable to a man in the middle attack, and with forward secrecy.

```
1: Alice and Bob run Algorithm 12.3 to obtain round key $g^{k_A k_B}$
2: Alice sends a random number $1 \leq x_A \leq p-2$ encrypted with $k_{A,B}$ as $c_A$
   to Bob, and Bob sends Alice a random number $1 \leq x_B \leq p-2$ encrypted
   with $k_{A,B}$ as $c_B$ a challenge, respectively
3: Alice and Bob decrypt the respective messages, and Alice sends $x_B + 1$
   encrypted with $k_{A,B}$ to Bob as a response (and Bob as well with $x_A + 1$)
4: If decryption yields $x_A + 1$ for Alice, or $x_B + 1$ for Bob, respectively, they
   accept the round key $g^{k_A k_B}$
```

Lemma 12.12. *Algorithm 12.11 has the property of forward secrecy and is not vulnerable to a man in the middle attack, if encryption with $k_{A,B}$ is secure.*

Proof. For a man in the middle attack, Eve needs to be able to decrypt and encrypt with $k_{A,B}$ to convince Alice and Bob that they directly communicated with each other, which is a contradiction to the security assumption.

Regarding forward secrecy, if the attacker Eve gathers the secret key $g^{k_A k_B}$ of a communication round, she can decrypt the messages of this communication round. Even if Eve gains access to $k_{A,B}$, she cannot gain access to the keys generated in past communication rounds. □

Remarks:

- Observe that forward secrecy only applies to communication rounds in the past. If Eve gains access to $k_{A,B}$, she can perform man in the middle attacks in future communication rounds.
- However, we have a new inconvenience: Alice and Bob need to agree on a secret key $k_{A,B}$ beforehand. Furthermore, with n participants, everyone needs $n-1$ different keys.

12.3 Public Key Cryptography

"*Love all, trust a few.*" – William Shakespeare

Definition 12.13 (Public Key Cryptography)**.** *A public key cryptography system uses two keys: A public key k_p, to be disseminated to everyone, and a secret (private) key k_s, only known to the owner. A message encrypted with the secret key can be decrypted with the corresponding public key. Analogously, a message encrypted with the public key can be decrypted with the corresponding secret key.*

Remarks:

- Popular public key cryptosystem include RSA and elliptic curve cryptography.
- With public key cryptography, we have reduced the number of keys – everyone just needs a secret and a public key.
- A conceptual way to think of public key cryptography is as follows: The secret key is a physical (secret) key that opens a specific type of padlock, and this type of padlock is freely available. The public key is a physical key too, freely available, but it opens only a (secret) specific type of padlock. If Alice wants to send Bob an encrypted message, she applies his public padlock to the message container, and only Bob can open it. Similarly, if Alice wants to authenticate her message to Bob, she locks the container with her secret padlock, and only Alice's public key can unlock it. Lastly, if Alice wants to ensure both encryption and authentication, she applies both her own secret padlock and Bob's public padlock to the message container.
- We will now extend the Diffie-Hellman algorithm to public key cryptography.

Algorithm 12.14 Elgamal Public Secret Key Generation

Input: Publicly known prime p and a primitive root g of p.
Result: Alice generates a public and a secret key

```
1: Alice picks random $k_s$ with $1 \le k_s \le p-2$ as her secret key
2: Alice calculates $k_p = g^{k_s} \mod p$ as her public key
```

Remarks:

- Alice can publish p, g, k_p, but should keep k_s to her own.
- We will now start with encryption, before covering authentication.

Algorithm 12.15 Elgamal Public Key Encryption and Decryption

Input: Alice and Bob know p, g, k_p, Alice knows k_s.
Result: Bob sends Alice an encrypted message, which she can decrypt.

```
1: Bob picks a message $1 \le m \le p-2$ and a random $1 \le x \le p-2$
2: Bob sends $g^x \mod p$ and $c = m \cdot k_p^x \mod p$ to Alice
3: Alice first calculates $y = (g^x)^{p-k_s-1} \mod p$
4: Alice then obtains $m = y \cdot c \mod p$
```

Theorem 12.16 (Fermat's little theorem)**.** *Let p be a prime number. Then, for any $a \in \mathbb{N}$ holds: $a^p = a \mod p$. If a is not divisible by p, then $a^{p-1} = 1 \mod p$.*

Lemma 12.17. *Algorithm 12.15 is correct.*

Proof.

$$\begin{aligned} y \cdot c &= (g^x)^{p-k_s-1}\left(m \cdot k_p^x\right) \mod p \\ &= (g^x)^{p-k_s-1}\left(m \cdot \left(g^{k_s}\right)^x\right) \mod p \quad (\text{using } k_p = g^{k_s} \mod p) \\ &= (g^x)^{p-k_s-1} m \cdot (g^x)^{k_s} \mod p \\ &= m\,(g^x)^{p-1} \mod p \\ &= m \mod p \quad (\text{using Theorem 12.16}). \end{aligned}$$

□

Remarks:

- We can now send someone an encrypted message using public key cryptography, but what about authentication?
- Again, we first need some number theoretic preliminaries.

Definition 12.18 (Greatest Common Divisor, gcd)**.** *The greatest common divisor (gcd) of two integers i_1, i_2 is the largest integer that divides i_1 and i_2 without a remainder.*

Theorem 12.19. *Let p be a prime and i be an integer with $gcd(i, p) = 1$. Let $a_1, a_2 \in \mathbb{N}$. If $a_1 = a_2 \mod (p-1)$, then $i^{a_1} = i^{a_2} \mod p$.*

Algorithm 12.20 Elgamal Authentication

```
Input: Alice and Bob know $p, g, k_p$, Alice knows $k_s$.
Result: Alice signs a message $1 \leq m \leq p-2$, which Bob authenticates.

1: Alice picks a random $1 \leq x \leq p-1$, with $gcd(x, p-1) = 1$
2: Alice calculates $a = g^x \mod p$ and $b = x^{-1} \mod (p-1)$
3: Alice calculates $d = (m - ak_s)b \mod (p-1)$
4: Alice sends the message $m$ and the signature $(a, d)$ to Bob
5: Bob checks if $1 \leq a \leq p-1$, else he rejects
6: Bob accepts Alice's signature for $m$ if $k_p^a a^d = g^m \mod p$
```

Remarks:

- A multiplicative inverse modulo p (in this algorithm: $x^{-1} \mod p$), can be calculated using, e.g., the extended Euclidean algorithm.

Lemma 12.21. *Algorithm 12.20 is correct.*

Proof. With $d = (m - ak_s)x^{-1} \mod (p-1)$, it follows that:

$$dx = m - ak_s \mod (p-1) \Rightarrow m = dx + ak_s \mod (p-1).$$

Using Theorem 12.19, we now obtain $g^{dx+ak_s} = g^m \mod p$. Hence,

$$k_p^a a^d \mod p = \left(g^{k_s}\right)^a \left(g^x\right)^d = g^{ak_s} g^{dx} \mod p = g^m \mod p.$$

□

Remarks:

- The security of the Elgamal public key cryptography again depends on the hardness of the discrete logarithm problem.
- We can now authenticate a message using public key cryptography, e.g., we can check that the public key of Alice corresponds to Alice's secret key.
- However, we are back still at our old problem: How do I know that Alice's public key really belongs to Alice? Maybe Eve pretended to be Alice? To use a famous saying by Peter Steiner: "*On the Internet, nobody knows you're a dog*".
- What can we do, unless we personally meet with everyone to exchange secret keys? The answer lies in trusting a few, in order to trust many: Let's say that you don't know Alice, but both Alice and you know Doris. If you trust Doris, then Doris can verify Alice's public key for you. In the future, you can ask Alice to vouch for her friends as well, etc.
- Trust is not limited to real persons though, especially since Alice and Doris are represented by their keys. Take a website like PayPal for example. How do you know that you give them your credit card information, and not some infamous Nigerian princess Eve? You probably don't know anybody who personally knows PayPal...

Definition 12.22 (Web of Trust). *Let $G = (V, E)$ be a graph, where an edge between two nodes u, v represents trust between u, v. For any two nodes u, w, we say u trusts w if there is a path from u to w in G.*

Remarks:

- Hence, if you want someone to authenticate themselves, you need to find a path in the Web of Trust to them.
- In practice, the Web of Trust is a bit more sophisticated, as you can assign various levels of trust – and you might only trust someone in short distance.
- The whole situation is a bit of a chicken and egg dilemma though. In the beginning, you don't trust anyone, and nobody trusts you. You may want to find some well-connected nodes and gain their trust. This is the motivation for certificate authorities.

Definition 12.23 (Certificate Authority, CA). *A certificate authority is a node in a web of trust that is trusted by many other nodes.*

Remarks:

- A main distinction between a CA (or nodes in general) and your real-life friends is that trust is not needed to be mutual, edges in the web of trust can also be directed. As such a node u might trust v, but v does not necessarily need to trust u.
- You will find trust for some certificate authorities pre-installed on your system/browser, known as root certificates. When you want to know if you can trust a node, the node can supply you with a path (chain of trust) from the CA. More specifically, you will be supplied with signatures which you can check (as you trust the CA).
- Again, one can implement various levels of trust, e.g., you might only trust short paths.
- Moreover, a CA might get compromised. This leads to the idea of key revocation, where one can check if a key for a signature has been compromised – the corresponding certificate can be generated by anyone holding the respective secret key. Another idea is to also generate expiration dates for keys.
- A totally different problem is that your own set of root certificates might be compromised, e.g., if malicious software adds new root certificates to one's device.

12.4 Secret Sharing & Bulk Encryption

"*Three may keep a secret, if two of them are dead.*" – Benjamin Franklin

Definition 12.24 (Perfect Secrecy). *An encryption algorithm has perfect secrecy, if the encrypted message reveals no information to an attacker, except for the possible maximum length of the message.*

Definition 12.25 (Threshold Secret Sharing)**.** *Let $t, n \in \mathbb{N}$ with $1 \leq t \leq n$. An algorithm that distributes a secret among n participants such that t participants need to collaborate to recover the secret is called a (t,n)-threshold secret sharing scheme.*

Algorithm 12.26 (n, n)-Threshold Secret Sharing

```
Input: A secret $k$, encoded in binary representation of length $l(k)$.

Secret distribution
1: Generate $n-1$ random binary numbers $k_i$ of length $l(k)$ and distribute them
   among $n-1$ participants
2: Give participant $n$ the value $k_n$ as the result of XOR of $k$ and $k_1, \ldots, k_{n-1}$,
   i.e., $k_n = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1}$

Secret recovery
1: Collect all $n$ values $k_1, \ldots, k_n$ and obtain $k = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1} \oplus k_n$
```

Theorem 12.27. *Algorithm 12.26 has perfect secrecy even if $n-1$ participants collaborate.*

Proof. The theorem holds as applying the *XOR* operation $\oplus$ to a random bitstring and k results in a random bitstring. □

Remarks:

- How can we achieve a (t, n)-threshold secret sharing scheme with perfect secrecy?

Algorithm 12.28 (t, n)-Threshold Secret Sharing

```
Input: A secret $k$, represented as a real number.

Secret distribution
1: Generate $t-1$ random $a_1, \ldots, a_{t-1} \in \mathbb{R}$
2: Obtain a polynomial $f$ of degree $t-1$ with $f(x) = k + a_1 x + \cdots + a_{t-1} x^{t-1}$
3: Generate $n$ distinct $x_1, \ldots, x_n \in \mathbb{R} \setminus 0$
4: Distribute $(x_1, f(x_1))$ to participant $P_1$, ..., $(x_n, f(x_n))$ to $P_n$

Secret recovery
1: Collect $t$ pairs $(x_i, f(x_i))$ from at least $t$ participants
2: Use Lagrange's interpolation formula to obtain $f(0) = k$
```

Remarks:

- With at most $t-1$ pairs $(x_i, f(x_i))$, there are infinitely many possible polynomials with different values for $f(0)$.
- There are many other (t, n)-threshold secret sharing schemes, e.g., with intersecting hyperplanes.

- Note that in practice, a finite field of prime order instead of real numbers is used.
- We can now use the ideas in this section so far to develop a bulk encryption algorithm with perfect secrecy.

Definition 12.29 (Bulk Encryption Algorithm)**.** *A bulk encryption algorithm can securely encrypt a message of any size.*

Algorithm 12.30 One-Time Pad

```
Input: A message $m$ known to Alice, and a symmetric key $k$ (as a random bitstring) of length $l(k)$ known by both Alice and Bob.

Encryption
1: Alice sends $c = m \oplus k$ to Bob

Decryption
1: Bob obtains $m$ by $m = c \oplus k$
```

Corollary 12.31. *Algorithm 12.30 has perfect secrecy.*

Remarks:

- Note that Algorithm 12.30 has one big disadvantage – Alice and Bob need to agree on a large random number first! While this is feasible for, e.g., secret agents, it is quite impractical for everyday usage.
- One can use padding to also remove information about the length of the message, e.g., by adding random bits to the secret.

Definition 12.32 (Electronic Code Book, ECB)**.** *Given a method to encrypt a block of x bits, ECB encrypts a message of length rx by splitting the message into r blocks of length x, encrypting each block separately.*

Remarks:

- Do we now have a secure method to easily encrypt a large message, if we can encrypt small blocks, each using the same one-time pad?
- Suppose you have two message blocks m_1, m_2 of the same length, encrypted with k, resulting in c_1, c_2. However, you can obtain $m_1 \oplus m_2 = c_1 \oplus c_2$, giving you information about m_1 and m_2.

Definition 12.33 (Cipher Block Chaining, CBC)**.** *Given a method f to encrypt a block of x bits, CBC encrypts a message of length rx by splitting the message into r blocks of length x, $m_1, m_2, \ldots, m_r$, encrypting (the plaintext of) each block XORed with the previous encrypted block, i.e., $c_i = f(m_i \oplus c_{i-1})$. The first block c_0 is initialized randomly.*

Remarks:

- Are we secure now? Using the same technique as in the last remark, you can again get, e.g., $m_4 \oplus m_5$.
- CBC is still one of the standard techniques though when encrypting blocks successively, as more advanced algorithms are not susceptible to this simple attack for one-time pads. An example would be the advanced encryption standard (AES). Using AES with CBC is an example of a bulk encryption algorithm. The operation of AES is beyond the scope of this short chapter however.

12.5 Message Authentication & Passwords

"*I've been imitated so well I've heard people copy my mistakes.*" – Jimi Hendrix

Definition 12.34 (Replay Attack)**.** *In a replay attack a previously valid message from Alice to Bob is sent again from an eavesdropper Eve to Bob.*

Remarks:

- An easy way to prevent replay attacks is to include time stamps in messages. Bob can detect a replay attack, if the time stamp is too old or multiple messages with the same time stamp arrive. Another idea is to use *nonces* (numbers only used once), with the sender and receiver keeping track of the nonces used so far.
- Another issue is that an attacker could change an encrypted message without knowing the content

Definition 12.35 (Malleability)**.** *If ciphertext c can be changed to c' such that the receiver decrypts it into a different message m' without noticing, the encryption algorithm is malleable.*

Remarks:

- The Elgamal encryption Algorithm 12.15 is malleable: An attacker can relay $c = m \cdot k_p^x \mod p$ as $z \cdot c$, resulting in a valid decryption of zm.
- Thus, we need a way to ensure that the messages cannot be changed by an attacker. A natural solution are hash functions. However the hash functions described in Chapter 6 are not secure.

Definition 12.36 (One-Way Hash Function)**.** *A hash function $h : U \to S$ is called one-way, if for a given $z \in S$ it is computationally hard to find an element $x \in U$ with $h(x) = z$.*

Definition 12.37 (Collision Resistant Hash Function)**.** *A hash function $h : U \to S$ is called collision resistant, if it is computationally hard to find elements $x \neq y$, $x, y \in U$, with $h(x) = h(y) \in S$.*

Remarks:

- It can be shown that a collision resistant hash function is also a one-way hash function.

Theorem 12.38 (Example for a Collision Resistant Hash Function)**.** *Let* $p = 2q + 1$ *be a safe prime, with primitive roots* $g_1 \neq g_2$ *of* p*. The hash function* $h : \{0, \ldots, q-1\} \times \{0, \ldots, q-1\} \to \mathbb{Z} \setminus \{0\}$ *with* $h(x_1, x_2) = g_1^{x_1} g_2^{x_2} \mod p$ *is a collision resistant hash function.*

Remarks:

- Popular hash functions used in cryptography include the Secure Hash Algorithm (SHA) and the Message-Digest Algorithm (MD).
- For a small example, let us pick $p = 5$ with primitive roots $g_1 = 2$ and $g_2 = 3$. We choose $x_1 = 3$ and $x_2 = 4$, obtaining the hash $h(3, 4) = 2^3 3^4 \mod 5 = 3 \mod 5$.
- It can be shown that finding a collision for the hash function described in Theorem 12.38 is equivalent to solving the discrete logarithm problem for $\log_{g_1} g_2$. Thus, the hash function is a collision resistant hash function, as we assume the discrete logarithm problem to be computationally hard.
- One might think that using a collision resistant hash function is good enough to store passwords for a service. E.g., store the hash of each password, and then compare it to the input of the user. Even if the hashes are leaked, an attacker Eve cannot recover the passwords – or can she?
- In practice, many users use short passwords, trading security for convenience. Eve can sample the hashes of common passwords such as "*password*", revealing the passwords of all users using these simple passwords. To counter this attack, one uses a technique called *salting*: The service adds a random bitstring (the *salt*) to each password before storing the hash (or, less secure, but simpler, the username). Even if the salt is known for each user, Eve needs to attack the hash of each user individually.
- To make life for Eve even harder, it is good practice to use hash functions that provably need a lot of computation and memory to execute. However, there is still a trade off as the real user wants to log in fast as well.
- Many web services already offer secure two-factor authentication (e.g., via mobile phones) instead of just passwords or challenge-response systems. However, there is a trade-off between security and convenience.
- Are we resistant against malleability now, if we include a hash of the encrypted message? No: An attacker changing the message can change the hash as well, as the hash function is not assumed to be secret. How do we prevent the hash from being modified without being noticed? The answer are HMACs:

Definition 12.39 (Message Authentication Code, MAC)**.** *A message authentication code is a bitstring that verifies that a message comes from the desired sender and was not changed until reaching the receiver.*

Definition 12.40 (Hash-Based Message Authentication Code, HMAC)**.** *A hash-based message authentication code is a MAC that uses a collision resistant hash function in combination with a secret key.*

Algorithm 12.41 Hash-Based Message Authentication Code Generation

Input: An encrypted message c, to be sent from Alice to Bob, the publicly known hash function h from Theorem 12.38, and a secret key $1 \leq k \leq c$ known to Alice and Bob.
Result: An HMAC for c, checkable by Bob.

```
1: Alice computes $h_A = h(k, h(k, c))$, and sends $c, h_A$ to Bob
2: Bob computes $h_B = h(k, h(k, c))$, and checks if $h_A = h_B$
```

Remarks:

- In practice, if $k > c$, then k will be hashed to have a smaller size. Also, the key will be padded for extra security.
- If an attacker wants to change the message, he needs to change the HMAC too. To change the HMAC, he needs to know the secret key k
- Algorithm 12.41 can be also used with any other collision resistant hash function.

Chapter Notes

The Diffie-Hellman Key Exchange was published in the seminal paper [6], parallel unpublished work also existed from Ellis et al. at the British intelligence service GCHQ. For some works showing the hardness of breaking the Diffie-Hellman key exchange, we refer to, e.g., [5], [10], [17]. For some more recommendations on how to choose the parameters of the Diffie-Hellman key exchange see RFC 3526 at `http://tools.ietf.org/html/rfc3526`. The currently fastest algorithms to solve the discrete logarithm problem still have non-practical runtime, e.g., [1]. The idea of challenging the other party to return an encrypted version of one's random number incremented by one in Algorithm 12.11 is taken from the Kerberos protocol. The Elgamal cryptosystem was published by Elgamal in 1984 [8], some years after RSA [14].

The first deterministic polynomial primality test, by Agrawal, Kayal, and Saxena, was published in [9], with an improved runtime of $\tilde{O}(\log^6 p)$ available at `https://math.dartmouth.edu/~carlp/aks041411.pdf`. The Miller-Rabin primality test is from Rabin [13] and Miller [11]. For an introduction to number theory, we recommend, e.g., [15].

The idea for the web of trust was proposed by Zimmermann in 1992. For certificate chains and key revocation, we refer to RFC 5280 at `http://tools.ietf.org/html/rfc5280`.

The Chaum-van-Heijst-Pfitzmann hash function described in Theorem 12.38 was published in [4] by Chaum et al., for the reduction to the discrete logarithm problem see, e.g., [18]. However, the runtime of the Chaum-van-Heijst-Pfitzmann hash function is too high in practice, it is chosen in this chapter as it is easier to understand compared to other related work. The subsequently described HMAC Algorithm 12.41 is from RFC 2104 at `https://tools.ietf.org/html/rfc2104`, with further security updates in RFC 6151, cf. `https://tools.ietf.org/html/rfc6151`.

The secret sharing variant discussed in this chapter is from Shamir [16], Blakley developed similar work in parallel in 1979 [3], and also discussed its relation to one-time pads [2].

While CBC seems superior to ECB, there is one downside: Decryption of ECB can be parallelized, but the decryption of CBC has to be sequential. The in this context mentioned AES encryption is a symmetric key algorithm, based on the Rijndael cipher of Daemen and Rijmen. Details of the Advanced Encryption Standard can be found in `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. AES, with a key length of 128,192, or 256 bits, replaced DES (Data Encryption Standard), as its key length of just 56 was no longer secure enough against brute-force attacks.

For a general overview of the topic of computer security, we recommend [12] and [7]. Lastly, as a very general recommendation, we urge you not to implement your own cryptosystem unless you really know what you are doing – there is just too much that can easily be missed.

This chapter was written in collaboration with Klaus-Tycho Förster.

Bibliography

[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 55–60, Washington, DC, USA, 1979. IEEE Computer Society.

[2] G. R. Blakley. One time pads are key safeguarding schemes, not cryptosystems fast key safeguarding schemes (threshold schemes) exist. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 108–113. IEEE Computer Society, 1980.

[3] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[4] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 1991.

[5] Bert den Boer. Diffie-hillman is as strong as discrete log for certain primes. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th*

Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings, volume 403 of *Lecture Notes in Computer Science*, pages 530–539. Springer, 1988.

[6] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[7] Niels Ferguson and Bruce Schneier. *Practical cryptography.* Wiley, 2003.

[8] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[9] Nitin Saxena Manindra Agrawal, Neeraj Kayal. PRIMES Is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[10] Ueli M. Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete algorithms. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer, 1994.

[11] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, December 1976.

[12] Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of computer security.* Springer, 2003.

[13] M.O. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128 – 138, 1980.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[15] Winfried Scharlau and Hans Opolka. *From Fermat to Minkowski: lectures on the theory of numbers and its historical development.* Undergraduate Texts in Mathematics. Springer, New York, 1985.

[16] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[17] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[18] Douglas R. Stinson. *Cryptography - theory and practice.* Discrete mathematics and its applications series. CRC Press, 1995.