Chapter 1

Network Layer

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

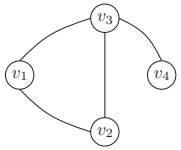
1.1 Graphs

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 $1 \leq i < k$. 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 $1 \leq i < k$, 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 20$^{\text{th}}$ 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, ω)
2: Let S = {u} be a set of visited nodes, initialized with any node u ∈ V
3: Let T be a tree just consisting of the single node u ∈ S, no edges
4: while S ≠ V do
5:    Find minimum weight edge e = {v, w} with v ∈ S and w ∈ V \ 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

Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
Set a parent node $p_v = \text{null}$ for every node $v \in V$
Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
Let $S = \{r\}$ be the set of visited nodes
while $S \neq V$ **do**
  Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
  Set $p_w = v$
  Set $d_w = d_v + \omega(e)$
  $S = S \cup \{w\}$
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?

Definition 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.

Definition 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*.
- Packets are not the only approach to network communication. In the old school telephone network, 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*.

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.

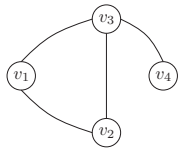
Definition 1.23 (Routing)**.** *Every node v has a* ***routing table*** *that maps every destination address to a neighbor of v.*

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.



Definition 1.25 (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 $100,000$.
- 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 1.26 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 1.26, 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.

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 v_1, v_2, v_3 and edges with weights $\omega(\{v_1, v_2\}) = \omega(\{v_2, v_3\}) = 1$ and $\omega(\{v_1, v_3\}) = \infty$. Eventually Algorithm 1.27 will compute $D(v_1 \to v_3) = 2$ and $D(v_2 \to v_3) = 1$. Now, we set a new weight $\omega(\{v_2, v_3\}) = 100$ and let v_2 detect the weight change. According to Algorithm 1.27, node v_2 still thinks that there is a path to from v_1 and v_3 that has a cost of 2, since v_1 reported that $D(v_1 \to v_3) = 2$. Thus, v_2 tells v_1 that its cost of the path to v_3 is now 3. Once v_1 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). 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 Multiprotocol Label Switching (MPLS): Many ISPs implement virtual circuit style routing by prepending an MPLS header, to get more control which paths packets are taking.
- Tunnels are also used in security. One can send unencrypted traffic over a network in an encrypted tunnel.

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, internet service providers (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 18^{th} century. Prim's algorithm (Algorithm 1.11) dates back to 1957 [Pri57] and Dijkstra invented his famous algorithm (Algorithm 1.16) in 1959 [Dij59]. 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 [Flo62, War62], 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 [KR05, TW11]. More details of the IPv4 [Pos81] and IPv6 [DH98] can be found from the RFC specifications and similarly for the RIP protocol [Mal94], OSPF protocol [Moy91], the BGP protocol [RLH06] and the Domain Name System [Moc87].

Bibliography

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). Technical report, RFC Editor, 1998.

[Dij59] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[KR05] Jim Kurose and Keith Ross. *Computer Networking: A Top-down Approach Featuring the Internet*. Addison Wesley, 2005.

[Mal94] G. Malkin. RIP Version 2. Technical report, RFC Editor, 1994.

[Moc87] P. Mockapetris. Domain Names - Implementation and Specification. Technical report, RFC Editor, 1987.

[Moy91] J. Moy. OSPF Version 2. Technical report, RFC Editor, 1991.

[Pos81] Jon Postel. Internet Protocol. Technical report, RFC Editor, 1981.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[RLH06] Y. Rekhter, T. Li., and S. Hares. A Border Gateway Protocol 4 (BGP-4). Technical report, RFC Editor, 2006.

[TW11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall, 2011.

[War62] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.