Chapter 3

Transport Layer

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

3.1 Flows

Since data flows are inherently directed, we will only consider (weighted) directed graphs in this chapter. Moreover, in contrast to Chapter 2, the weights will not indicate the latency of edges, but the bandwidth capacity.

Definition 3.1 (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:*

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

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

Remarks:

- By $\texttt{in}(v)$ resp. $\texttt{out}(v)$ 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 3.2 (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.,*

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

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 3.3 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.

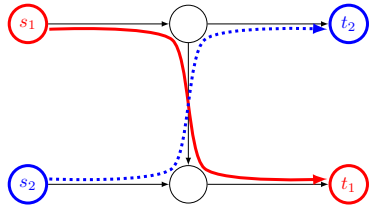

Figure 3.3: Given the depicted graph with a flow from s_1 to t_1, there is an augmenting path from s_2 to t_2 in the corresponding residual graph. If we now add a flow to the graph according to the augmenting path, then the flows starting in s_1 and s_2 will end up at the wrong destinations!

3.2 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 3.4: Linear program for throwing a party

Remarks:

- How is a linear program defined in general?

Definition 3.5 (Linear Program, LP)**.** *A **linear program (LP)** consists of a set of m inequalities*

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

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 3.6 Simplex Algorithm

```
1: choose a vertex x of the polytope
2: while there is a neighboring vertex y such that f(y) > f(x) do
3:     x := y
4: end while
5: return 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 $f(\mathbf{x}) = \sum_{e \in \mathtt{out}(s)} x_e$
subject to

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

Figure 3.7: 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 3.8 (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. If paths are not fixed, 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.

3.3 Fairness

Definition 3.9. *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.

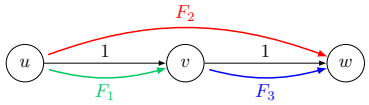

Figure 3.10: We have three flows, all with demand 1.

Remarks:

- What is a fair bandwidth allocation in Figure 3.10? 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 3.11 (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 3.12.

Algorithm 3.12 Max-Min-Fair Allocation

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 $d_1, \ldots, d_k$
while $\mathcal{F} \neq \emptyset$ **do**
  repeat
    increase rate of all flows in $\mathcal{F}$ evenly, but at most up to the respective demands
  until there is an edge $e \in E$ such that $\sum_{i:e \in p_i} F_i = c(e)$
  for all such edges e **do**
    for all i such that $e \in p_i$ **do**
      $\mathcal{F} := \mathcal{F} \setminus \{F_i\}$
    end for
    $E := E \setminus \{e\}$
  end for
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.

3.4 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 3.13 (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 3.14 (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 3.15 (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 2 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 handle packet loss.
- UDP does not provide any congestion control.

- Furthermore, UDP does not guarantee any order on the delivery of packets.
- Dealing with all 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.

3.5 TCP

Definition 3.16 (Connection)**.** *A **connection** is a bidirectional long-term relationship established between a client and a server in order to transmit data reliably.*

Protocol 3.17 (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 3.18 (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 3.19 (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 3.20 (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 3.21 (Round-Trip Time)**.** *The time it takes a packet to travel from sender to receiver and back is called **round-trip time** (**RTT**).*

Definition 3.22 (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.

sectionNATs

Definition 3.23 (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 2.20, but many nodes may have the same address. Nodes outside the private network cannot route to such a private address. %item 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 p arrives at the router, the router will switch the client's private IPv4 address and port with its own router IPv4 address and an arbitrary unused port. The router forward that modified packet p' to the server, and memorizes the triple (new unused port, client address, and client port). When the server's answer comes back to the router, the router will switch back the destination address/port to the client's address/port, before the router forwards the packet to the client.
- This is a nasty hack because it mixes concepts of the network and the transport layer.

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.