Technische Universität Ilmenau 24. Februar 2024

**Vorlesung**

# Netzalgorithmen

Prof. Dr.-Ing Günter Schäfer

NEXTEX

Zusammenfassung von
Adrian Schollmeyer

# Inhaltsverzeichnis

# 1. Introduction

## 1.1. Basic Types of Transmissions

**Web**
- Bunch of data to be transmitted
- No guaranteed arrival times
- Simplest Case: Server and Client are directly connected by a cable

**Telephony**
- Continuous flow of information
- Information must arrive in time
- Simplest Case: Two telephones are directly connected via a cable

## 1.2. Structuring a Network

As pairwise connection of all entities with each other (thus building a complete graph of all entities) does not work, other structures need to be established. We distinguish between *end systems* (user devices) and *switching elements* (switches, routers, etc.).

**End systems** connect to some form of uplink to access/provide information on the network. They do not forward requests of other systems.

**Switching elements** Forward incoming packets onto the next hop towards its destination. The "best" next hop is decided by various routing/forwarding tables and algorithms.

## 1.3. Routing Algorithms

Routers execute *routing* algorithms to decide which output line an in coming packet should be transmitted on.

**Connection-oriented services** Run the routing algorithm during connection setup and only once to find one path to forward the packets along for the whole lifetime of the connection.

**Connectionless services** Run the routing algorithm either for each packet or periodically, updating the router's forwarding table in the process.

Routing algorithms can take a *metric* into account that assigns costs to network links and allows administrators to influence routing decisions. Some possible metrics are:

- Financial cost for sending a packet over a link (e. g. when the link is charged per unit of data transferred).
- Delay (useful to penalize using a link with high delay when trying to prefer links with low delay)
- Number of hops (commonly used in most routing algorithms deployed on the Internet, aims to reduce the number of routers/networks to traverse to reach a destination)

The cheapest path is also commonly referred to as the *shortest path*.

Basic types of routing algorithms:

**Non-adaptive routing algorithms** do not base routing decisions on the current state of the network.

**Adaptive routing algorithms** take into account the current network state (e. g. distance vector routing, link state routing).

## 1.4. Flooding

*Flooding* is a simple strategy, sending every incoming packet to every outgoing link except the one it arrived on. This leads to many duplicated packets in the network, but leads to the packet almost certainly arriving at the destination (the only exception being broken links partitioning the network into two parts).

To reduce the number of duplicated packets, strategies can be used:

**Solution 1: Hop counting** Have a hop counter in the packet header, which is decremented by each router. If the packet stays in the network for too long, the hop counter goes to 0 and the packet is dropped. Ideally, the hop counter should be initialized to the length of the shortest path from the source to the destination.

**Solution 2: Sequence numbers** Each router maintains a sequence number and a table of sequence numbers it has seen from other routers. The first-hop router increments and adds its sequence number to each incoming packet from a host. Each router only forwards incoming packets, if it hasn't seen this sequence number from the first-hop router, yet. Thus, packets that have already been seen are discarded.

## 1.5. Adaptive Routing Algorithms

Non-adaptive routing algorithms pose problems:

- Non-adaptive routing algorithms can't cope with dramatic changes in traffic levels in different parts of the network.
- Non-adaptive routing algorithsm are usually based on average traffic conditions, but lots of computer traffic us extremely *bursty* (i. e. very variable in intensity).

Thus, adaptive routing algorithms are commonly used to make routing decisions.
Three types can be distinguished:

**Centralized adaptive routing** Has only one central routing controller making routing decisions.

**Isolated adaptive routing** Is based on information local to each router. No exchange of information between routers is required.

**Distributed adaptive routing** Uses periodic exchanges of information between routers to compute and update routing information to be stored in the local forwarding table.

### 1.5.1. Centralized Adaptive Routing

At the heart of Centralized Adaptive Routing is a central routing controller, which

- periodically collects link state information from routers
- calculates routing tables for each router
- dispatches updated routing tables to each router

The centralized approach is severely limited by the routing controller. If it goes down, the routing becomes non-adaptive, making the network vulnerable to outages. Furthermore, the controller needs to handle a great deal of routing information, making it not only a single point of failure, but also a bottleneck for scalability and performance of the network.

### 1.5.2. Isolated Adaptive Routing

The basic idea is to make routing decisions solely based on information available locally in each router, e. g.:

- *Hot potato*
- *Backward learning*

Hot potato routing:

- Forward the incoming packet to the output link with the shortest queue
- Do not care where the selected output link leads
- Not very effective

Backward learning:

- Maintain a local forwarding table with next hop, hop count and output link

- Incoming packets update the fowarding table entry of the sender if their hop count is better than the entry's current hop count
- Forward packets based on the forwarding table, random route (hot potato, flood) the packet if no entry for the destination exists.
- Remove/forget stale entries in the forwarding table to account for deterioration of routes (e. g. in case of link failures)

Ethernet switches commonly use backward learning to maintain forwarding tables for MAC addresses, usually falling back to flooding packets if no entry for the destination MAC exists in their local forwarding table.

### 1.5.3. Distributed Adaptive Routing

The central goal is to determine a "good" path (i. e. a sequence of routers) through the network from source to destination. For calculations, the network is abstracted into a graph consisting of:

**Routers** represented by nodes

**Links** represented by edges

**Costs** assigned to edges, representing link costs

Routing algorithms can be classified in several ways:

- Global or decentralized information

  **Decentralized** All routers know only a portion of the network, i. e. their physically connected neighbors and link costs to their neighbors. By exchanging information with neighbors, routes to other destinations can be calculated. Examples: BGP (path vector), RIP (distance vector)

  **Global** By exchanging information, routers gain knowledge of the full network topology and the cost of each link. This is used to compute cheap routes to each destination in the network. Examples: OSPF, IS-IS

- Static or dynamic routes

  **Static** Routes change only slowly over time, e. g. if they are statically configured by network administrator.

  **Dynamic** Routes change more quickly in response to link cost changes and require periodic updates of routing information.

### 1.5.4. Graph Model for Routing Algorithms

- $V = \{v_1, v_2, \ldots, v_n\}$ the set of nodes (routers)
- $E = \{e_1, e_2, \ldots, e_m\} \subseteq V^2$ the set of edges (links)
- $c : V \times V \to \mathbb{Z}_{>0}$ cost of an edge
- $s \in V$ start node (i. e. the node for which the shortest path shall be found)
- $d[i]$ cost from $s$ to node $v_i$
- $p[i]$ index $j$ of the predecessor $v_j$ of $v_i$ on the shortest path from $s$ to $v_i$.
- $\delta(s, v)$ cost of the shortest path from $s$ to $v$

### 1.5.5. Dijkstra's Algorithm for Shortest Paths

Maintain a set $N$ (initially empty) of nodes for which shortest have been found. For each node $i$ that is not directly connected to $s$, set $d[i] = \infty$, otherwise $d[i] = c(s, v_i)$. Starting from $s$, take $v_i \in V \setminus N$ with the lowest $d[i]$. The path from $v_i$'s predecessor to $v_i$ is the shortest path. Update $d[j]$ for neighbors $v_j \in V \setminus N$ of $v_i$, wherever $d[i] + c(v_i, v_j) < d[j]$ (i. e. the path from $s$ to $v_j$ via $v_i$ is shorter than the previously known path from $s$ to $v_j$). Set $N := N \cup \{v_i\}$. This results in finding the shortest paths from $s$ to each node in the network.

Complexity:

- $\mathcal{O}(|V|^2)$
- Optimal in dense graphs with $|E| \approx |V|^2$
- Efficient implementations with $\mathcal{O}(|V| \cdot \log |V| + |E|)$ are possible when using Fibonacci-Heaps.

### 1.5.6. Distance Vector Routing

For distance vector routing, each node has its own table for $D^X(Y, Z)$, listing the cost from $X$ to $Y$ via $Z$ as next hop. Distance vector routing has a few favorable properties, useful in large networks like the Internet:

**Iterative** The algorithm works iteratively, until no nodes exchange information (i. e. no updated information is transmitted through the network)

**Asynchronous** Information does not need to be exchanged in lock step. Instead any node can send updated information at any time.

**Distributed** Each node only needs to communicate with its directly attached neighbors.

Initially, all routers only know the costs to their neighbors and set the cost to any other destination to $\infty$ (aka. unreachable). Afterwards they notify neighbors of their costs to each destination they know of. Then, distance vector routing algorithms continuously wait for changes in local link costs or link update messages from neighbors. Whenever such a change occurs, the information is used to update the local distance table. If any changes to shortest (!) paths have occured, neighbors are notified of the updated shortest path costs.

The main problem of distance vector routing is the *count to infinity* problem. If the link cost for a link suddenly increases (possibly to $\infty$), the network might now have a shortest path to a destination going in a circle for some time, while the change in link cost is continuously increased in the network until the new cost for the link is reached or an alternative, shorter path is found. This increases the time to find a new shortest path dramatically! This issue can be mitigated with *poisoned reverse*, i. e. when $Z$ routes to $X$ via $Z$ and $Y$ notifies $Z$ that the cost to $X$ changed, $Z$ notifies $Y$ that its cost to $X$ is $\infty$ to prevent $Y$ from routing to $X$ via $Z$ (as $Z$ would want to route to $X$ via $Y$, making the packets go in a loop).

### 1.5.7. The Bellman-Ford Algorithm

The *Bellman-Ford algorithm* is capable of solving the problem of computing shortest path in graphs with edges with negative costs and is the basis for distance-vector routing. Its only limitation is that there must be no cycles with negative total cost, as otherwise the shortest path's cost will go to $-\infty$ (as continuously traversing such a cycle will inifinitely reduce the total cost). The algorithm can detect if such cycles with negative total cost exist.

The algorithm iteratively improves the estimated cost to reach a node by iterating $|V|-1$ times over all edges and check if the current estimate of the node can be improved by taking any of the connected edges, given the current estimate cost. As this algorithm always checks all edges, it has a higher running time of $\mathcal{O}\big(\,|V| \cdot |E|\big)$.

Negative cost cycles can be detected by running the algorithm $|V|$ times. If the cost to any node has changed in the $|V|$th iteration, there must be a negative cost cycle.

### 1.5.8. Comparison of Link State and Distance Vector Algorithms

**Message complexity** How many messages are exchanged?

- Link State: with $n$ nodes, $E$ links, $\mathcal{O}\left(n \cdot E\right)$ messages are sent by each node
- Distance Vector: exchange only between neighbors, but variable number of messages and variable convergence time

**Speed of Convergence** How long does it take until the routing table doesn't change after a link state change has occured?

- Link State: $\mathcal{O}\left(n^2\right)$ algorithm requires $\mathcal{O}\left(n \cdot E\right)$ messages
- Distance Vector: variable convergence time, in part caused by routing loops and the count-to-infinity problem

**Robustness** What happens if a router malfunctions?

- Link State:
  - Node can advertise incorrect link cost
  - Invalid routing table calculations only affect the malfunctioning router
- Distance Vector:
  - Nodes can advertise incorrect path costs
  - Each node's table is used (in part) by other routers, so errors can propagate through the network

## 1.6. Hierarchical Routing and Interconnected Networks

So far, an idealized scenario with identical routers and a "flat" network was assumed. In practice, networks scale to hundreds millions of destinations, making storing detailed routing tables for all destinations in the whole network technically impossible, due to memory limitations in routers and link overloads caused by routing table exchanges between routers. Furthermore, administrative autonomy in parts of the network (like in the Internet's autonomous systems) should allow for network administrators to control the routing (especially the routing protocol in use) in their network, independent of the rest of the network.

In *interconnected networks*, data transmission usually involves multiple networks. Routing can be distinguished into two levels:

**Intradomain routing** inside autonomouse systems.

**Interdomain routing** between autonomous systems

In the Internet, interdomain routing is done using the Border Gateway Protocol (BGP), which operates on the AS level and considers every AS as one hop. For intradomain routing, each network administrator can choose their AS's interior routing protocol (e. g. OSPF, RIP, iBGP, IS-IS).

For sending traffic between ASes, *Internet Service Providers* (ISPs) have peering agreements and connections with and to each other, making a data transfer possible. Depending on the policies and available links, traffic may not be able to be sent directly from the source ISP to the destination ISP, but needs to be sent to a different transport provider network first (transit).

Each network operator has to make decisions regarding how to handle the traffic in their network, including

- allocating enough capacity of routers and links,
- choosing a routing algorithm,
- setting link costs.

This requires estimation of *traffic demand* in the network. This can be displayed as *demand volume matrix* $H : \{1, \ldots, n\}^2 \to \mathbb{N}$, denoting the traffic demand volume between nodes $v_i$ and $v_j$ as $H[i, j]$, also abbreviated $h_{ij}$ later on.

## 1.7. Considerations on Traffic Demand and Link Utilization

To understand constraints on maximum link utilization, a few basic facts about the natur of Internet traffic need to be recapitulated:

- Packets are delayed in every router due to store-and-forward processing and queueing.
- Traffic congestion can occur in parts of the Internet.
- Packets may be dropped if arriving at a router with full output queues.

The task of a network designer is to design the network such that delay, congestion and the probability of packet drops are minimized, while also allowing for a reasonable utilization of the network. This is complicated by the fact that traffic arrival patterns and packet sizes in the Internet are random. In order to characterize Internet traffic behavior, large scale measurements are needed to gain insights about traffic arrival distribution and packet size distribution.

**Important observation:** Internet traffic does not follow commonly known distributions like normal or exponential distributions, but shows self-similar characteristics and can have *heavy-tailed* distributions, i. e. distributions with high skewness.

For simplicity, a few assumptions are made:

- Packets arrive according to a Poisson process with rate $\lambda$:

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{1.1}$$

  (on average, one arrival in every time interval of length $\frac{1}{\lambda}$).

- Packet size is exponentially distributed, leading to exponentially distributed service times with rate $\mu$

Considering only one router, such a system can be thought of as an M/M/1 queueing system.

## 1.8. The Poisson Process

Let $A(t)$ $(t \geq 0)$ be the number of packets arriving in $(0, t]$. Requirements:

- $A(0) = 0$

- Idependence of teh number of arrivals in disjoint time periods
- Singularity of arrival events (packets never arrive in parallel)
- Stationary process of arrivals: the probability how many arrivals happen in a time interval only depends on the interval length.

Denote the probablity that $n$ packets arrive in $(0, t]$ as follows:

$$P_n(t) := \Pr[A(t) = n] \tag{1.2}$$

Due to the aformentioned requirements, we have

$$P_0(0) = 1 \qquad \forall n > 0 : P_n(0) = 0 \tag{1.3}$$

After a bunch of math that no one in their right mind can memorize, we obtain:

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{1.4}$$

## 1.9. Little's Law

Let Arrival$(T)$ be the number of packets arrived until time $T$, $W_i(T)$ be the waiting time of packet $i$ at time $T$, $N(T)$ be the number of packets in the system at time $T$.

We are interested in the accumulated (total) waiting time of all jobs that ever arrived in the system until time $T$. This can be computed either as the sum of waiting times of all packets arrived until time $T$ or as the integral over the number of packets in the system during $(0, T]$. Both methods lead to the same result.

Now, let $\lambda(T)$ be the average number of packets in $(0, T]$, $\overline{W}(T)$ be the average waiting time of a packet and $\overline{N}(T)$ be the average number of packets in the system. Then we get *Little's Law*:

$$\lambda \cdot \overline{W} = \overline{N} \tag{1.5}$$

## 1.10. M/M/1 System

The number of packets in the system (queue size) at discrete points in time $\delta$ can be described as a *Markov chain*, with the probability of the queue size increasing being $\lambda\delta$ and the probability of the queue size decreasing being $\mu\delta$. Let $p_n$ denote the probability of the system having queue size $n$. We obtain

$$p_n = (1 - \rho)\rho^n \qquad \rho = \frac{\lambda}{\mu} \tag{1.6}$$

$$\overline{N} = \frac{\lambda}{\mu - \lambda} \tag{1.7}$$

$$\overline{W} = \frac{1}{\mu - \lambda} \tag{1.8}$$

Thus, with $\rho \to 1$ (i. e. the system load is so high that on average each packet takes as long to process as new packets arrive on average), the average waiting time and queue size go to infinity. Since in reality the queue size is limited, this will lead to packets being dropped.

If packets have average size $K_p$ bits and link capacity is $C$ bits per second, then the average service rate of the link is

$$\mu_p = \frac{C}{K_p} \text{ pps (packets per second)} \tag{1.9}$$

If the average arrival rate is $\lambda_p$ pps, then the average delay is given by

$$D(\lambda_p, \mu_p) = \frac{1}{\mu_p - \lambda_p} \tag{1.10}$$

This leads to an important insight: To maintain low delays, link utilization should be kept low, e. g. below 50 %. Thus, when link utilization reaches a certain threshold, it should be upgraded. From a delay perspective, it's better to have one high bandwidth link than multiple lower bandwidth links. This is often referred to as the *statistical multiplexing gain*.

Contrary to this, fault tolerance may call for having multiple links. Also, on a single link, misbehaving traffic flows are difficult to control.

## 1.11. Notion of Routing and Flows

Routing can not only be interpreted as the decision how an individual packet may be transported in the networks, but also how ensemble traffic may be routed between the same two points (e. g. points of presence, data centers). For the remainder of the lecture, this second notion is used and instead of making routing decisions for individual packets, decisions for whole flows are made. Routing decisions then need to stay withing capacity constraints or can influence capacity decisions.

## 1.12. Multi-Level Networks

When doing interconnects over transit providers, the network architecture can be viewed both in the *transport view* (i. e. the (underlay) network of the transport provider) as well as the *traffic view* (i. e. the flow of traffic between nodes in the network).

Links in the traffic network are *logical links* and must be mapped to links/paths in the transport network. The mapping can change the properties of the network from one view to the other. There can be logical links between nodes in the traffic view for nodes that are not physically connected in the transport view, e. g. if traffic between these nodes needs to be transported over a few switches.

# 2. Modeling Network Design Problems

## 2.1. Link-Path Formulation

- $\hat{h}_{12} = 5$… undirected demand between node 1 and 2 is 5, also noted as $\langle 1, 2 \rangle$
- $\hat{x}_{132}$… amount of flow over path 1, 3, 2
- $\hat{c}_{12}$… capacity of link 1–2
- $a^*$… optimal solution for variable $a$ (e. g. $\hat{x}^*_{132}$)

These can be combined to obtain systems of equations, which usually have multiple solutions. The answer to the question, which of these solutions is of best interest, depends on the goal of network design, e. g.:

- Minimize the total routing cost (if links are annotated with a link cost)
- Minimize congestion of the most congested link

If the objective is to minimize the total routing cost and the cost of routing one unit of traffic over one link is set to 1 for all links (i. e. the goal is to minimize the number of hops for each route), an objective function might be:

$$\mathbf{F} = \hat{x}_{12} + 2\hat{x}_{132} + \hat{x}_{13} + 2\hat{x}_{123} + \hat{x}_{23} + 2\hat{x}_{213} \tag{2.1}$$

Note that flows routed over two links are weighted with factor 2. Such an optimization task is called a *multi-commodity flow problem*. The inverse objective (try to avoid direct links) can also occur, e. g. in air travel networks.

Link-path formulation is one of multiple ways to describe network optimization problems. It is appropriate for networks with undirected links as well as with directed links.

## 2.2. Node-Link Formulation

In this scenario, links and demands are assumed to be directed, so a link 1–2 is substituted with two directed links ("*arcs*") $1 \to 2$ and $2 \to 1$. Instead of tracing all path flows realising the demand, the total link flow for the demand on each link is considered. Undirected demands $\langle 1, 2 \rangle$ are replaced with directed demands $\langle 1 : 2 \rangle$.

Looking from the point of view of a fixed node that is not end point of the flow, there are flows coming in and going out of that node. The total incoming flow must then be equal to the total outgoing flow. The demand of source nodes is the total outgoing flow

minus the total incoming flow, while for sink nodes the demand is the total incoming flow minus the total outgoing flow.

This gives the following notation:

- $\tilde{x}_{13,12}$... flow over arc $1 \to 3$ for demand $\langle 1:2 \rangle$

For each demand and each node, all possible direction of paths (including backflows) are part of the total flow equation, although backflows can be safely set to 0, as they make no sense from a practical viewpoint. Additionally, the source node includes a $-\hat{h}$ term and the sink node includes a $\hat{h}$ term to account for the imbalance in incoming/outgoing flow caused by the demand (account for the conservation of flow).

A simple example for a demand $\langle 1:2 \rangle$ with nodes $1, 2, 3$ might be:

$$\tilde{x}_{12,12} + \tilde{x}_{13,12} = \hat{h}_{12} \tag{2.2}$$

$$-\tilde{x}_{13,12} + \tilde{x}_{32,12} = 0 \tag{2.3}$$

$$-\tilde{x}_{12,12} - \tilde{x}_{32,12} = -\hat{h}_{12} \tag{2.4}$$

Capacity constraints are modeled as before:

$$\tilde{x}_{12,12} + \tilde{x}_{12,13} \leq \hat{c}_{12} \tag{2.5}$$

Note that the two notations shown until now can become very cumbersome for larger networks:

- There might be no demand between some or many pairs of nodes
- There are no links between most pairs of nodes

Still, the two formulations would require inclusion of these cases, although they are not relevant to the solution at all.

## 2.3. Link-Demand-Path-Identifier-Based Notation

This formulation assignes indices to demands and capacities, yielding a simpler notation:

- $h_i$... the demand with index $i$
- $c_j$... the (known) capacity of the $j$th link
- $y_j$... the unknown capacity of the $j$th link (dimensioning problem)
- $x_{ij}$... the flow of demand $i$ over link $j$
- $v_1 - v_2 - \ldots - v_{n+1}$... undirected path in node representation
- $v_1 \to v_2 \to \ldots \to v_{n+1}$... directed path in node representation
- $\{e_1, e_2, \ldots, e_n\}$... undirected path in link representation

- $(e_1, e_2, \ldots, e_n)$... directed path in link representation
- $\xi_i$... cost of link $e_i$
- $P_{ij}$... candidate path $j$ for demand $i$
- $\delta_{edp} : e \times P_{ij} \to \{0, 1\}$... is link $e_k$ on candidate path $P_{ij}$, *link-path incidence relation*

The vector of all flows is called the *flow allocation vector* or *flow vector*:

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x_2}, \ldots, \mathbf{x}_D) \tag{2.6}$$
$$= (x_{11}, x_{12}, \ldots, x_{1P_1}, \ldots, x_{D1}, x_{D2}, \ldots, x_{DP_D}) \tag{2.7}$$
$$= (x_{dp} \mid d = 1, 2, \ldots, D; p = 1, 2, \ldots, P_d) \tag{2.8}$$

The table for the link-path incidence relation $\delta_{edp}$ contains a 1 whenever a link $e$ is used for satisfying a demand $d$ over a path $p$, and 0 if the link is not used in that path. Note that $\delta_{edp}$ is not a variable, but fixed!

This us a notation for the *load* $\underline{y}_e$ of link $e$ and capacity constraints:

$$\underline{y}_e = \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \tag{2.9}$$

$$\forall e \in \{1, 2, \ldots, E\} : \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \leq y_e \tag{2.10}$$

The general formulation of the simple dimensioning problem is:

$$\min \mathbf{F} = \sum_{e=1}^{E} \xi_e y_e \tag{2.11}$$

subject to

$$\forall d \in \{1, \ldots, D\} : \sum_{p=1}^{P_d} x_{dp} = h_d \qquad \text{demand} \tag{2.12}$$

$$\forall e \in \{1, \ldots, E\} : \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \leq y_e \qquad \text{capacity} \tag{2.13}$$

$$\mathbf{x} \geq 0 \qquad \text{variables} \tag{2.14}$$

$$\mathbf{y} \geq 0 \tag{2.15}$$

Depending on whether link capacites are known (fixed) or unknown (to be chosen), $c_i$ and $y_i$ are used, respectively. Problems with unknown link capacities are referred to as *dimensioning problems* or *uncapacitated problems*, contrary to *capacitated problems* where link capacities are known. When variables can take continuous values, then for any optimal solution, the capacity constraints become equalities, as otherwise cost would arise for unused capacity (which is never optimal).

The *cost* of a path $P_{dp}$ is given by:

$$\zeta_{dp} = \sum_{e=1}^{E} \delta_{edp}\xi_e \qquad d \in \{1,\ldots,D\}, p \in \{1,\ldots,P_d\} \tag{2.16}$$

**Shortest-Path Allocation Rule for Dimensioning Problems:** For each demand, allocate its entire demand to its shortest path with respect to link costs and candidate paths. If there is more than one shortest path for a given demand, then the demand volume can be arbitrarily split among shortest paths.

This rule works for simple dimensioning problems, but might not work if further constraints are to be taken into account. Further constraints might very well be imposed on the problem:

- *Non-bifurcated flows* require each demand to be satisfied by exactly one path flow.
- To ensure graceful degradation in case of node or link failures, flows might need to be partitioned among several node-disjoint paths.

Depending on the demands and capacities, non-bifurcated solutions might not even be possible, although bifurcated solutions exist.

## 2.4. Capacitated Problems

In some cases, link capacities are given and the task is to find a solution that satisfies the specified demands, while staying within the capacity bounds. Such problems can be formulated in the following general notation:

$$\forall d \in \{1,\ldots,D\} : \sum_{p=1}^{P_d} x_{dp} = h_d \qquad \text{demand constraints} \tag{2.17}$$

$$\forall e \in \{1,\ldots,E\} : \sum_{d=1}^{D}\sum_{p=1}^{P_d} \delta_{edp}x_{dp} \leq c_e \qquad \text{capacity constraints} \tag{2.18}$$

$$\mathbf{x} \geq 0 \qquad \text{constraints on variables} \tag{2.19}$$

Sometimes there might be no objective function, rendering any feasible solution acceptable. If flow routing cost is to be minimized, these problems are similar to the first problem.

## 2.5. Shortest-Path Routing

Shortest-Path routing is commonly used in networks. Thus, the network design needs to anticipate that demands will only be routed on their shortest paths. The *length* of the path is determined by adding up link costs $w_e$ according to some weight system $\mathbf{w}$.

Setting the link capacities to be equal to the computed link loads of the shortest-path routing solution gives a (trivially) feasible solution. In general, however, the objective is to find a solution that allows to respect the given link capacities and instead looks for the appropriate weight system.

**Single Shortest Path Allocation Problem** For given link capacities $\mathbf{c}$ and demand volumes $\mathbf{h}$, find a link weight system $\mathbf{w}$ such that the resulting shortest paths are unique and the resulting flow allocation vector is feasible.

This problem is usually complex, as non-bifurcated solutions may not exist even though bifurcated solutions do, non-bifurcated solutions (if they exist) are usually hard to determine and a weight system inducing an existing single-path flow solution might be impossible to find.

If there are multiple shortest paths, one might be interested in splitting demand volumes amoung multiple shortest paths. Such a rule which is used in OSPF routing is the *equal-cost multi-path* (ECMP) rule, aiminng to equally split the outgoing demand volume over all outgoing next hops with equal cost for a fixed destination. However, such a simple might fail if link weights are not set appropriately.

## 2.6. Fair Networks

In the Internet, demands are often not fixed but *elastic*, meaning that each demand can consume any bandwidth assigned to its path. In such a case, capacity constraints are given, for the demands $h_d$ no particular values are assumed.

An obviously initial solution is to assign each demand volume on its lower bound. If this does not satisfy the capacity constraints, there is no feasible solution at all. If, however, feasibility is assured, being able to carry more than the minimum required bandwidth while at the same time giving a fair share of bandwidth to all flows might be desired.

The best-known general fairness criterion is *Max-Min-Fairness* (MMF), also called *equity*:

- If no lower bounds are specified, assign the same maximal value to all demands.
- If there is still capacity left, assign the same maximal value to all demands that can still make use of that capacity.

One might be interested in a compromise between MMF and greedily maximizing network throughput. One such fair allocation principle is *Proportional Fairness* (PF) and is realized by maximizing a logarithmic revenue function:

$$\forall d \in \{1, \ldots, D\} \, \forall p \in \{1, \ldots, P_d\} : \mathbf{R}(x) = \sum_d r_d \ln \left( \sum_p x_{dp} \right) \tag{2.20}$$

with $r_d$ being the revenue associated with demand $d$. If all demands are of equal importance, then $r_d = 1$ for all demands $d$. This function is no longer linear. However, it

ensures that no demand is allocated an overall path flow sum of 0 and makes assigning (unfairly) high values "unattractive". By introducing a linear approximation of **R**, the PF problem can be solved, as is shown later.

Solving the MMF capacitated problem is more complicated, since in general it is not enough to find a flow allocation vector that maximizes the minimal flow $X_d$ over all demands $d$. Even if such a flow vector $X$ is found, then in general some link capacities might still be free and can be used to increase flow allocations for at least a subset of demands.

## 2.7. Topological Design

When installing a link in a network, there is usually a fixed cost that is independent of the capacity of the link (e. g. cabling cost). In order to account for this, such an *opening cost* $\kappa_e$ needs to be modeled in the objective function (which is to be minimized):

$$\mathbf{F} = \sum_e \xi_e y_e + \sum_e \kappa_e u_e \tag{2.21}$$

where $u_e$ is a binary variable indicating whether link $e$ is installed or not.

To force the capacity $y_e$ to be 0 whenever the link $e$ is not installed, a large additional constant $\Delta$ together with additional constraints is introduced,

$$\forall e \in \{1, \ldots, E\} : y_e \leq \Delta u_e \tag{2.22}$$

## 2.8. Restoration Design

So far, the network was always considered to be in operational state, without accounting for link or node failures. Now, let's assume the following failure model:

- Links can be either fully functional or completely failed
- No more than one link fails at a time
- Failure state $s$ ($s \in \{1, \ldots, |E|\}$) indicates that $s$ links have failed

To solve the *restoration design problem* (RDP), additional indexes $s$ are introduced to the path flow variables $x_{dps}$, referring to that particular flow in case of failure state $s$. This also requires reformulating the capacity constraints, e. g.:

$$s = 0 : \qquad x_{120} + x_{310} \leq y_1 \tag{2.23}$$

$$s = 1 : \qquad x_{121} + x_{311} \leq 0 \tag{2.24}$$

$$s = 2 : \qquad x_{122} + x_{312} \leq y_1 \tag{2.25}$$

Additionally, the notation $\alpha_{es}$ is introduced, indicating whether link $e$ is up or not obtaining the following constraints:

$$\forall s \in \{0, \ldots, S\} \, \forall e \in \{1, \ldots, E\} : \sum_d \sum_p \delta_{edp} x_{dps} \leq \alpha_{es} y_e \tag{2.26}$$

A robust network can be considerably more expensive than the cheapest network without failure considerations.

## 2.9. Intra-Domain Traffic Engineering for IP Networks

In this scenario, intra-domain routing is operated by an ISP that has control over the network topology, routing algorithm and link weight system. Due to service level agreements or experience obtained via measurements, the ISP knows the demands between nodes of their network. A common objective of intra-domain routing optimization is to minimize the (average) delay experience by data packets. Thus, the goal is to minimize the maximum utilization over all links.

A commonly used intra-domain routing protocol is OSPF, which is based on Dijkstra's algorithm, which calculates shortest paths based on some weight system $\mathbf{w}$. Thus, the goal is to identify a weight system $\mathbf{w}$ such that the maximum link utilization of the network is minimized while satisfying all given demands and staying within capacity constraints. This results in path flows and total link loads begin defined with respect to $\mathbf{w}$, as these are now induced by the weight system influencing how OSPF distributes traffic:

$$\forall d \in \{1, \ldots, D\} : \sum_{p} x_{dp}(\mathbf{w}) = h_d \tag{2.27}$$

$$\forall e \in \{1, \ldots, E\} : \underline{y}_e(\mathbf{w}) = \sum_{d} \sum_{p} \delta_{edp} x_{dp}(\mathbf{w}) \leq c_e \tag{2.28}$$

The maximum $r$ over all link utilizations can be computed and is needed to ensure that all link loads stay below $c_e r$:

$$r = \max \left\{ \frac{\underline{y}_e(\mathbf{w})}{c_e} \,\middle|\, e = 1, \ldots, E \right\} \tag{2.29}$$

$$\forall e \in \{1, \ldots, E\} : \underline{y_e}(\mathbf{w}) = \sum_{d} \sum_{p} \delta_{edp} x_{dp}(\mathbf{w}) \leq c_e r \tag{2.30}$$

This leads to the following optimization problem:

$$\min \mathbf{F} = r \tag{2.31}$$

subject to

$$\forall d \in \{1, \ldots, D\} : \sum_{p=1}^{P_d} x_{dp}(\mathbf{w}) = h_d \tag{2.32}$$

$$\forall e \in \{1, \ldots, E\} : \underline{y}_e(\mathbf{w}) = \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp}(\mathbf{w}) \leq c_e r \tag{2.33}$$

$$r = \max \left\{ \frac{\underline{y}_e(\mathbf{w})}{c_e} \,\middle|\, e = 1, \ldots, E \right\} \tag{2.34}$$

With $r$ being continuous and $w_e$ being non-negative integers.

For this to work, $k$ shortest paths for every attempted weight system vector $\mathbf{w}$ need to be found. If $r^* < 1$, no link will be overloaded. However, if $r^*$ is close to 1, congestion is likely to occur. For $r^* > 1$ there is at least one overloaded link, i. e. the demands can not be satisifed appropriately.

## 2.10. Tunnel Optimization for MPLS Networks

*Multi-Protocol Label Switching* (MPLS) is an approach that introduces virtual connections into packet switched networks in order to speed up processing times for routers and allow for traffic engineering. In order to transport traffic in an MPLS network, a so-called *label switched path* is set up from the source (ingess MPLS node) to the destination (egress MPLS node). Tunneling (by making use of label stacking) can be used to handle "similar" traffic in an aggregated way, allowing for different traffic capabilities like putting traffic of similar QoS classes into the same tunnels for special treatment and easy re-routing in case of congestion or link failures.

In order not to overload routers with too many tunnels, which would increase the processing overhead, it is desirable to limit the number of tunnels per router and/or link. Thus the optimization challenge is to carry different traffic classes in an MPLS network through the creation of tunnels such that the number of tunnels per node/link is minimized and the load is balanced amoung routers/links. For this, the same notation as before can be used, with $x_{dp}$ now denoting the fraction of demand that is routed over path $P_{dp}$, resulting in the demand constraint:

$$\forall d \in \{1, \ldots, D\} : \sum_{p=1}^{P_d} x_{dp} = 1 \tag{2.35}$$

Note that $x_{dp} \in [0, 1]$ and the absolute flow transported is now $h_d \cdot x_{dp}$.

To avoid path flows with very low fractions, a lower bound $\varepsilon$ is introduced together with binary variables $u_{dp}$ indicating whether the lower bound is satisfied or not:

$$\forall d \in \{1, \ldots, D\} \, \forall p \in \{1, \ldots, P_d\} : \varepsilon u_{dp} \leq x_{dp} \tag{2.36}$$

$$\forall d \in \{1, \ldots, D\} \, \forall p \in \{1, \ldots, P_d\} : x_{dp} \leq u_{dp} \tag{2.37}$$

Capacity feasibility constraints:

$$\forall e \in \{1, \ldots, E\} : \sum_{d=1}^{D} h_d \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \leq c_e \tag{2.38}$$

The number of tunnels on link $e$ will be:

$$\sum_{d} \sum_{p} \delta_{edp} u_{dp} \tag{2.39}$$

The goal is now to minimize the number $r$ representing the maximum number of tunnels over all links:

$$\min \mathbf{F} = r \tag{2.40}$$

subject to

$$\forall d \in \{1, \ldots, D\} : \sum_{p=1}^{P_d} x_{dp} = 1 \tag{2.41}$$

$$\forall e \in \{1, \ldots, E\} : \sum_{d=1}^{D} h_d \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \leq c_e \tag{2.42}$$

$$\forall d \in \{1, \ldots, D\} \, \forall p \in \{1, \ldots, P_d\} : \varepsilon u_{dp} \leq x_{dp} \tag{2.43}$$

$$\forall d \in \{1, \ldots, D\} \, \forall p \in \{1, \ldots, P_d\} : x_{dp} \leq u_{dp} \tag{2.44}$$

$$\forall e \in \{1, \ldots, E\} : \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} u_{dp} \leq r \tag{2.45}$$

and $x_{dp}$ continuous and non-negative, $u_{dp}$ binary and $r$ integer.

This problem has both continuous and binary variables, whilch the constraints and objective function are linear. It is an example for a *mixed-integer linear programming problem* (MIP) . Finding exact solutions for MIP is more difficult than for linear optimization problems, branch-and-bound and banch-and-cut being established techniques to solve such problems.

# A. Letter Salad Decryption Manual

In case one does no longer know which letter means what, here is a semi-comprehensive list of letters used throughout the lecture:

$v$ Node

$e$ Edge/Link

$P$ Path

$\delta$ Indicator wheter a link is on a path

$x$ Flow

$y$ Link capacity (uncapacitated problems)

$c$ Link capacity (capacitated problems)

$h$ Demand

$\xi$ Link cost

$w$ Weight

$\kappa$ Opening cost

$s$ Failure state

$\alpha$ Link up in failure state

$u$ binary variable

$\varepsilon$ Lower bound

# Stichwortverzeichnis