Traffic Engineering of Unicast Services

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2023/2024

Traffic engineering of unicast services

A unicast service is defined by a set of point-to-point traffic flows on a given telecommunication network.

- Consider a network composed by a set of point-to-point links and supporting one unicast service defined by a set of traffic flows T, such that the packets of all flows have the same statistics.
  - The network is modelled by a graph $G=(N,A)$. Set N is the set of network nodes. Set A is the set of network links: the arc $(i,j) \in A$ represents the link between nodes $i \in N$ and $j \in N$ from i to j whose capacity is given by c_{ij} in bps (usually $c_{ij} = c_{ji}$).
  - Each traffic flow $t \in T$ is defined by its origin node o_t, destination node d_t, average throughput from origin to destination b_t (in bps) and average throughput from destination to origin $\underline{b_t}$ (in bps).
  - For each flow $t \in T$, P_t is the set of the candidate routing paths in graph G from its origin node o_t to its destination node d_t.

The traffic engineering task is the task of choosing for each flow $t \in T$ the percentage of its average throughput that must be routed through each of its candidate routing paths of P_t in each direction.

Traffic engineering with single path routing

- In single path routing, each traffic flow must be routed through one single path (no flow bifurcation is allowed).
- Symmetrical routing might be required or not; when required, the routing path from a node $j \in N$ to a node $i \in N$ must use the same links as the routing path from node $i \in N$ to node $j \in N$.

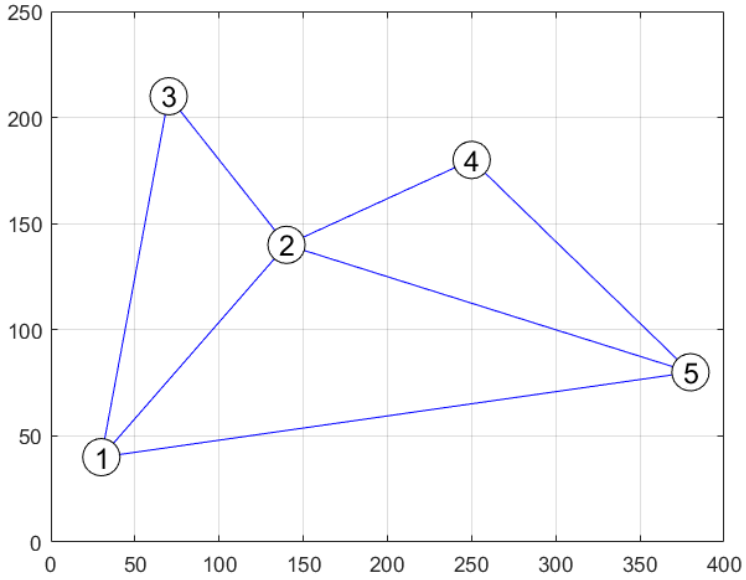
Consider a binary variable x_{tp} associated to each traffic flow $t \in T$ and each routing path $p \in P_t$ that, when is 1, indicates that traffic flow t is routed through path p.

Any traffic engineering solution with single path routing must be compliant with the following constraints:

- For each flow $t \in T$, one of its associated variables x_{tp} must be 1 and all other associated variables must be 0.
- At each arc $(i,j) \in A$, the sum of the throughput values (either b_t or $\underline{b}_t$) of all flows routed through it cannot be higher than its capacity c_{ij}.

Example

Example network:



All links with 10 Gbps of capacity
(in general, these values can be different)

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

In general, these values are different

Flow 1:
Path 1 = 1 2 4
Path 2 = 1 3 2 4
Path 3 = 1 5 4
Path 4 = 1 2 5 4
Path 5 = 1 3 2 5 4
Path 6 = 1 5 2 4

Flow 2:
Path 1 = 1 5
Path 2 = 1 2 5
Path 3 = 1 2 4 5
Path 4 = 1 3 2 5
Path 5 = 1 3 2 4 5

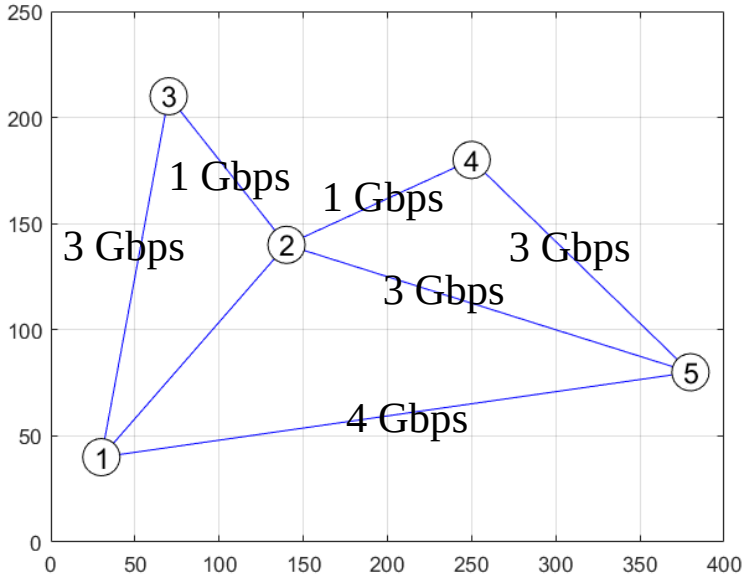
Flow 3:
Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Flow 4:
Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

Routing paths ordered from shortest to longest lengths

Example: one possible solution

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- **Path 2 = 1 3 2 4**
- Path 3 = 1 5 4
- Path 4 = 1 2 5 4
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- Path 3 = 1 2 4 5
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

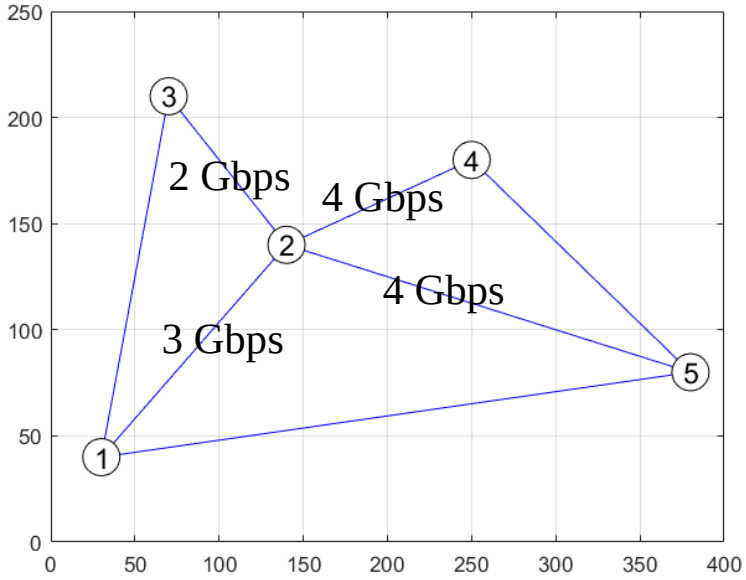
Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- **Path 3 = 3 1 5**
- Path 4 = 3 1 2 5
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

One routing path assigned for each traffic flow

Example: another possible solution

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:

Path 1 = 1 2 4
Path 2 = 1 3 2 4
Path 3 = 1 5 4
Path 4 = 1 2 5 4
Path 5 = 1 3 2 5 4
Path 6 = 1 5 2 4

Flow 2:

Path 1 = 1 5
Path 2 = 1 2 5
Path 3 = 1 2 4 5
Path 4 = 1 3 2 5
Path 5 = 1 3 2 4 5

Flow 3:

Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Flow 4:

Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

Different assignments provide different solutions

Traffic engineering objectives

The traffic engineering task aims to:

- optimize at least one parameter related with either the performance or the operational cost of the network;
- optionally, guarantee (maximum or minimum) values for other parameters.

Examples of optimization parameters:

- the average service packet delay (to minimize the delay performance of the service);
- the worst average packet delay among all traffic flows (to minimize the delay performance fairness among all traffic flows);
- the worst link load (to maximize the robustness of the network to unpredictable traffic growth);
- the energy consumption of the network (to minimize operational costs).

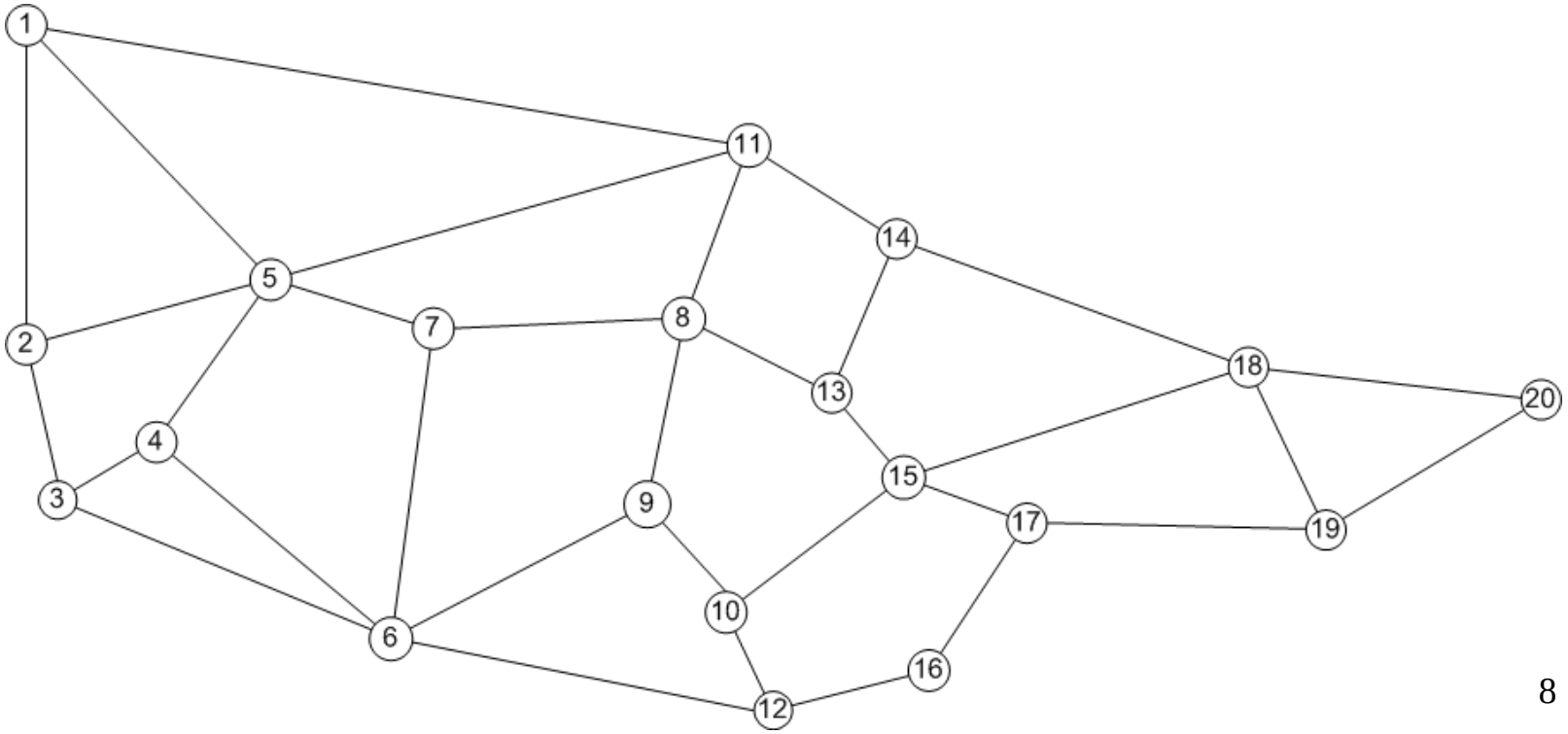
The best traffic engineering solution depends on the optimization objective of interest. Different optimization objectives might be conflicting:

- for example, to reduce the energy consumption, more links must be put in sleeping mode; consequently, the same traffic is concentrated in less links and the worst link load increases.

Example - network

Consider the following network with 20 routers and 33 links where all links have a capacity of 1 Gbps.

The length of the links varies between 88 km (between nodes 10 and 12) and 759 km (between nodes 1 and 11) and the link propagation delay is determined by the speed of light over a optical fibre (approximately 2 x 10^8 meters/s).



Example – traffic flow matrix

Consider the following flow matrix (values in Mbps) where the packets of all flows are exponentially distributed with an average packet size B = 1000 Bytes:

0.0	47.7	64.4	13.6	10.6	45.5	10.6	12.9	9.8	9.8	13.6	11.4	11.4	41.7	12.9	10.6	9.1	12.9	11.4	22.0
47.0	0.0	68.2	32.6	33.3	189.4	59.8	47.0	49.2	38.6	59.1	53.0	38.6	212.1	31.8	48.5	40.9	43.9	54.5	74.2
65.9	69.7	0.0	8.3	13.6	53.0	13.6	14.4	18.9	14.4	11.4	36.4	12.9	63.6	13.6	14.4	11.4	13.6	15.9	22.7
13.6	46.2	13.6	0.0	12.9	31.1	14.4	12.1	11.4	12.9	31.1	12.1	9.1	31.8	11.4	10.6	14.4	12.9	17.4	24.2
7.6	31.8	10.6	14.4	0.0	55.3	11.4	9.1	9.8	12.9	36.4	11.4	12.9	46.2	12.9	7.6	12.1	15.9	16.7	28.0
52.3	174.2	53.8	55.3	38.6	0.0	39.4	47.0	41.7	40.9	53.8	44.7	42.4	212.1	40.9	71.2	62.9	46.2	56.8	72.0
13.6	32.6	11.4	11.4	12.9	54.5	0.0	7.6	12.1	12.9	12.1	14.4	56.8	55.3	9.8	9.1	13.6	15.9	9.8	21.2
13.6	47.0	14.4	11.4	9.8	37.9	10.6	0.0	12.1	9.1	11.4	13.6	12.1	57.6	9.8	10.6	9.8	17.4	12.9	34.1
9.8	33.3	16.7	12.1	14.4	38.6	12.9	9.8	0.0	11.4	13.6	9.1	13.6	56.1	11.4	13.6	12.1	15.2	18.9	18.9
12.9	53.0	10.6	9.8	11.4	46.2	13.6	10.6	10.6	0.0	9.1	9.8	11.4	42.4	10.6	11.4	10.6	15.9	10.6	25.8
9.1	36.4	9.8	31.1	33.3	40.9	9.8	10.6	12.1	14.4	0.0	9.8	10.6	47.7	9.1	11.4	8.3	9.8	12.9	32.6
10.6	61.4	35.6	9.8	9.8	59.8	14.4	8.3	8.3	10.6	12.1	0.0	9.8	33.3	9.8	28.0	9.8	9.1	14.4	25.0
10.6	32.6	9.1	12.1	9.1	35.6	59.8	10.6	9.8	14.4	9.8	13.6	0.0	41.7	11.4	12.9	13.6	15.9	16.7	40.9
40.9	181.8	49.2	56.1	42.4	189.4	55.3	64.4	57.6	31.8	31.8	33.3	46.2	0.0	40.9	57.6	40.2	48.5	51.5	69.7
12.1	37.1	10.6	9.8	10.6	37.1	8.3	14.4	8.3	10.6	9.8	12.1	11.4	47.7	0.0	11.4	10.6	10.6	9.1	28.8
10.6	47.7	9.8	11.4	11.4	44.7	9.8	11.4	10.6	9.1	9.1	12.9	9.1	56.8	14.4	0.0	11.4	9.1	12.9	30.3
13.6	34.1	10.6	10.6	13.6	55.3	12.1	12.9	9.8	11.4	10.6	12.9	9.1	44.7	10.6	9.1	0.0	10.6	9.8	20.5
13.6	40.9	11.4	9.8	18.9	40.9	11.4	18.2	13.6	18.9	12.9	10.6	17.4	40.9	11.4	12.9	9.8	0.0	34.1	24.2
7.6	49.2	18.9	15.9	12.9	53.0	12.1	9.8	15.9	13.6	15.2	10.6	18.9	47.0	12.9	9.8	9.8	30.3	0.0	18.9
23.5	68.2	26.5	28.8	34.1	65.9	25.8	30.3	15.9	22.7	30.3	28.0	34.1	65.2	34.1	25.8	24.2	21.2	15.9	0.0

Example – one possible solution

One possible solution is to route each traffic flow $t \in T$ by the routing path with the shortest length (minimizing, in this way, the propagation delay of each flow).

Using the Kleinrock approximation, we obtain the following performance parameters:

Worst average packet delay = 6.06 ms

Worst link load = 99.3%

Number of active links = 33 out of 33

However, it is possible to obtain better traffic engineering solutions through appropriate optimization algorithms.

Example – optimal solutions

Minimization of the worst average packet delay:

Worst average packet delay = 5.21 ms
Worst link load = 93.6%
Number of active links = 33 out of 33

Minimization of the worst link load:

Worst average packet delay = 8.63 ms
Worst link load = 69.9%
Number of active links = 33 out of 33

Minimization of the number of active links:

Worst average packet delay = 10.54 ms
Worst link load = 82.4%
Number of active links = 26 out of 33

Conclusion:

- Each traffic engineering solution is a different trade-off between the different optimization objectives.
- It is up to the operator to select the best routing solution.

Optimization methods

Exact methods

- Based on mathematical models (for example, Integer Linear Programming)
- In the general case, computationally hard
- Theoretically, they are able to compute the optimal solutions
- Inefficient for large problem instances (they either take too long to even compute feasible solutions or finish due to out-of-memory)

Heuristic methods

- Based on simple programming algorithms
- Easy to implement and quick to find solutions
- Do not guarantee optimality
- For larger runtimes, they find better solutions (than exact methods)
- Efficient for large problem instances

Heuristic method versus heuristic algorithm

Heuristic method: a generic approach to search for good solutions that can be applied to any optimization problem.

Heuristic algorithm: an optimization algorithm that has resulted from applying an heuristic method to a particular optimization problem.

Many heuristic methods (usually, also the simplest ones) are based on two algorithmic strategies:

1. To build a solution starting from the scratch.
   - Examples: *random*, *greedy, greedy randomized, etc…*

2. To get a better solution from a known solution.
   - Examples: *hill climbing, tabu search, simulated annealing, etc...* (we will address only the hill climbing strategy).

Building a solution from the scratch

Building one solution from the scratch (I)

1. Random strategy:

- The solution is built by assigning a random routing path $p \in P_t$ for each flow $t \in T$
- We might obtain better solutions if we consider higher probabilities to routing paths $p \in P_t$ with “better characteristics”
  - For example, paths with a smaller number of links, paths containing links of larger capacity, etc…

2. Greedy strategy:

- Start by considering the network without any routing path
- Then, for each flow $t \in T$:
  - assign the first routing path $p \in P_t$ that, together with the previous assigned routing paths, gives the best objective function value

Building one solution from the scratch (II)

3. Greedy randomized strategy:

The aim is to obtain a different solution on different runs.

First alternative:

- First, choose a random order of the flows $t \in T$
- Then, apply the greedy strategy (previous slide) by the chosen order

Second alternative:

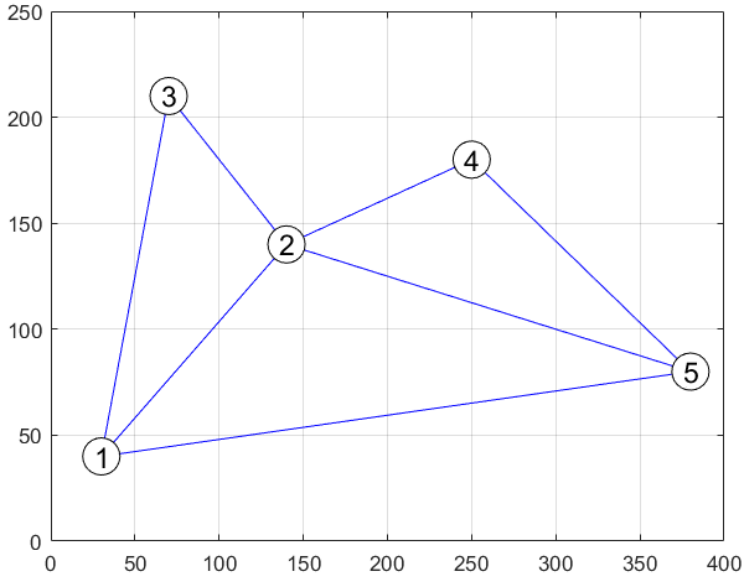
- Start by considering the network without any routing path
- Then, for each flow $t \in T$:
  - compute the α routing paths of P_t that, each one together with the previous assigned routing paths, give the best objective values
    - α is an integer parameter of the algorithm (with $\alpha \geq 2$)
  - assign randomly one of the previous α routing paths to flow $t \in T$

Third alternative:

- To combine the 2 previous alternatives

Minimizing the worst link load: greedy strategy

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:

Path 1 = 1 2 4
Path 2 = 1 3 2 4
Path 3 = 1 5 4
Path 4 = 1 2 5 4
Path 5 = 1 3 2 5 4
Path 6 = 1 5 2 4

Flow 2:

Path 1 = 1 5
Path 2 = 1 2 5
Path 3 = 1 2 4 5
Path 4 = 1 3 2 5
Path 5 = 1 3 2 4 5

Flow 3:

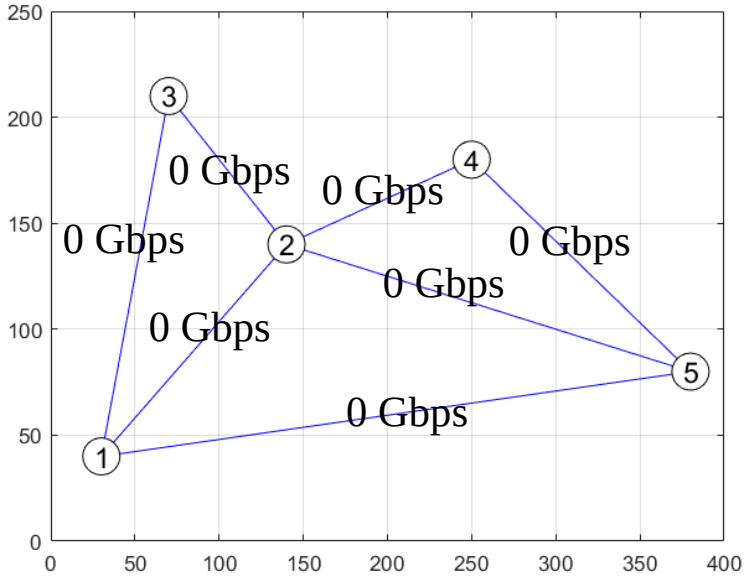
Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Flow 4:

Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

Greedy strategy: step 1

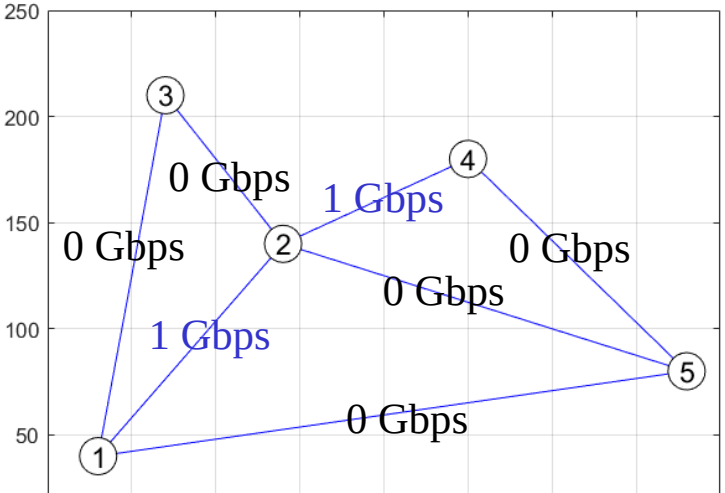
Current solution:



Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Updated solution:



Flow 1:

Path 1 = 1 2 4
Path 2 = 1 3 2 4
Path 3 = 1 5 4
Path 4 = 1 2 5 4
Path 5 = 1 3 2 5 4
Path 6 = 1 5 2 4

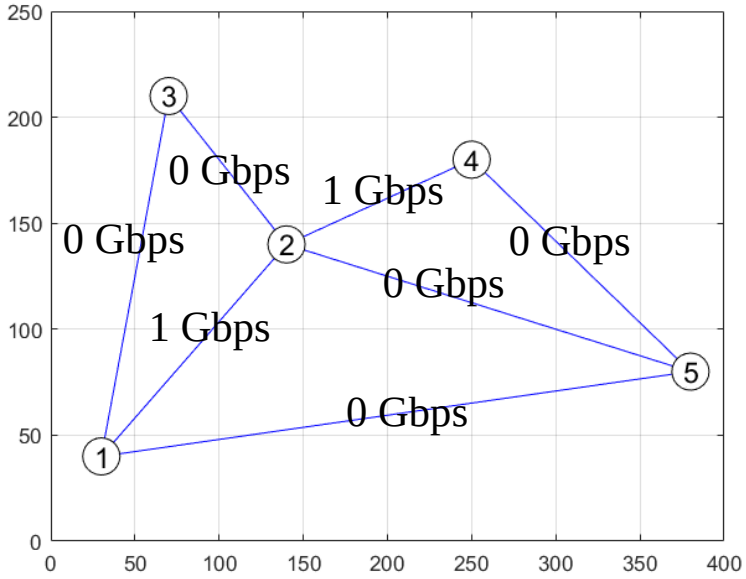
Selected path:

Path 1 = 1 2 4

Path minimizing the worst link load in the updated solution

Greedy strategy: step 2

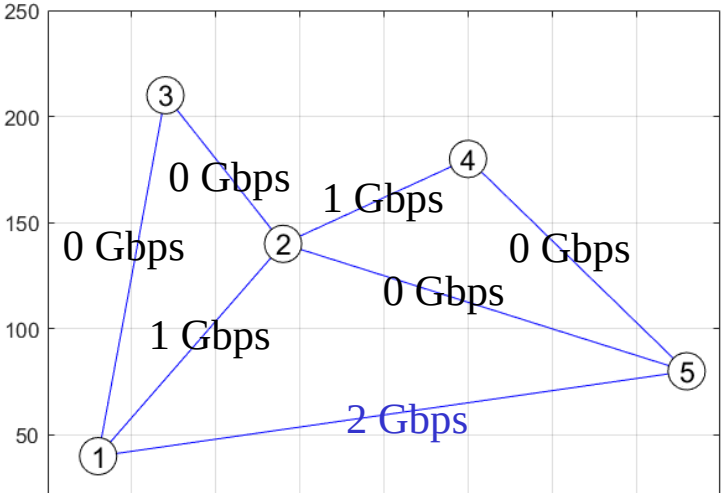
Current solution:



Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Updated solution:



Flow 2:

Path 1 = 1 5
Path 2 = 1 2 5
Path 3 = 1 2 4 5
Path 4 = 1 3 2 5
Path 5 = 1 3 2 4 5

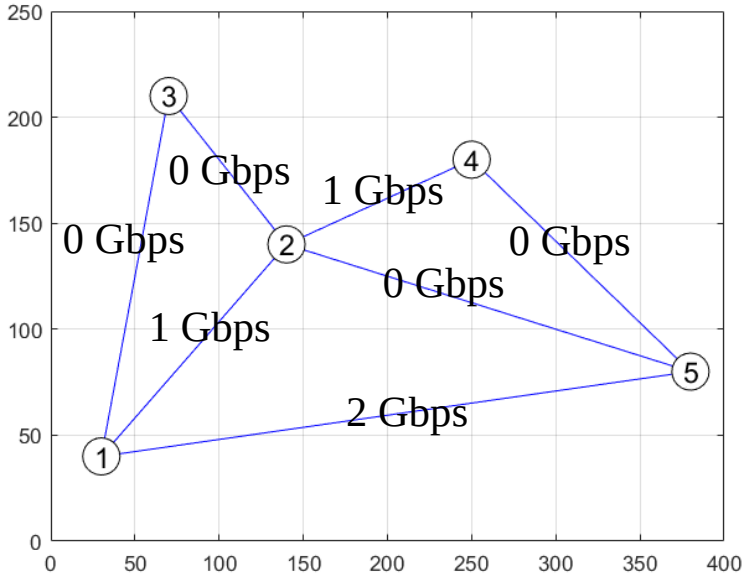
Selected path:

Path 1 = 1 5

Path minimizing the worst link load in the updated solution

Greedy strategy: step 3

Current solution:



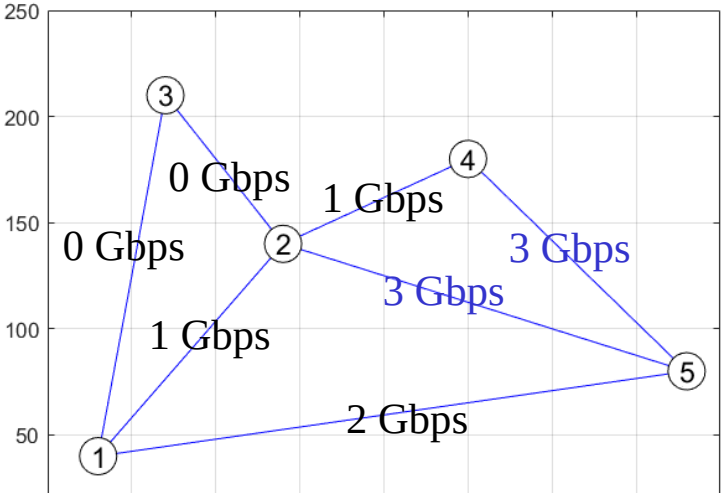
Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 3:

Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Updated solution:



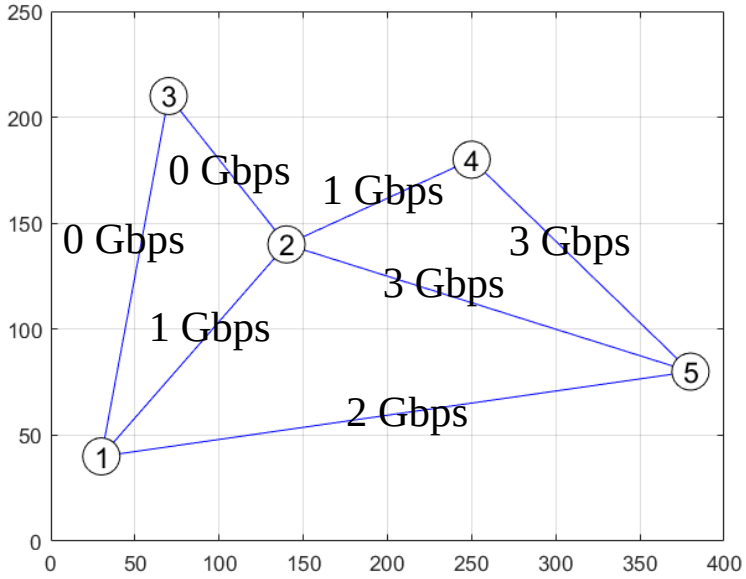
Selected path:

Path 2 = 2 5 4

Path minimizing the worst link load in the updated solution

Greedy strategy: step 4

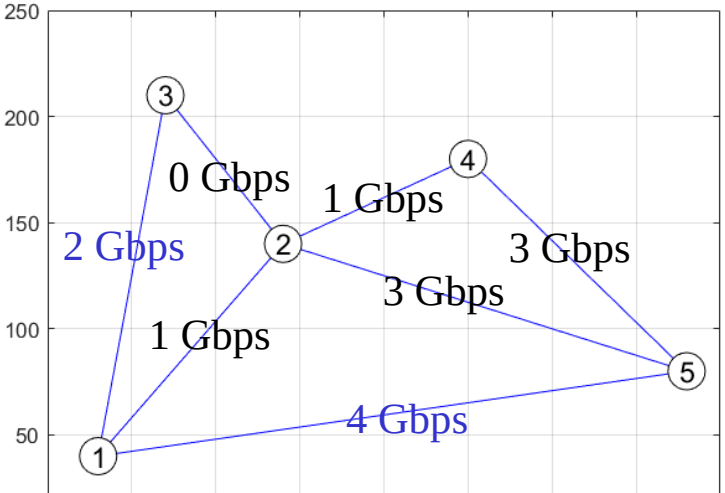
Current solution:



Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Updated solution:



Flow 4:

Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

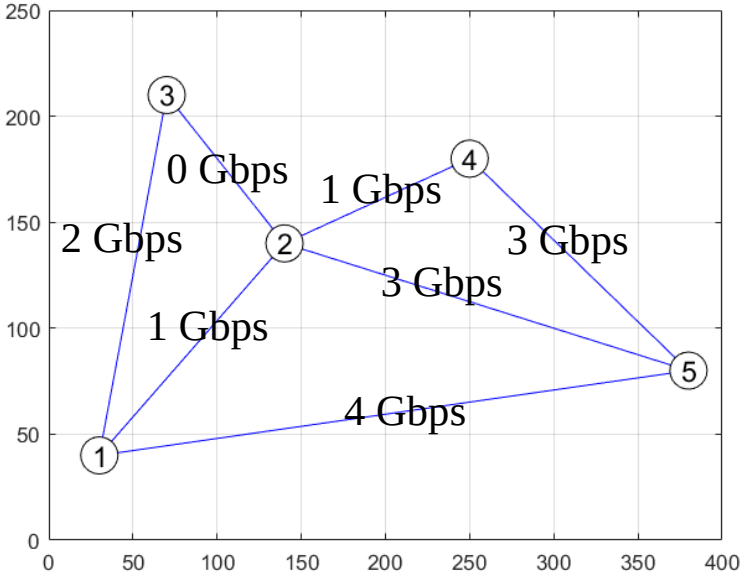
Selected path:

Path 3 = 3 1 5

Path minimizing the worst link load in the updated solution

Greedy strategy: FINAL SOLUTION

Current solution:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

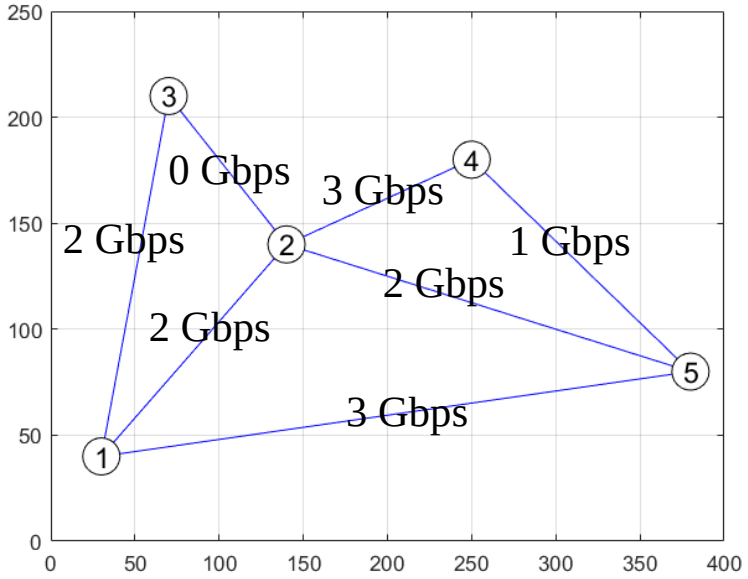
FINAL SOLUTION :

Flow 1:
Path 1 = 1 2 4
Flow 2:
Path 1 = 1 5
Flow 3:
Path 2 = 2 5 4
Flow 4:
Path 3 = 3 1 5

Worst link load →	4 Gbps (= 40%)
Link(s) with the worst link load →	(1,5) and (5,1)
Unused links →	{2,3}

Greedy randomized strategy

Current solution:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Taking the random order 3 → 1 → 2 → 4 :

Flow 3:
Path 1 = 2 4
Flow 1:
Path 3 = 1 5 4
Flow 2:
Path 3 = 1 2 5
Flow 4:
Path 3 = 3 1 5

Worst link load → 3 Gbps (= 30%)
Link(s) with the worst link load → (1,5), (5,1), (2,4) and (4,2)
Unused links → {2,3}

Optimization algorithm

- In a problem aiming to minimize function $F(x)$, it works as follows:

$f_{best} = +\infty$

repeat

$x = BuildSolution\ ()$

$f = F(x)$

if $f < f_{best}$ **then**

$x_{best} = x$

$f_{best} = f$

endif

until Stopping Criteria is met

Random strategy or
Greedy Randomized strategy

If the aim is to maximize $F(x)$
$f_{best} = -\infty$
$f > f_{best}$

- Examples of Stopping Criteria:
  - Run a predefined time duration
  - Run a predefined number of iterations
  - Run until f_{best} not improving a predefined number of iterations

Getting a better solution from a known solution

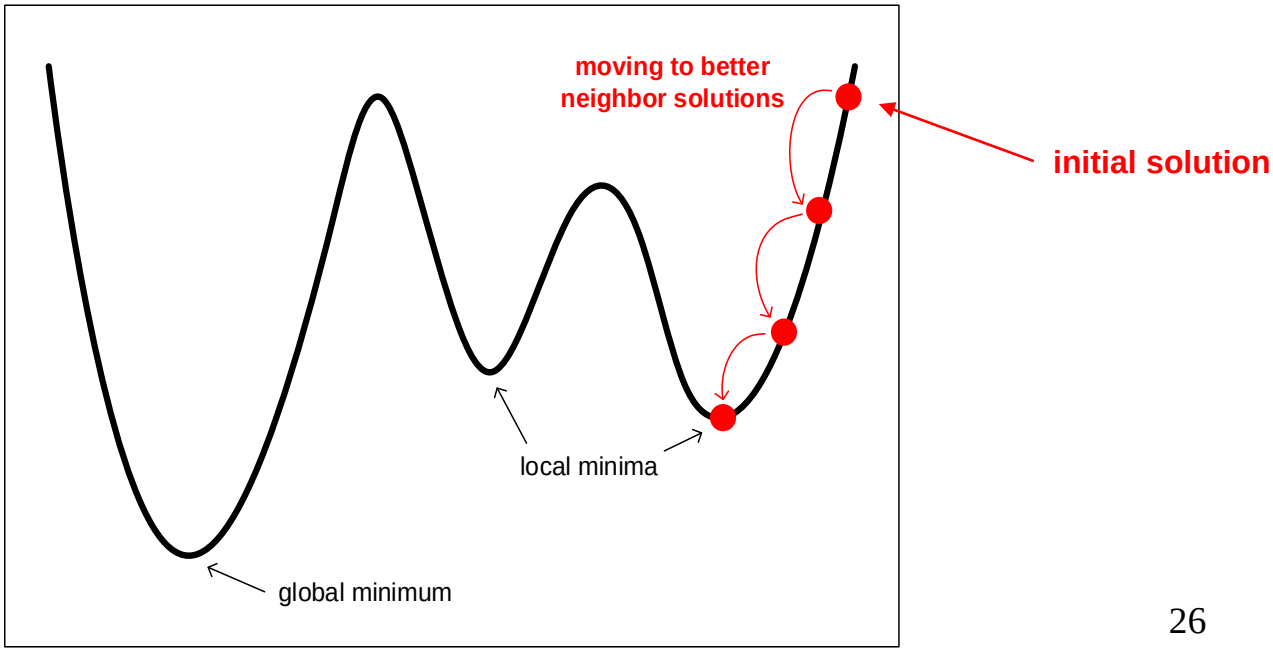
Getting a better solution from a known solution (I)

Hill climbing strategy

Start by an initial solution and try to move to a better solution by making one local change.

- The possible local changes are defined by the neighbour set.
- The moves are repeated until no possible local change produces a better solution (we obtain a local optimum solution).

Minimization problem



Getting a better solution from a known solution (II)

Hill climbing strategy – best neighbour move variant

In this variant, all neighbour solutions are evaluated at each iteration. It provides the best possible improvement on each move.

This strategy is defined as an iterative procedure with the following steps:

1. Set the current solution with the initial solution.
2. For the current solution, evaluate all neighbour solutions and select the best one.
3. If the best neighbour solution is better than the current solution, move to this solution (*i.e.*, set the current solution with the best neighbour solution) and go to step 2 (*i.e.*, a new iteration starts).
4. If not, stop and the current solution is the final solution (we say it is a local optimum solution).

NOTE: If the initial solution is a local optimum solution, the algorithm stops in the first iteration. In this case, the final solution is the initial solution.

Getting a better solution from a known solution (III)

Hill climbing strategy – first best neighbour move variant

If the evaluation of all neighbour solutions is computationally hard (either because each neighbour solution is hard to evaluate or because the neighbour set has too many solutions), the previous variant might be non efficient.

This strategy variant is defined by the following steps:

1. Set the current solution with the initial solution.
2. For the current solution, evaluate the neighbour solutions until a solution better than the current one is found or until all neighbour solutions are evaluated.
3. If the found neighbour solution is better than the current one, set the current solution with the neighbour solution and go to step 2.
4. If not, stop and the current solution is the final solution.

NOTE: Usually, it is more efficient to use the best neighbour move variant, although in some problems it requires a careful definition of the neighbour set.

Hill climbing algorithm

Consider an optimization problem with a solution set S, an optimization function $F(x)$ and a set $V(x)$ of neighbours of each solution $x \in S$

$x' \leftarrow$ Initial($x \in S$)

$f' \leftarrow F(x')$

improved $\leftarrow$ TRUE

While *improved* **do**

  $x \leftarrow$ Best($x \in V(x')$)

  $f \leftarrow F(x)$

  If f is better than f' **do**

    $x' \leftarrow x$

    $f' \leftarrow f$

  Else do

    improved $\leftarrow$ FALSE

  EndIf

EndWhile

x' is set with an initial known solution and f' is its objective value

the algorithm stops when no improvement can be obtained by a neighbour solution

the best (or first best) neighbour x of solution x' is selected

if $F(x)$ is better than $F(x')$, x becomes the current solution x' and f' its objective value

otherwise, no improvement can be obtained by a neighbor solution

The final result is solution x' whose function value is $f' = F(x')$

Hill climbing strategy: defining the set of neighbour solutions

- The set of neighbour solutions (of a given solution) depends on the addressed optimization problem.
- The neighbour set must be carefully defined in order to allow the algorithm to compute all neighbour solutions in reasonable running time.

Recall that in traffic engineering of telecommunication networks:

- P_t is the set of candidate routing paths of flow $t \in T$
- A solution defines the routing path $p \in P_t$ assigned to traffic flow $t \in T$

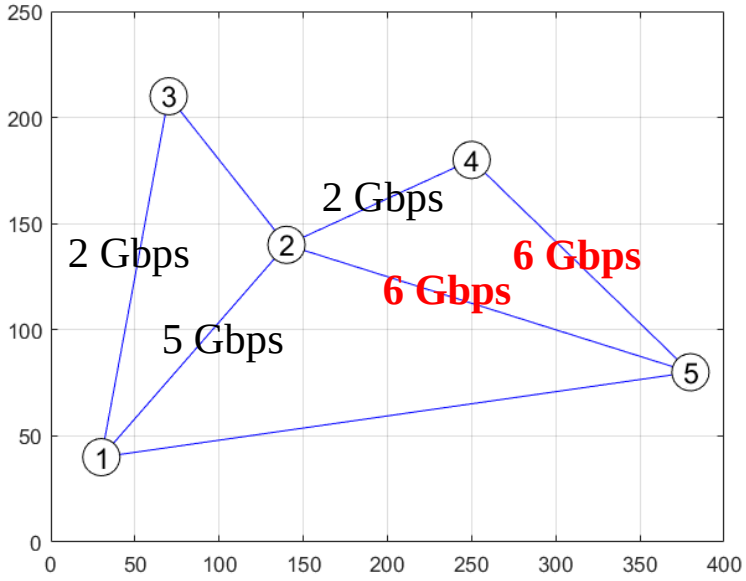
The neighbour set of a current solution is usually defined as:

- all solutions that differ from the given solution in the routing path of a single flow.
- so, the total number of neighbour solutions is: $\sum_{t \in T} (|P_t| - 1)$

  where $|P_t|$ is the number of routing paths of set P_t.

Minimizing the worst link load
Hill climbing: beginning

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
Path 1 = 1 2 4 7
Path 2 = 1 3 2 4 4
Path 3 = 1 5 4 6
Path 4 = 1 2 5 4 5+6+6
Path 5 = 1 3 2 5 4 2+6+6
Path 6 = 1 5 2 4 6+2

Flow 2:
Path 1 = 1 5
Path 2 = 1 2 5 5+6
Path 3 = 1 2 4 5 5+2+6
Path 4 = 1 3 2 5 2+6
Path 5 = 1 3 2 4 5 2+2+6

Flow 3:
Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Flow 4:
Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

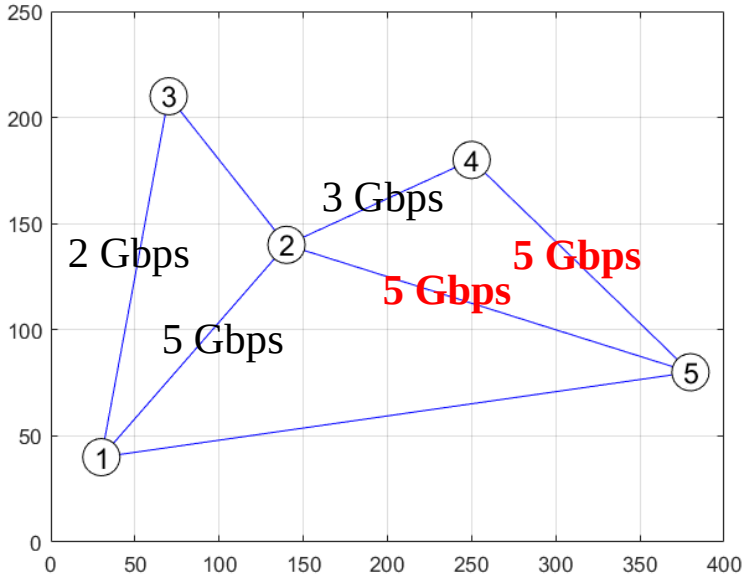
Worst link load: 6 Gbps

Current solution:
Flow 1 – Path 4
Flow 2 – Path 3
Flow 3 – Path 2
Flow 4 – Path 4

Current solution is the initial known solution

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- **Path 1 = 1 2 4**
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- Path 1 = 1 5
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

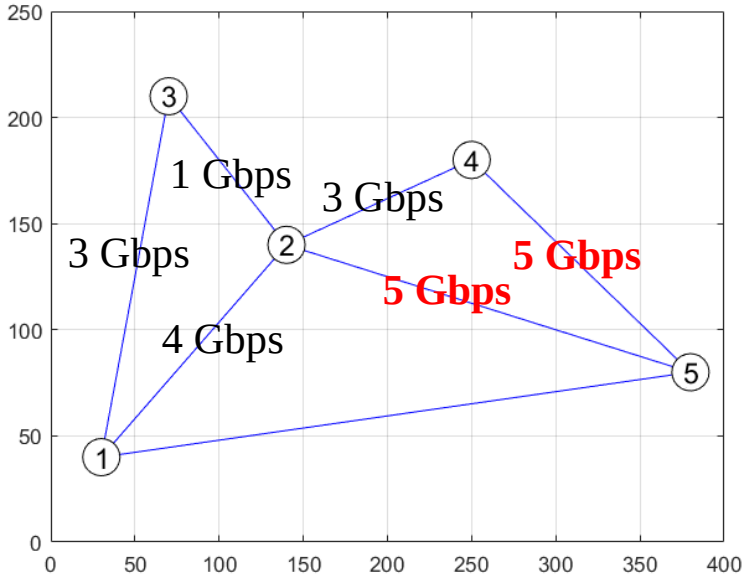
Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Better neighbour solution

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- **Path 2 = 1 3 2 4**
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- Path 1 = 1 5
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

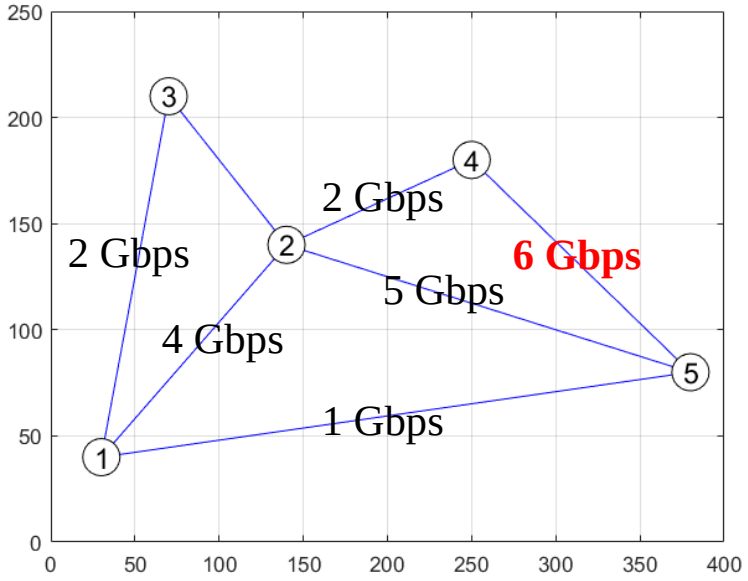
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- **Path 3 = 1 5 4**
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- Path 1 = 1 5
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

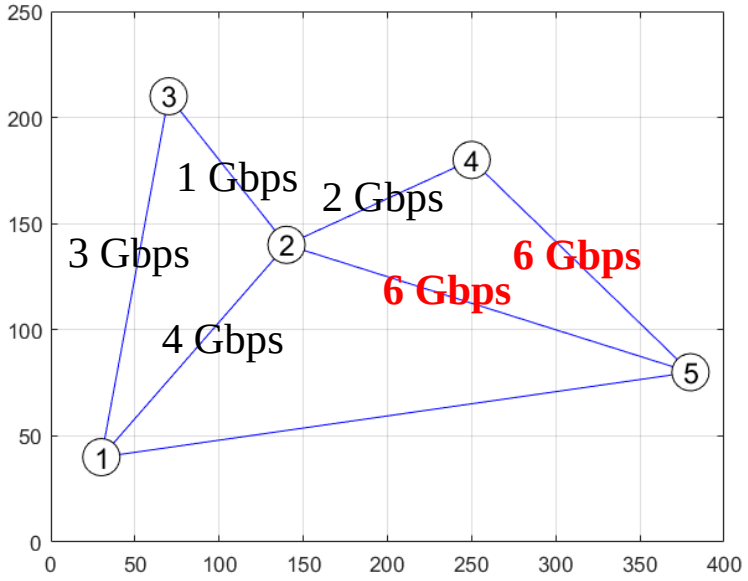
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- **Path 5 = 1 3 2 5 4**
- Path 6 = 1 5 2 4

Flow 2:
- Path 1 = 1 5
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

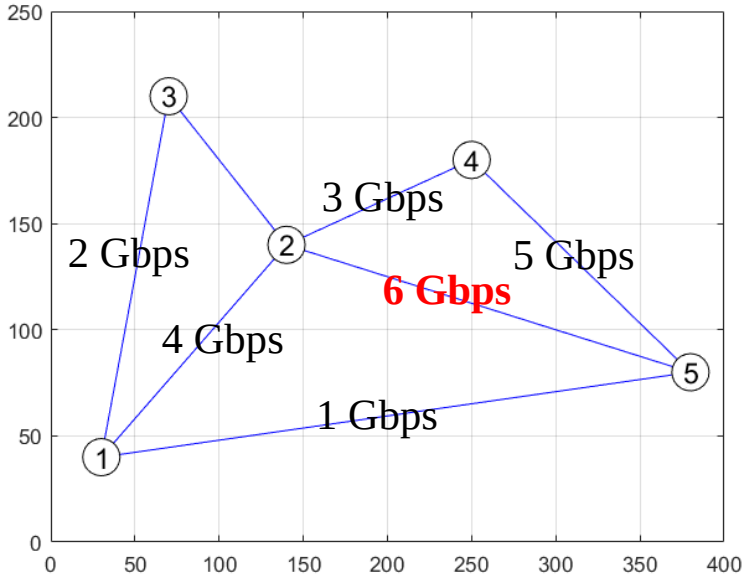
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- **Path 6 = 1 5 2 4**

Flow 2:
- Path 1 = 1 5
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

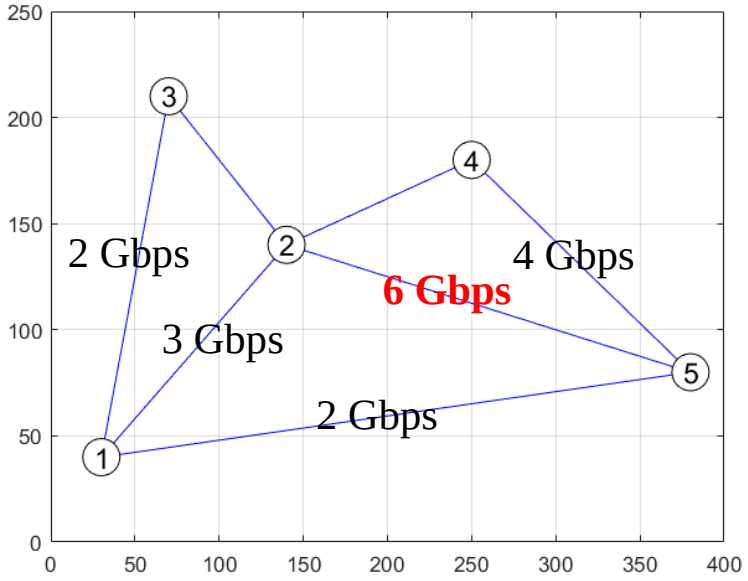
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

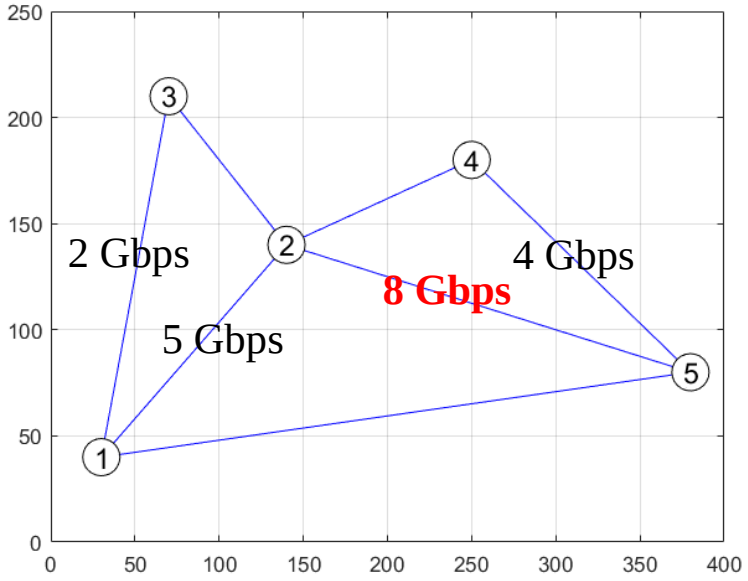
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- **Path 2 = 1 2 5**
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

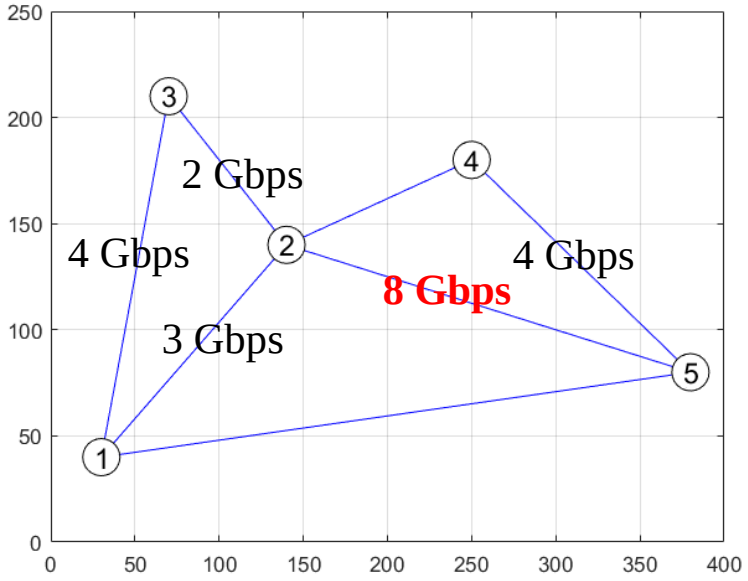
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- **Path 4 = 1 3 2 5**
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

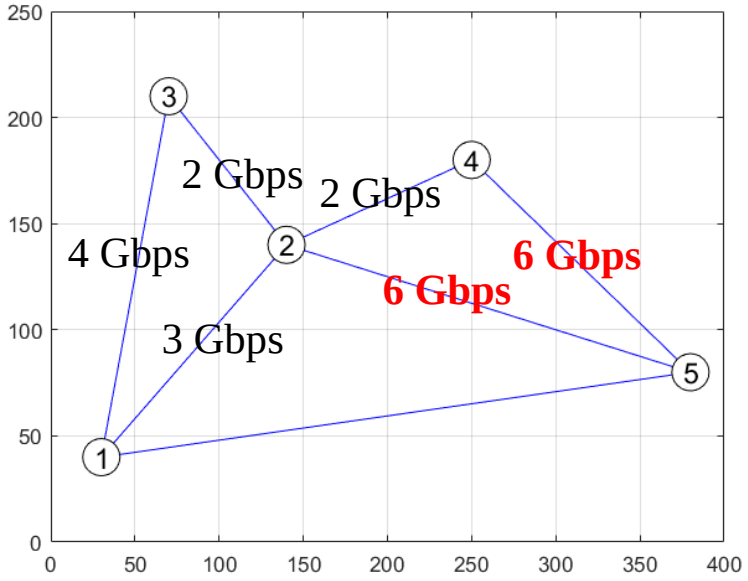
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- **Path 5 = 1 3 2 4 5**

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

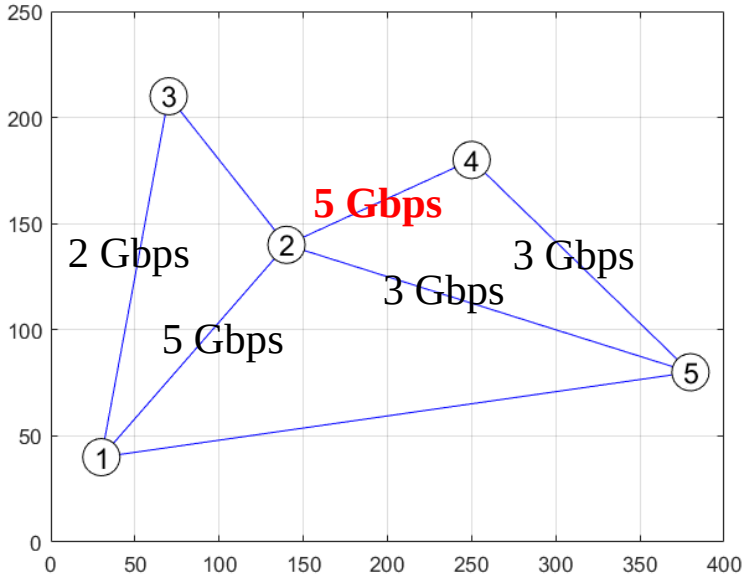
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- **Path 1 = 2 4**
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

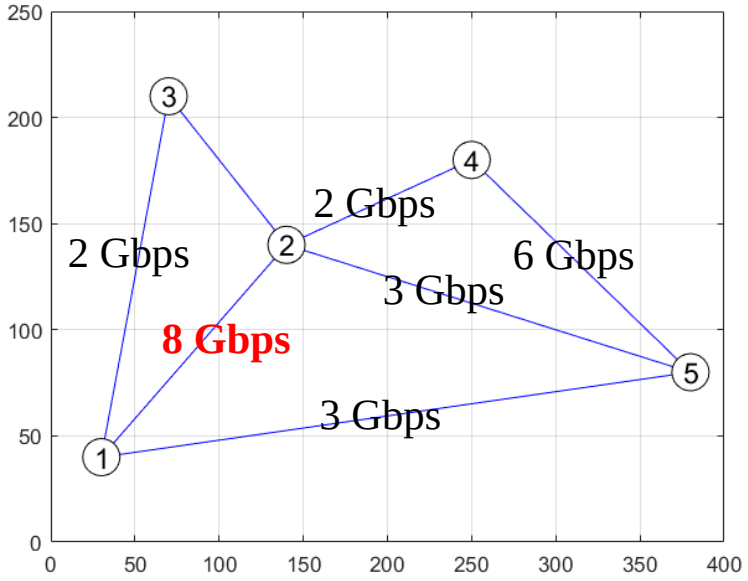
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- **Path 3 = 2 1 5 4**
- Path 4 = 2 3 1 5 4

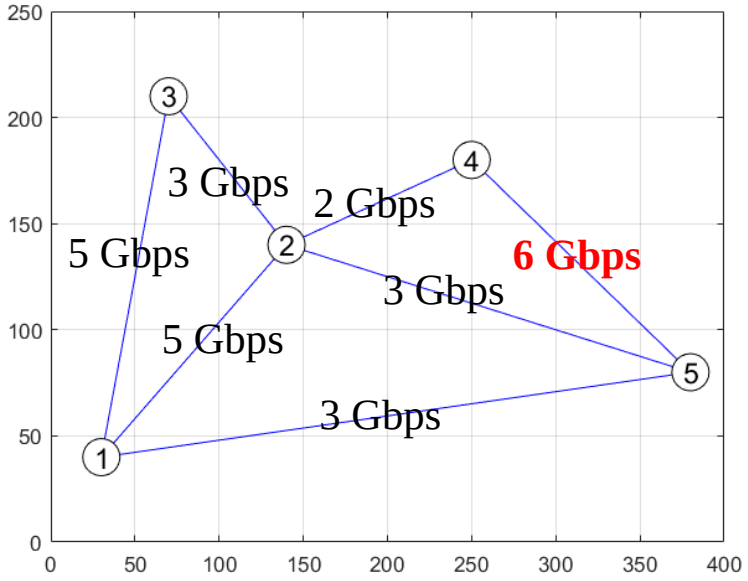
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- **Path 4 = 2 3 1 5 4**

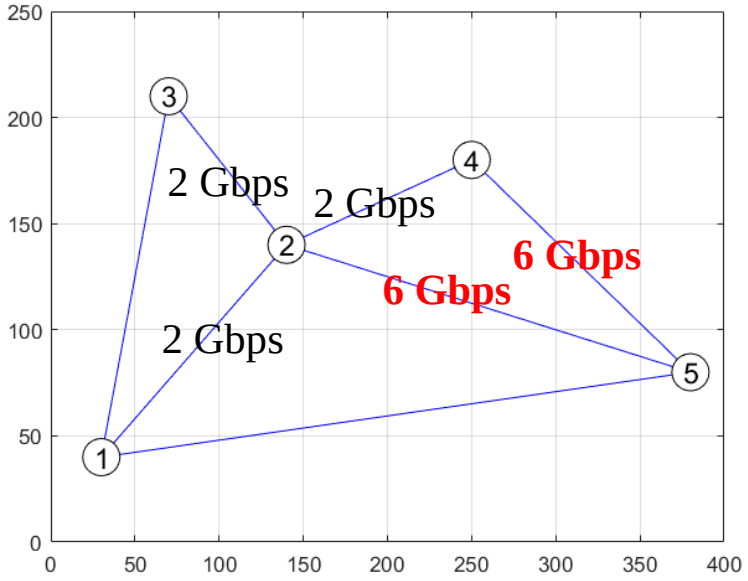
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

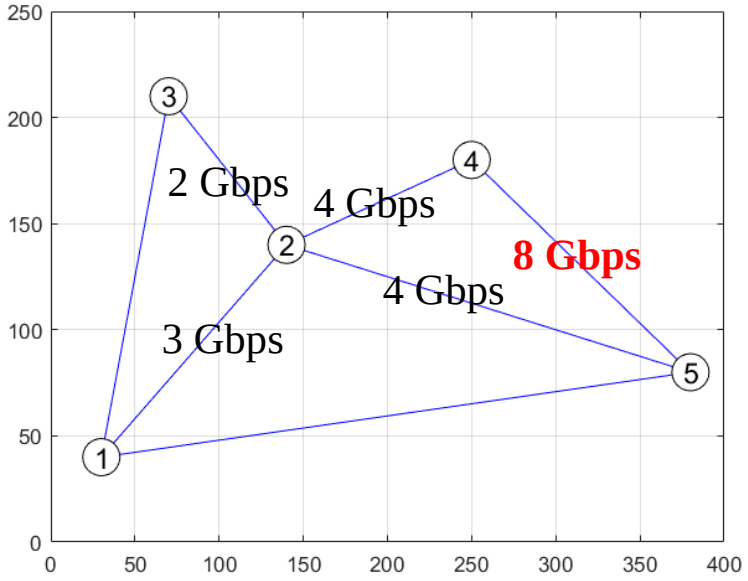
Flow 4:
- **Path 1 = 3 2 5**
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

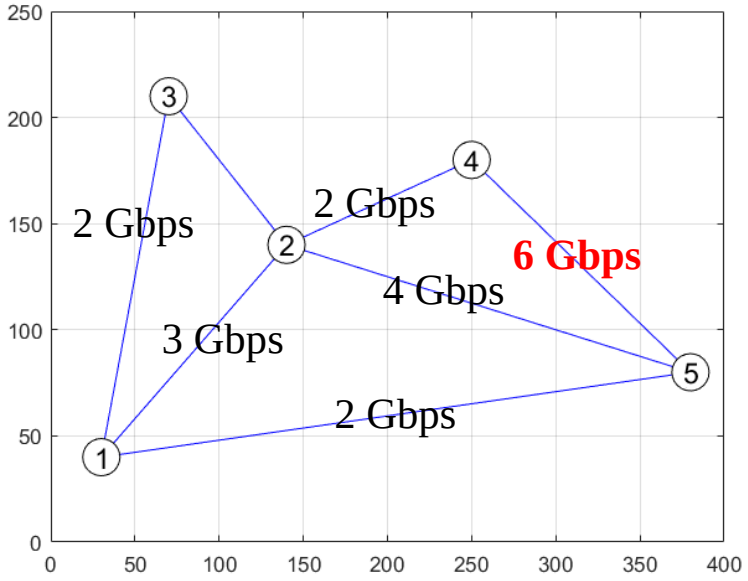
Flow 4:
- Path 1 = 3 2 5
- **Path 2 = 3 2 4 5**
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

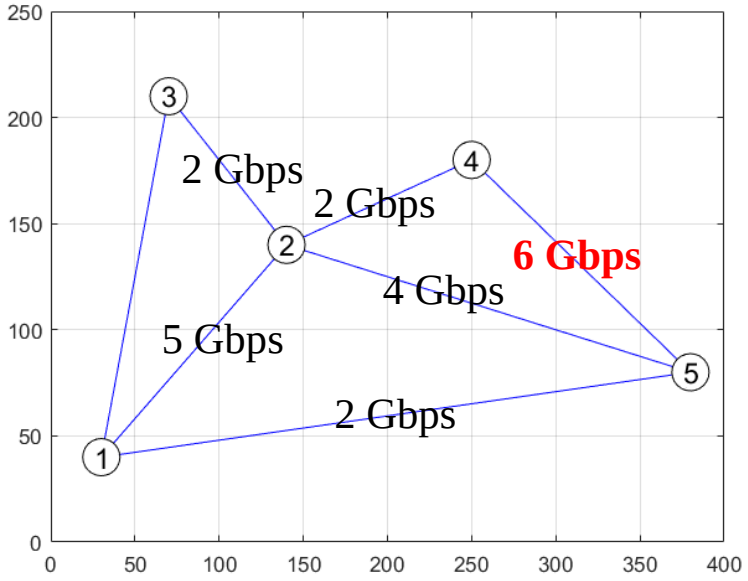
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- **Path 3 = 3 1 5**
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

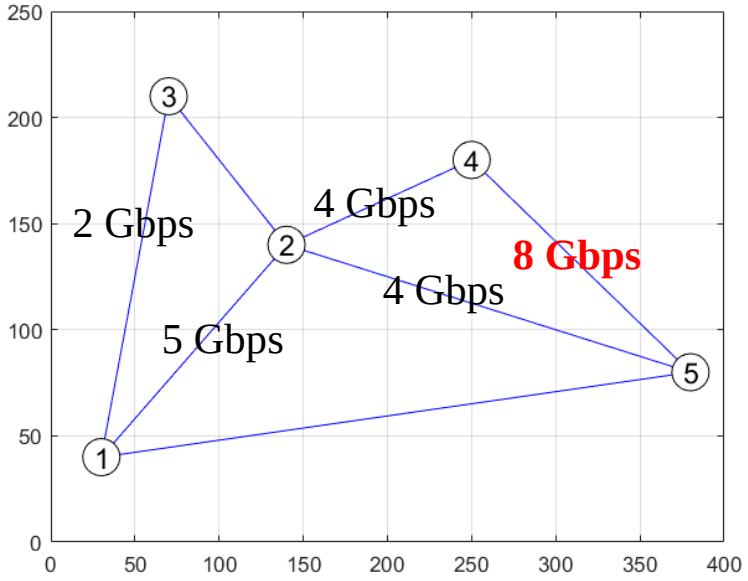
Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- **Path 5 = 3 2 1 5**
- Path 6 = 3 1 2 4 5

Worst link load: **5 Gbps**

Best move so far:
Flow 1: Path 4 →Path1

Minimizing the worst link load
Hill climbing: step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
- Path 1 = 1 2 4
- Path 2 = 1 3 2 4
- Path 3 = 1 5 4
- **Path 4 = 1 2 5 4**
- Path 5 = 1 3 2 5 4
- Path 6 = 1 5 2 4

Flow 2:
- **Path 1 = 1 5**
- Path 2 = 1 2 5
- **Path 3 = 1 2 4 5**
- Path 4 = 1 3 2 5
- Path 5 = 1 3 2 4 5

Flow 3:
- Path 1 = 2 4
- **Path 2 = 2 5 4**
- Path 3 = 2 1 5 4
- Path 4 = 2 3 1 5 4

Flow 4:
- Path 1 = 3 2 5
- Path 2 = 3 2 4 5
- Path 3 = 3 1 5
- **Path 4 = 3 1 2 5**
- Path 5 = 3 2 1 5
- **Path 6 = 3 1 2 4 5**

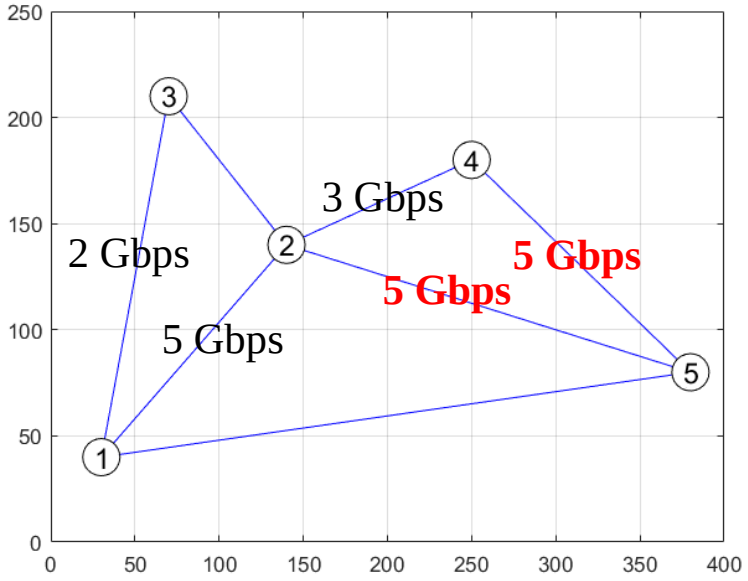
Worst link load: **5 Gbps**

Best move:
Flow 1: Path 4 →Path1

Last neighbour solution

Minimizing the worst link load
Hill climbing: end of step 1

Example network:



All links with 10 Gbps of capacity

Traffic flows:

t	o_t	d_t	b_t (Gbps)	$\underline{b}_t$ (Gbps)
1	1	4	1.0	1.0
2	1	5	2.0	2.0
3	2	4	3.0	3.0
4	3	5	2.0	2.0

Flow 1:
Path 1 = 1 2 4
Path 2 = 1 3 2 4
Path 3 = 1 5 4
Path 4 = 1 2 5 4
Path 5 = 1 3 2 5 4
Path 6 = 1 5 2 4

Flow 2:
Path 1 = 1 5
Path 2 = 1 2 5
Path 3 = 1 2 4 5
Path 4 = 1 3 2 5
Path 5 = 1 3 2 4 5

Flow 3:
Path 1 = 2 4
Path 2 = 2 5 4
Path 3 = 2 1 5 4
Path 4 = 2 3 1 5 4

Flow 4:
Path 1 = 3 2 5
Path 2 = 3 2 4 5
Path 3 = 3 1 5
Path 4 = 3 1 2 5
Path 5 = 3 2 1 5
Path 6 = 3 1 2 4 5

Worst link load: 5 Gbps

Current solution:
Flow 1 – Path 1
Flow 2 – Path 3
Flow 3 – Path 2
Flow 4 – Path 4

Current solution is updated changing Flow 1 from Path 4 to Path 1

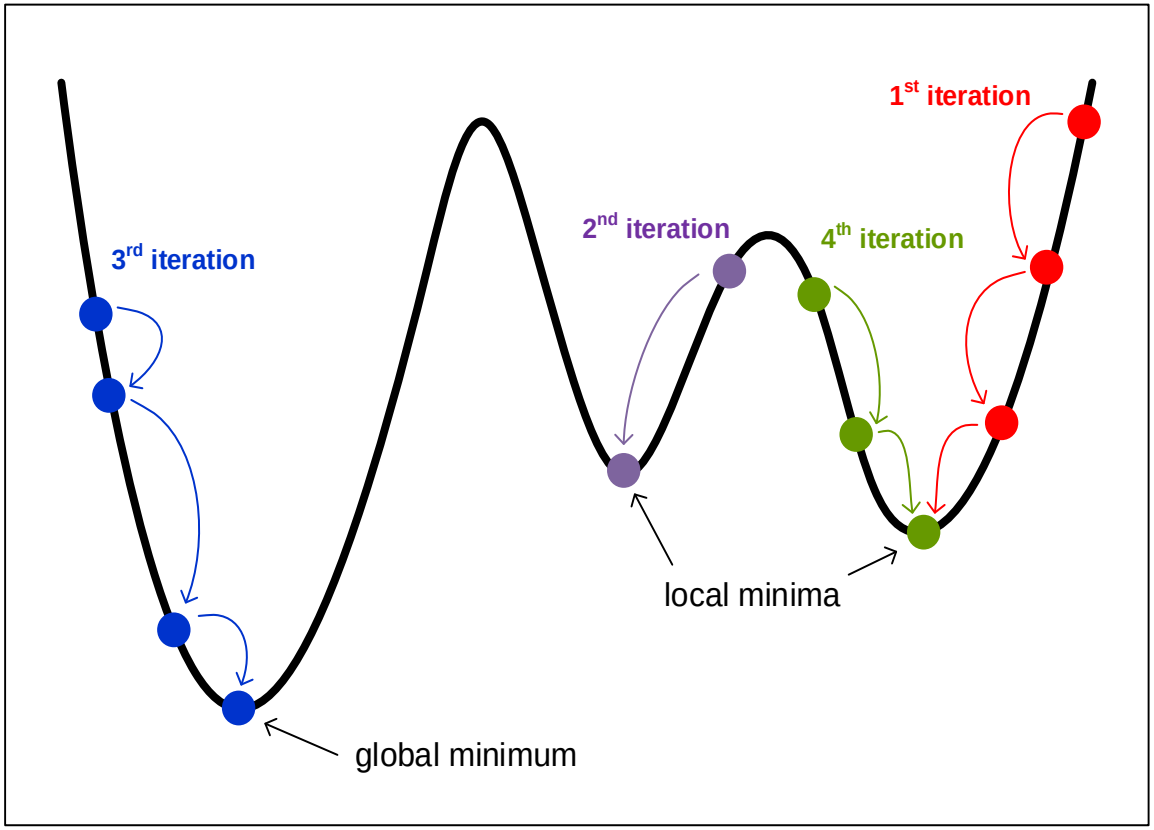
Building a solution from the scratch

+

Getting a better solution from a known solution

Multi Start Hill Climbing Heuristic (I)

- This heuristic combines the two algorithmic strategies:
    1. to build a solution from the scratch
    2. to get a better solution from a known solution



Multi Start Hill Climbing Heuristic (II)

- This heuristic combines the two algorithmic strategies:
  1. to build a solution from the scratch
  2. to get a better solution from a known solution
- In a problem aiming to minimize function $F(x)$, it works as follows:

$f_{best} = +\infty$

repeat

$z = BuildSolution\ ()$

$x = HillClimbing\ (z)$

$f = F(x)$

if $f < f_{best}$ **then**

$x_{best} = x$

$f_{best} = f$

endif

until Stopping Criteria is met

Random strategy or
Greedy Randomized strategy

Hill Climbing starts by the initial solution z and returns solution x

If the aim is to maximize $F(x)$

$f_{best} = -\infty$

$f > f_{best}$

Disponibilidade e Robustez a Falhas de Redes e Serviços

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2023/2024

Noção de disponibilidade

- A disponibilidade de um elemento é a probabilidade de o elemento estar operacional em qualquer instante de tempo.
- Para um dado elemento *i*, seja:
  - $MTBF_i$ (*Mean Time Between Failures*): tempo médio entre falhas do elemento *i*
  - $MTTR_i$ (*Mean Time To Repair*): tempo médio de reparação do elemento *i*
- então, a disponibilidade a_i do elemento *i* é dada por:

$$a_i = \frac{MTBF_i}{MTBF_i + MTTR_i}$$

- Exemplos:
  - se uma ligação falha em média ao fim de 1 ano (*MTBF* = 365.25 × 24 = 8766 horas) e demora em média 2 dias a ser reparada e voltar a estar operacional (*MTTR* = 2 × 24 = 48 horas), então a sua disponibilidade é 0.99455 (= 99.455%);
  - se um router falha em média ao fim de 90 dias (*MTBF* = 90 × 24 = 2160 horas) e demora em média 3 horas a ser reparado (ou substituído) e reconfigurado para ficar operacional (*MTTR* = 3 horas), então a sua disponibilidade é 0.99862 (= 99.862%).

Noção de disponibilidade

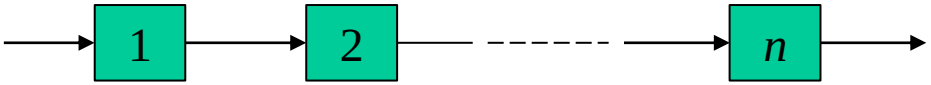
- A medição da disponibilidade de um elemento é definida para um determinado intervalo de tempo

Disponibilidade	Downtime por ano	Downtime por mês
90% (um nove)	36.53 dias	73.05 horas
99% (dois noves)	3.65 dias	7.31 horas
99.9% (três noves)	8.77 horas	43.83 minutos
99.99% (quarto noves)	52.6 minutos	4.38 minutos
99.999% (cinco noves)	5.26 minutos	26.3 segundos
99.9999% (seis noves)	31.56 segundos	2.63 segundos

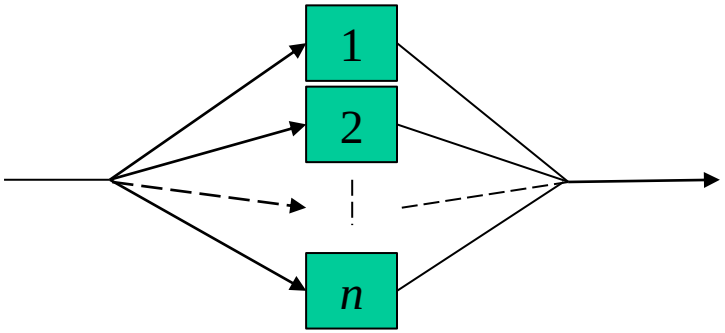
Downtime: tempo total em que o elemento não está disponível

Disponibilidade de um sistema composto por múltiplos elementos

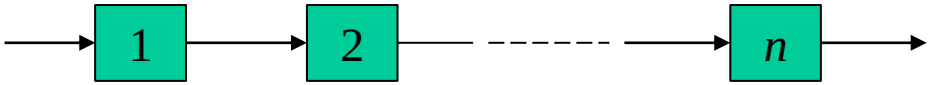
- A disponibilidade de sistema composto por múltiplos elementos é calculada modelando o sistema como uma interligação de elementos em série ou em paralelo.
- As regras para decidir se os elementos devem ser colocados em série ou em paralelo são:
  - Um conjunto de elementos é colocado em série se a falha de um elemento fizer com que o sistema falhe:



  - Um conjunto de elementos é colocado em paralelo se o sistema falha apenas quando todos os elementos falham simultaneamente:



Disponibilidade de um sistema com os elementos em série

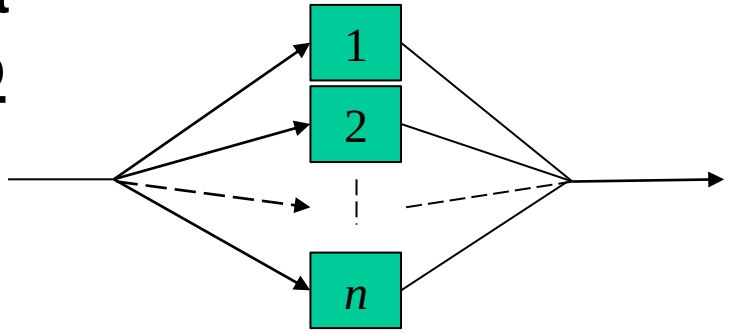


- A disponibilidade A do sistema é a probabilidade de todos os elementos estarem disponíveis (i.e., a funcionar).
- Sabendo a disponibilidade a_i de cada elemento i (e considerando que as falhas entre diferentes elementos são estatisticamente independentes), a disponibilidade do sistema é dada pelo produto das disponibilidades de todos os elementos:

$$A = a_1 \times a_2 \times \cdots \times a_n$$

- Propriedade:
  - A disponibilidade do sistema é menor (i.e., pior) ou igual do que a disponibilidade do elemento menos disponível
- Exemplo:
  - Um sistema com 3 elementos em série (com disponibilidade de 99.9% cada) tem uma disponibilidade de 99.7%

Disponibilidade de um sistema com os elementos em paralelo



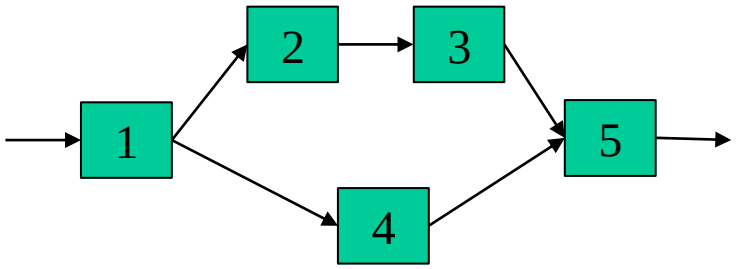
- A disponibilidade A do sistema é a probabilidade de pelo menos um elemento estar disponível (i.e., a funcionar).
- Sabendo a disponibilidade a_i de cada elemento i (e considerando que as falhas entre os diferentes elementos são estatisticamente independentes), a disponibilidade do sistema é dada por 1 – (a probabilidade de todos os elementos estarem indisponíveis):

$$A = 1 - [(1 - a_1) \times (1 - a_2) \times \cdots \times (1 - a_n)]$$

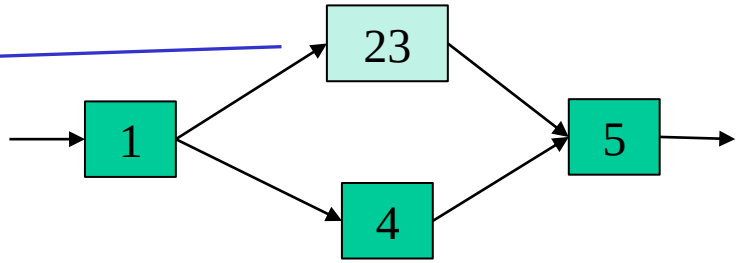
- Propriedade:
  - A disponibilidade do sistema é maior (i.e., melhor) ou igual do que a disponibilidade do elemento mais disponível
- Exemplo:
  - Um sistema com 3 elementos em paralelo (com disponibilidade de 99.0% cada) tem uma disponibilidade de 99.9999% (seis noves)

Disponibilidade de um sistema composto por múltiplos elementos

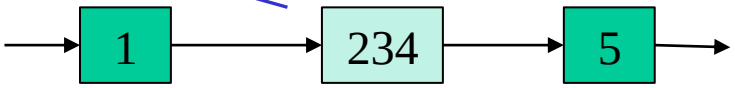
No caso geral, vai-se reduzindo o modelo geral num modelo com menos elementos substituindo um conjunto de elementos (em série ou em paralelo) por um elemento único cuja disponibilidade é calculada de acordo com o tipo de conjunto.



$$A_{23} = a_2 \times a_3$$



$$A_{234} = 1 - [(1 - A_{23}) \times (1 - a_4)]$$



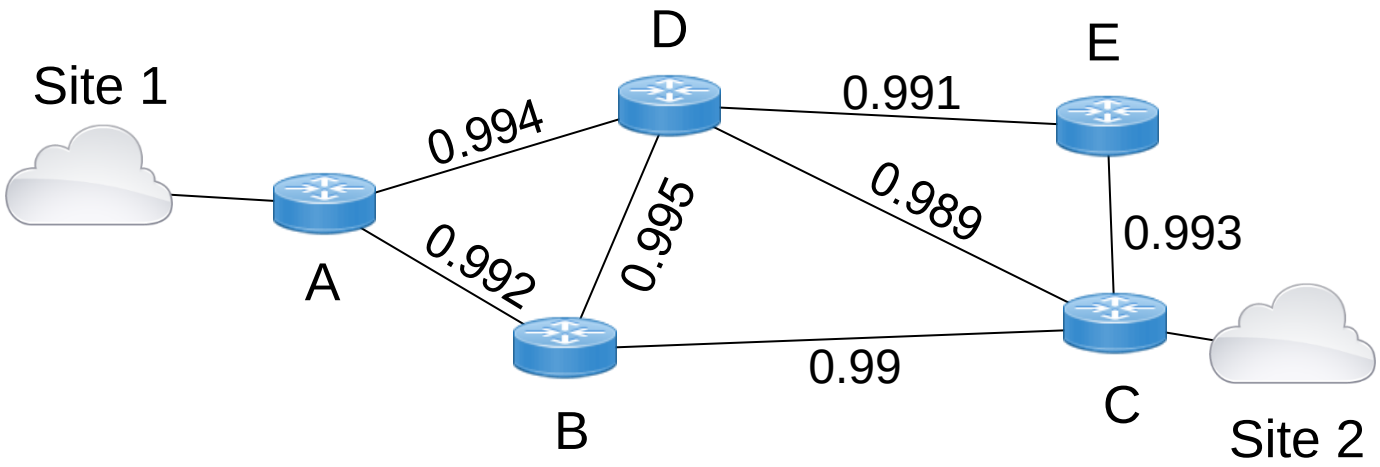
$$A = a_1 \times A_{234} \times a_5$$

Disponibilidade de um serviço unicast numa rede de telecomunicações

A disponibilidade de um serviço unicast depende dos percursos de encaminhamento na rede de cada fluxo ponto-a-ponto do serviço.

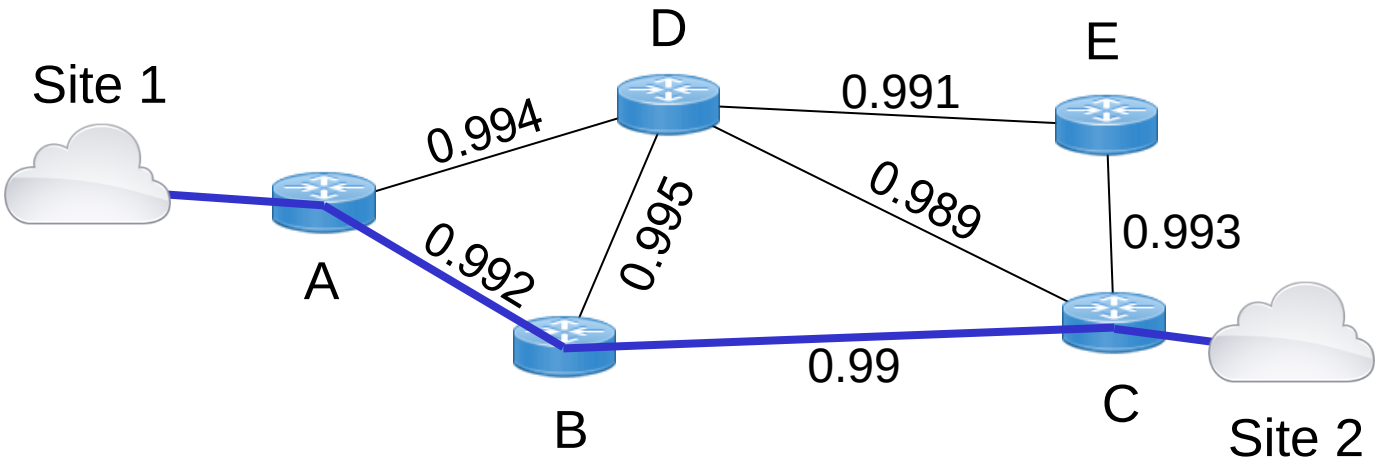
Considere-se o seguinte exemplo de uma rede de um operador a suportar um serviço VPN entre dois sites de uma empresa (serviço com um único fluxo ponto-a-ponto) em que:

- todos os routers do operador têm uma disponibilidade de 99.99%;
- a figura indica a disponibilidade de cada ligação.



Disponibilidade do exemplo

Se o serviço VPN entre dois sites for encaminhado pelo percurso A → B → C (do site 1 para o site 2) e pelo mesmo percurso no sentido inverso (do site 2 para o site 1),

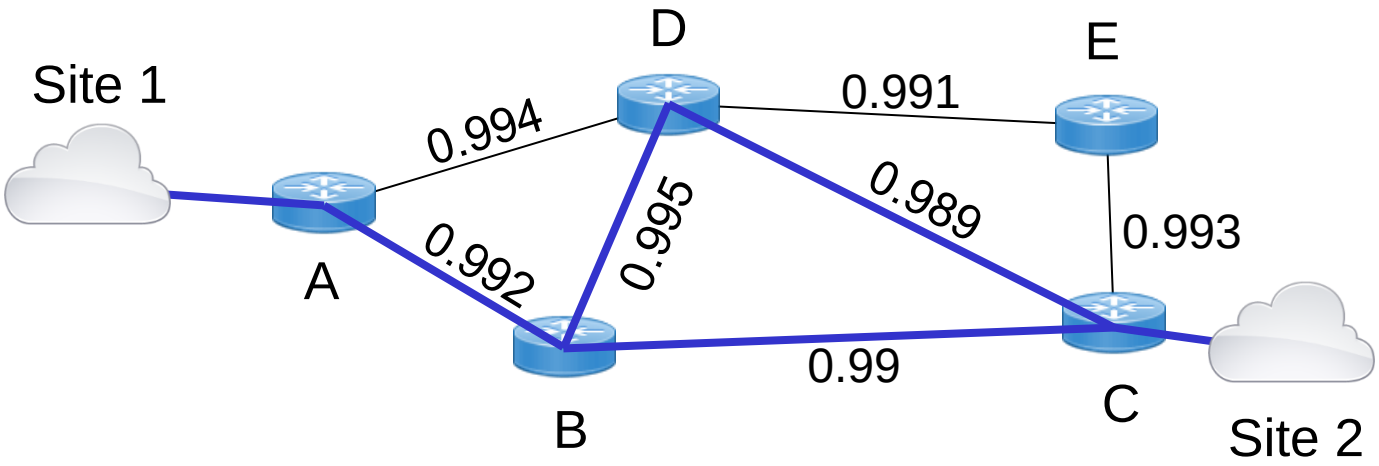


então, a disponibilidade da rede para este serviço VPN é:

$$A = a_A \times a_{AB} \times a_B \times a_{BC} \times a_C$$
$$= 0.9999 \times 0.992 \times 0.9999 \times 0.99 \times 0.9999 = 0.9818$$

Disponibilidade do exemplo

Se o serviço VPN entre dois sites for encaminhado por um dos percursos A → B → C ou A → B → D → C (do site 1 para o site 2) e pelos mesmos percursos no sentido inverso (do site 2 para o site 1),



então, a disponibilidade da rede para este serviço VPN é:

$$A_{BDC//BC} = 1 - [(1 - a_{BD} \times a_D \times a_{DC}) \times (1 - a_{BC})] =$$

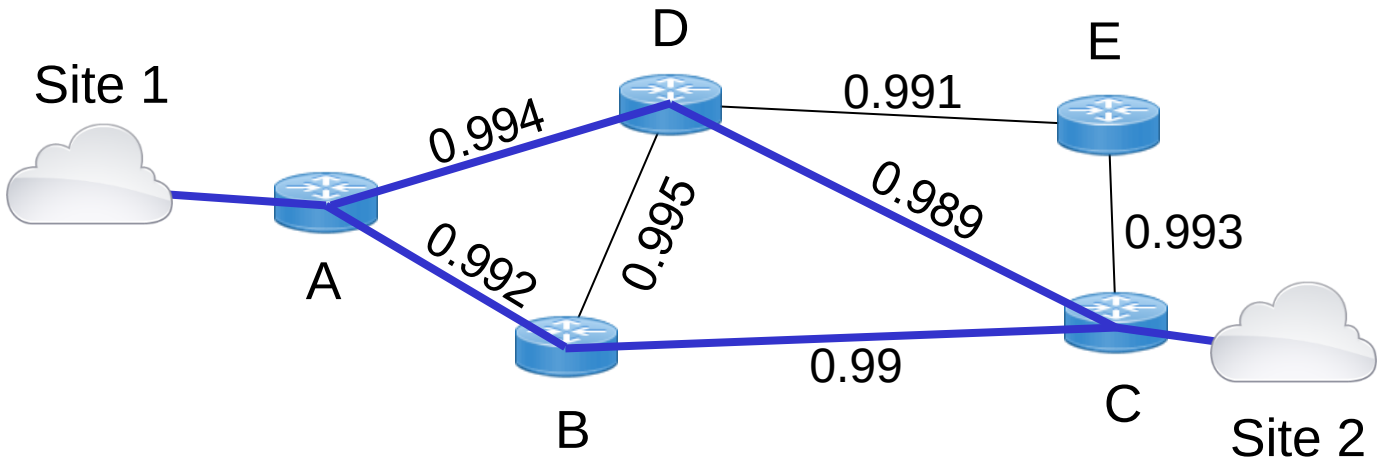
$$= 1 - [(1 - 0.995 \times 0.9999 \times 0.989) \times (1 - 0.99)] = 0.99984$$

$$A = a_A \times a_{AB} \times a_B \times A_{BDC//BC} \times a_C$$

$$= 0.9999 \times 0.992 \times 0.9999 \times 0.99984 \times 0.9999 = 0.9915$$

Disponibilidade do exemplo

Se o serviço VPN entre dois sites for encaminhado por um dos percursos A → B → C ou A → D → C (do site 1 para o site 2) e pelos mesmos percursos no sentido inverso (do site 2 para o site 1),



então, a disponibilidade da rede para este serviço VPN é:

$$A_{ADC//ABC} = 1 - [(1 - a_{AD} \times a_D \times a_{DC}) \times (1 - a_{AB} \times a_B \times a_{BC})] =$$

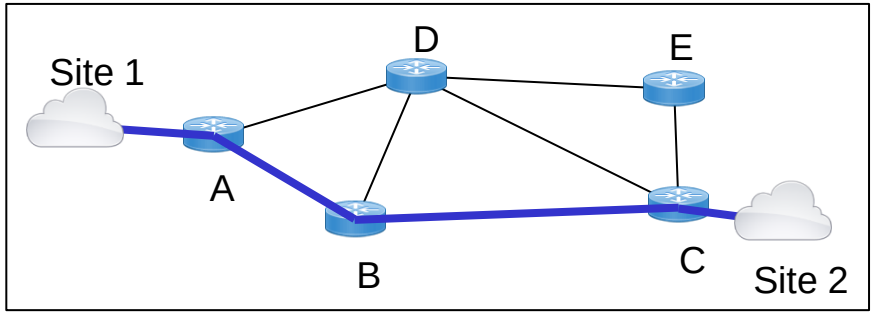
$$= 1 - [(1 - 0.994 \times 0.9999 \times 0.989) \times (1 - 0.992 \times 0.9999 \times 0.99)] = 0.9997$$

$$A = a_A \times A_{ADC//ABC} \times a_C$$

$$= 0.9999 \times 0.9997 \times 0.9999 = 0.995$$

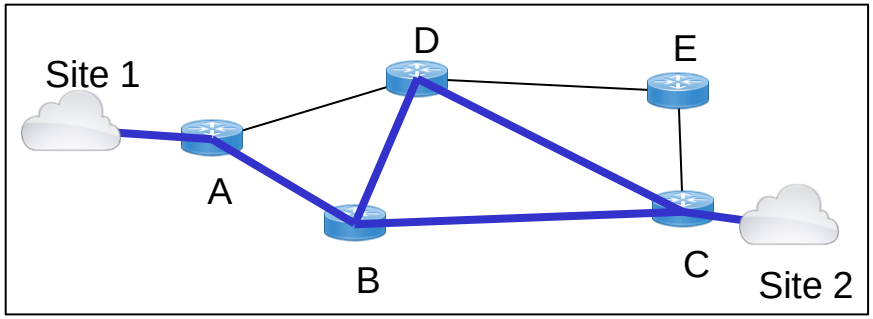
Disponibilidade do exemplo

Comparando as 3 soluções anteriores:



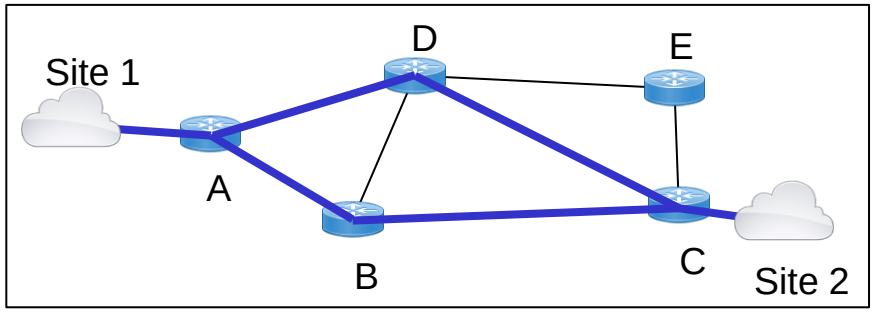
$$A = 0.9818$$

O serviço falha em média
(1 – 0.9818) × 365.25
= 6.65 dias/ano



$$A = 0.9915$$

O serviço falha em média
(1 – 0.9915) × 365.25
= 3.1 dias/ano



$$A = 0.995$$

O serviço falha em média
(1 – 0.995) × 365.25
= 1.8 dias/ano

Modelo de disponibilidade de ligações em redes de telecomunicações

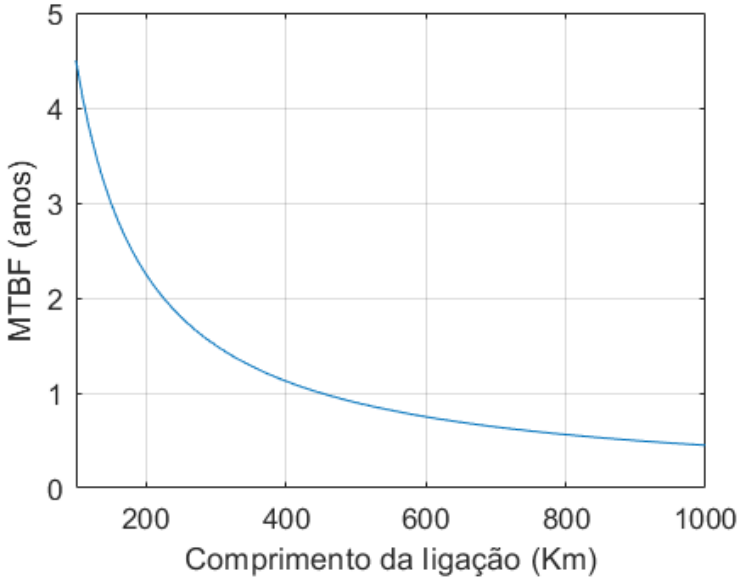
De acordo com [1], um modelo de disponibilidade das ligações de redes óticas nos EUA no início deste século era aproximadamente:

$$\frac{MTBF}{MTBF + MTTR}$$

$MTTR$ = 24 horas

$$MTBF = \frac{CC \times 365 \times 24}{\text{comprimento da ligação [Km]}} \text{ [horas]}$$

CC ($Cable\ Cut\ metric$) = 450 Km



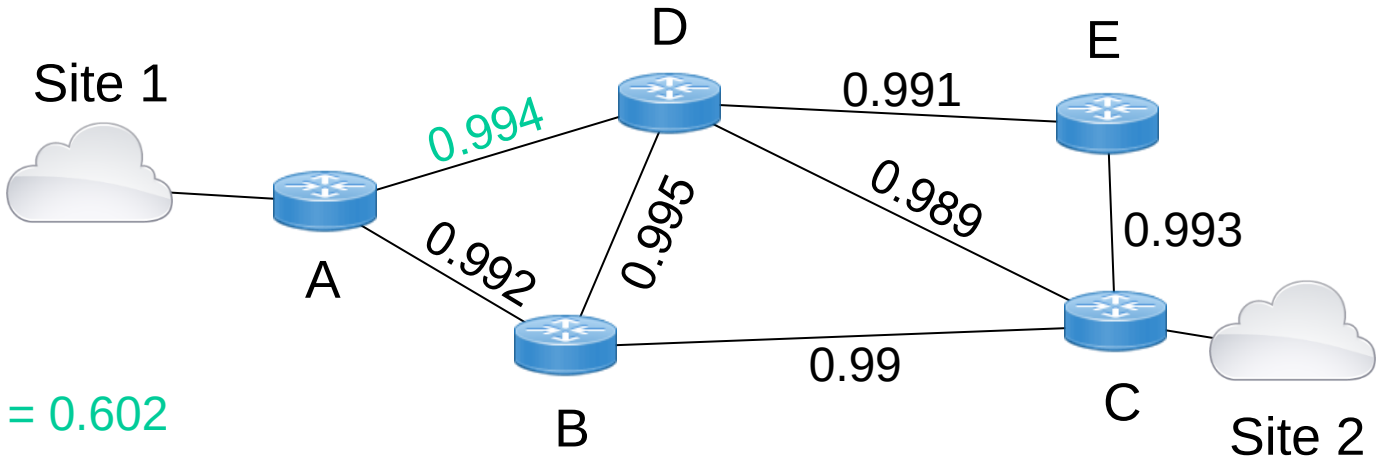
[1] J.-P. Vasseur, M. Pickavet and P. Demeester, “Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS”, Elsevier (2004)

Cálculo de percursos de maior disponibilidade

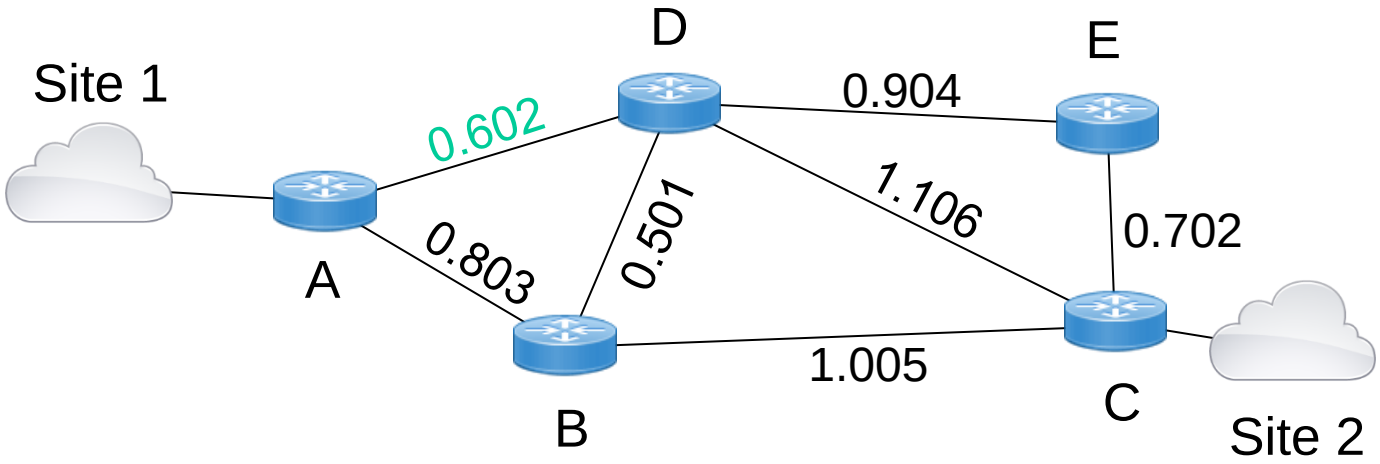
- Considere-se:
  - uma rede de telecomunicações em que a disponibilidade dos nós (*routers*, *switches*, *optical cross-connects*) é 1.0.
  - o conjunto P de todos os percursos de encaminhamento possível na rede de um determinado nó origem para um determinado nó destino.
- A disponibilidade a_p de cada percurso $p \in P$ é o produto das disponibilidades dos links que pertencem ao percurso.
- O logaritmo da disponibilidade de um percurso $log(a_p)$ é então a soma dos logaritmos das disponibilidades dos links.
- A função logaritmo é monotonamente crescente. Assim, o k-ésimo percurso mais disponível (com o maior valor de disponibilidade) é também o k-ésimo percurso com o maior valor do logaritmo da sua disponibilidade.
- O valor $log(a_p)$ é negativo. Assim, considerando o "comprimento" de cada ligação como $-log(a_p)$, os k percursos de maior disponibilidade podem ser calculados por um algoritmo de k percursos mais curtos.

Cálculo de percursos de maior disponibilidade

Considere-se o exemplo de um serviço VPN entre dois sites de uma empresa. A figura indica a disponibilidade de cada ligação.



– $log(0.994) \times 100 = 0.602$



Disponibilidade de uma rede de telecomunicações

Considere-se uma rede com *n* ligações em que cada ligação tem uma disponibilidade *a* (igual para todas as ligações). O número *i* de ligações indisponíveis é uma variável aleatória binomial com probabilidade $p = 1 - a$.

Assim, a probabilidade de haver *i* ligações indisponíveis é:

$$f(i) = \binom{n}{i} p^i (1-p)^{n-i}, i = 0, 1, 2, ..., n$$

A probabilidade *P* de estarem 2 ou mais ligações indisponíveis é:

a	*n*	*P*
0.995	10	0.110%
0.995	20	0.447%
0.995	40	1.719%
0.999	10	0.004%
0.999	20	0.019%
0.999	40	0.076%

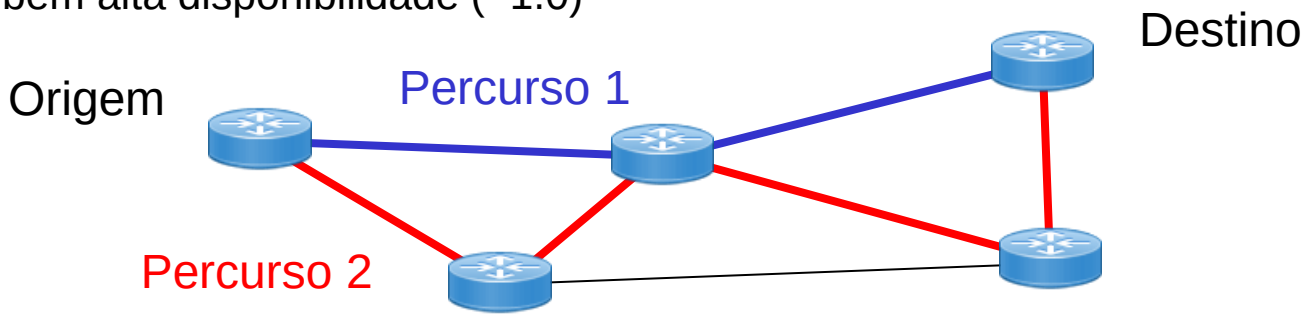
CONCLUSÃO:
a esmagadora maioria das vezes que há falhas de ligações, apenas está uma ligação indisponível.

Robustez de serviços a falhas da rede

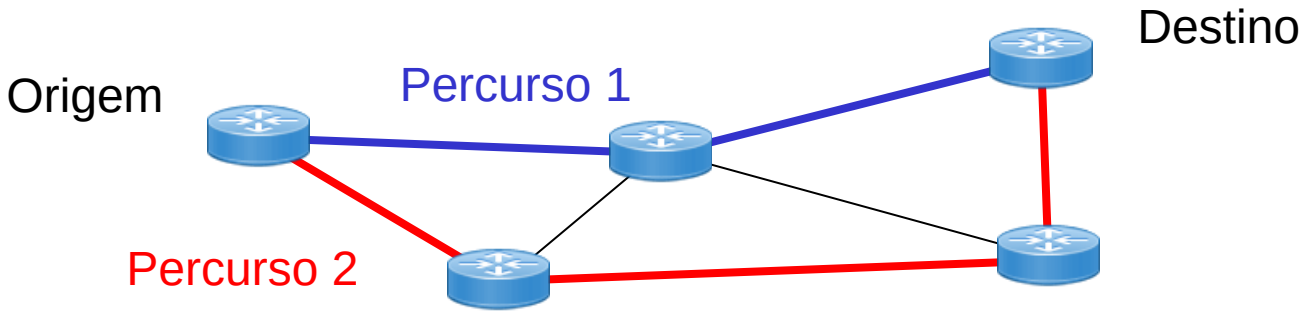
- A robustez de uma rede de telecomunicações é genericamente definida como a capacidade da rede em manter os serviços que suporta quando um ou mais dos seus elementos (nós e/ou ligações) falham.
- A robustez de uma rede pode ser melhorada com dois tipos de mecanismos.
- **Mecanismos de restauro**: os serviços são suportados assumindo que não há falhas; quando uma falha acontece, a rede tenta reencaminhar o mais possível os fluxos dos serviços afetados pelos recursos que se mantêm disponíveis (i.e., nós e ligações que não falharam).
  - Exemplos: as redes IP com protocolos tais como o RIP e o OSPF
- **Mecanismos de proteção**: os recursos da rede são atribuídos (aos diferentes fluxos dos diferentes serviços) não só para o caso de nenhuma falha mas também para um subconjunto de casos de possíveis falhas; se acontecer uma das falhas do subconjunto considerado, é garantido que os serviços continuam a ser suportados.

Mecanismos de proteção baseados em pares de percursos disjuntos

- Cada fluxo (de um nó origem para um nó destino) é suportado por dois percursos disjuntos (ambos a iniciar no nó origem e a terminar no nó destino do fluxo):
  - disjuntos nas ligações (i.e., sem ligações comuns) usado quando os nós exibem alta disponibilidade (~1.0)



  - disjuntos nos nós e ligações (i.e., sem ligações nem nós intermédios comuns) usado quando a disponibilidade dos nós não é próxima de 1.0.



Mecanismos de proteção baseados em pares de percursos disjuntos

- Um par de percursos disjuntos nas ligações protege o fluxo para todas as falhas individuais de ligação.
- Um par de percursos disjuntos nos nós e nas ligações protege o fluxo para todas as falhas individuais de um elemento (nó ou ligação) exceto a falha do nó origem ou do nó destino do fluxo.

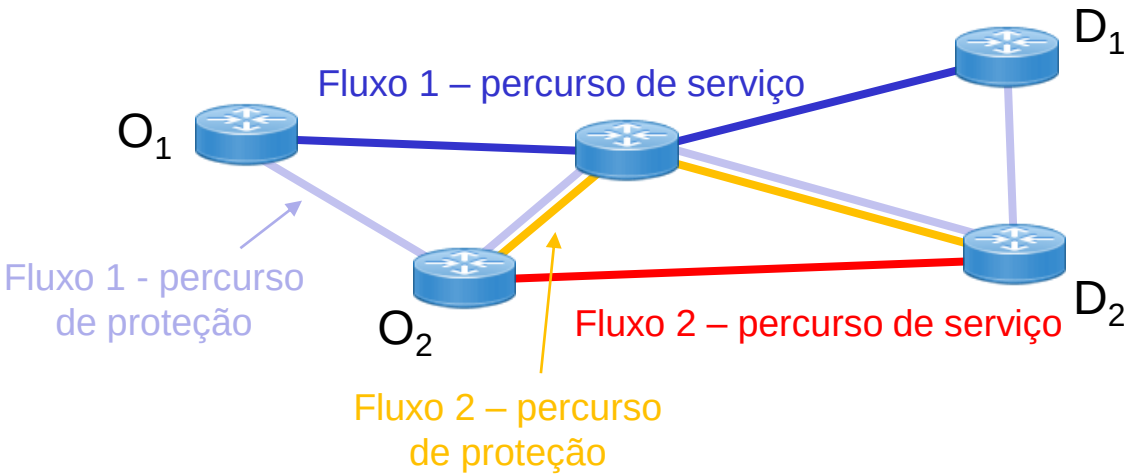
- Distinguem-se 2 casos:
- **Proteção 1+1** (um mais um): o fluxo é enviado duplicado pelos 2 percursos.
  - O tempo de recuperação a falhas é virtualmente nulo (i.e., muito curto).
  - Exige recursos da rede dedicados a cada fluxo.
- **Proteção 1:1** (um para um): o fluxo é enviado por um dos percursos (designado por percurso de serviço) e o outro percurso (designado por percurso de proteção) só é usado em caso de falha do primeiro.
  - O tempo de recuperação a falhas é maior (o nó origem tem de ser notificado da falha do percurso de serviço para passar a transmitir o fluxo pelo percurso de proteção).
  - Os recursos do percurso de proteção podem ser partilhados entre diferentes fluxos.

Mecanismos de proteção baseados em pares de percursos disjuntos - EXEMPLO

Considere-se a rede seguinte a suportar dois fluxos:

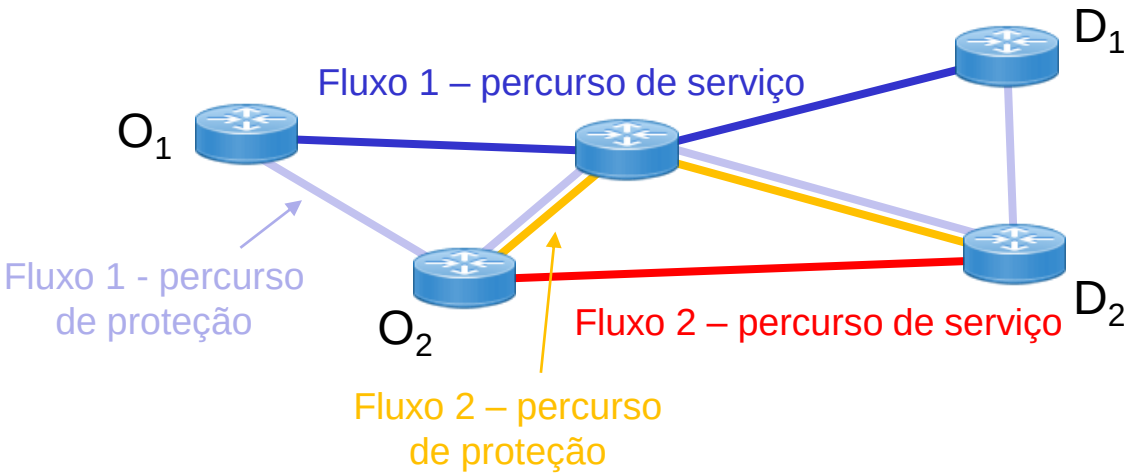
- fluxo 1 de 10 Gbps do router O_1 para o router D_1
- fluxo 2 de 20 Gbps do router O_2 para o router D_2

A figura mostra o par de percursos de encaminhamento usado para encaminhar cada um dos fluxos.



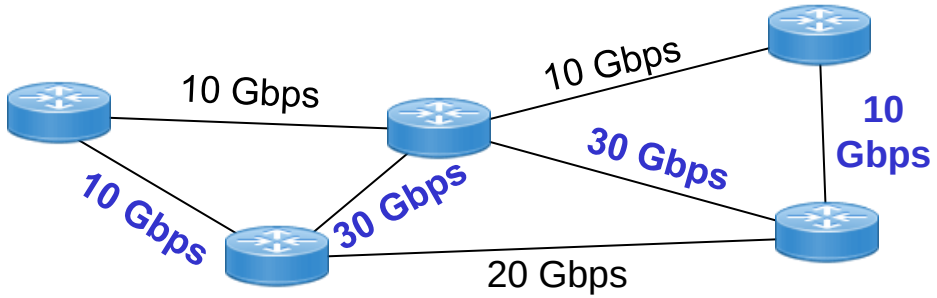
Pergunta: que recursos são necessários em cada ligação para os dois fluxos serem protegidos por um mecanismo 1+1 ou um mecanismo 1:1?

Mecanismos de proteção baseados em pares de percursos disjuntos - EXEMPLO

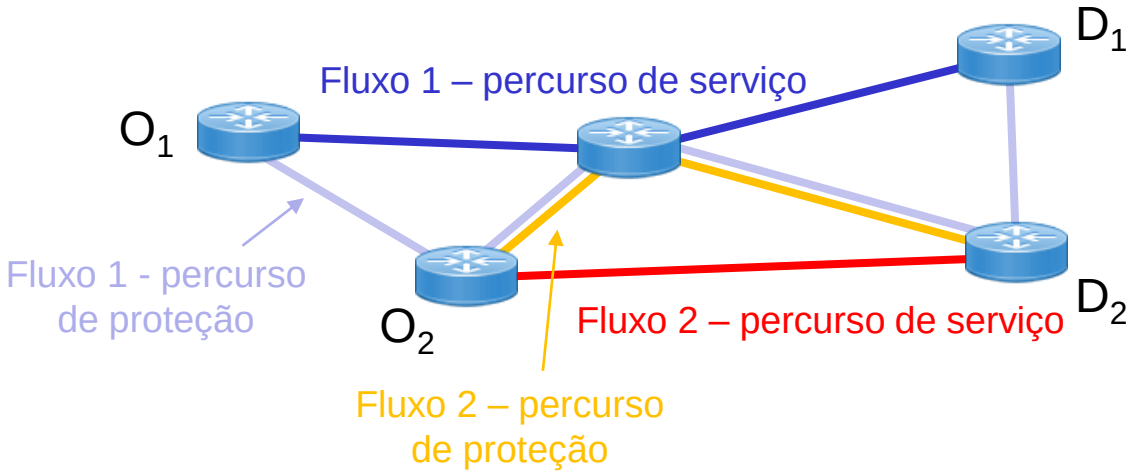


Proteção 1+1 - os 10 Gbps do fluxo 1 e os 20 Gbps do fluxo 2 são transmitidos em ambos os pares de percursos de cada fluxo.

Recursos ocupados em cada ligação:

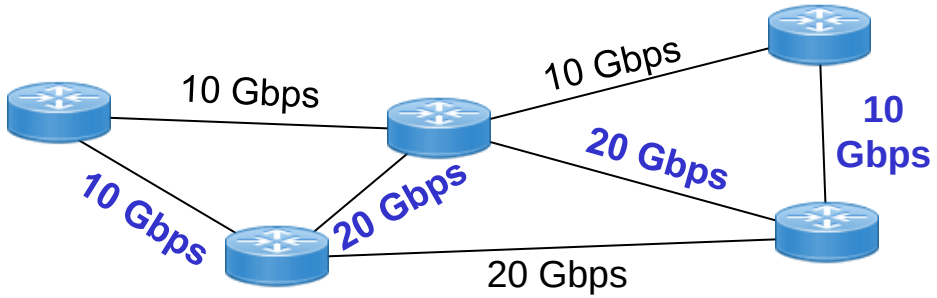


Mecanismos de proteção baseados em pares de percursos disjuntos - EXEMPLO



Proteção 1:1 – como os percursos de serviço são disjuntos, uma falha individual apenas pode afetar um dos percursos. Assim, nas ligações dos percursos de proteção, apenas basta ter os recursos para o maior valor dos percursos de serviço.

Recursos ocupados em cada ligação:



Qualidade de serviço em redes com comutação de pacotes: controle de fluxo, escalonamento e descarte de pacotes

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2023/2024

Controlo de fluxos em redes com comutação de pacotes

Considere-se uma rede com comutação de pacotes em que cada ligação tem uma capacidade máxima de transmissão.

Em serviços de dados baseados em transferência de ficheiros:

- o emissor de cada fluxo ambiciona enviar cada ficheiro à taxa de transmissão máxima que a rede lhe permite;
- a taxa de transmissão de cada fluxo pode ser dinâmica ou pode ser atribuída previamente.

Controlo de fluxo: como regular o emissor de forma dinâmica para que a taxa de transmissão seja a taxa máxima que a rede pode suportar em cada instante de tempo.

Controlo de taxa de transmissão: como regular o emissor de forma a que ele não ultrapasse a taxa de transmissão previamente atribuída.

Mecanismos de escalonamento e de descarte de pacotes

Considere-se uma rede com comutação de pacotes em que em cada interface de saída de cada ligação existe uma fila de espera para condicionar temporariamente os pacotes a transmitir pela interface. Em cada interface de saída de cada ligação:

Disciplina de escalonamento: algoritmo que decide a ordem pela qual são transmitidos pela ligação os pacotes de diferentes fluxos que estão na fila de espera

- impõe assim diferentes atrasos médios (*average delays*) a diferentes fluxos ao definir a ordem de transmissão dos pacotes.

Método de descarte de pacotes: método que decide como os pacotes dos diferentes fluxos são aceites na fila de espera quando a ligação está ocupada com a transmissão de outro pacote

- impõe assim diferentes taxas de perda de pacotes (*packet loss rates*) a diferentes fluxos ao definir que pacotes são descartados.

Sumário do Módulo

Controlo de Fluxos em Redes com Comutação de Pacotes:

Primeira Parte:

- Noções básicas de controlo de fluxos em redes com comutação de pacotes
- Controlo de fluxos de pacotes baseado em janelas extremo-a-extremo

Segunda Parte:

- Mecanismos de controlo de taxas de transmissão de fluxos de pacotes
- Atribuição de taxas de transmissão a fluxos de pacotes segundo o princípio de equidade do tipo max-min

Sumário do Módulo

Mecanismos de escalonamento e de descarte de pacotes

Primeira Parte:

- Caracterização das disciplinas de escalonamento de pacotes

Segunda Parte:

- Disciplinas de escalonamento de pacotes: FIFO, com prioridades e que funcionam de forma rotativa

Terceira Parte

- Disciplinas de escalonamento de pacotes que funcionam por aproximação ao sistema GPS

Quarta Parte

- Métodos de descarte de pacotes
- Ilustração da combinação de disciplinas de escalonamento com métodos de descarte de pacote na arquitectura *DiffServ* do IETF.

Controlo de Fluxos em Redes com Comutação de Pacotes

Primeira parte:

- **Noções básicas de controlo de fluxos em redes com comutação de pacotes**
- **Controlo de fluxos de pacotes baseado em janelas extremo-a-extremo**

Controlo de fluxo - introdução

O *tráfego efetivo* reflete a quantidade de serviço suportada por uma rede com comutação de pacotes.

O *atraso médio* reflete a qualidade de serviço proporcionada por uma rede com comutação de pacotes.

Controlo de fluxo: mecanismo de realimentação que estabelece um compromisso entre o tráfego efetivo e o atraso médio por forma a manter o atraso médio dentro de limites aceitáveis:

- Quando o tráfego oferecido é reduzido, é aceite na sua totalidade pelo algoritmo de controlo de fluxo e, neste caso,

  tráfego efetivo = tráfego oferecido

- Quando o tráfego oferecido é excessivo, o algoritmo de controlo de fluxo rejeita parte dele e, neste caso,

  tráfego efetivo = tráfego oferecido – tráfego rejeitado

- À medida que o algoritmo de encaminhamento aumenta o atraso médio, o controlo de fluxo reduz o tráfego efetivo.

Controlo de fluxo - introdução

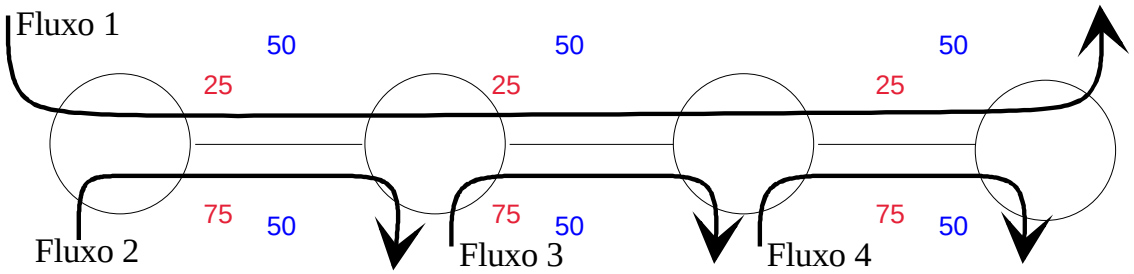
Os algoritmos de controlo de fluxo devem idealmente observar os seguintes requisitos:

- Estabelecer um bom compromisso entre:
  - a quantidade de serviço (o tráfego efetivo, sujeito eventualmente à garantia de uma taxa de transmissão mínima) e
  - a qualidade de serviço (medida, por exemplo, a partir do atraso médio e da taxa de pacotes perdidos)
- Garantir um tratamento equitativo dos diferentes fluxos de pacotes, ao fornecer a qualidade de serviço requerida.

  Fluxos com iguais necessidades devem ter o mesmo tratamento

Gestão de recursos: tráfego efetivo vs. equidade

Considere-se o exemplo da figura assumindo que a capacidade de cada ligação é 100.



Tráfego efetivo máximo:

Nao é aceite na rede, por com o trafego efetivo total o fluxo 1 não é suportado

Fluxo 1 = 0, Fluxos 2,3,4 = 100

Tráfego efetivo total = 0+100+100+100 = 300

Partilha equitativa dos recursos:

Fluxo 1 = 25, Fluxos 2,3,4 = 75

Equitativa

Tráfego efetivo total = 25+75+75+75 = 250

Máxima equidade (i.e., mesma taxa de transmissão a todos os fluxos):

Fluxos 1,2,3,4 = 50

Max equidade, melhor solução

Tráfego efetivo total = 50+50+50+50 = 200

Controlo de fluxo através de janelas

- Considere um fluxo de pacotes de um emissor A para um recetor B.
- Por cada pacote recebido, o recetor B notifica o emissor A através do envio para A de uma *permissão*:
  - Uma permissão pode ser transmitida num pacote de controlo dedicado ou pode ser encavalitada (*piggybacked*) num pacote de dados enviado no sentido contrário. Funciona como o ACK do TCP
- Quando recebe uma permissão, o emissor A fica autorizado a enviar mais um pacote para o recetor B.
- Um esquema de controlo de fluxos pode ser combinado com um protocolo ARQ (*Automatic Repeat Request*) de controlo de erros
  - neste caso, os pacotes são numerados (*sequence numbers*) e as permissões indicam o número de pacotes recebidos (*acknowledgment numbers*) sem erros

A permissão para além de notificar o emissor que pode enviar mais pacotes, indica que também recebeu o anterior sem erros:
-> Controlo de fluxos e de erros

Controlo de fluxo através de janelas

- Um fluxo de pacotes entre o emissor A e o recetor B diz-se *controlada através de janelas* se existir um limite máximo para o número de pacotes que, tendo sido transmitidos por A, não foram ainda notificadas como tendo sido recebidos por B.
- O limite máximo é designado por *tamanho da janela*, ou simplesmente, *janela*.
- O emissor e o recetor podem ser dois nós da rede, um terminal e o nó de entrada da rede ou os dois terminais que estão nos extremos do fluxo.

De seguida, considera-se a estratégia de ***janelas extremo-a-extremo*** (*end-to-end*):

- para cada fluxo de pacotes, o controlo de fluxos é implementado entre o seu emissor e o seu recetor
- estratégia usada pelo TCP nas redes TCP/IP

Janelas extremo-a-extremo

- No controlo de fluxos através de janelas, a taxa de transmissão do emissor é reduzida à medida que as permissões demoram mais tempo a regressar.
- Assim, se o percurso de encaminhamento do fluxo estiver congestionado, a diferença de tempo entre o envio de cada pacote e a receção da sua permissão aumenta o que obriga o emissor a reduzir a sua taxa de transmissão (aliviando o congestionamento do percurso).
- Além disso, o recetor pode atrasar intencionalmente o envio de permissões para reduzir a taxa de transmissão do fluxo com o objetivo de, por exemplo, evitar a sobrecarga do seu *buffer* de receção.

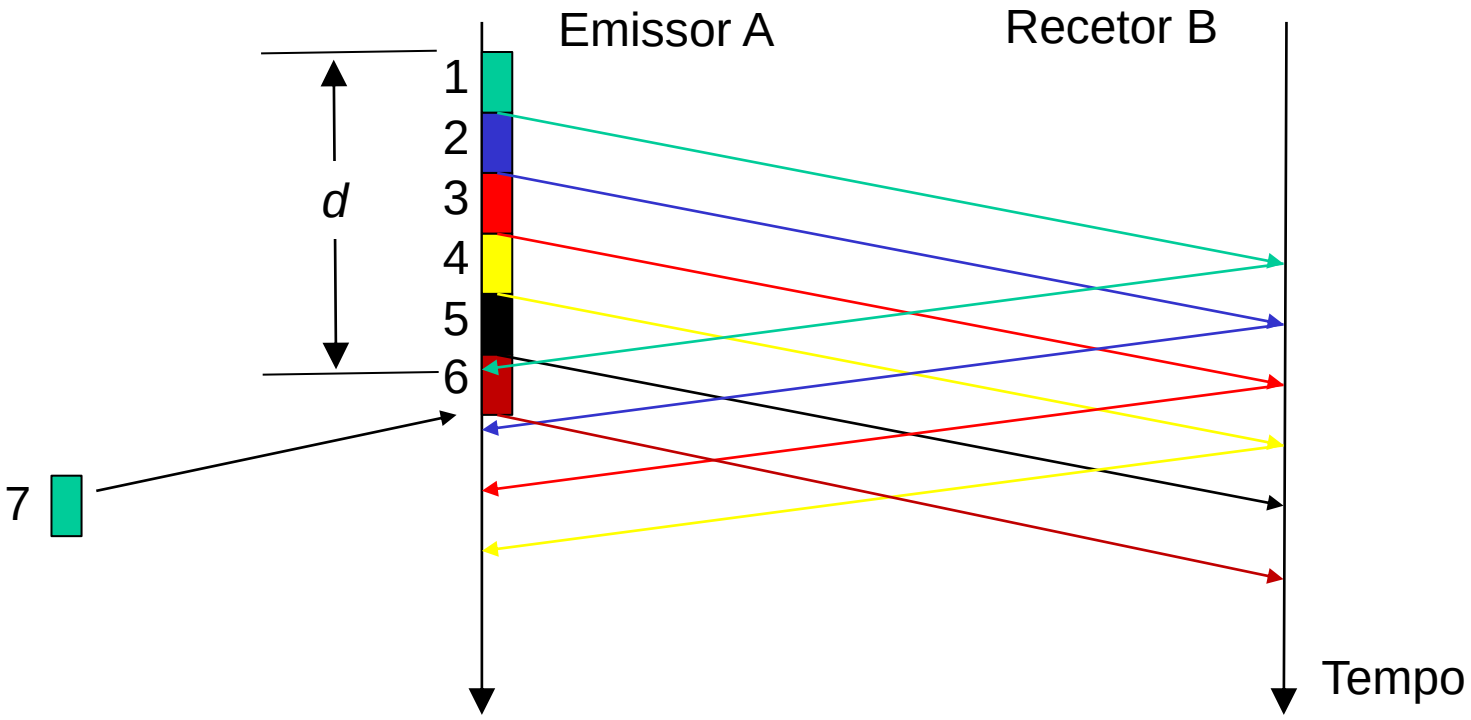
Janelas extremo-a-extremo

- Considere-se o tamanho da janela dado por *W*, em número de pacotes (pode ser noutras unidades como por exemplo Bytes no TCP).
  - Cada vez que um pacote é recebido no nó destino, é enviada uma permissão autorizando o envio de um novo pacote.
- Considere-se o atraso de ida-e-volta dado por *d* e o tempo de transmissão médio de cada pacote dado por *X* (i.e., o tráfego efetivo máximo disponível na rede é 1/*X*, em pacotes por segundo):
  - ✓ Se $d \leq WX$, a transmissão de *W* pacotes demora mais que o atraso de ida-e-volta; assim, o emissor pode transmitir à velocidade máxima de 1/*X* pacotes por segundo.
  - ✓ Se $d > WX$, o controlo de fluxos está ativo pois o atraso de ida-e-volta é tão elevado que *W* pacotes são transmitidos antes da receção da permissão relativa ao primeiro dos pacotes.

Então, o ritmo de transmissão é dado por: $r = \min\left\{\frac{1}{X}, \frac{W}{d}\right\}$

Ilustração das janelas extremo-a-extremo

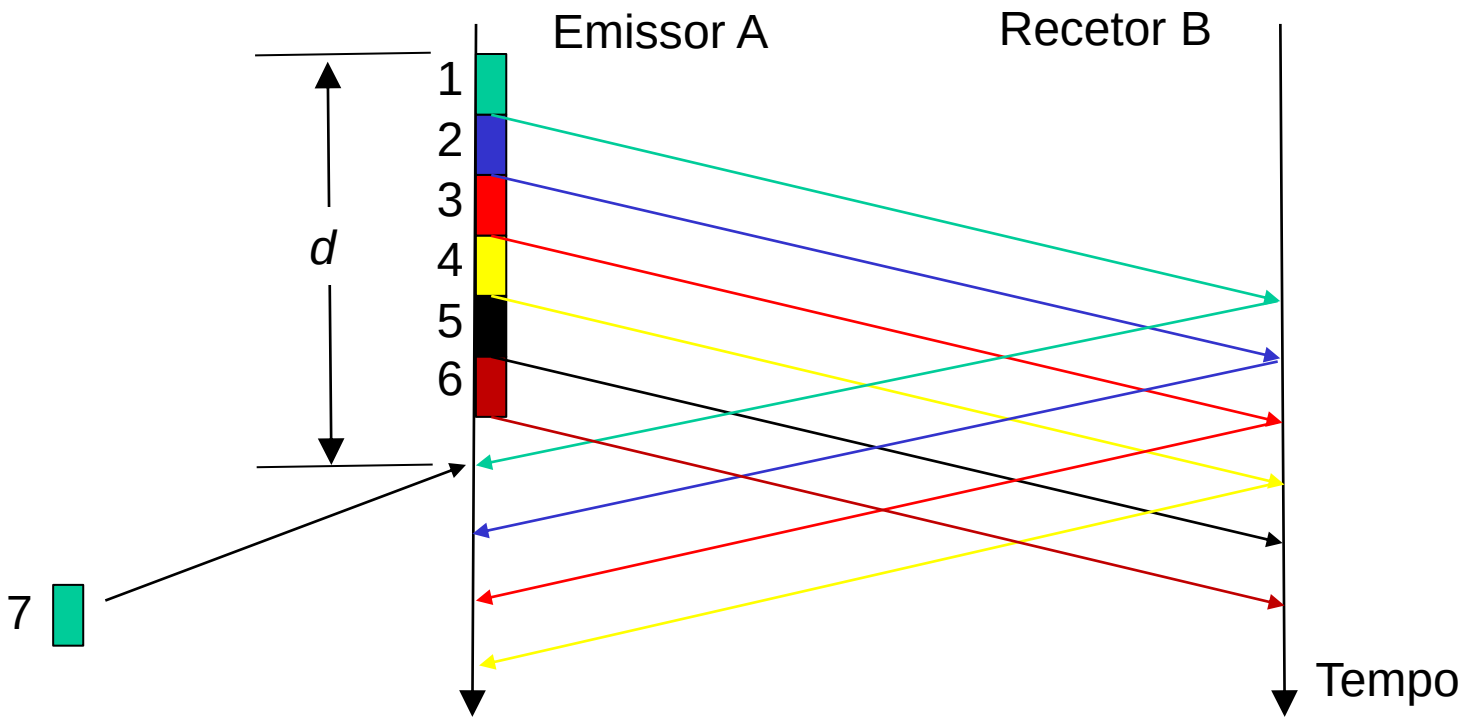
Considere-se W = 6 pacotes do emissor A para o recetor B.



$d \leq WX$ (a transmissão de 6 pacotes demora mais tempo que o atraso de ida-e-volta d) $\rightarrow$ o 7º pacote pode ser transmitido logo após o 6º pacote

Ilustração das janelas extremo-a-extremo

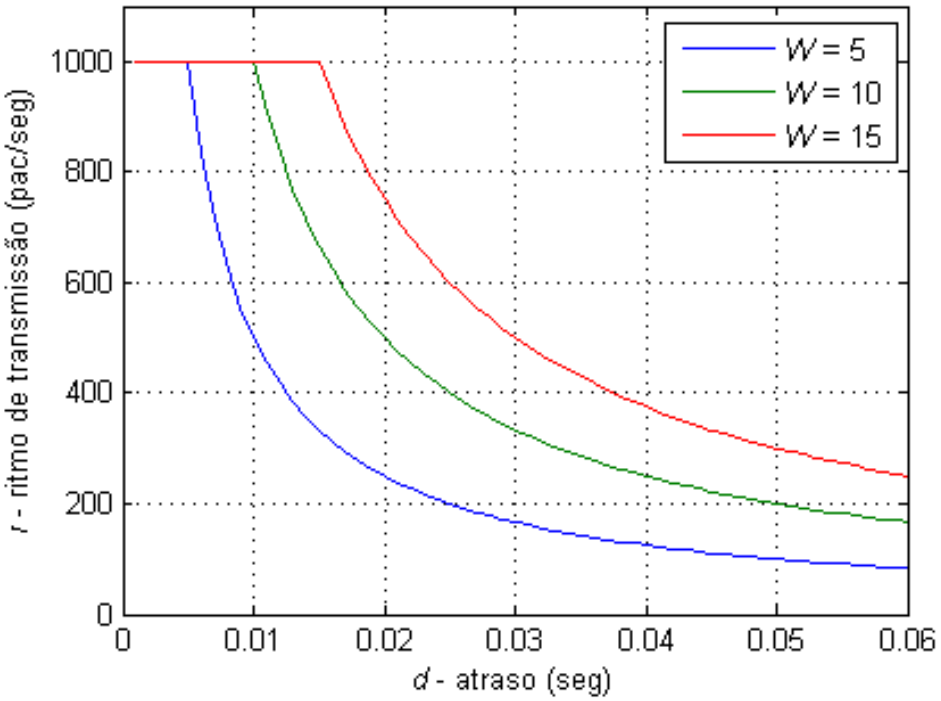
Considere-se W = 6 pacotes do emissor A para o recetor B.



$d > WX$ (a transmissão de 6 pacotes demora menos tempo que o atraso de ida-e-volta d) $\rightarrow$ o 7º pacote só pode ser transmitido quando o emissor A recebe a permissão do 1º pacote

Janelas extremo-a-extremo

Exemplo: X = 1 mseg. e janela W = 5, 10 e 15 pacotes.



$$r = \min\left\{\frac{1}{X}, \frac{W}{d}\right\}$$

- ✓ Para valores $d \leq WX$, o emissor transmite ao ritmo máximo $r = 1/10^{-3} = 1000$ (em pacotes/segundo)
- ✓ Para valores $d > WX$, o controlo de fluxos está ativo e o emissor transmite ao ritmo $r = W/d$ (em pacotes/segundo)

Dimensionamento do tamanho da janela

Existe um compromisso entre tráfego efetivo e atraso:

- por um lado, a janela deve ser pequena para limitar o número de pacotes na rede, evitando assim grandes atrasos e congestão;
- por outro, a janela deve ser grande para permitir a transmissão ao ritmo máximo (i.e., tráfego efetivo máximo) a todos os fluxos em condições de tráfego moderado na rede.

De qualquer modo, é sempre desejável que cada fluxo possa transmitir ao ritmo máximo quando não existe nenhum outro fluxo ativo na rede.

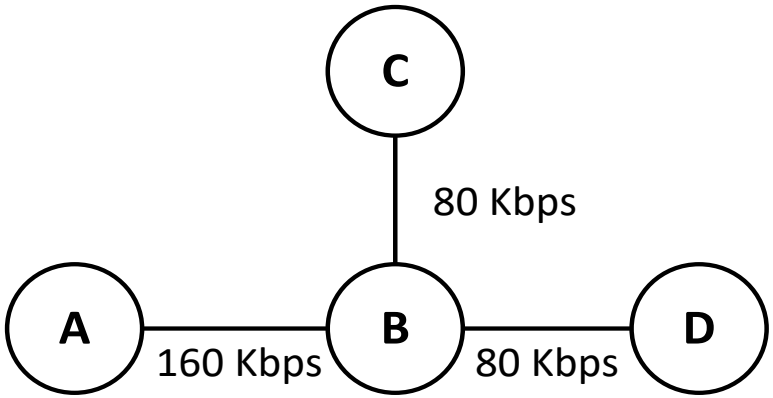
Esta condição impõe um limite inferior ao tamanho da janela. Se $d \leq WX$ então o fluxo pode transmitir à velocidade máxima pelo que o tamanho da janela (em número de pacotes) deverá ser dado por

$$W = \left\lceil \frac{d}{X} \right\rceil$$

onde $\lceil z \rceil$ representa o menor inteiro não inferior a z e d deverá ser o menor atraso de ida-e-volta proporcionado pela rede.

Exemplo 1

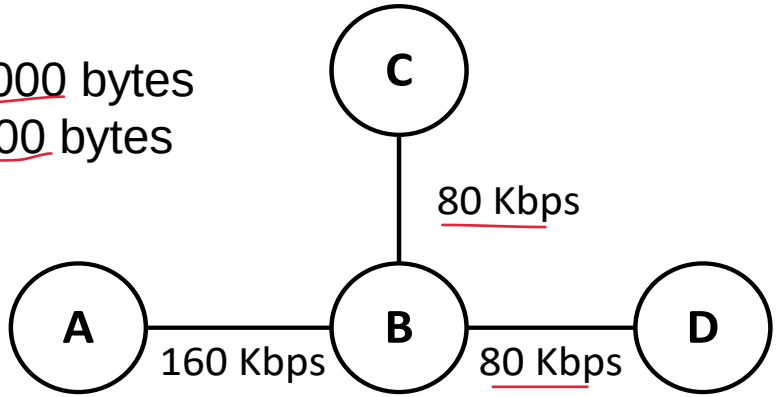
Considere a rede com comutação de pacotes da figura em que o atraso de propagação de cada ligação é 10 mseg em cada sentido. A rede suporta dois fluxos: A→D com pacotes de tamanho médio 1000 bytes e C→D com pacotes de tamanho médio 500 bytes. A ambos os fluxos é aplicado um mecanismo de controle de fluxos baseado no método das janelas extremo-a-extremo e em ambos os casos, as permissões têm um tamanho fixo de 100 Bytes. Determine o tamanho mínimo (em número de pacotes) das janelas de emissão garantindo que cada fluxo pode emitir ao ritmo máximo quando o outro não está a emitir pacotes.



Exemplo 1 - resolução

A → D com pacotes de tamanho médio 1000 bytes
C → D com pacotes de tamanho médio 500 bytes

$$W \geq \left\lceil \frac{d}{X} \right\rceil$$



$$W_{AD} \geq \left\lceil \frac{\frac{8\times1000}{160000}+0.01+\frac{8\times1000}{80000}+0.01+\frac{8\times100}{80000}+0.01+\frac{8\times100}{160000}+0.01}{\frac{8\times1000}{80000}} \right\rceil = \left\lceil \frac{0.205}{0.1} \right\rceil = 3 \text{ pacotes}$$

$$W_{CD} \geq \left\lceil \frac{\frac{8\times500}{80000}+0.01+\frac{8\times500}{80000}+0.01+\frac{8\times100}{80000}+0.01+\frac{8\times100}{80000}+0.01}{\frac{8\times500}{80000}} \right\rceil = \left\lceil \frac{0.16}{0.05} \right\rceil = 4 \text{ pacotes}$$

Limitações do controlo de fluxo baseado em janelas extremo-a-extremo

1. Não permite assegurar uma taxa mínima de transmissão. Quantos mais fluxos forem submetidos na rede, menor é o tráfego efetivo que cada fluxo obtém.
2. Não fornece um controlo adequado do atraso. Considerem-se n fluxos com controlo de fluxos ativo através de janelas com tamanho fixo W_1, …, W_n. O número total de pacotes e permissões é $\sum_{i=1}^{n} W_i$ e o número de pacotes é $\sum_{i=1}^{n} \beta_i W_i$ onde β_i é um valor entre 0 e 1.

   Pelo teorema de Little, o atraso médio por pacote é

   $$T = \frac{\sum_{i=1}^{n} \beta_i W_i}{\lambda}$$

   onde λ é o tráfego efetivo de todos os fluxos. À medida que o número de fluxos aumenta, o tráfego efetivo tende para um valor constante (limitado pela capacidade das ligações). Assim, o atraso médio por pacote aumenta aproximadamente de forma proporcional ao número de fluxos.

Controlo de Fluxos em Redes com Comutação de Pacotes

Segunda parte:

- **Mecanismos de controlo de taxas de transmissão de fluxos de pacotes**
- **Atribuição de taxas de transmissão a fluxos de pacotes segundo o princípio de equidade do tipo max-min**

Controlo de taxas de transmissão

- A função de controlo de fluxos pode atribuir a cada fluxo uma taxa de transmissão máxima compatível com as suas necessidades.

- Essa taxa pode, por exemplo, ser definida na fase de estabelecimento de um circuito virtual (redes IP com RSVP, redes MPLS).

- De seguida, consideram-se dois métodos para controlar a taxa de transmissão:
  - por janelas
  - através de *leaky bucket* (usado pela arquitetura *Integrated Services* (IntServ) nas redes IP)

Controlo de taxas de transmissão por janelas (I)

- Considere-se que foi atribuída uma taxa de transmissão de *r* pacotes por segundo a um determinado fluxo (de um emissor para um recetor).

- Uma possibilidade para garantir esta taxa poderia ser aceitar no emissor, quando muito, um pacote em cada 1/*r* segundos.

- No entanto, este esquema tende a introduzir grandes atrasos quando a fonte que gera os pacotes no emissor é em rajada.

- Neste caso, é preferível aceitar no emissor *W* pacotes em cada *W*/*r* segundos (permite rajadas de *W* pacotes).

Controlo de taxas de transmissão por janelas (II)

Se foi atribuído a um determinado fluxo: (i) uma taxa de transmissão de *r* pacotes/segundo e (ii) uma janela de *W* pacotes, então:

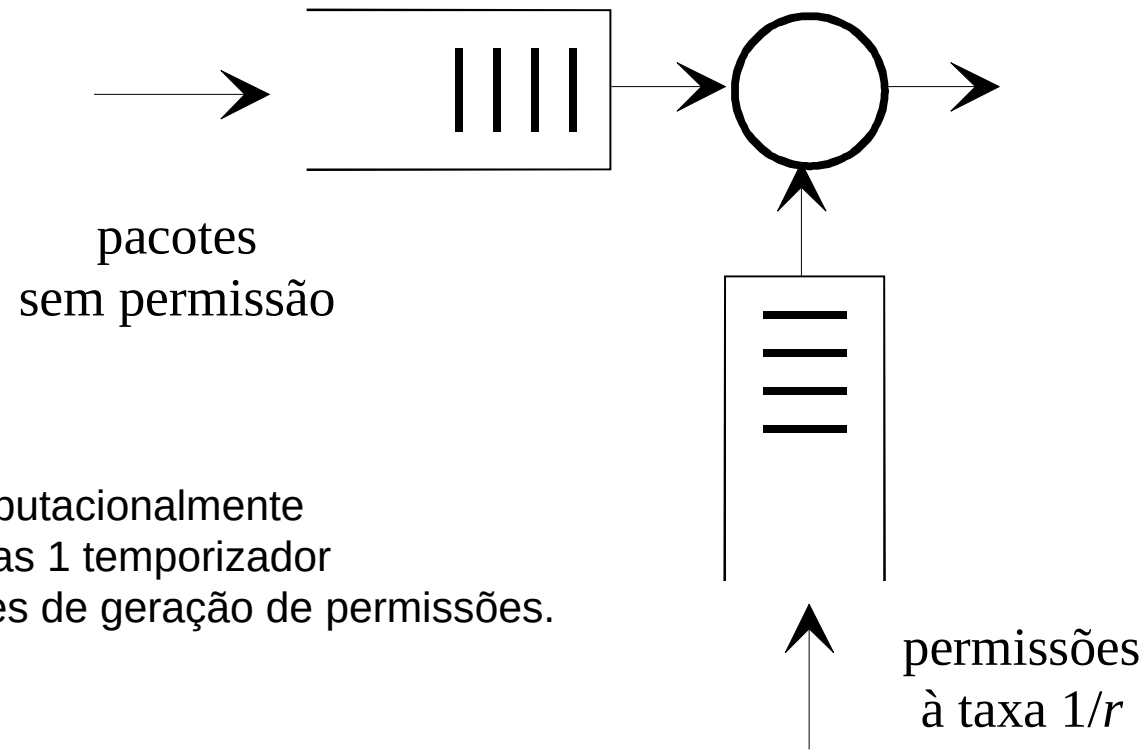
1. O emissor mantém um contador *x* que indica, em cada instante, o número de pacotes dessa janela que ainda pode ser transmitido (*x* é inicializado a *W*).
2. Sempre que um pacote é transmitido, o contador *x* é decrementado e passados *W*/*r* segundos é novamente incrementado (exige um temporizador por cada pacote transmitido).
3. Os pacotes só são enviados para a rede se $x > 0$ (o número máximo de temporizadores é *W*).

Nota: O método do controlo de fluxo por janelas extremo-a-extremo é semelhante a este com a diferença apenas de que o contador é incrementado por cada permissão recebida.

Desvantagem: este método é computacionalmente pesado pois exige *W* temporizadores simultâneos por cada fluxo.

Controlo de taxas de transmissão por *leaky bucket*

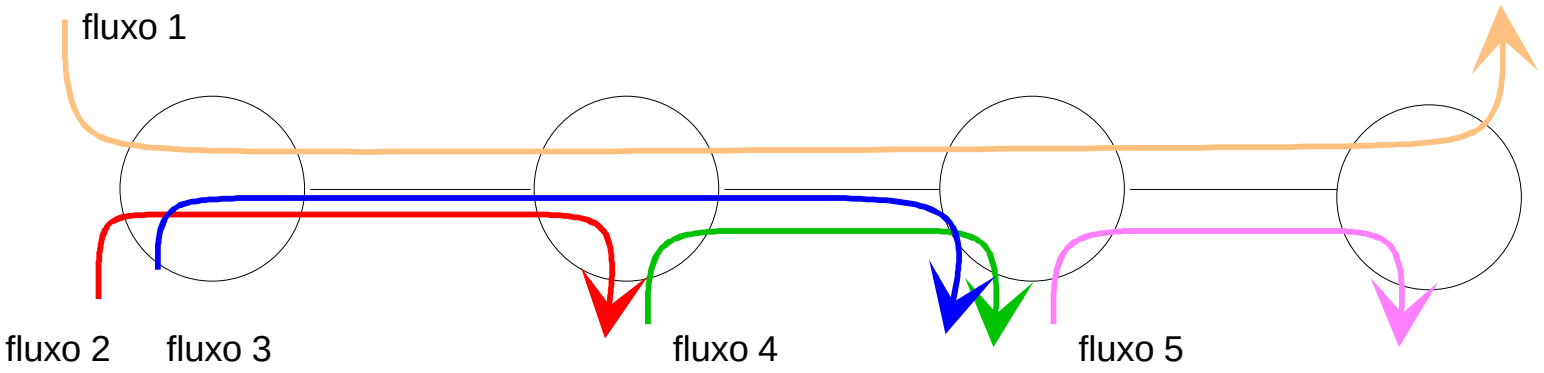
- Neste método, o contador é incrementado periodicamente em cada 1/*r* segundos, até um máximo de *W* pacotes.
- O método pode ser visto da seguinte forma (modelo *leaky bucket*):
  - existe uma fila de espera de pacotes e uma fila de espera de permissões, com capacidade para *W* permissões;
  - é gerada uma nova permissão em cada 1/*r* segundos;
  - os pacotes só são transmitidos quando existe uma permissão disponível na fila de espera respetiva.



Vantagem: este método é computacionalmente menos pesado pois exige apenas 1 temporizador por fluxo para definir os instantes de geração de permissões.

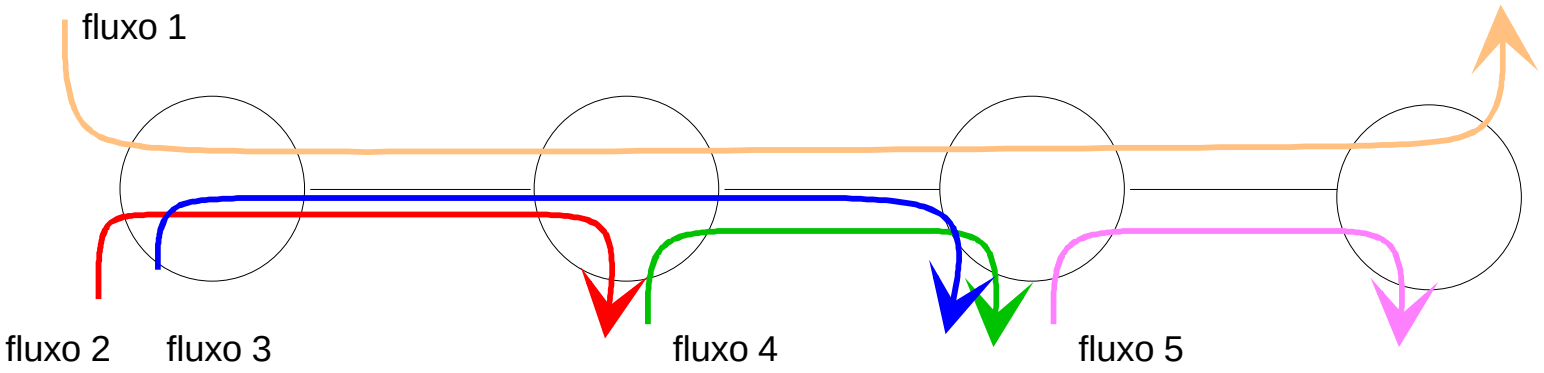
Atribuição de taxas de transmissão

- Considere a rede da figura em que as ligações têm todas capacidade para 120 pacotes/s.
- Uma solução equilibrada (*fair*) seria atribuir a todos os fluxos uma taxa de 1/3 $\times$ 120 = 40 pacotes/s.
- No entanto, não faz sentido restringir a taxa do fluxo 5 a 40 pacotes/s, pois este fluxo pode usar 80 pacotes/s sem prejudicar os fluxos 1, 2, 3 e 4.



Equidade do tipo *max-min*

- Surge assim o conceito de equidade do tipo max-min (*max-min fairness*).
- Segundo este princípio, maximizam-se os recursos atribuídos aos fluxos que podem usar menos recursos.
- Uma forma alternativa de formular este princípio:
  - Maximizam-se as taxas atribuídas a cada fluxo, respeitando a restrição segundo a qual um incremento na atribuição ao fluxo *i* não conduz a uma diminuição da taxa atribuída a qualquer outro fluxo cuja taxa seja menor ou igual que a de *i*.



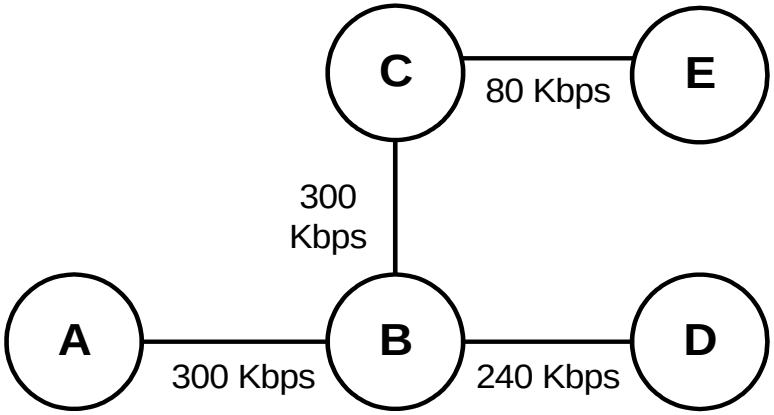
Exemplo 2

Considere a rede com comutação de pacotes da figura.

A rede suporta 5 fluxos de pacotes: de A para B, de A para C, de A para D, de B para D e de B para E.

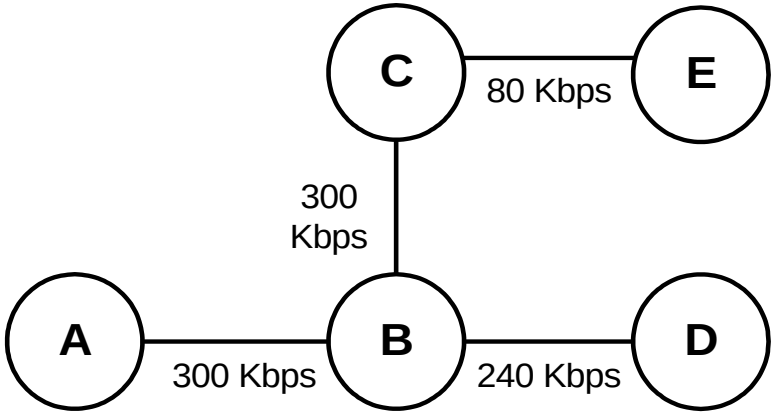
A rede permite controlar a taxa de transmissão máxima de cada fluxo através de um qualquer mecanismo adequado.

Calcular que taxas de transmissão máxima se devem atribuir a cada fluxo segundo o princípio de equidade do tipo *max-min*.



Exemplo 2 - resolução

5 fluxos de pacotes:
- de A para B
- de A para C
- de A para D
- de B para D
- de B para E



<u>1ª iteração:</u> - a ligação AB atribui 300/3 = 100 Kbps por fluxo
- a ligação BC atribui 300/2 = 150 Kbps por fluxo
- a ligação BD atribui 240/2 = 120 Kbps por fluxo
- a ligação CE atribui 80/1 = 80 Kbps por fluxo

O menor valor é o da ligação CE: é atribuído 80 Kbps ao fluxo B→E.

<u>2ª iteração:</u> - a ligação AB atribui 300/3 = 100 Kbps por fluxo
- a ligação BC atribui (300–80)/1 = 220 Kbps por fluxo
- a ligação BD atribui 240/2 = 120 Kbps por fluxo

O menor valor é o da ligação AB: é atribuído 100 Kbps aos fluxos A→B, A→C e A→D.

<u>3ª iteração:</u> - a ligação BD atribui (240–100)/1 = 140 Kbps por fluxo

É atribuído 140 Kbps ao fluxo B→D.

Mecanismos de Escalonamento e de Descarte de Pacotes em Redes com Comutação de Pacotes

Primeira parte:

- **Caracterização das disciplinas de escalonamento de pacotes**

Equidade das disciplinas de escalonamento

Quando uma ligação está congestionada (*i.e.*, a sua fila de espera não está vazia), o problema mais básico que se coloca à função de escalonamento é:

> divisão de um recurso escasso por fluxos com iguais direitos mas com diferentes necessidades de utilização desse recurso.

Idealmente, a atribuição deve ser feita de acordo com o princípio de equidade *max-min*:

- Os recursos são atribuídos aos fluxos por ordem crescente de necessidade.
- A nenhum fluxo é atribuída uma quantidade de recursos maior do que a sua necessidade.
- A fluxos cuja necessidade não tenha sido satisfeita é atribuída uma igual quantidade de recursos.

Equidade max-min com direitos iguais

Considere-se:

- um conjunto de fluxos 1, 2, ..., *n* com necessidades x_1, x_2, ..., x_n e ordenados pelas suas necessidades ($x_1 \leq x_2 \ldots \leq x_n$);
- uma ligação com capacidade *C*.

A atribuição dos recursos da ligação é efetuada do seguinte modo:

- Inicialmente todos os fluxos têm direito a $d = C/n$
- *d* é menor que x_1?
  - se sim, atribui-se *d* a todos os fluxos, i.e., aos fluxos 1, 2, …, *n*;
  - se não, atribui-se x_1 ao fluxo 1 e os fluxos 2, 3, …, *n* têm direito a $d = d + (d - x_1)/(n - 1)$
- *d* é menor que x_2?
  - se sim, atribui-se *d* aos fluxos 2, 3, …, *n*;
  - se não, atribui-se x_2 ao fluxo 2 e os fluxos 3, 4, …, *n* têm direito a $d = d + (d - x_2)/(n - 2)$
- E assim sucessivamente...

Exemplo 1

Considere-se uma ligação com capacidade de 128 Mbps e 4 fluxos de tráfego de 8, 36, 48 e 128 Mbps. Determine que recursos são atribuídos a cada fluxo pelo princípio de equidade max-min quando todos os fluxos têm direitos iguais.

i) O fluxo 1 tem direito a d = 128/4 = 32 Mbps.

Como o fluxo 1 gera menos que 32 Mbps, o fluxo 1 fica com 8 Mbps. Sobram 32 – 8 = 24 Mbps.

ii) O fluxo 2 tem direito a d = 32 + 24/3 = 40 Mbps.

Como o fluxo 2 gera menos que 40 Mbps, o fluxo 2 fica com 36 Mbps. Sobram 40 – 36 = 4 Mbps.

ii) O fluxo 3 tem direito a d = 40 + 4/2 = 42 Mb/s.

Como o fluxo 3 (e o fluxo 4) geram mais de 42 Mbps, os fluxos 3 e 4 ficam com 42 Mbps.

Equidade max-min com direitos diferentes

São atribuídos pesos aos fluxos proporcionais aos seus direitos.

A atribuição de recursos é feita de acordo com o princípio *weighted max-min* fair.

Neste caso:

- Os recursos são atribuídos aos fluxos por ordem crescente de necessidade, estando esta normalizada em relação ao peso.
- A nenhum fluxo é atribuído uma quantidade de recursos maior do que a sua necessidade.
- A fluxos cuja necessidade não tenha sido satisfeita é atribuída uma quantidade de recursos proporcional ao seu peso.

Exemplo 2

Considere uma ligação com capacidade de 128 Mbps e 4 fluxos de tráfego de 8, 36, 48 e 128 Mbps. Determine que recursos são atribuídos a cada fluxo quando os fluxos têm pesos 1, 1, 3 e 3, respetivamente.

i) Fluxo 1 : $1/(1+1+3+3) \times 128 = 16$ Mbps

Fluxo 2 : $1/(1+1+3+3) \times 128 = 16$ Mbps

Fluxo 3 : $3/(1+1+3+3) \times 128 = 48$ Mbps

Fluxo 4 : $3/(1+1+3+3) \times 128 = 48$ Mbps

Atribui-se 8 Mbps ao fluxo 1 (<16 Mbps) e 48 Mbps ao fluxo 3.

Sobram $(16 – 8) + (48 – 48) = 8$ Mbps.

ii) Fluxo 2 : $16 + 1/(1+3) \times 8 = 18$ Mbps

Fluxo 4 : $48 + 3/(1+3) \times 8 = 54$ Mbps

Atribui-se 18 Mbps ao fluxo 2 (<36 Mbps) e 54 Mbps ao fluxo 4 (<128 Mbps).

Comparação dos Exemplos 1 e 2

Capacidade da ligação: 128 Mbps

Fluxos:	1	2	3	4
Débito de transmissão (Mbps):	8	36	48	128
Pesos:	1	1	1	1
Atribuição (Mbps):	8	36	42	42
Pesos:	1	1	3	3
Atribuição (Mbps):	8	18	48	54

- Quando os pesos são todos iguais, os fluxos 1 e 2 conseguem todo o seu débito porque são os fluxos com menor débito de transmissão
- Quando os fluxos 3 e 4 têm 3 vezes maior peso que os fluxos 1 e 2, conseguem maior débito enquanto que o fluxo 2 já não tem todo o seu débito de transmissão.

Proteção nas disciplinas de escalonamento

Idealmente, a função de escalonamento deve procurar proteger os fluxos bem comportados dos fluxos mal comportados.

Um fluxo mal comportado é um fluxo que envia tráfego a uma taxa superior à taxa a que tem direito (de acordo com o princípio de atribuição de recursos em vigor).

Como veremos à frente:

- as disciplina de escalonamento do tipo FIFO ou com prioridades não protegem os fluxos bem comportados dos fluxos mal comportados;
- por exemplo, as disciplinas de escalonamento do tipo *round-robin* conseguem.

Disciplinas de escalonamento

As disciplinas de escalonamento podem classificar-se em disciplinas com e sem conservação de trabalho (*work conserving*):

- numa disciplina com conservação de trabalho, a ligação só está inativa (i.e., não está a seu usada para transmitir pacotes) se não houver qualquer pacote à espera de ser transmitido;
- numa disciplina sem conservação de trabalho, a ligação pode estar inativa mesmo que haja pacotes na fila de espera.

Todas as disciplinas de escalonamento que iremos abordar são disciplinas com conservação de trabalho e são as seguintes:

(1) por ordem de chegada (FIFO),

(2) com base em prioridade estrita,

(3) de uma forma rotativa (RR, WRR, DRR),

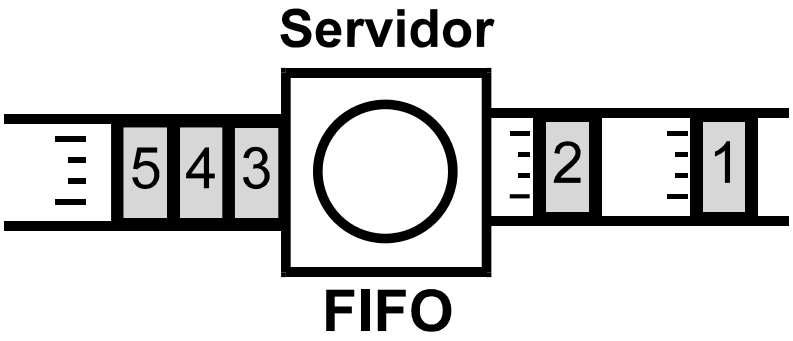
(4) por aproximação ao sistema GPS (WFQ, SCFQ).

Mecanismos de Escalonamento e de Descarte de Pacotes em Redes com Comutação de Pacotes

Segunda parte:

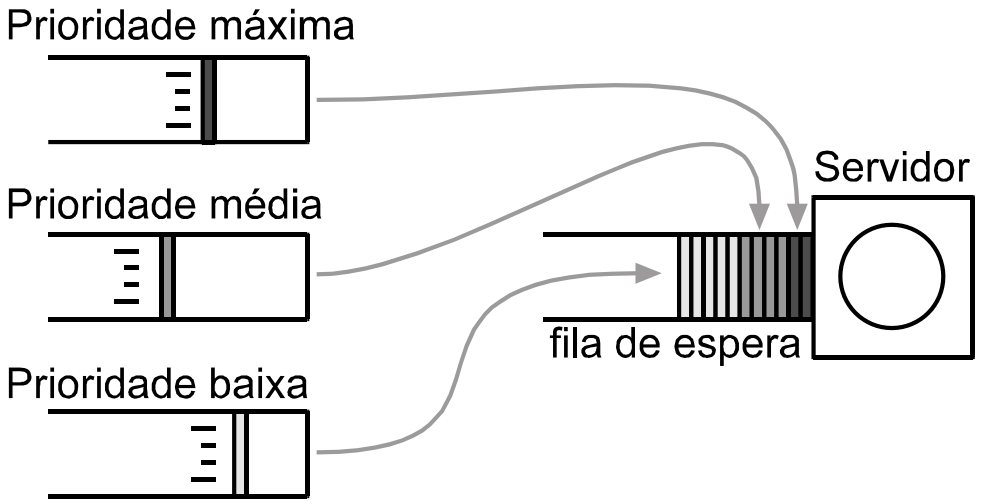
- **Disciplinas de escalonamento de pacotes: FIFO, com prioridades e que funcionam de forma rotativa**

First-In-First-Out (FIFO)



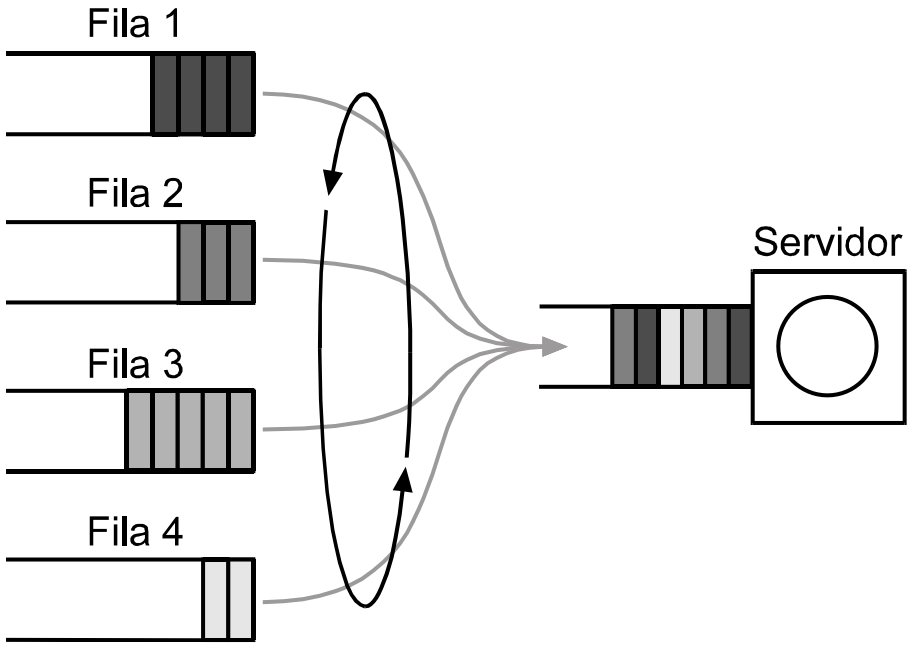
- Os pacotes de todos os fluxos são transmitidos pela sua ordem de chegada.
- Não envolve processamento de ordenação nem de classificação de pacotes.
- Não permite diferenciação de qualidade de serviço (o atraso médio na fila de espera é igual para os pacotes de todos os fluxos).
- Quando a fila de espera não está vazia, fluxos com *n* vezes mais tráfego recebem *n* vezes mais taxa de serviço pelo que os fluxos bem comportados não são protegidos.

Prioridade Estrita



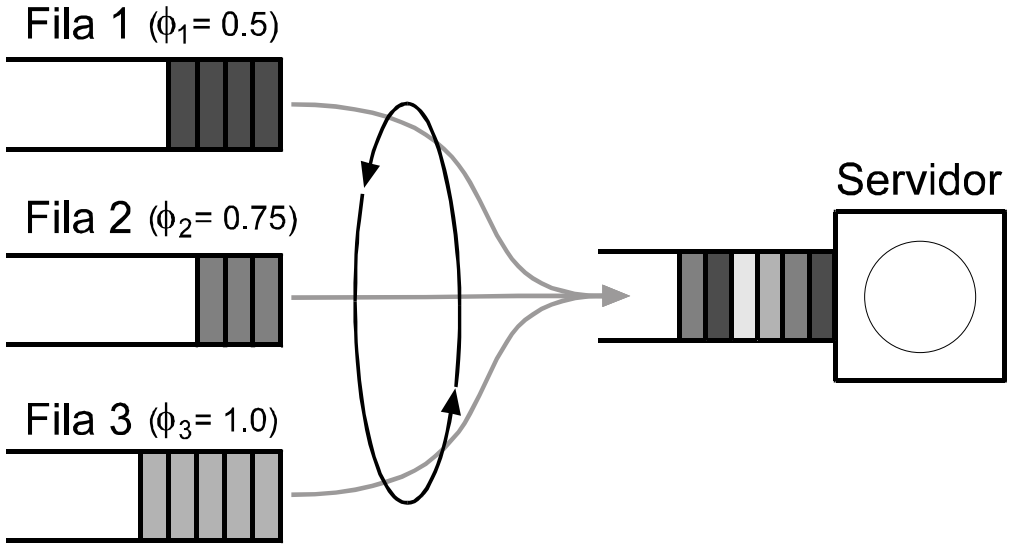
- Os pacotes classificados como de maior prioridade são sempre transmitidos antes dos pacotes de menor prioridade (os pacotes com a mesma prioridade são transmitidos com a disciplina FIFO).
- Não envolve processamento de ordenação.
- Envolve classificação dos pacotes de acordo com a prioridade.
- Permite diferenciação da qualidade de serviço (o atraso médio na fila de espera é menor para os pacotes de maior prioridade).
- Fluxos de pacotes de maior prioridade podem impedir que os fluxos de menor prioridade recebam qualquer serviço.

Round Robin (RR)



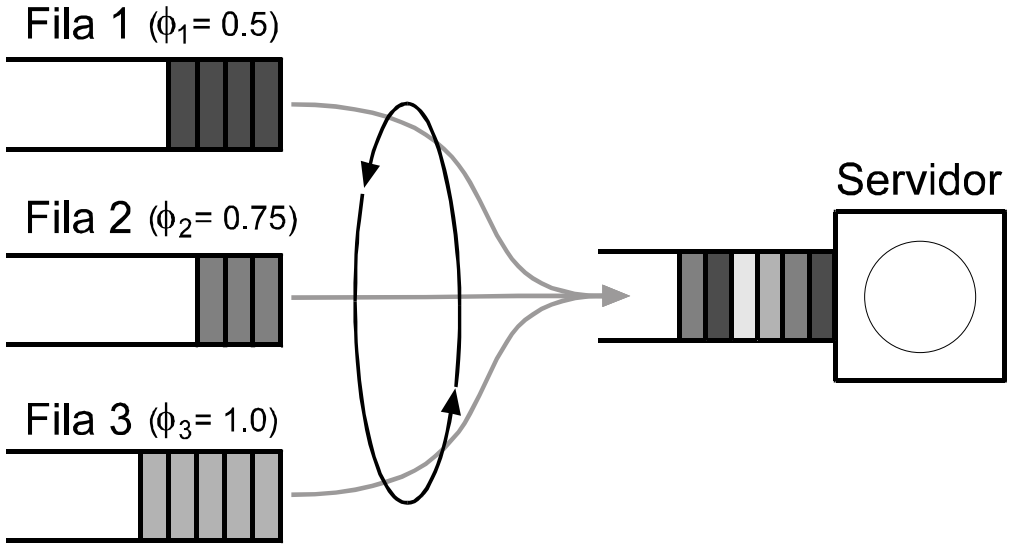
- Existe uma fila por fluxo de pacotes e o algoritmo seleciona um pacote de cada fila não vazia de forma rotativa.
- Não permite diferenciação de qualidade de serviço.
- Ao contrário do FIFO, o RR serve o mesmo número de pacotes de todos os fluxos ativos (i.e. fluxos com pacotes na fila).
- Fluxos de pacotes maiores têm maior taxa de serviço.
- Protege os fluxos bem comportados (os fluxos mal comportados apenas penalizam o seu próprio atraso na fila de espera).

Weighted Round Robin (WRR)



- É atribuído um peso ϕ_i a cada fila de espera proporcional à taxa de serviço a proporcionar a cada fluxo em situação de congestão.
- Em cada ciclo, o WRR serve um número de pacotes de cada fila de espera tal que a soma dos seus tamanhos (em bytes) é proporcional ao peso da fila.
- É necessário conhecer a priori o comprimento médio dos pacotes.
- A ligação pode ficar demasiado tempo a servir cada fluxo de pacotes o que tem um impacto negativo no *jitter* introduzido pela ligação.

Weighted Round Robin (WRR)



No exemplo da figura, se o comprimento médio (em Bytes) dos pacotes de cada fluxo for:

$L_1 = 50, L_2 = 500, L_3 = 1500$

Os pesos normalizados são:

$\varphi_1 = 0.5/50 = 1/100 = 60/6000$

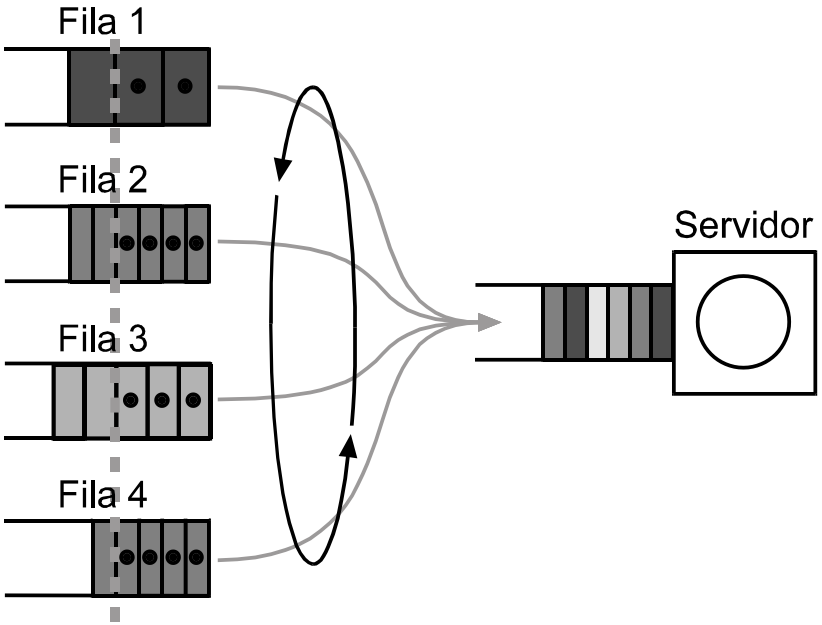
$\varphi_2 = 0.75/500 = 3/2000 = 9/6000$

$\varphi_3 = 1/1500 = 4/6000$

Número de pacotes por ciclo:

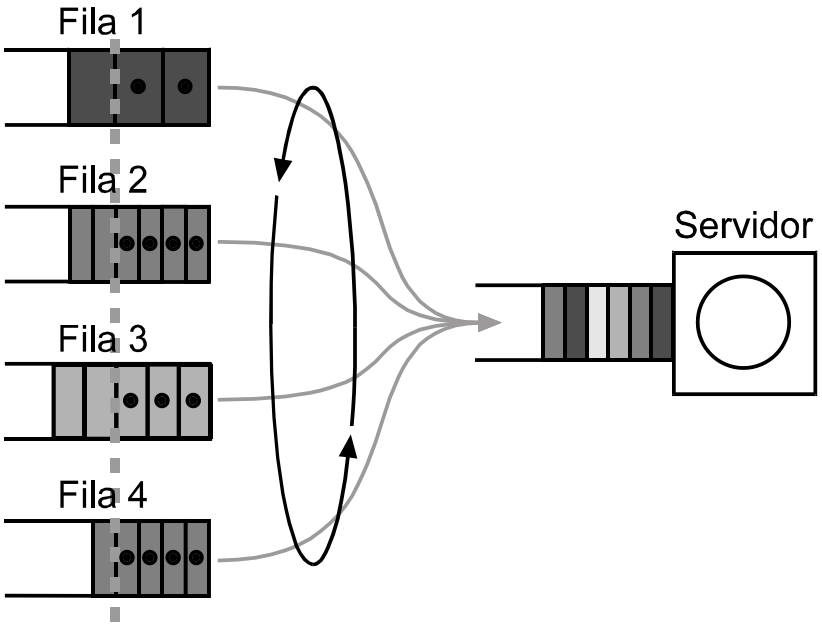
$\Phi_1 = 60,\ \Phi_2 = 9,\ \Phi_3 = 4$

Deficit Round Robin (DRR)

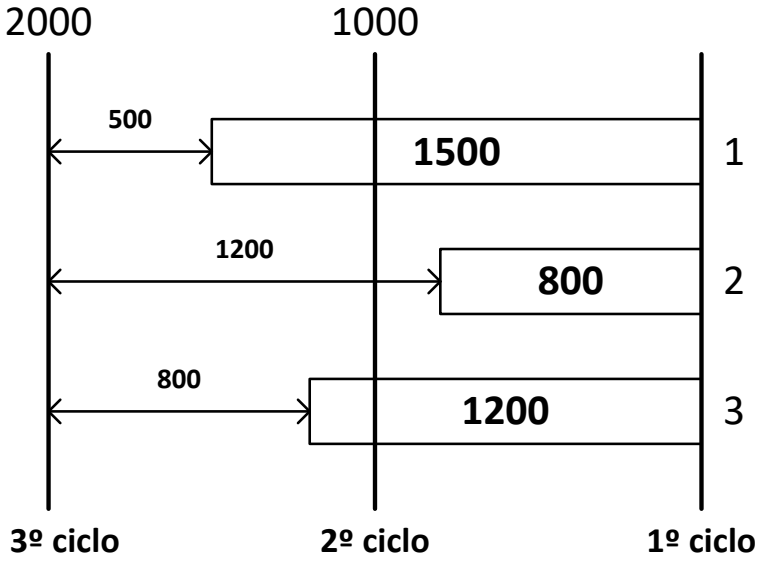


- Em cada ciclo, o DRR serve uma quantidade de bytes até um valor máximo designado por *limiar*.
- A diferença entre a quantidade servida e o limiar é contabilizada em forma de *crédito* para o ciclo seguinte.
- Quando uma fila está vazia, o crédito respetivo é colocado a zero.
- Se se considerarem limiares diferentes para as diferentes filas, a taxa de serviço de cada fluxo é proporcional ao limiar da sua fila de espera.
- Ao contrário do WRR, não é necessário saber o comprimento médio dos pacotes.

Deficit Round Robin (DRR)



limiar = 1000 bytes (para todos os fluxos)



1º Ciclo:

a) fila 1 não é servida, obtém crédito de 1000
b) fila 2 é servida, obtém crédito de 200
c) fila 3 não é servida, obtém crédito de 1000

2º Ciclo:

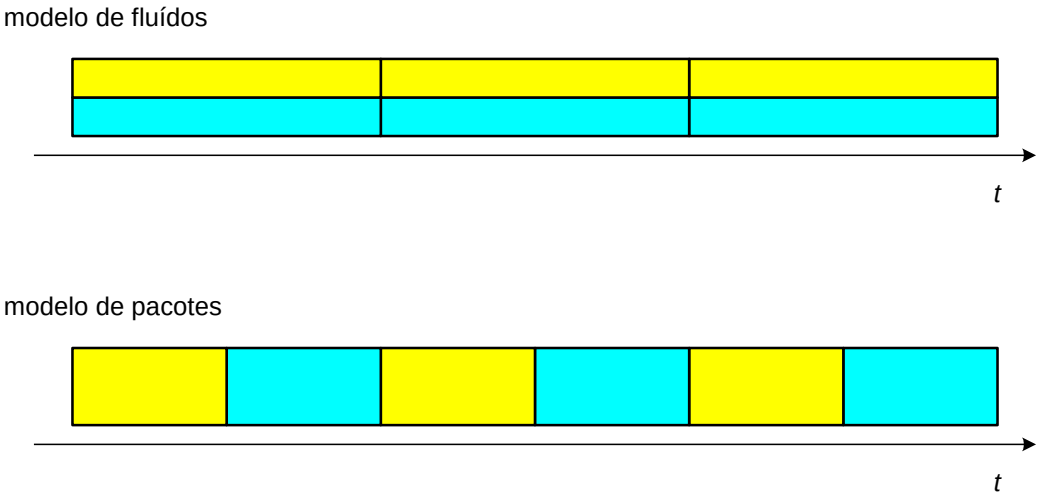
a) fila 1 é servida, obtém crédito de 500
b) fila 2 está vazia, fica com crédito a 0
c) fila 3 é servida, obtém crédito de 800

Mecanismos de Escalonamento e de Descarte de Pacotes em Redes com Comutação de Pacotes

Terceira parte:

- **Disciplinas de escalonamento de pacotes que funcionam por aproximação ao sistema GPS**

Generalized Processor Sharing (GPS)



- Algoritmo ideal que proporciona equidade perfeita, baseado num modelo de fluídos, em que o tráfego é considerado infinitamente divisível.
  - Exemplo: num dado instante, 50% da capacidade de uma ligação é utilizada por um fluxo e 50% por outro fluxo.
- Existe uma fila de espera por fluxo e é atribuído um peso ϕ_i a cada fluxo.
- Quando um pacote chega a uma fila, se nenhum outro pacote da mesma fila estiver a ser transmitido, este começa imediatamente a ser transmitido, em paralelo com os pacotes das outras filas, a uma taxa de serviço proporcional ao seu peso.
- É um algoritmo impossível de realizar na prática, mas constitui uma boa base teórica para o desenvolvimento de outros algoritmos.

Exemplo 3

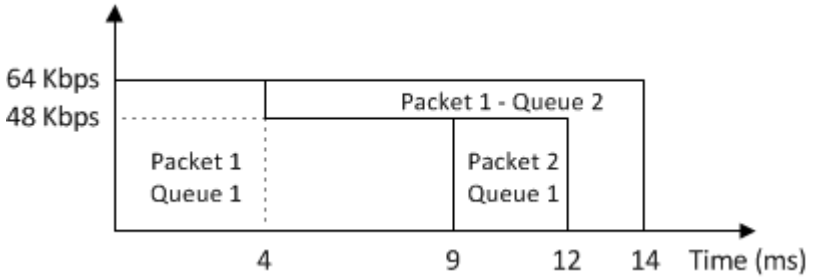
Considere-se uma ligação de 64 Kbps com 2 filas de espera de pesos ϕ_1 = 3 e ϕ_2 = 1, em que os 2 fluxos de pacotes são servidos pela disciplina de escalonamento ideal GPS. Chegam a esta ligação os seguintes pacotes:

- pacote 1 à fila 1 com 62 Bytes em t = 0,
- pacote 1 à fila 2 com 32 Bytes em t = 4 ms e
- pacote 2 à fila 1 com 18 Bytes em t = 6 ms.

Determinar os instantes em que os pacotes são servidos (i.e., os instantes de tempo em que termina a transmissão de cada pacote).

Resolução do Exemplo 3

Ligação: 64 Kbps
2 filas de espera: $\phi_1 = 3$ e $\phi_2 = 1$
Chegam:
- pacote 1 à fila 1 com 62 Bytes ($t = 0$),
- pacote 1 à fila 2 com 32 Bytes ($t = 4$ ms) e
- pacote 2 à fila 1 com 18 Bytes ($t = 6$ ms).



- O pacote 1 da fila 1 é servido inicialmente a 64 Kb/s. Em $t = 4$ ms, foram servidos (64Kb/s)×(4ms) = 256 bits = 32 Bytes do pacote 1 da fila 1. A partir daqui, a fila 1 é servida a (3/4)×64 Kb/s = 48 Kb/s e a fila 2 a (1/4)×64 Kb/s = 16 Kb/s.
- Com estas taxas, o pacote 1 da fila 1 demora ((62–32)×8)/(48Kb/s) = 5 ms a finalizar a sua transmissão e o pacote 1 da fila 2 demora (32×8)/(16Kb/s) = 16 ms. Assim, o pacote 1 da fila 1 termina a sua transmissão em $t = 4 + 5 = 9$ ms. Neste instante, inicia-se a transmissão do pacote 2 da fila 1 porque chegou no instante $t = 6$ ms.
- O pacote 2 da fila 1 demora (18×8)/(48Kb/s) = 3 ms a ser transmitido. Assim, o pacote 2 da fila 1 termina a sua transmissão em $t = 9 + 3 = 12$ ms.
- A partir de $t = 12$ ms, o pacote 1 da fila 2 é transmitido a 64 Kb/s. Como até este instante foram transmitidos (16Kb/s)×(8ms) = 128 bits = 16 Bytes, os restantes 16 Bytes demoram (16×8)/(64Kb/s) = 2 ms. Assim, o pacote 1 da fila 2 termina a transmissão em $t = 12 + 2 = 14$ ms.

Weighted Fair Queuing (WFQ)

É uma aproximação ao sistema GPS: o WFQ tenta servir os pacotes pela ordem em que terminariam de ser transmitidos no sistema GPS.

Sempre que chega um pacote a uma fila, é atribuído ao pacote um ***Finish Number*** (*FN*) que indica a ordem pela qual ele será enviado relativamente aos outros pacotes.

Round Number (*RN*) é uma variável real que cresce no tempo a uma taxa inversamente proporcional aos pesos dos fluxos ativos.

Num intervalo de tempo $[\tau_i,\ \tau_{i+1})$ em que o número de fluxos ativos se mantenha constante:

$$RN(\tau_i + t) = RN(\tau_i) + \frac{1}{\sum_{j\ ativos} \phi_j} t \qquad t \in \left[\tau_i, \tau_{i+1}\right)$$

O *RN* é processado sempre que o número de fluxos ativos se altera:

- quando um pacote chega de um fluxo que não tem pacotes no sistema;
- quando um pacote de um fluxo termina de ser transmitido e o fluxo não tem nenhum outro pacote na fila de espera.

Quando o pacote k com comprimento L_k pertencente à fila i chega, é-lhe atribuído o *finish number* $FN_{i,k}$ dado por:

$$FN_{i,k} = \max\left(FN_{i,k-1}, RN\right) + \frac{L_k / C}{\phi_i}$$

Self Clock Fair Queuing (SCFQ)

A principal desvantagem do WFQ é o peso computacional do cálculo do *RN*.

Por forma a evitar o cálculo do *RN* do WFQ, o SCFQ substitui este parâmetro pelo valor do *FN* do pacote que está a ser transmitido, FN_s, qualquer que seja o fluxo a que pertence.

Assim, quando o pacote k com comprimento L_k pertencente à fila i chega, é-lhe atribuído o *finish number* $FN_{i,k}$ dado por:

$$FN_{i,k} = \max\left(FN_{i,k-1}, FN_s\right) + \frac{L_k}{\phi_i}$$

Não se utiliza o valor da capacidade da ligação (*C*), uma vez que não é necessário saber o tempo que o pacote demoraria a ser servido no sistema GPS.

Apesar do SCFQ ser de muito menor complexidade que o WFQ, pode não ser tão justo para pequenos intervalos de tempo (i.e., não se aproxima tão bem ao GPS como o WFQ).

Exemplo 4

Considere-se uma ligação de 64 Kbps com 2 filas de espera de pesos $\phi_1 = 3$ e $\phi_2 = 1$. Chegam a esta ligação os seguintes pacotes:

- pacote 1 à fila 1 com 62 Bytes em $t = 0$,
- pacote 1 à fila 2 com 32 Bytes em $t = 4$ ms e
- pacote 2 à fila 1 com 18 Bytes em $t = 6$ ms.

Determinar os instantes em que os pacotes são servidos (i.e., os instantes de tempo em que termina a transmissão de cada pacote) considerando que os 2 fluxos de pacotes são servidos por uma:

(a) uma disciplina de escalonamento WFQ

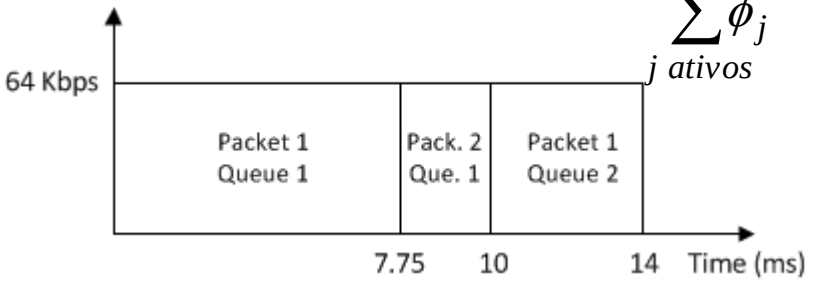
(b) uma disciplina de escalonamento SCFQ

Exemplo 4 – resolução de (a)

$$FN_{i,k} = \max\left(FN_{i,k-1}, RN\right) + \frac{L_k / C}{\phi_i}$$

$$RN(\tau_i + t) = RN(\tau_i) + \frac{1}{\sum_{j\ ativos} \phi_j} t$$

Ligação: 64 Kbps
2 filas de espera: $\phi_1 = 3$ e $\phi_2 = 1$
Chegam:
- pacote 1 à fila 1 com 62 Bytes ($t = 0$),
- pacote 1 à fila 2 com 32 Bytes ($t = 4$ ms) e
- pacote 2 à fila 1 com 18 Bytes ($t = 6$ ms).

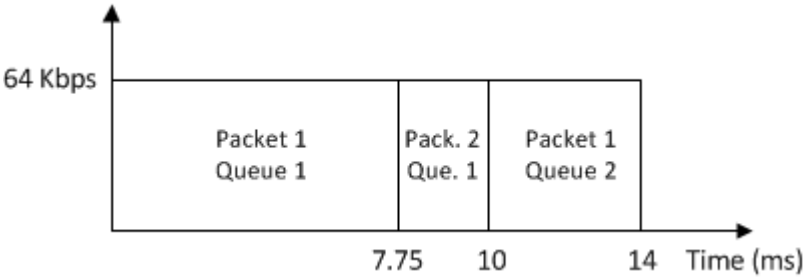


- Em $t = 0$ ms, $RN = 0$ e $FN_{1,1} = 0 + (62\times8)/64000/3 = 2.58\times10^{-3}$. O pacote 1 da fila 1 é transmitido em (62×8)/(64Kb/s) = 7.75 ms. Assim, o pacote 1 da fila 1 termina a sua transmissão em $t = 0 + 7.75 = 7.75$ ms.
- Em $t = 4$ ms : $RN = 0 + (4\times10^{-3})/3 = 1.33\times10^{-3}$
  $FN_{2,1} = 1.33\times10^{-3} + (32\times8)/64000/1 = 5.33\times10^{-3}$
- Em $t = 6$ ms : $RN = 1.33\times10^{-3} + (6\times10^{-3} - 4\times10^{-3})/4 = 3.33\times10^{-3}$
  $FN_{1,2} = \max(2.58\times10^{-3}, 3.33\times10^{-3}) + (18\times8)/64000/3 = 4.08\times10^{-3}$
- Em $t = 7.75$ ms, como $FN_{1,2} < FN_{2,1}$, o pacote 2 da fila 1 começa a ser transmitido. O pacote 2 da fila 1 é transmitido em (18×8)/(64Kb/s) = 2.25 ms. Assim, o pacote 2 da fila 1 termina a sua transmissão em $t = 7.75 + 2.25 = 10$ ms.
- Em $t = 10$ ms, o pacote 1 da fila 2 começa a ser transmitido. O pacote 1 da fila 2 é transmitido em (32×8)/(64Kb/s) = 4 ms. Assim, o pacote 1 da fila 2 termina a sua transmissão em $t = 10 + 4 = 14$ ms.

Exemplo 4 – resolução de (b)

$$FN_{i,k} = \max\left(FN_{i,k-1}, FN_s\right) + \frac{L_k}{\phi_i}$$

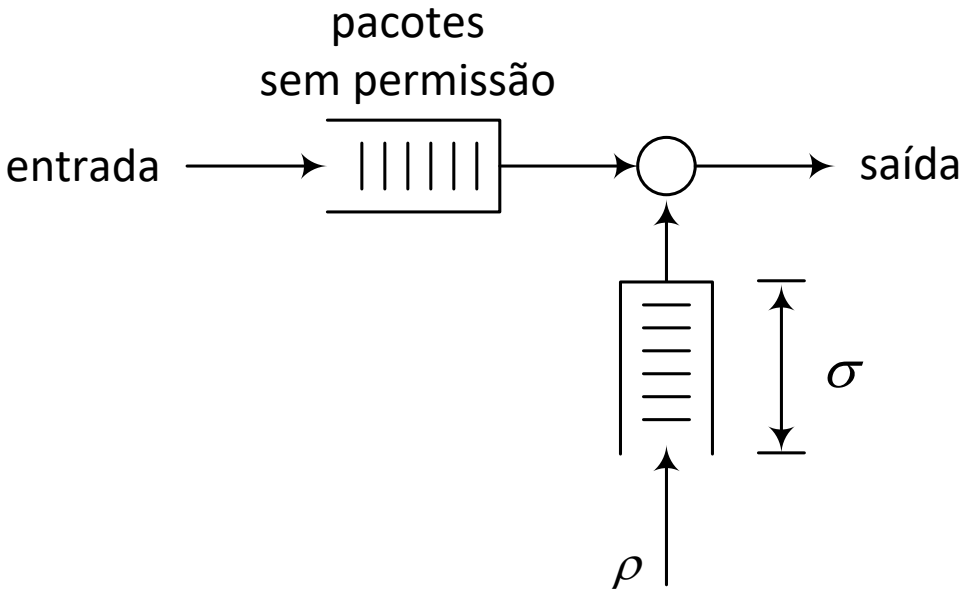
Ligação: 64 Kbps
2 filas de espera: $\phi_1 = 3$ e $\phi_2 = 1$
Chegam:
- pacote 1 à fila 1 com 62 Bytes ($t = 0$),
- pacote 1 à fila 2 com 32 Bytes ($t = 4$ ms) e
- pacote 2 à fila 1 com 18 Bytes ($t = 6$ ms).



- Em $t = 0$ ms, $FN_{1,1} = 0 + (62\times8)/3 = 165.3$. O pacote 1 da fila 1 é transmitido em $(62\times8)/(64\text{Kb/s}) = 7.75$ ms. Assim, o pacote 1 da fila 1 termina a sua transmissão em $t = 0 + 7.75 = 7.75$ ms.
- Em $t = 4$ ms : $FN_{2,1} = 165.3 + (32\times8)/1 = 421.3$
- Em $t = 6$ ms : $FN_{1,2} = \max(165.3, 165.3) + (18\times8)/3 = 213.3$
- Em $t = 7.75$ ms, como $FN_{1,2} < FN_{2,1}$, o pacote 2 da fila 1 começa a ser transmitido. O pacote 2 da fila 1 é transmitido em $(18\times8)/(64\text{Kb/s}) = 2.25$ ms. Assim, o pacote 2 da fila 1 termina a sua transmissão em $t = 7.75 + 2.25 = 10$ ms.
- Em $t = 10$ ms, o pacote 1 da fila 2 começa a ser transmitido. O pacote 1 da fila 2 é transmitido em $(32\times8)/(64\text{Kb/s}) = 4$ ms. Assim, o pacote 1 da fila 2 termina a sua transmissão em $t = 10 + 4 = 14$ ms.

Desempenho do GPS com controlo de taxa de transmissão por *leaky bucket*

O *Leaky Bucket* é um mecanismo de controlo de taxas de transmissão que permite impor um majorante ao tráfego gerado por um dado fluxo.



Se $A_i(\tau,t)$ representar a quantidade de tráfego (em Bytes) do fluxo i que é submetido à rede no intervalo de tempo [τ, t], então:

$$A_i(\tau,t) \leq \sigma_i + \rho_i(t-\tau)$$

Desempenho do GPS com controlo de taxa de transmissão por *leaky bucket*

Numa disciplina GPS, se designarmos por $S_i(\tau,t)$ o tráfego (em Bytes) de um fluxo i que é servido num intervalo de tempo [τ, t), então:

$$S_i(\tau,t) \geq r_i(t-\tau) \qquad \text{em que} \qquad r_i = \frac{\phi_i}{\sum_j \phi_j} C$$

A quantidade máxima de tráfego em espera $Q_{i,\max}(t)$ do fluxo i, desde um instante em que o fluxo não tinha tráfego no sistema ($\tau = 0$) até um qualquer instante t é:

$$\begin{aligned} Q_{i,\max}(t) &= A_{i,\max}(0,t) - S_{i,\min}(0,t) \\ &= \sigma_i + \rho_i t - r_i t \\ &\leq \sigma_i \qquad\qquad \Leftarrow \; r_i \geq \rho_i \end{aligned}$$

O atraso máximo D_i é o tempo necessário para transmitir todo o tráfego em espera, que na pior das hipóteses é servido à taxa mínima de serviço r_i. Assim, se $r_i \geq \rho_i$, o atraso máximo de qualquer pacote do fluxo i é:

$$D_i = \frac{\sigma_i}{r_i}$$

Desempenho do WFQ com controlo de taxa de transmissão por *leaky bucket*

Numa disciplina WFQ, o atraso máximo é maior que no GPS porque a informação é transmitida em pacotes.

Considere um fluxo *i* formatado por um *leaky bucket* com parâmetros σ_i e ρ_i que atravessa *n* ligações:

C_j - capacidade da ligação *j*

r_i - largura de banda reservada para o fluxo *i* em todas as ligações ($r_i \geq \rho_i$)

L_i - tamanho máximo dos pacotes do fluxo *i*

$L_{\max}$ - tamanho máximo dos pacotes de todos os fluxos

Prova-se que o atraso máximo (D_i) que os pacotes do fluxo *i* sofrem é:

$$D_i = \frac{\sigma_i + (n-1)L_i}{r_i} + \sum_{j=1}^{n} \frac{L_{\max}}{C_j} + \Gamma$$

em que Γ é o atraso total de propagação de todas as ligações.

Exemplo 5

Considere um fluxo de pacotes de comprimento máximo de 200 Bytes formatado por um *leaky bucket* com parâmetros σ = 1000 bytes e ρ = 150 Kbps. O fluxo atravessa 8 ligações todas com capacidade 100 Mbps servidas por uma disciplina WFQ. O comprimento máximo dos pacotes de todos os fluxos é de 1500 bytes. O atraso de propagação total é 2 mseg. Qual a taxa (em Mbps) que é necessário reservar em todas as ligações para este fluxo, por forma a garantir um atraso máximo extremo-a-extremo de 20 mseg?

$$D_i = \frac{\sigma_i + (n-1)L_i}{r_i} + \sum_{j=1}^{n} \frac{L_{\max}}{C_j} + \Gamma$$

$$0.02 = \frac{1000 \times 8 + 7 \times 200 \times 8}{r} + 8 \times \frac{1500 \times 8}{100 \times 10^6} + 0.002$$

$$r = \frac{1000 \times 8 + 7 \times 200 \times 8}{0.018 - 8 \times \frac{1500 \times 8}{100 \times 10^6}} = 1127 \text{ Kbps} = 1.127 \text{ Mbps}$$

Mecanismos de Escalonamento e de Descarte de Pacotes em Redes com Comutação de Pacotes

Quarta parte:

- **Métodos de descarte de pacotes**
- **Ilustração da combinação de disciplinas de escalonamento com métodos de descarte de pacote na arquitectura DiffServ do IETF**

Métodos de Descarte de Pacotes

Os métodos de descarte de pacotes podem ser classificados quanto a:

- Posição de descarte
- Prioridade de descarte
- Grau de agregação
- Descarte antecipado

Métodos de Descarte de Pacotes

Posição de descarte

- Cauda da fila – Normalmente usado por omissão; mais simples de implementar (o pacote não chega a entrar na fila).
  - Em muitos casos, a fila tem muitos pacotes pertencentes a poucos fluxos. Se o pacote que chega não pertence a nenhum desses fluxos, a estratégia não é justa.
- Posição aleatória – Escolhe-se aleatoriamente um pacote (entre todos os da fila + o novo) para ser eliminado (computacionalmente pesado).
  - Os fluxos com mais pacotes na fila são mais penalizados: estratégia mais justa.
- Cabeça da fila – Retira-se o pacote mais antigo da fila e aceita-se o que chegou (computacionalmente leve).
  - Tão bom como a posição aleatória em termos de justiça.
  - Útil quando o controle de fluxo é baseado em perdas de pacotes (porquê? relembrar controlo de congestão do TCP!)

Métodos de Descarte de Pacotes

Prioridades de descarte

- O emissor ou a rede (o policiador de um domínio DiffServ) podem marcar alguns pacotes com maior prioridade de descarte. Estes, em situação de congestionamento serão os primeiros a ser descartados.
- Quando um pacote é fragmentado e um dos fragmentos é descartado, os restantes fragmentos podem (e devem) também ser descartados pois deixam de ter qualquer utilidade.
    - Podia ser usado no protocolo IP? Relembrar utilização da flag '*more fragments*' e do campo *Fragment Offset*.
- Um método de descarte possível consiste em dar maior prioridade de descarte aos pacotes que passaram por menos ligações (*i.e.*, usaram menos recursos).
    - Este método não pode ser implementado no protocolo IP (porquê? relembrar utilização do campo TTL no IPv4)

Métodos de Descarte de Pacotes

Grau de agregação

Agregação de fluxos

- O método de descarte pode considerar os fluxos individualmente ou de forma agregada.
    - Na forma agregada, o método é aplicado a cada pacote do agregado, sem tomar em consideração o fluxo a que pertence.
    - Quanto mais fluxos forem agregados, menor a proteção entre os fluxos pertencentes ao mesmo agregado.

Agregação da memória dedicada às filas de espera

- Se existe uma fila de espera por fluxo de pacotes e a memória é partilhada por todas as filas, consegue-se uma atribuição de memória *max-min fair* quando se descarta o último pacote da fila mais longa (*i.e.*, da fila com um maior número de pacotes).
    - Com o WFQ, isto corresponde a descartar o pacote com maior *Finish Number* de entre todos os fluxos.

Métodos de Descarte de Pacotes

Descarte antecipado

Descarte quando a fila de espera está cheia:

- Quando a fila enche por um largo período (a ligação está muito com-gestionada), múltiplos pacotes são descartados provocando a reação simultânea de todas as ligações TCP afetadas; o tráfego tende a variar ciclicamente entre períodos de baixo débito e períodos de congestão.

Descarte antecipado (RED - *Random Early Discard*):

- Quando cada pacote chega à fila, ele é descartado com uma probabilidade proporcional à ocupação da fila; evita-se o sincronismo do controle de congestão das ligações TCP.
- Não proporciona diferenciação de qualidade de serviço.

Descarte antecipado pesado (WRED – *Weighted* RED):

- Atribuem-se diferentes probabilidades de descarte a pacotes pertencentes a diferentes fluxos (ou agregados de fluxos).
- Quanto menor a probabilidade de descarte, menor é a taxa de perda de pacotes que o fluxo (ou o agregado) sofre.

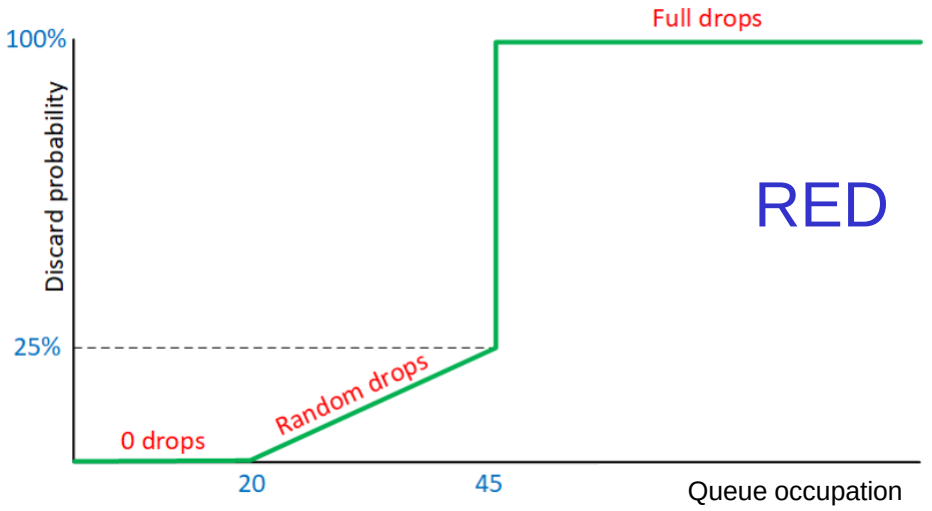
RED e WRED

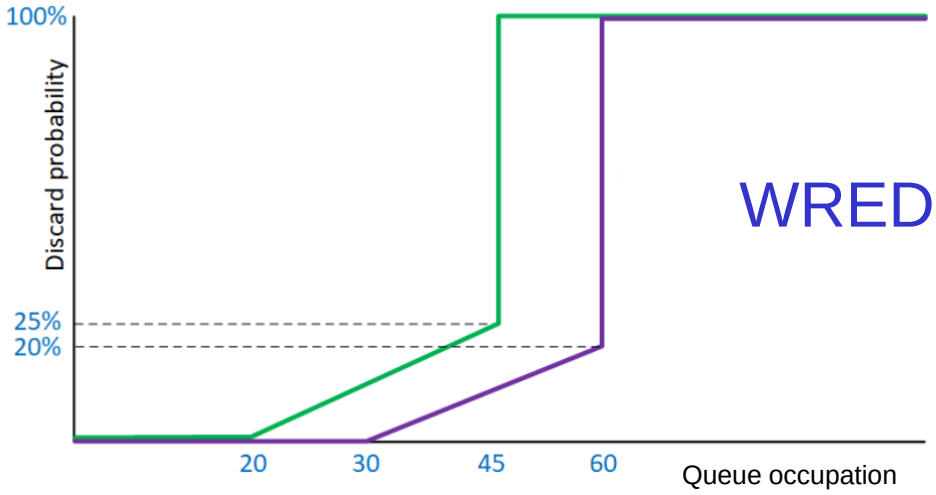
No RED:

- Limite Mínimo (*m*): quando um pacote chega e a ocupação da fila *f* é menor que o limite mínimo (*f* < *m*), o pacote é sempre aceite na fila.
- Limite Máximo (*M*): quando um pacote chega e a ocupação da fila *f* é maior que o limite máximo (*f* > *M*), o pacote é sempre descartado.
- *Mark Probability Denominator* (*MPD*): quando um pacote chega e a ocupação *f* está entre os limites mínimo e máximo ($m \leq f \leq M$), o pacote é descartado com probabilidade $(f–m)/(M–m)\times MPD$

No WRED:

- São atribuídos diferentes valores de *m*, *M* e *MPD* a diferentes fluxos (ou agregados de fluxos)





Figuras adaptadas de: https://networklessons.com/cisco/ccie-routing-switching-written/wred-weighted-random-early-detection

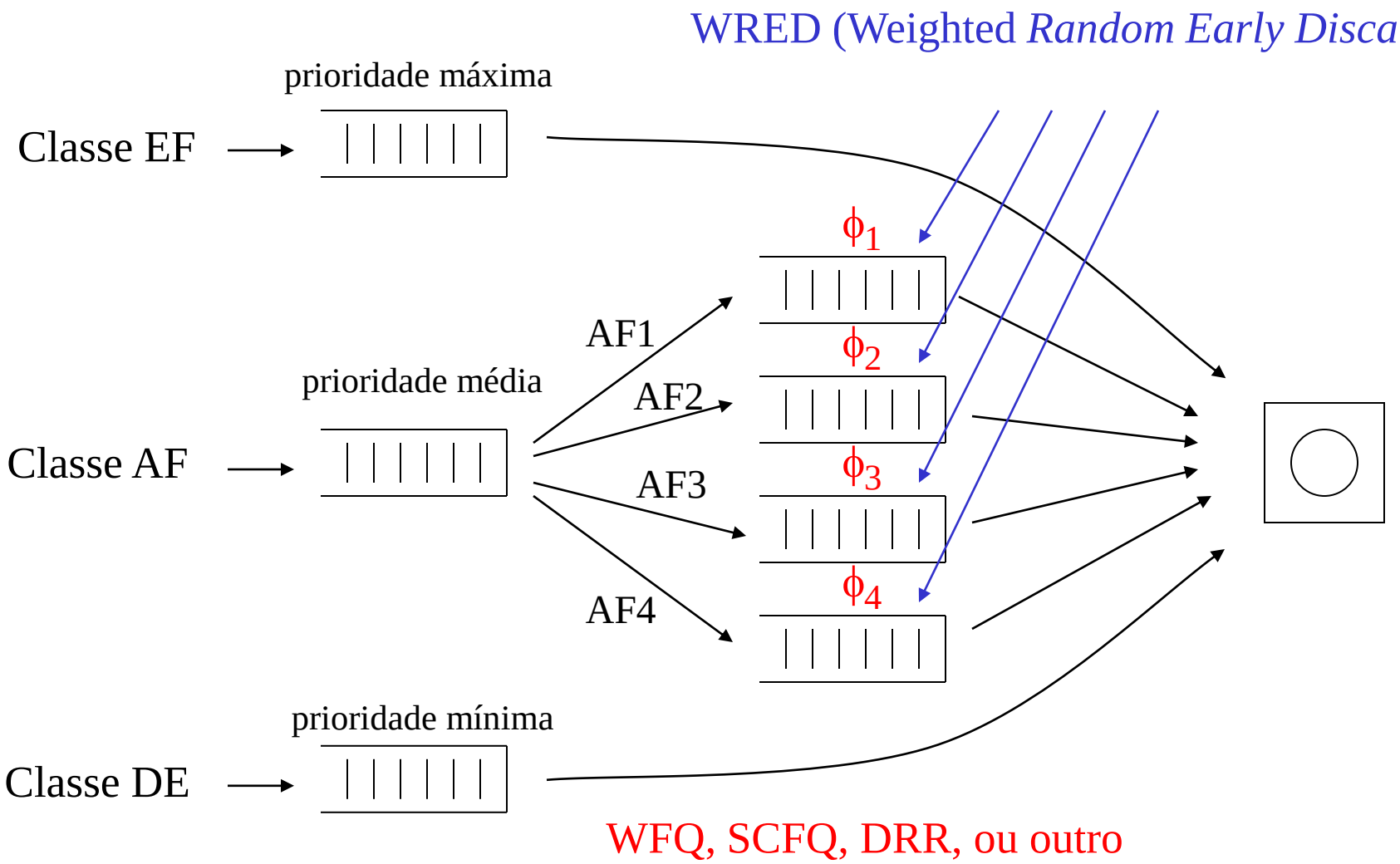
Exemplo – Arquitectura *DiffServ*

Classes de Serviço

- *Default* (DE) $\rightarrow$ DSCP = 000000
  - serviço *best-effort* com uma única fila de espera do tipo FIFO
- *Expedited Forwarding* (EF) $\rightarrow$ DSCP = 101110
  - serviço tipo “linha alugada virtual”
  - disponibiliza controle de perdas, do atraso e da variância do atraso dentro de uma determinada largura de banda máxima
- *Assured Forwarding* (AF)
  - fornece uma Qualidade de Serviço relativa entre até 4 classes AF
  - em cada classe AF, pode haver até 3 níveis de precedência para descarte de pacotes (em caso de congestionamento)

AF Codepoints	**AF1**	**AF2**	**AF3**	**AF4**
Low drop precedence	**001010**	**010010**	**011010**	**100010**
Medium drop precedence	**001100**	**010100**	**011100**	**100100**
High drop precedence	**001110**	**010110**	**011110**	**100110**

Possível Esquema de Escalonamento do DiffServ



Energy efficient networking

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2023/2024

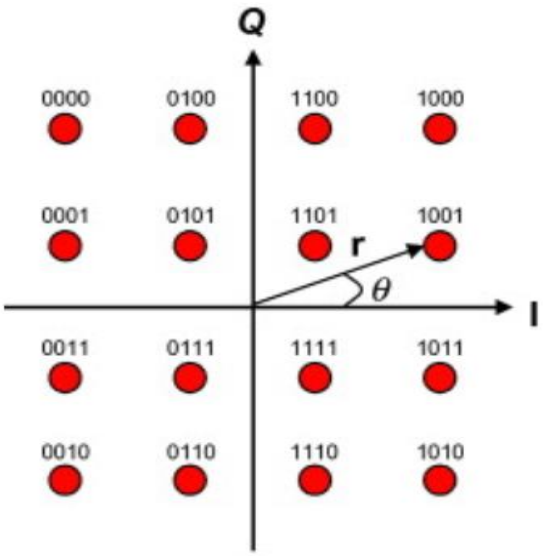
Energy consumption of network links

- Consider a network modelled by a graph $G=(N,A)$
  - Set N is the set of network nodes.
  - Set A is the set of network links: the arc $(i,j) \in A$ represents the link between nodes $i \in N$ and $j \in N$ from i to j whose capacity is given by c_{ij} in bps (with $c_{ij} = c_{ji}$).
- In wide area networks, network links are optical transmission systems. Energy consumption of network links is related with two factors:
  - *Transmission factor*: the energy required in the NICs (Network Interface Cards) for the conversion of the digital information from the electronic domain to the optical domain (in the transmitter side) and from the optical domain to the electronic domain (in the receiver side);
  - *Progapagion factor*: the energy required in the transmission lasers to send the optical signal with enough strength (i.e., enough optical power) so that the signal-to-noise ratio at the receiver is enough to recover the transmitted information.

Modulation formats in optical transmission systems

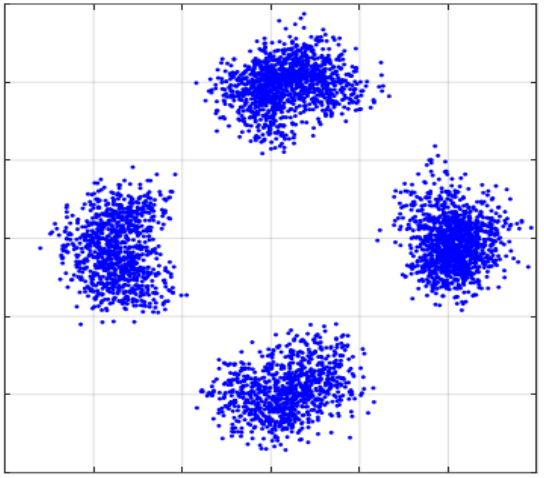
16-QAM at the transmitter side

- Each optical symbol represents a different combination of 4 bits
- There is a total of 16 possible combinations
- The resulting symbol rate (also known as baud rate) is ¼ of the bit rate



QPSK at the receiver side

- Each optical symbol represents a different combination of 2 bits
- Due to the impairments of the light propagation in fibers, the symbols are received distorted
- A higher transmitted optical power enlarges the amplitude distances between symbols



Operation modes of optical links

Consider that the network links can be put in a sleeping mode if they do not support traffic in none of its two directions.

When a link is in full operational mode, i.e., the link is supporting one or more traffic flows, the energy consumption of the link is:

- *Transmission factor*: the energy required on each of the two end NICs is a value which depends on the link capacity.
  - The modulation format order (defining the number of bits coded by each transmission symbol) and the baud rate (defining the number of transmitted symbols per second) determines the capacity of the link.
  - The higher both parameters are, the more energy they require for the electrical-optical and optical-electrical conversion.
- *Propagation factor*: the energy required in the transmission laser of each NIC is a value which depends on the modulation format order, baud rate and length of the link.
  - The longer the link is, the higher the optical power of the laser needs to be so that the signal-to-noise ratio at the receiver is enough to recover the transmitted information.

Operation modes of optical links

Consider that the network links can be put in a sleeping mode if they do not support traffic in none of its two directions.

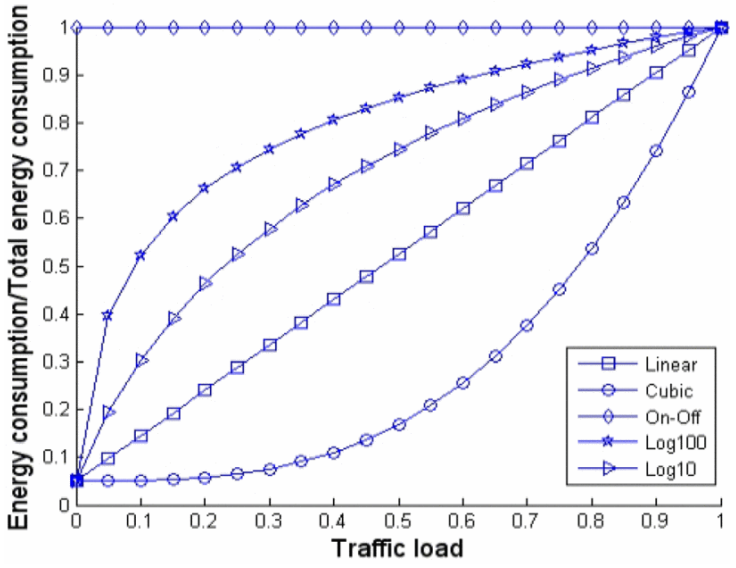
When a link is in sleeping mode, i.e., the link is not supporting any traffic flow, the energy consumption of the link is:

- *Transmission factor*: the energy required in each of the end NICs is a value to maintain the interface cards in a state that allows the card to be quickly activated.
- *Propagation factor*: the energy required in the transmission laser of each NIC is a value to maintain the laser in a state that allows the laser to be quickly activated.

Energy consumption of network nodes

As illustrated in the picture below[1], depending on the equipment (switching architecture, energy reduction techniques, cooling system, etc), the energy consumption of the nodes might depend on its total traffic load (sum of all flow demands crossing it).

- The On-Off model is when the router is always consuming its maximum energy (independently of its load).
- Other energy consumption models depend on each particular router equipment.



[1] J. C. Cardona Restrepo, C. G. Gruber and C. Mas Machuca, "Energy Profile Aware Routing," *IEEE Int. Conf. on Communications Workshops*, pp. 1-5, 2009

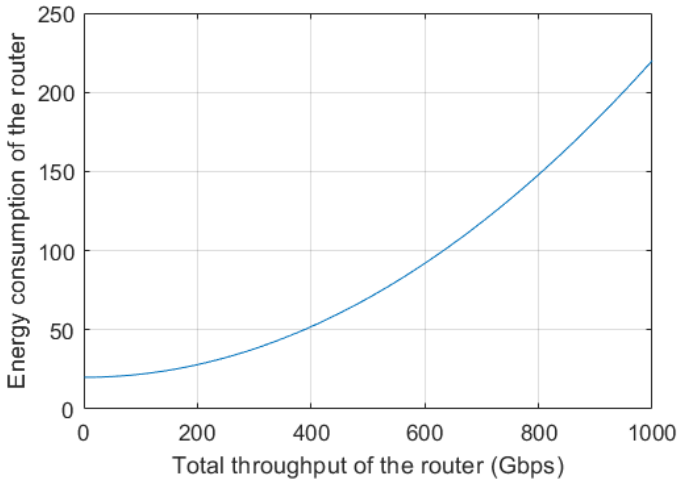
Energy consumption of network nodes - example

Consider a router equipment with a throughput capacity of 1 Tbps (= 1000 Gbps). The energy consumption of the router equipment is $E = 20 + 200 \times t^2$, where t is the total throughput supported by the router divided by its capacity.

1. Determine the energy consumption of the router when it is supporting a total throughput of 400 Gbps.

$$E = 20 + 200 \times t^2 = 20 + 200 \times \left(\frac{400}{1000}\right)^2 = 52$$

2. Draw a plot of the energy consumption of the router as a function of its supported throughput.



Single layer energy efficient routing

Consider that the energy consumption of each link is known when it is in full operational mode and when it is in sleeping mode.

The aim is to route the flows through routing paths so that:

- the set of non-used links (and, therefore, that can be put in sleeping mode) minimizes the energy consumption of all links;
- the set of full operational links has enough capacity to support the demand of all flows.

If the energy consumption of nodes is not the On-Off model, the energy consumption must also consider the total traffic load of each node and the aim is to minimize the network energy consumption (links + nodes).

Why links in sleeping mode are not removed from the network?

- Traffic flows of supported services can change over time.
- New services (with new traffic flows) can be introduced
- Failures in full operational links can require the activation of links in sleeping mode to maintain the services

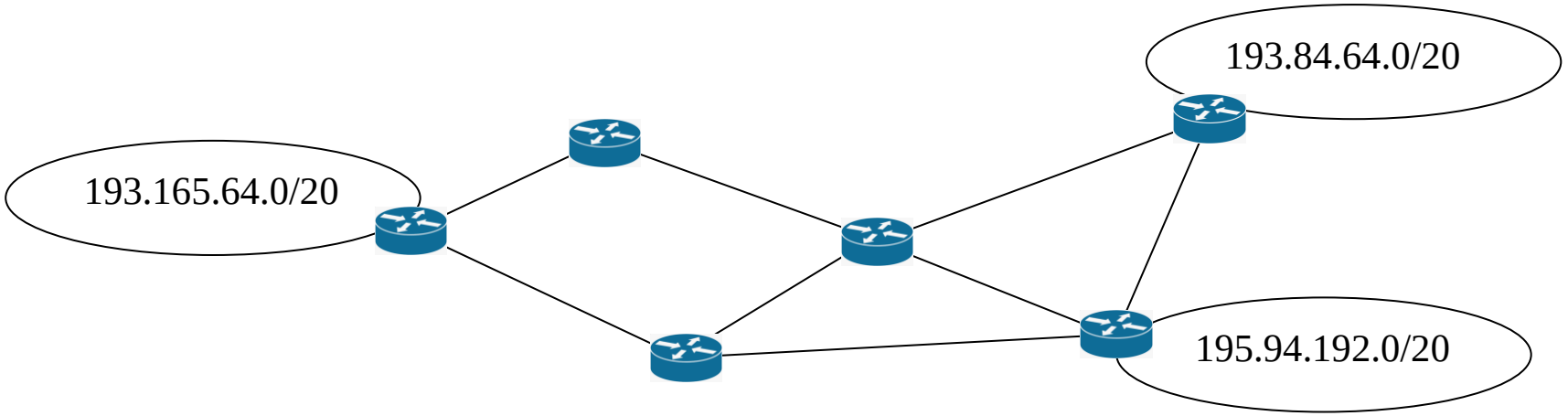
Single layer energy efficient routing

Consider the network below supporting one unicast service with the following traffic flows:

193.165.60.0/20 ↔ 193.84.64.0/20: 100 Gbps

193.165.60.0/20 ↔ 195.94.192.0/20: 100 Gbps

193.84.64.0/20 ↔ 195.94.192.0/20: 200 Gbps



Router

Single layer energy efficient routing

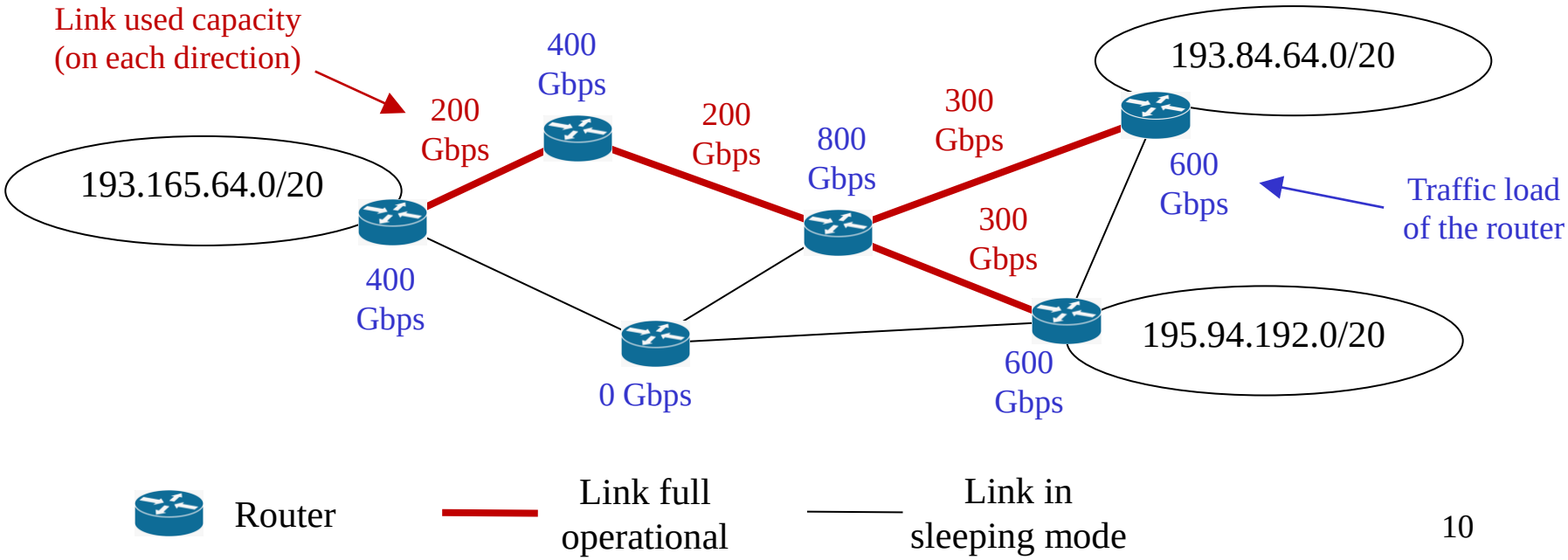
Consider the network below supporting one unicast service with the following traffic flows:

193.165.60.0/20 ↔ 193.84.64.0/20: 100 Gbps

193.165.60.0/20 ↔ 195.94.192.0/20: 100 Gbps

193.84.64.0/20 ↔ 195.94.192.0/20: 200 Gbps

A possible solution:



Two-layer networks

Recently, the wide area networks have evolved to have two layers:

- A top layer network composed by routers operating in the electronic domain.
- A bottom layer network composed by optical nodes (named OXCs – Optical Cross Connects) operating in the optical domain, and fibre links between them.

Switching optical signals (known as lightpaths) from input ports to output ports in OXCs require much less energy than routing traffic flows from input NICs to output NICs in routers.

The lightpaths' capacity depends on the length of the routing paths[2]:

Modulation Format	BPSK	QPSK	8-QAM	16-QAM
Transmission reach (km)	6300	3500	1200	600
Bitrate capacity (Gbps)	50	100	150	200

[2] F. Barbosa, A. de Sousa, A. Agra, K. Walkowiak, R. Goścień, RMSA algorithms resilient to multiple node failures in dynamic EONs, *Optical Switching and Networking*, Volume 42, 2021

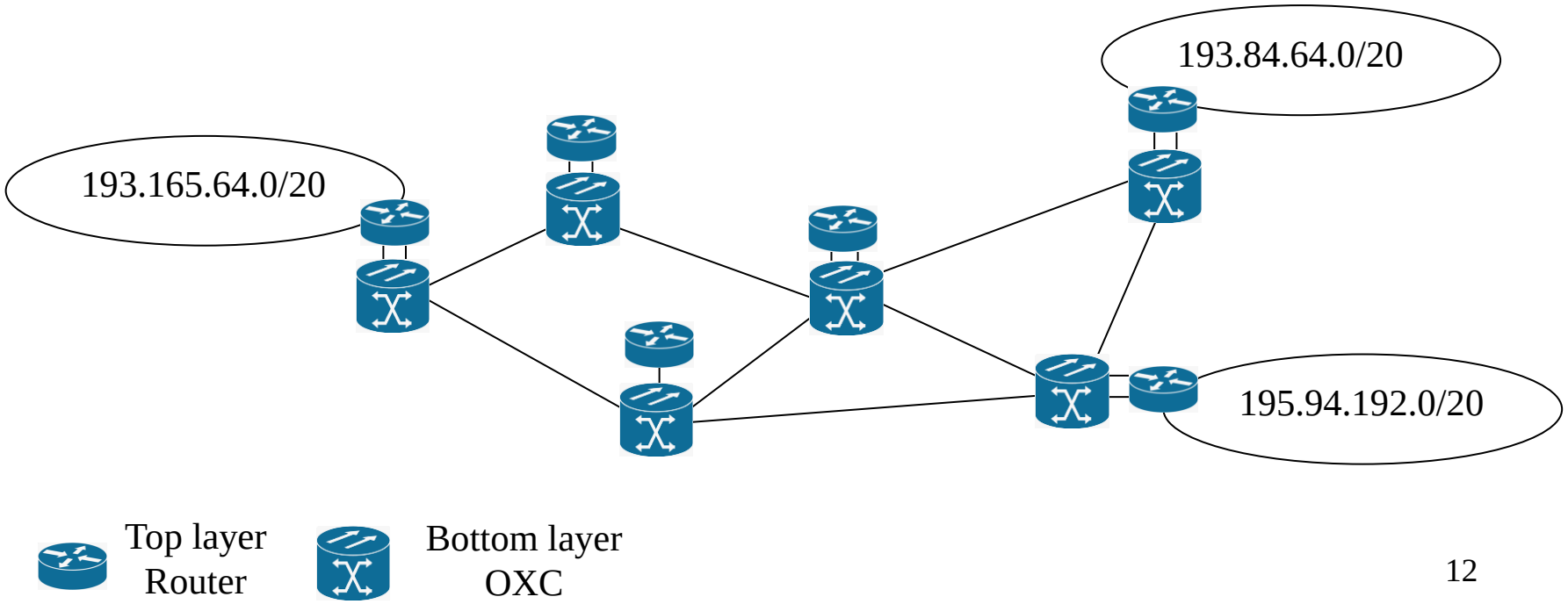
Two-layer energy efficient routing

Again, consider the network below supporting one unicast service with the following traffic flows:

193.165.60.0/20 ↔ 193.84.64.0/20: 100 Gbps

193.165.60.0/20 ↔ 195.94.192.0/20: 100 Gbps

193.84.64.0/20 ↔ 195.94.192.0/20: 200 Gbps



Two-layer energy efficient routing

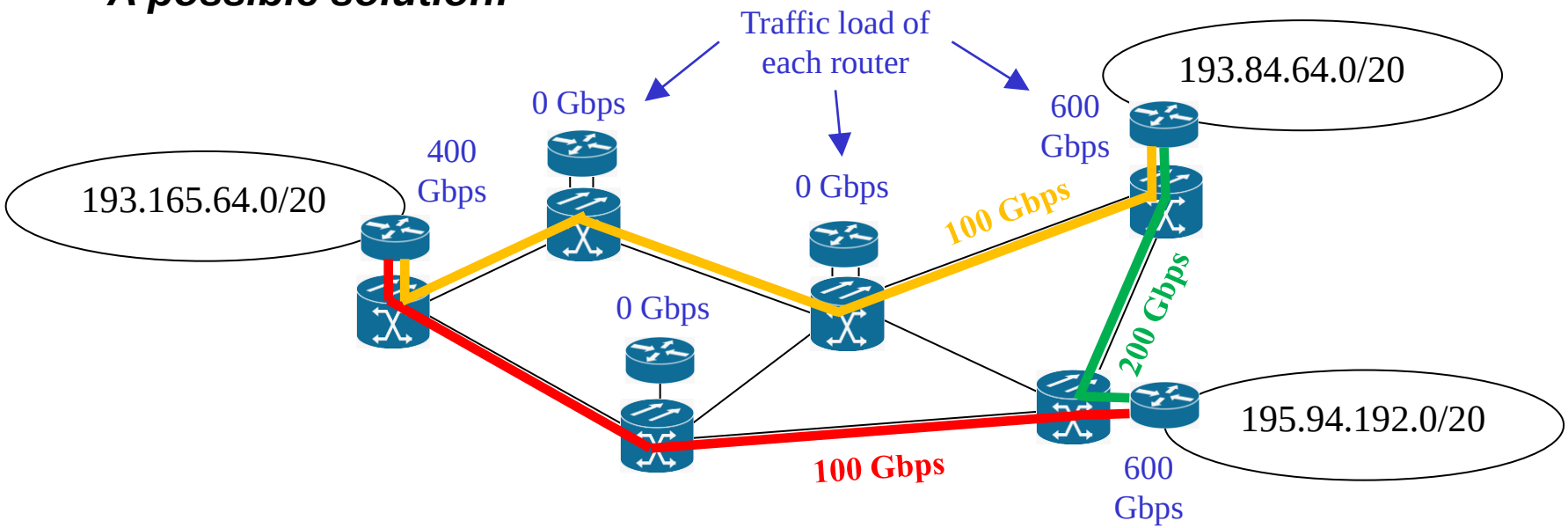
Again, consider the network below supporting one unicast service with the following traffic flows:

193.165.60.0/20 ↔ 193.84.64.0/20: 100 Gbps

193.165.60.0/20 ↔ 195.94.192.0/20: 100 Gbps

193.84.64.0/20 ↔ 195.94.192.0/20: 200 Gbps

A possible solution:



Engenharia de Tráfego de Serviços *Anycast*

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2023/2024

Serviços *Unicast* vs. *Anycast*

Serviço *Unicast*:

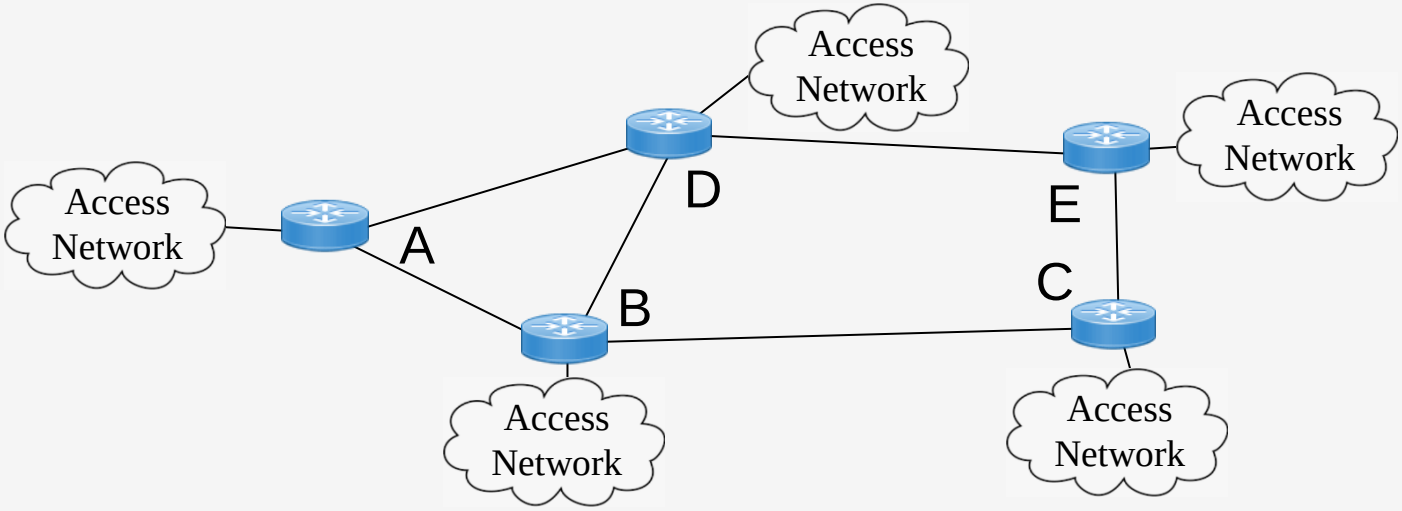
- O tráfego de um serviço *unicast* é definido por um conjunto de fluxos: um (ou mais) de cada nó origem para cada nó destino.

Serviço *Anycast*:

- Exemplos: serviços de *streaming* de filmes e/ou de música (tais como o Netflix, Youtube, Amazon Prime Video, Spotify)
- Num serviço *anycast*, existe um conjunto de nós (designados por nós *anycast*) associados ao serviço, i.e., os nós da rede ao qual estão ligados os servidores do serviço, tipicamente hospedados em *Data Centers* (DCs).
- O tráfego de um serviço anycast é definido por um conjunto de fluxos: um (ou mais) de cada nó origem.
- O destino de cada fluxo de tráfego pode ser qualquer um dos nós *anycast*, i.e., depende do encaminhamento dos fluxos na rede que liga os diferentes nós.

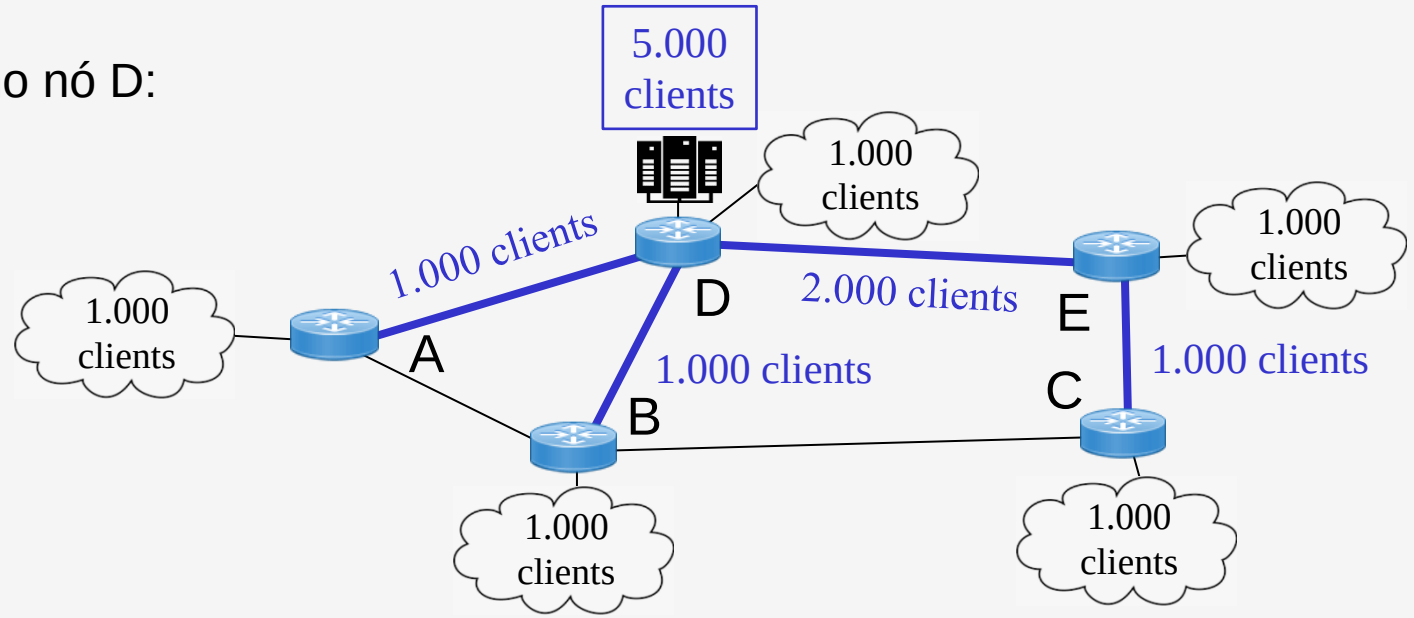
Serviços *anycast*

- Quando um cliente se liga a um serviço *anycast*, por norma, a rede encaminha a comunicação para o nó *anycast* mais próximo (em número de ligações ou em termos de atraso).
- O número de nós *anycast* e a sua localização na rede influenciam:
  - os recursos necessários na rede para suportar o serviço;
  - o desempenho da rede tanto em termos de atraso como de disponibilidade do serviço.
- Considere-se a seguinte rede a suportar um serviço *anycast* com 1000 clientes ligados em cada rede de acesso:



Serviços *anycast*

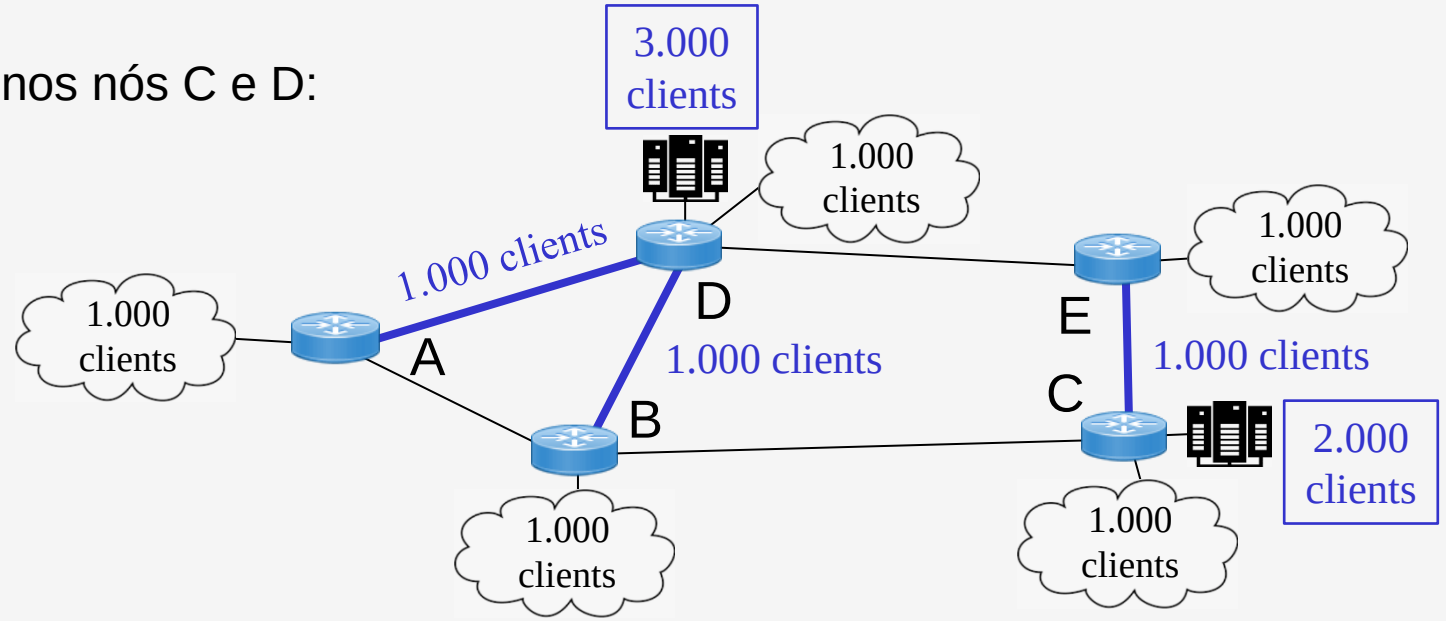
1 DC no nó D:



- 3 ligações têm de ter capacidade suficiente para suportar o tráfego de 1000 clientes e 1 ligação tem de ter capacidade suficiente para suportar o tráfego de 2000 clientes.
- O DC tem de ter a capacidade para suportar o tráfego de 5000 clientes.
- Se o DC falhar, o serviço *anycast* falha completamente.

Serviços *anycast*

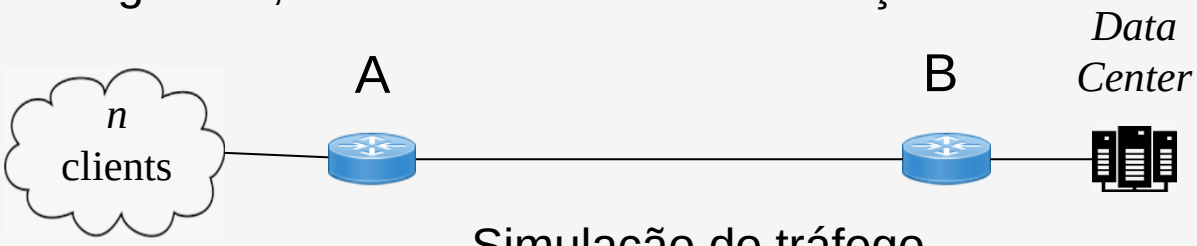
2 DCs nos nós C e D:



- 3 ligações têm de ter capacidade suficiente para suportar o tráfego de 1000 clientes (ocupa menos recursos de rede que no caso anterior).
- O DC no nó D tem de ter a capacidade para suportar o tráfego de 3000 clientes e o DC no nó C tem de ter a capacidade para suportar o tráfego de 2000 clientes.
- Se um dos DCs falhar, os fluxos são reencaminhados para o outro DC: o serviço *anycast* sofre uma degradação mas não falha completamente.

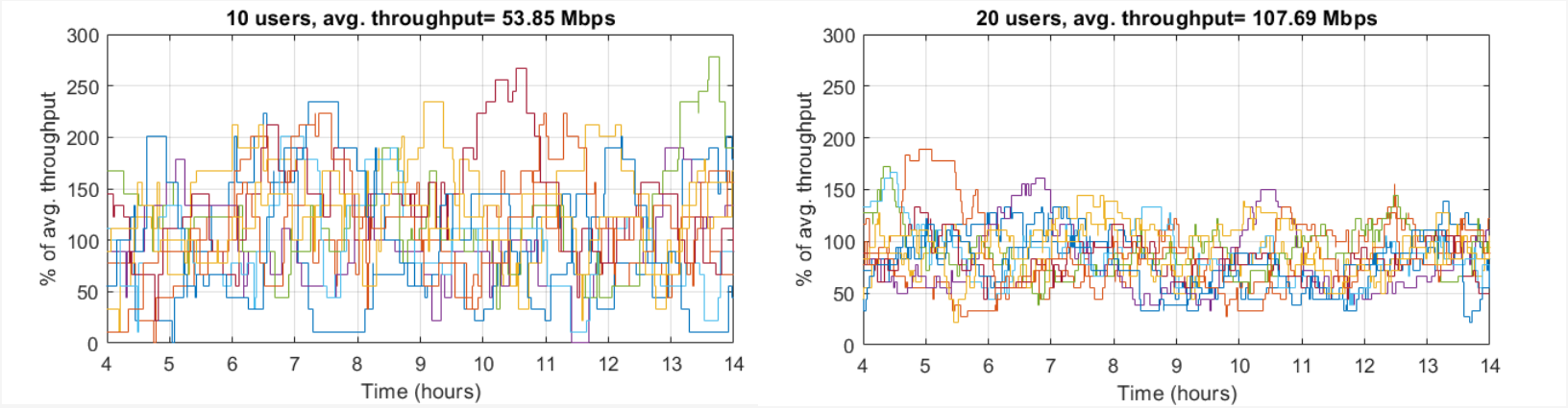
Agregação de tráfego

- Considere-se a simulação do tráfego *downstream* de *n* clientes numa rede de acesso a aceder a um servidor de *streaming* de filmes (hospedado num DC remoto) caracterizada por:
  - dependendo do terminal, o filme é transmitido num *stream* de 6, 12 ou 24 Mbps (todos os formatos igualmente prováveis)
  - duração do acesso de cada cliente ao serviço (em contínuo) entre 0.5 e 2 horas (segundo uma distribuição uniforme)
  - intervalo de tempo entre acessos ao serviço de cada cliente exponencialmente distribuído com média de 2 horas
  - em cada acesso, uma pausa entre 2 e 18 minutos (segundo uma distribuição uniforme) com probabilidade *p*.
- Nos slides seguintes, são visualizadas 10 simulações.



Simulação do tráfego agregado na ligação de B para A

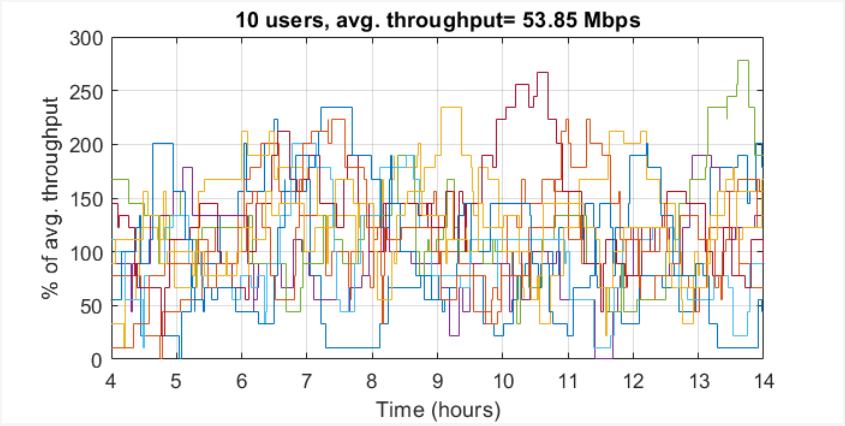
Agregação de tráfego sem pausas

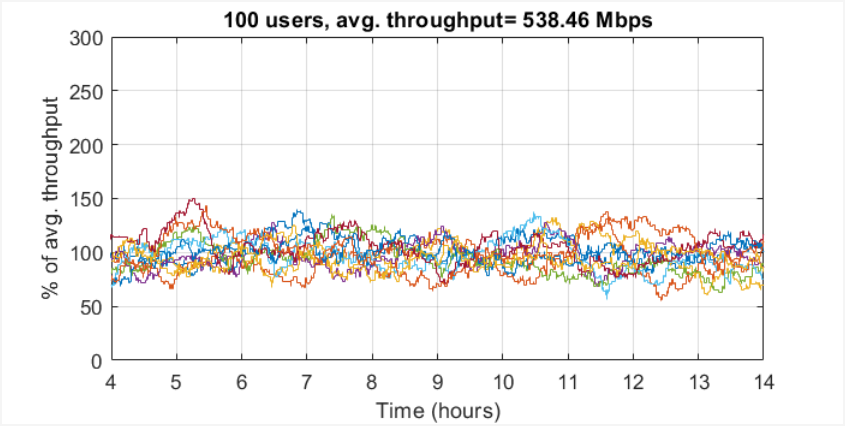


Para um número de clientes grande, o tráfego agregado varia pouco (em termos percentuais) em torno do tráfego médio.

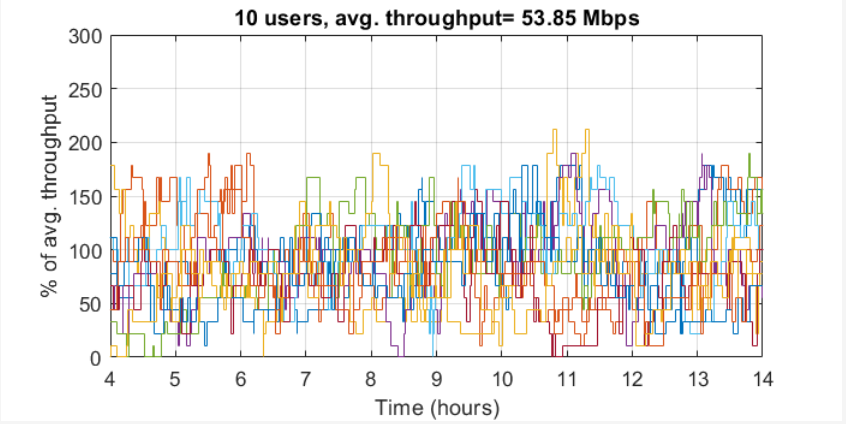
Agregação de tráfego

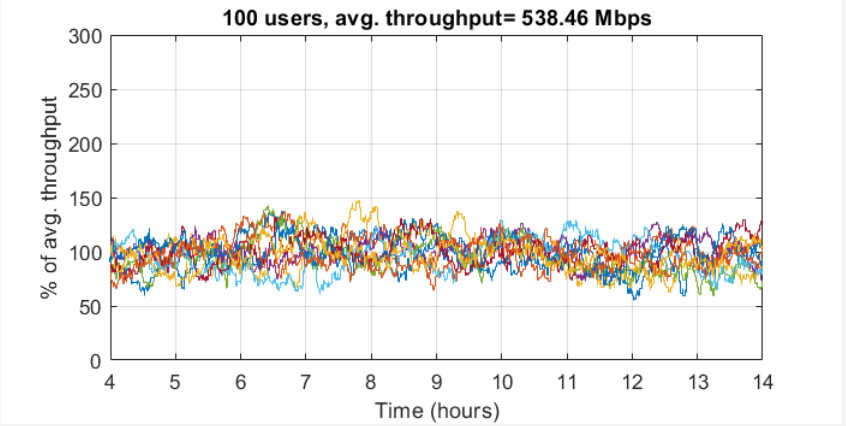
Sem Pausas (p = 0.0)





Com Pausas (p = 0.99)





As pausas reduzem pouco a variação do tráfego agregado em torno do tráfego médio e a redução é observável apenas para um número de clientes pequeno.

Tráfego médio de um agregado de fluxos

As simulações anteriores mostram que o tráfego suportado pela rede em cada fluxo é aproximadamente igual ao tráfego médio para serviços com muitos clientes.

Considere-se o tráfego *downstream* de *n* clientes (numa rede de acesso) a aceder a um servidor de *streaming* de filmes.

Os filmes têm uma duração média de 90 minutos e são disponibilizados em 3 formatos (por exemplo, HD, FHD ou 4K) cujo ritmo de transmissão é respetivamente de 6, 12 ou 24 Mbps.

Cada cliente vê em média 2 filmes por dia cujo formato é em HD com 20% de probabilidade, em FHD com 30% de probabilidade e em 4K com 50% de probabilidade.

1. Determine o ritmo médio *r* de transmissão *downstream* por cliente.
2. Determine o ritmo médio *R* de transmissão *downstream* para um agregado de *n* = 1000 clientes.

Tráfego médio de um agregado de fluxos

1. Determine o ritmo médio *r* de transmissão *downstream* por cliente.

   O ritmo médio de transmissão de um filme por cliente é:

   $0.2 \times 6 + 0.3 \times 12 + 0.5 \times 24 = 16.8$ Mbps

   A fração média de tempo em que o cliente está a ver filmes por dia é:

   $2 \times 1.5 / 24 = 0.125$ (=12.5%)

   Finalmente: $r = 16.8 \times 0.125 = 2.1$ Mbps

2. Determine o ritmo médio *R* de transmissão *downstream* para um agregado de $n = 1000$ clientes.

   $R = n \times r = 1000 \times 2.1 = 2100$ Mbps = 2.1 Gbps

Probabilidade de bloqueio de um servidor

- Considere-se um servidor com capacidade para atender *m* clientes. Qual a probabilidade de bloqueio do servidor, i.e., a probabilidade de um pedido ser recusado porque o servidor está a atender *m* clientes?
- Se os pedidos de serviço forem um processo de Poisson com taxa λ (pedidos por unidade de tempo) e o tempo de serviço de cada cliente for exponencialmente distribuído com média $1/\mu$ (unidades de tempo), o desempenho do servidor é caracterizado por um sistema *M*/*M*/*m*/*m*.
- Probabilidade do servidor estar a atender *n* clientes:

$$P_n = \frac{(\lambda/\mu)^n / n!}{\sum_{i=0}^{m} (\lambda/\mu)^i / i!} \qquad n = 0, 1, ..., m$$

- Probabilidade de bloqueio (fórmula de ErlangB): $P_m = \frac{(\lambda/\mu)^m / m!}{\sum_{i=0}^{m} (\lambda/\mu)^i / i!}$
- É possível demonstrar que as fórmulas são válidas para qualquer estatística do tempo de serviço desde que o tempo de serviço seja estatisticamente independente dos instantes de pedido de serviço.

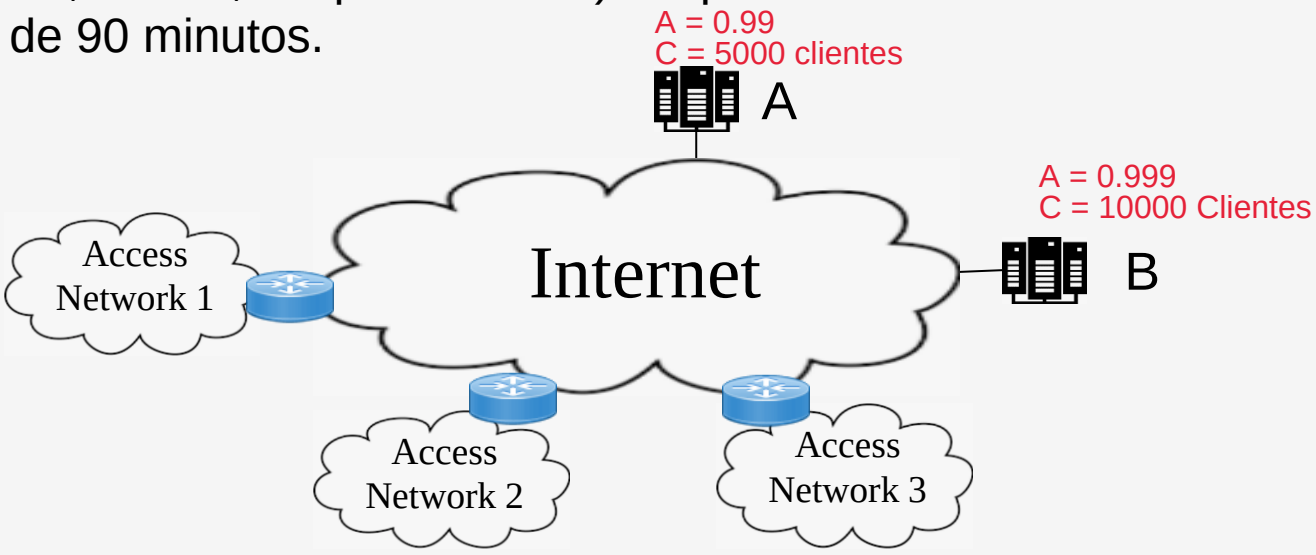
Probabilidade de bloqueio e disponibilidade de um serviço *anycast*

- Considere-se um serviço em que quando todos os servidores estão operacionais, um cliente pode ser servido por qualquer servidor.
- Assim, se o servidor mais próximo estiver totalmente ocupado, o pedido de serviço é encaminhado para o servidor disponível mais próximo
  - Geralmente, existe um módulo do serviço para balanceamento de carga entre servidores.
- Então, em termos de probabilidade de bloqueio, é conceptualmente equivalente a considerar um único servidor cuja capacidade é a soma das capacidades de todos os servidores.
- Se cada servidor tiver um valor de disponibilidade associado, então a probabilidade de bloqueio é a soma pesada da probabilidade de bloqueio da capacidade disponível em que o peso é a probabilidade de cada caso de falha.
- Exemplo no próximo slide.

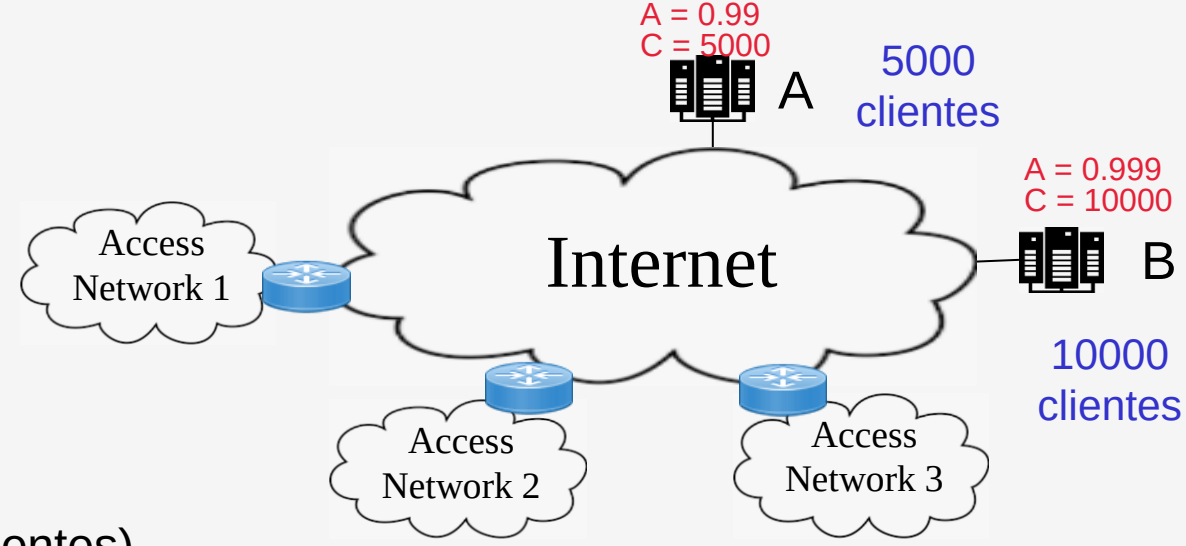
Exemplo

Considere-se um serviço de *streaming* de filmes com 2 servidores ilustrado na figura: um servidor hospedado no *Data Center* A com uma disponibilidade de 0.99 e uma capacidade de 5000 clientes e outro servidor hospedado no *Data Center* B com uma disponibilidade de 0.999 e uma capacidade de 10000 clientes.

1. Determine a capacidade média C do serviço (em número de clientes).
2. Determine a probabilidade de bloqueio P do serviço considerando que os clientes geram 5000, 3000 e 2000 pedidos por hora (nas redes de acesso 1, 2 e 3, respetivamente) e que os filmes têm uma duração média de 90 minutos.



Exemplo



1. Capacidade média C do serviço (em número de clientes).

Probabilidade dos 2 servidores estarem operacionais

Probabilidade do servidor A estar operacional e o servidor B estar inoperacional

Probabilidade do servidor A estar inoperacional e o servidor B estar operacional

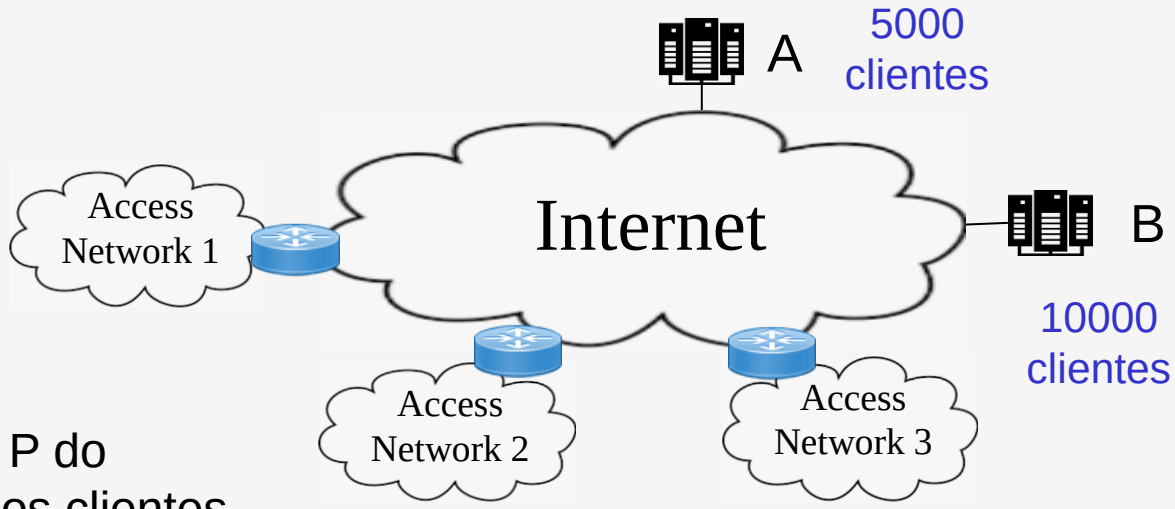
A e B — A e ~B — B e ~A

$$C = (5000+10000)\times 0.99\times 0.999 + 5000\times 0.99\times (1-0.999) + 10000\times (1-0.99)\times 0.999 =$$

$$= 15000\times 0.989 + 5000\times 0.00099 + 10000\times 0.00999 = 14939.85 \text{ clientes}$$

E(Y/U,15000)=(((1.5)^15000)/15000!)/
(SOMAT((1.5)^i) /i!)

Exemplo



2. Probabilidade de bloqueio P do serviço considerando que os clientes geram 5000, 3000 e 2000 pedidos por hora (nas redes de acesso 1, 2 e 3, respetivamente) e que os filmes têm uma duração média de 90 minutos.

λ = 5000+3000+2000 = 10000 pedidos/hora

$1/\mu$ = 1.5 horas

$\lambda/\mu = 10000 \times 1.5 = 15000$

$$E(\lambda/\mu, m) = \frac{(\lambda/\mu)^m / m!}{\sum_{i=0}^{m} (\lambda/\mu)^i / i!}$$

P = $E(\lambda/\mu,15000)\times0.989 + E(\lambda/\mu,5000)\times0.00999 + E(\lambda/\mu,10000)\times0.00099$ =

= $0.0065\times0.989 + 0.6667\times0.00099 + 0.3335\times0.00999 = 0.0104 = 1.04\%$

Probabilidade de bloqueio para capacidade de 15000 clientes

Probabilidade de bloqueio para capacidade de 5000 clientes

Probabilidade de bloqueio para capacidade de 10000 clientes