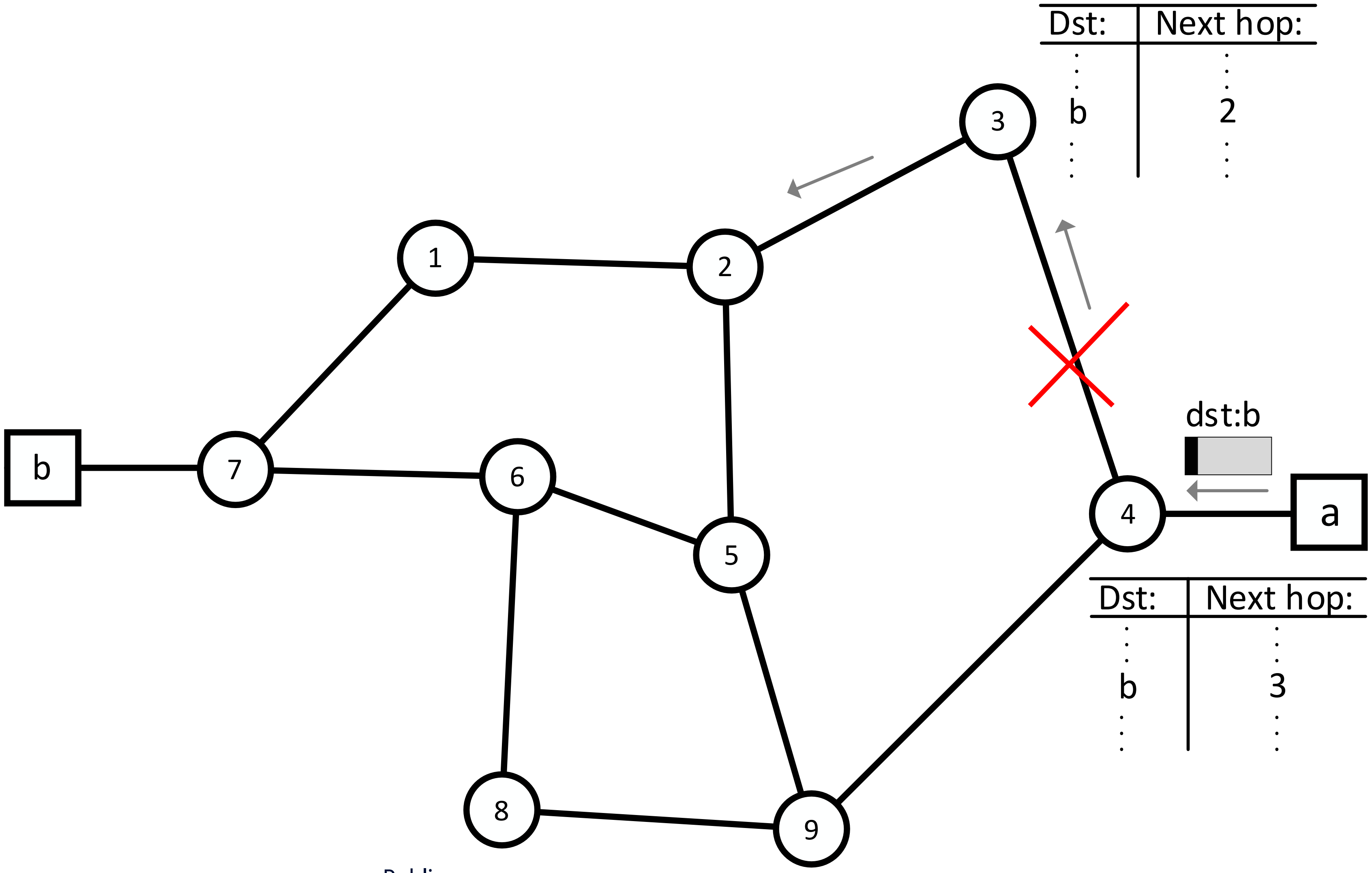
Path finding and dimensioning problems related to reliable telecommunication networks

Lajos Bajzik, Research Engineer

2022.10.18

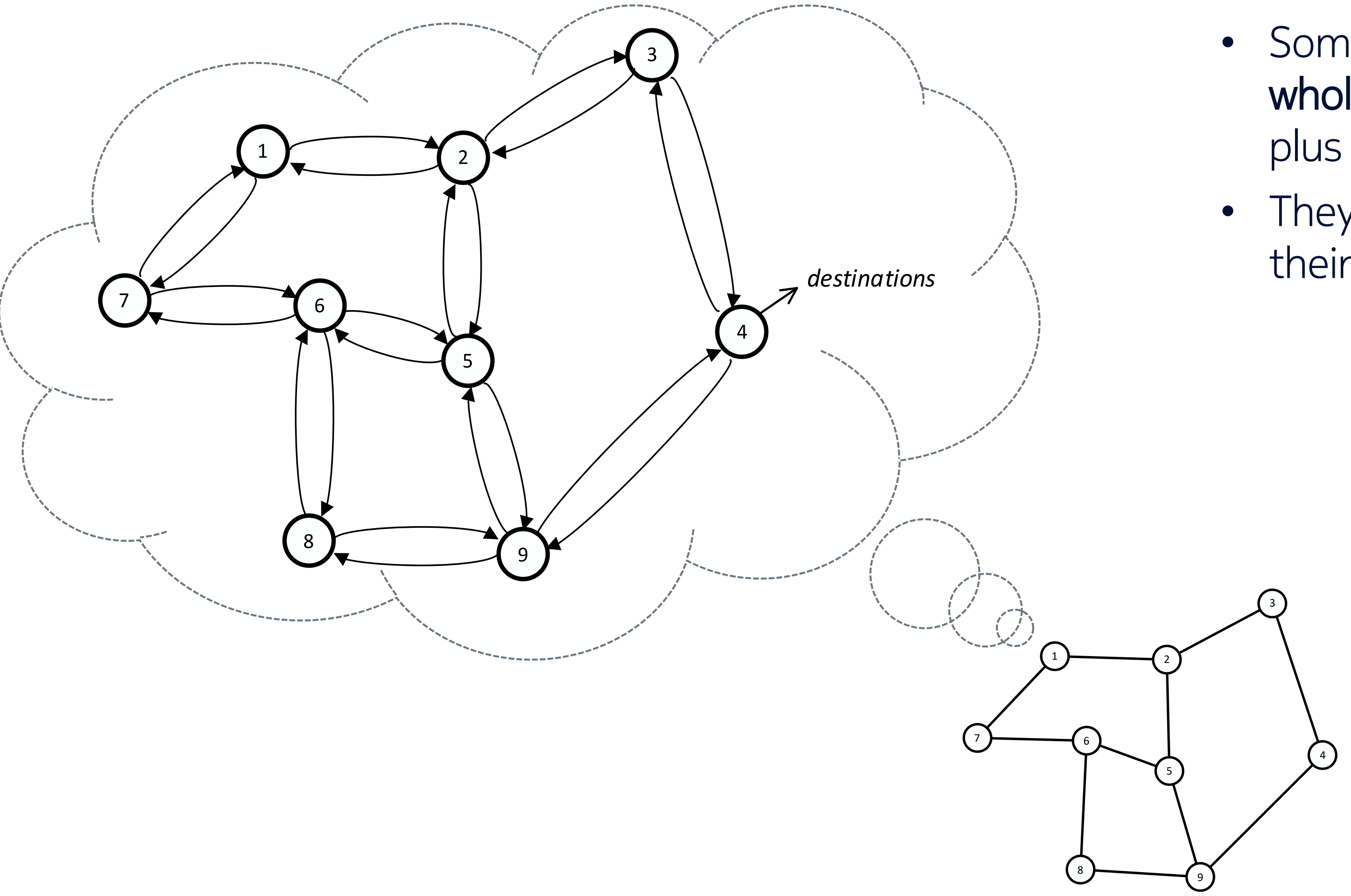
IP Network

A commonly known **transport** network technology which is **resilient** by design



NOKIA Bell Labs

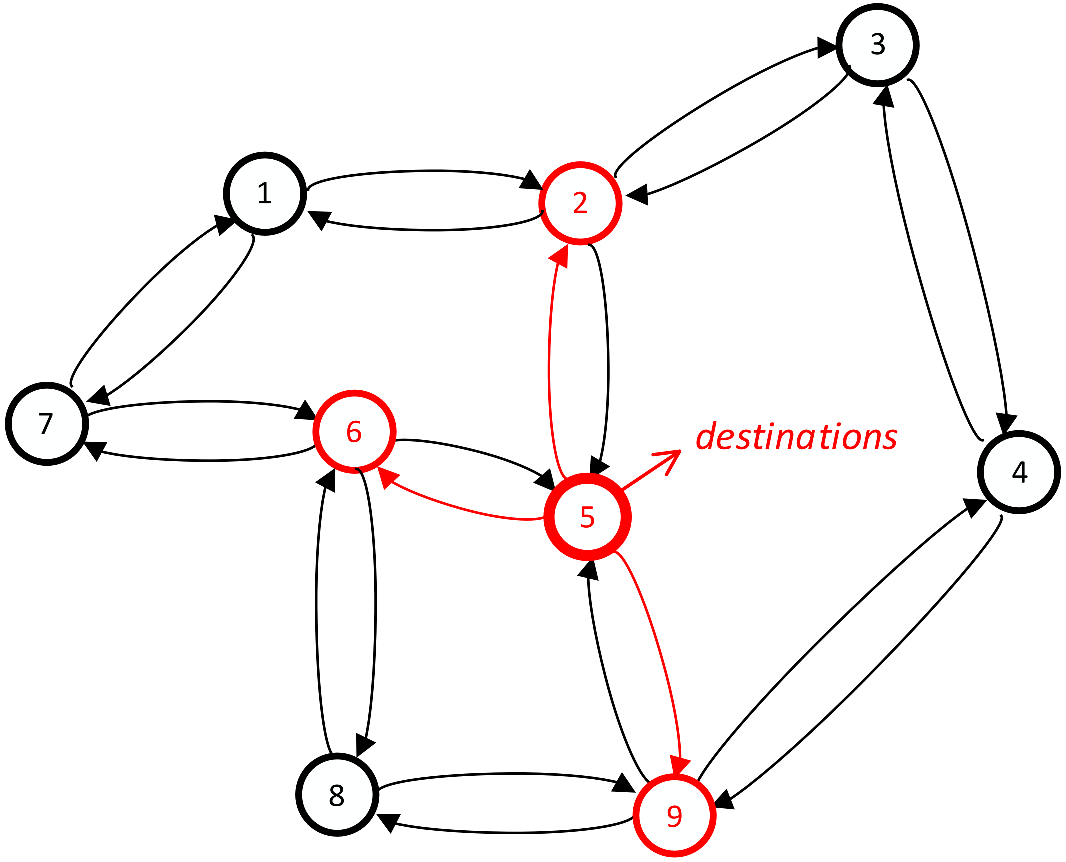
IP resiliency
Link state routing protocol



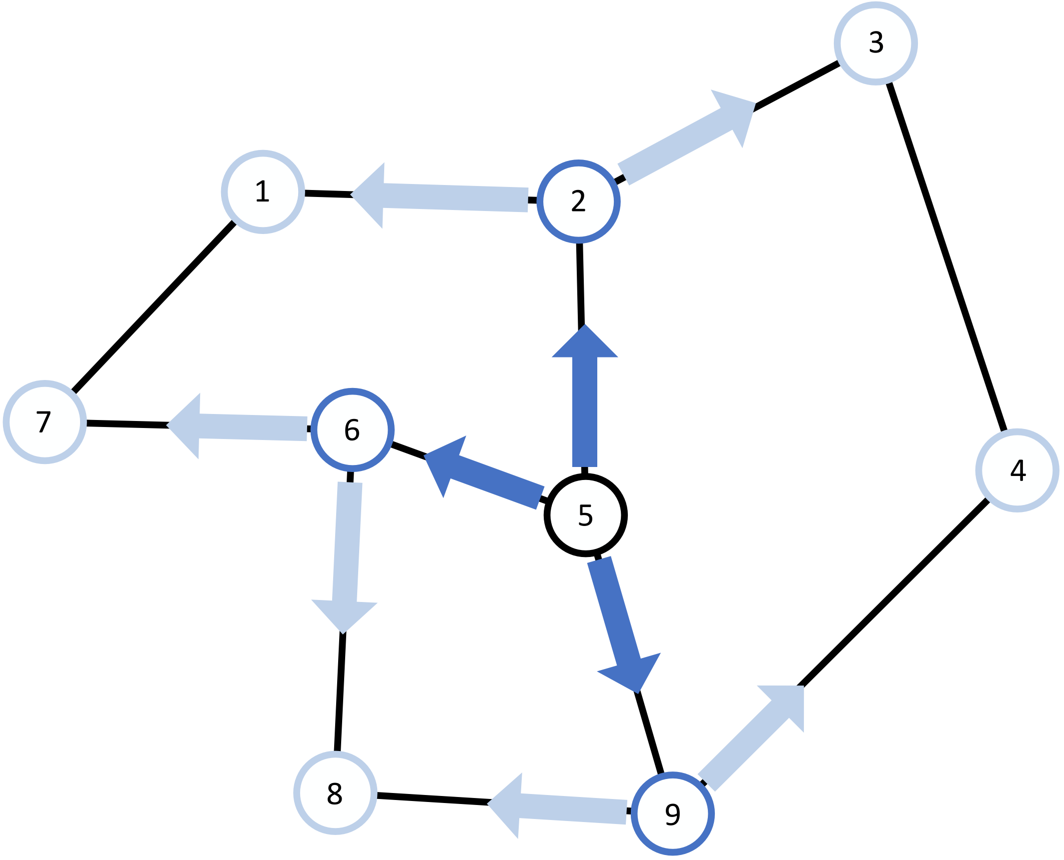
- Somehow each router knows the **whole** topology,
  plus the **destinations** behind others.
- They run Dijkstra's algorithm and set their routing table automatically.

IP resiliency

Link state routing protocol



One router's piece of the puzzle:
"Link State Advertisement"

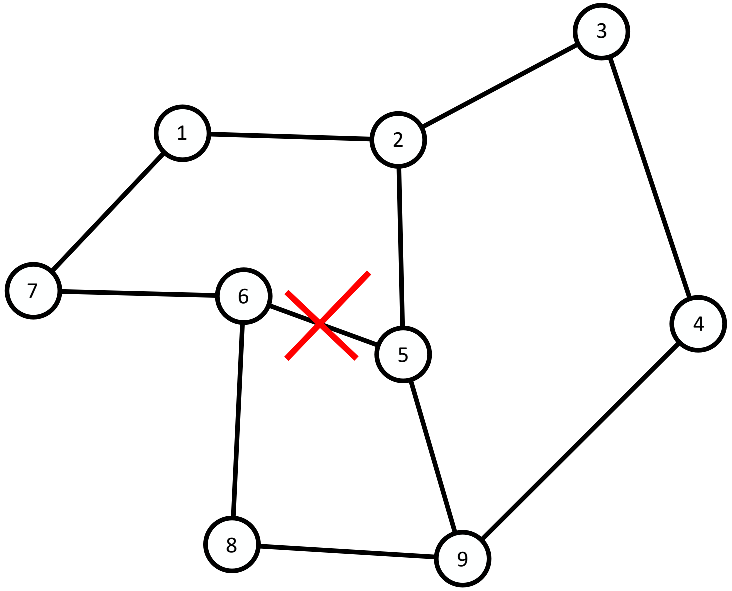


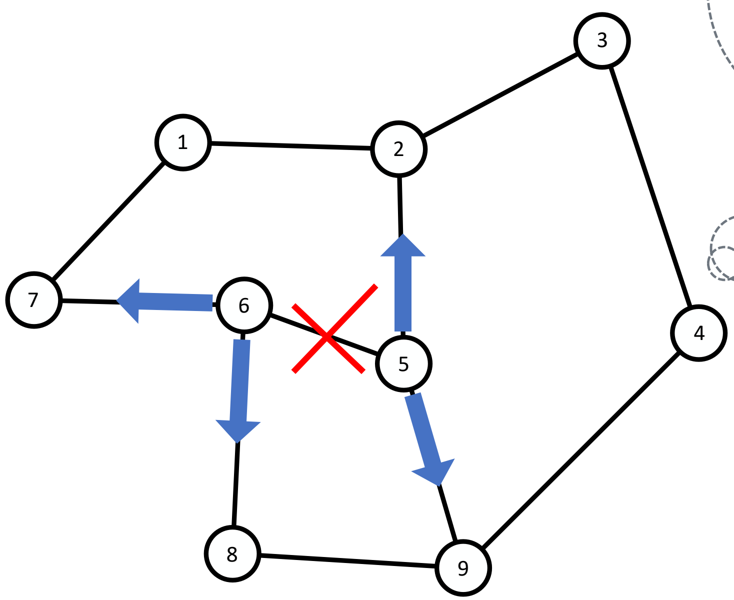
Flooding:
Everyone receives everyone's puzzle pieces

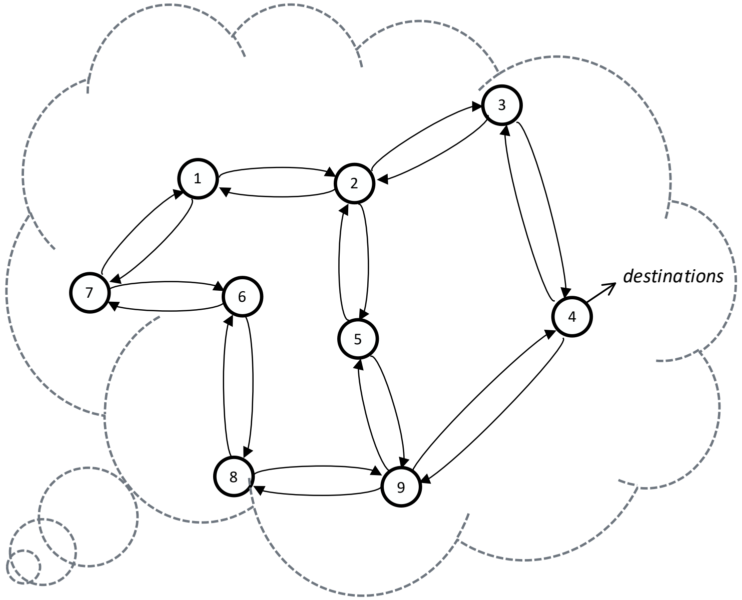
IP resiliency

Link state routing protocol

- The two routers at the ends of the failed link start flooding their updated LSA – without the link
- Soon each router know the updated topology and recalculate their routing table



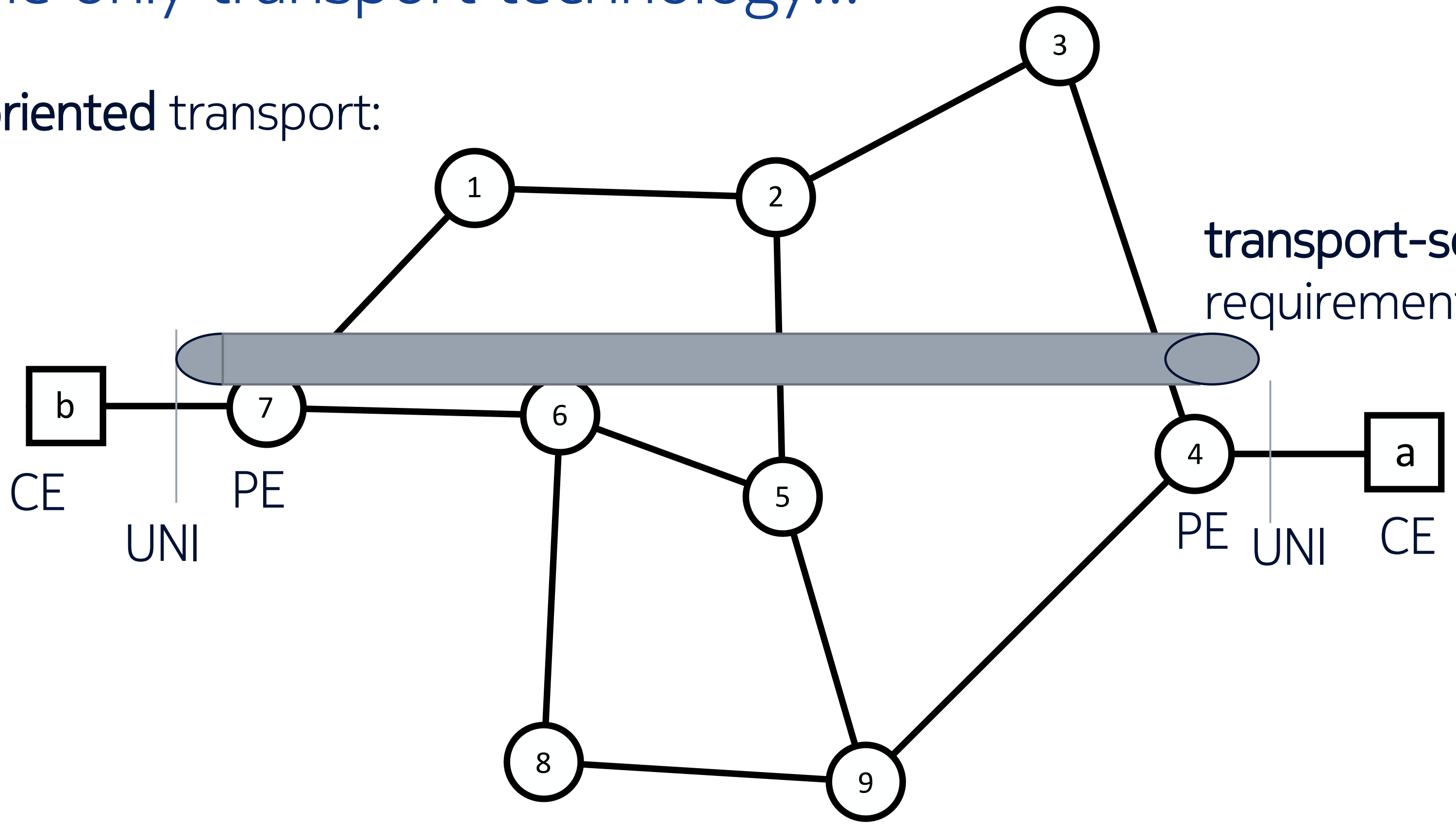




Eventually consistent state

IP is not the only transport technology…

Connection-oriented transport:



transport-service: SLA, incl. protection requirement…

Connection-oriented

Circuit-switched (CS)
TDM, **WDM**,…

Packet-switched (PS)
IP/**MPLS**, …

1+1 protection

1+1 protection

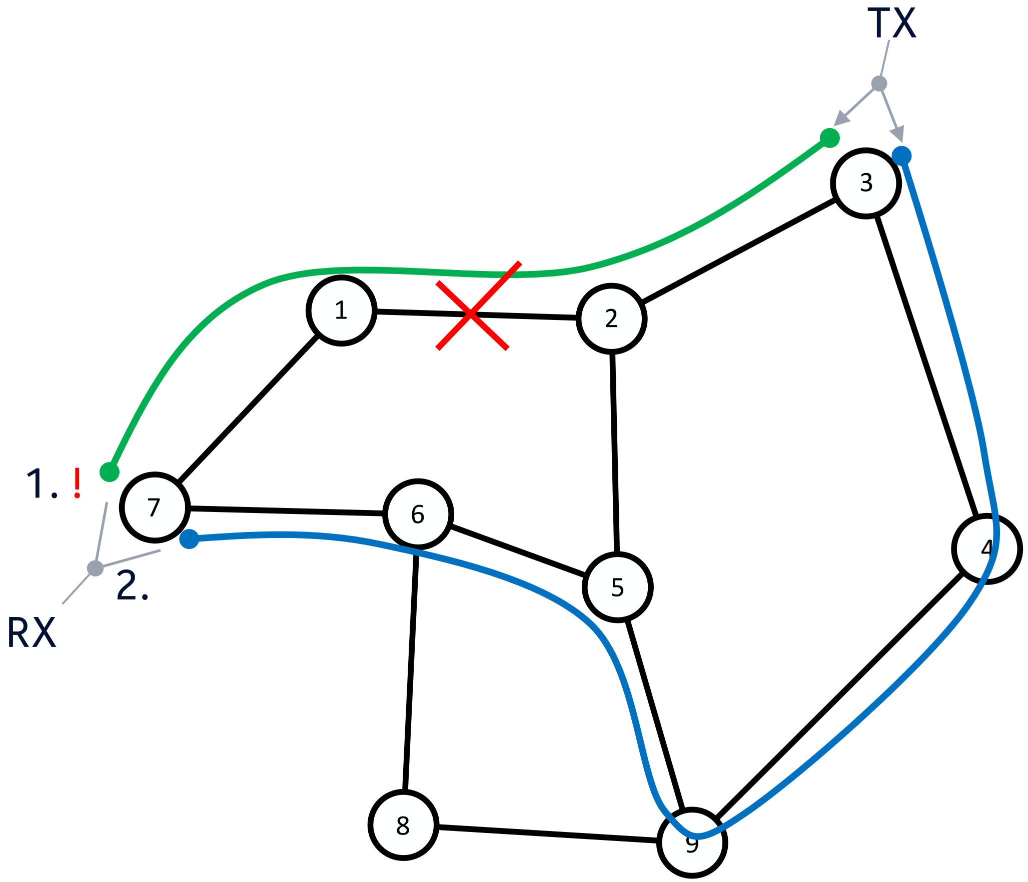
Operation

Failure-free operation:

- Information is transmitted on **both paths**
- Receiving end selects the data from the *working path*

In case of failure along the working path:

1. Receiving end gets notified – technology specific mechanism
2. Switches over to receiving from the *protection path*



Fastest service recovery time possible (for a global, path-level mechanism),on the expense of **dedicated protection resource** reservation

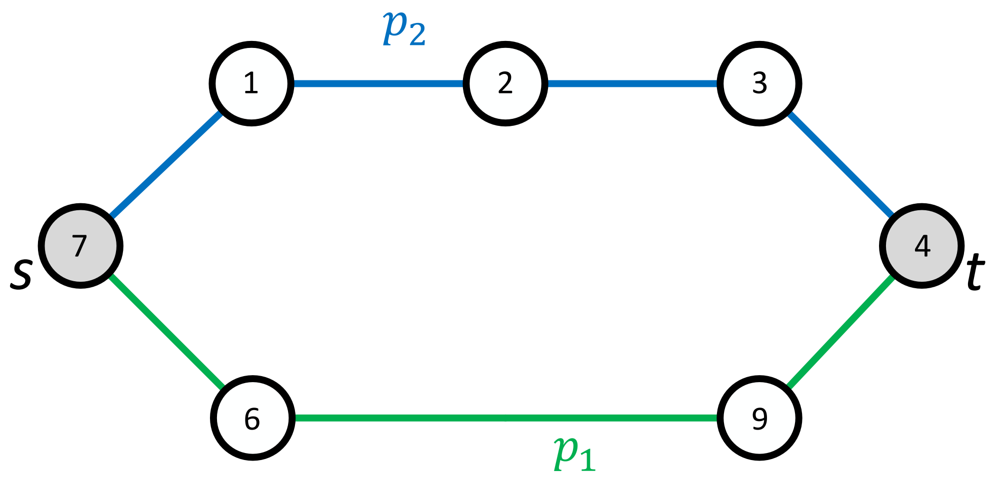
Least cost path pair
Link-disjoint problem

- Network model: directed weighted graph $G(V,E)$ with symmetrical, non-negative edge costs, $c(i,j) = c(j,i)$. A network link is represented by a pair of opposite directed edges.
- Find *working* path p_1 and *protection* path p_2 between source node s and destination node t, such that the two paths have no common edge and the total edge cost $C(p_1, p_2) = \sum_{(i,j)\in p_1} C(i,j) + \sum_{(i,j)\in p_2} C(i,j)$ is minimal.

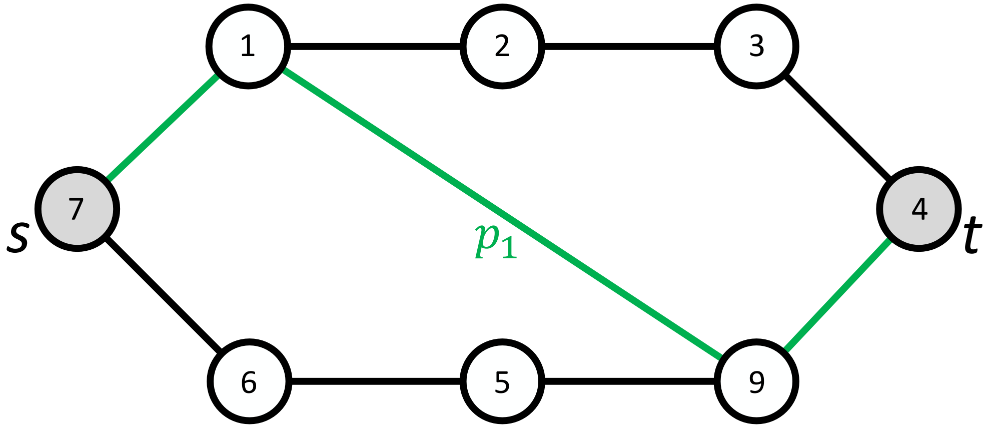
Least cost path pair

Link-disjoint: naive, two-step approach

Find p_1 as the shortest path in G, then exclude the edges of p_1 and find p_2 as the shortest path in this modified graph G'.



Works



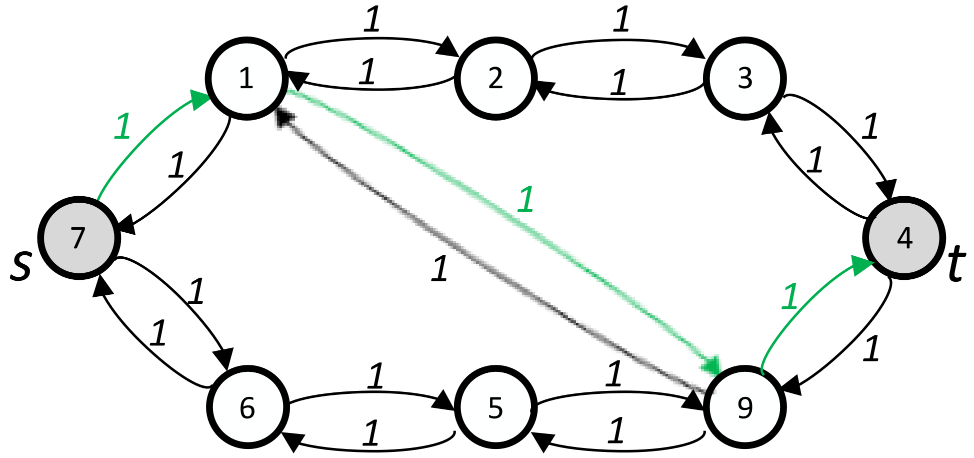
Doesn't work – *trap topology*

Even when it is not trapped, it doesn't guarantee optimal solution.

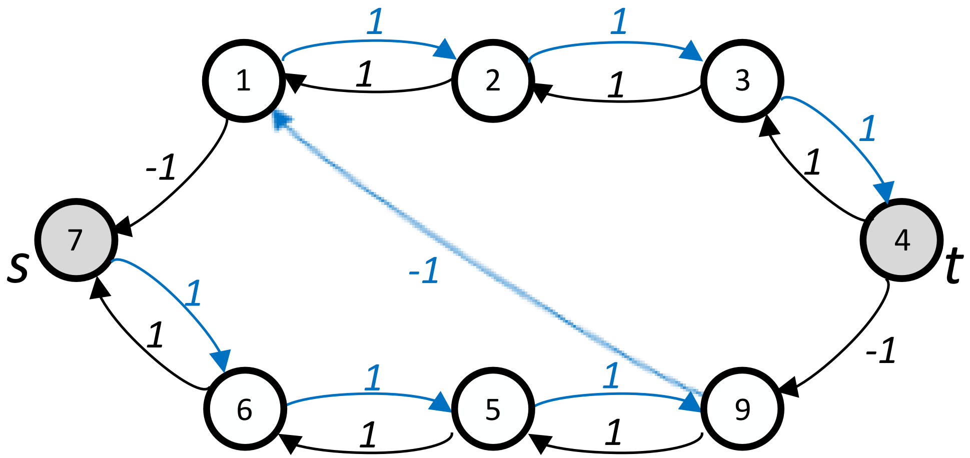
Least cost path pair

Link-disjoint: Suurballe's algorithm

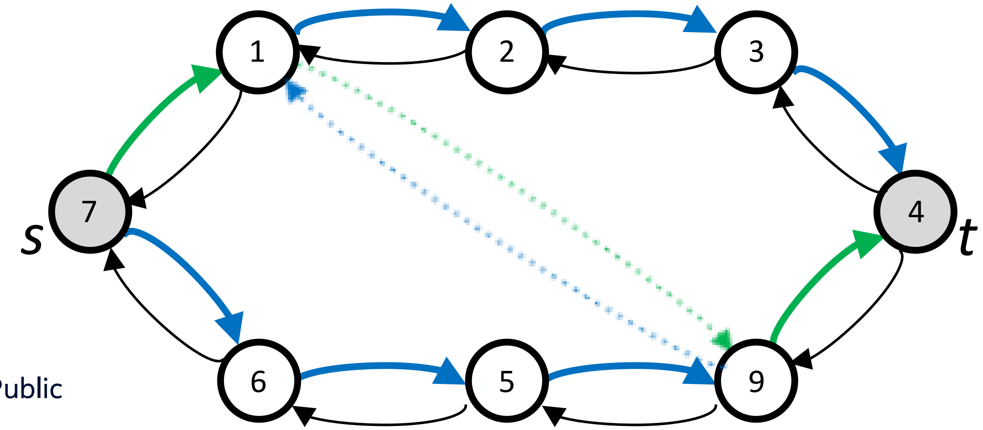
1) Find p_1' shortest path in G:



2) Create residual graph G', find p_2' shortest path in G':



3) Combine the non-interlacing edges of p_1' and p_2' into final the result (p_1, p_2):

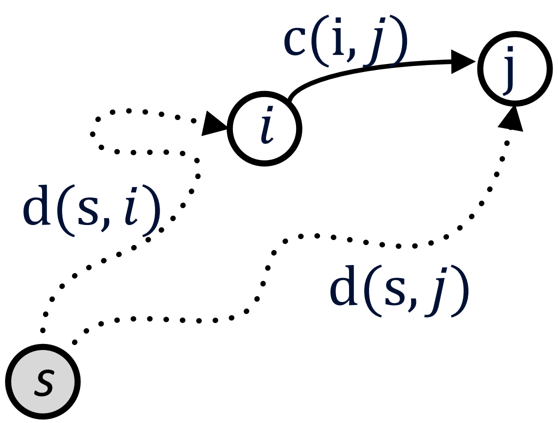


Least cost path pair

Link-disjoint: Suurballe's algorithm

The residual graph contains some negative edge weights – how to deal with it?

- Node potentials: a way to modify edge weights without changing the shortest path:
  - Assign some number $\mathrm{p(v)}$ to each node $\mathrm{v} \in V$
  - Modify the edge weights: $c(i,j) = \mathrm{c(i,j)} - (\mathrm{p(j)} - \mathrm{p(i)})$
  - The length of all paths from s to t change by the same amount, decreased by $\mathrm{p(t)} - \mathrm{p(s)}$
- If we use $\mathrm{p(v)} = \mathrm{d(s,v)}$, the length of the shortest path from s to v in the original graph, then after modification the weights of the residual graph will be non-negative!



$$\mathrm{d}(\mathrm{s},j) \leq d(s,i) + c(i,j)$$

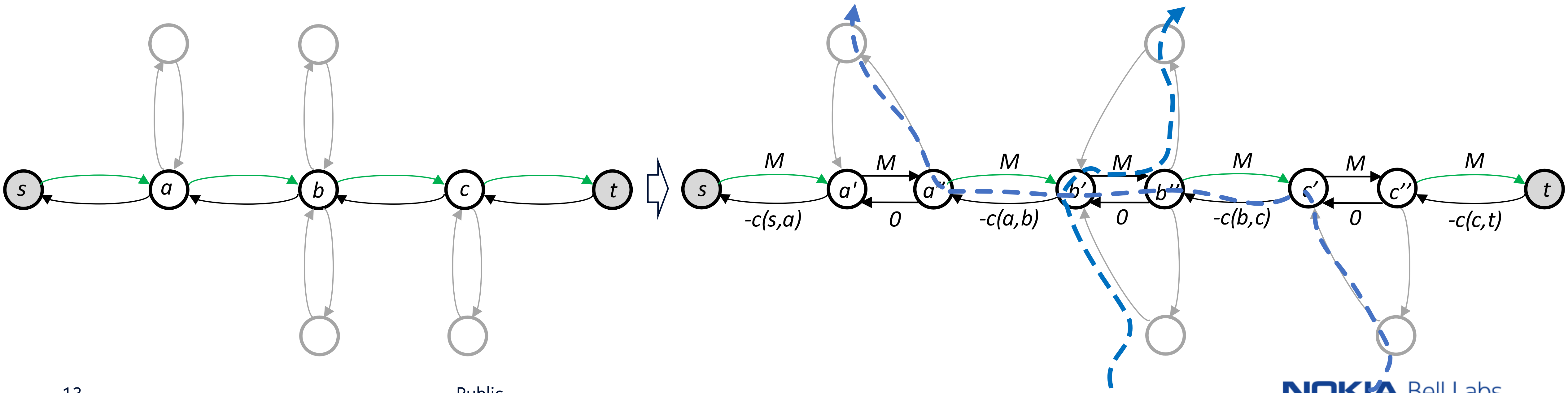
$$\mathrm{d}(\mathrm{s},j) - d(s,i) \leq c(i,j)$$

If we use **Dijskra's** algorithm for finding the $p_1{}'$ shortest path, then $\mathrm{d}(s,v)$ has been already calculated for a subset of nodes and for the rest we can use $\mathrm{d}(s,t)$ and it still results in non-negative modified weights.

Least cost path pair

Extending to other types of disjointedness

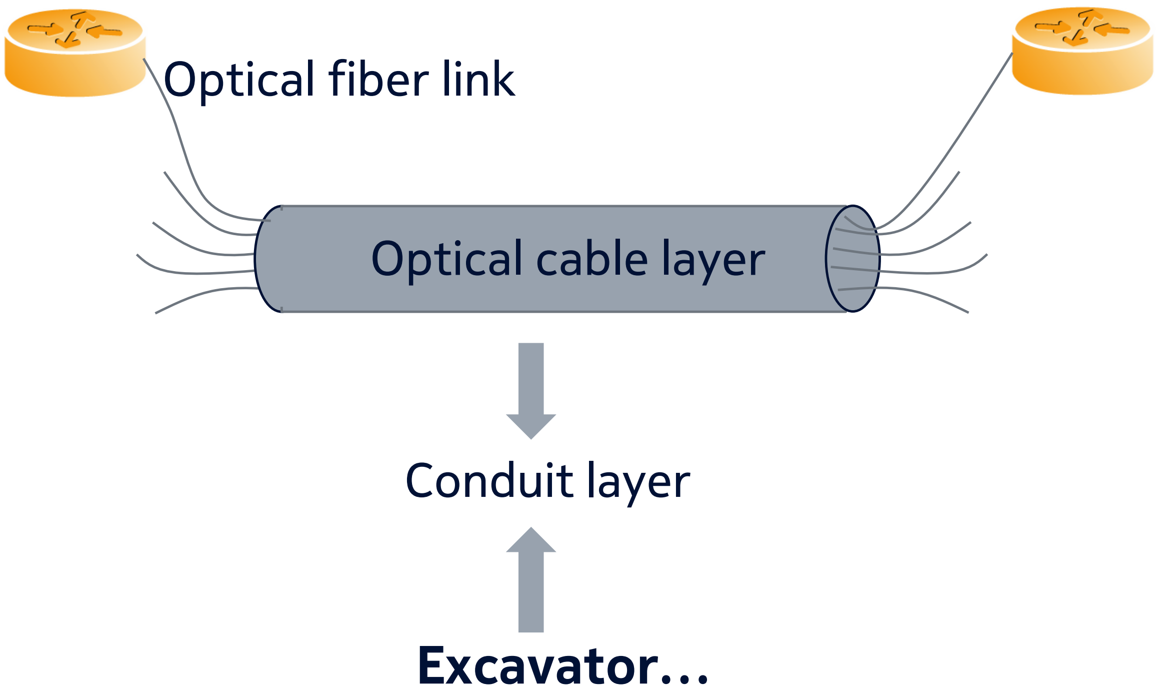
- Maximally link-disjoint:
  - the network topology may not allow totally link-disjoint path pair
  - instead of removing the directed edges in p_1', just set a sufficiently large weight M for them
- Node-disjoint:
  - with splitting the nodes on p_1', it can be reduced back to the edge-disjoint constraint

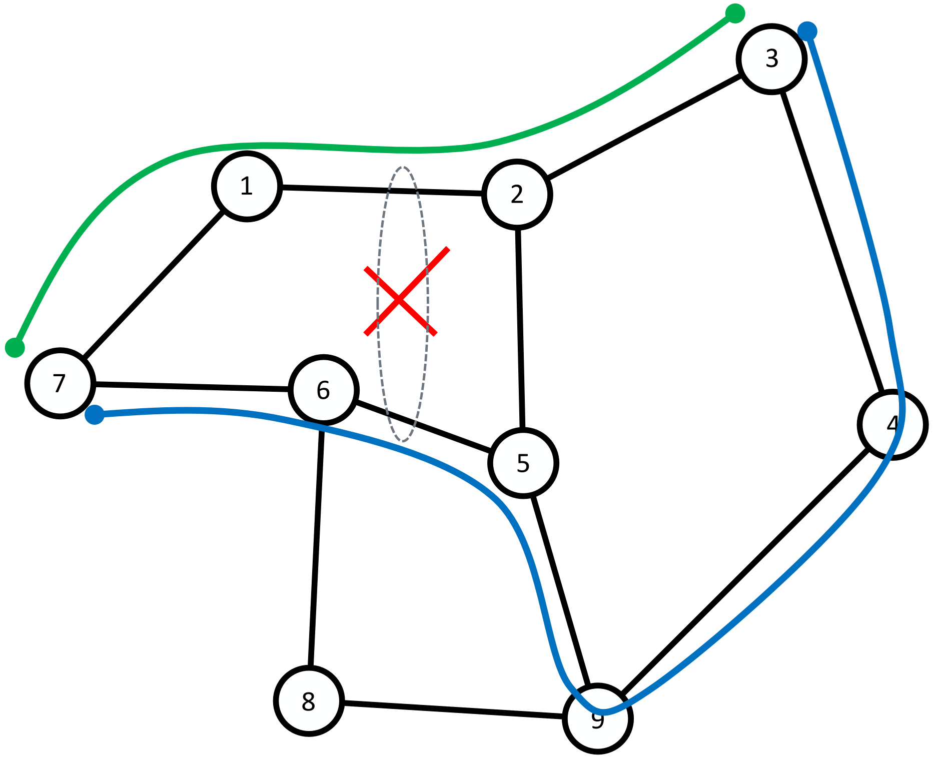


NOKIA Bell Labs

Least cost path pair

Shared Risk Link Group (SRLG)





Link disjoint is not enough

Least cost path pair

SRLG diverse routing

- Directed weighted graph $G(V,E)$ as before, plus the set of SRLG-s R, each SRLG is a set of two or more network links i.e. two or more pairs of opposite directed edges.
- Find *working* path p_1 and *protection* path p_2 between source node s and destination node t, such that the two paths have **no common edge**, there is **no SRLG in *R* which contains edge from both paths**, and the total edge cost $C(p_1, p_2) = \sum_{(i,j)\in p_1} C(i,j) + \sum_{(i,j)\in p_2} C(i,j)$ is minimal.
- The problem is NP-complete
- IMSH (Iterative Modified Suurballe's Heuristic)*

*A. Todimala and B. Ramamurthy, "IMSH: an iterative heuristic for SRLG diverse routing in WDM mesh networks," *Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969)*, 2004, pp. 199-204, doi: 10.1109/ICCCN.2004.1401627.

Least cost path pair
IMSH

MSH (Modified Suurballe's Heuristic)

For a seed path p from s to t in G:

- Create residual graph G':
  - Exclude the edges of p
  - Set the **0** weight for the opposite edges along p (negative cost can result in negative cycle when p is not the shortest path)
  - Set sufficiently large weight $\boldsymbol{M}$ for all edges in SRLG conflict with p
- Calculate shortest path p' in G'
- Get (p_1, p_2) from the non-interlaced edges of (p, p')
- Check whether (p_1, p_2) is SRLG disjoint

Least cost path pair
IMSH

Process:

- Generate seed paths of increasing length with Yen's algorithm
- Execute MSH for the next seed path p_i
- If the MSH result is SRLG disjoint, then compare it to the current best solution, update if better
- Terminate:
  - after max K number of seed paths checked, or
  - optimality verification criterion satisfied: $\text{Cost}(p_i) \geq C_{curr-opt}/2$

Yen's algorithm: the Swiss Army knife of network planning…

Shared mesh restoration

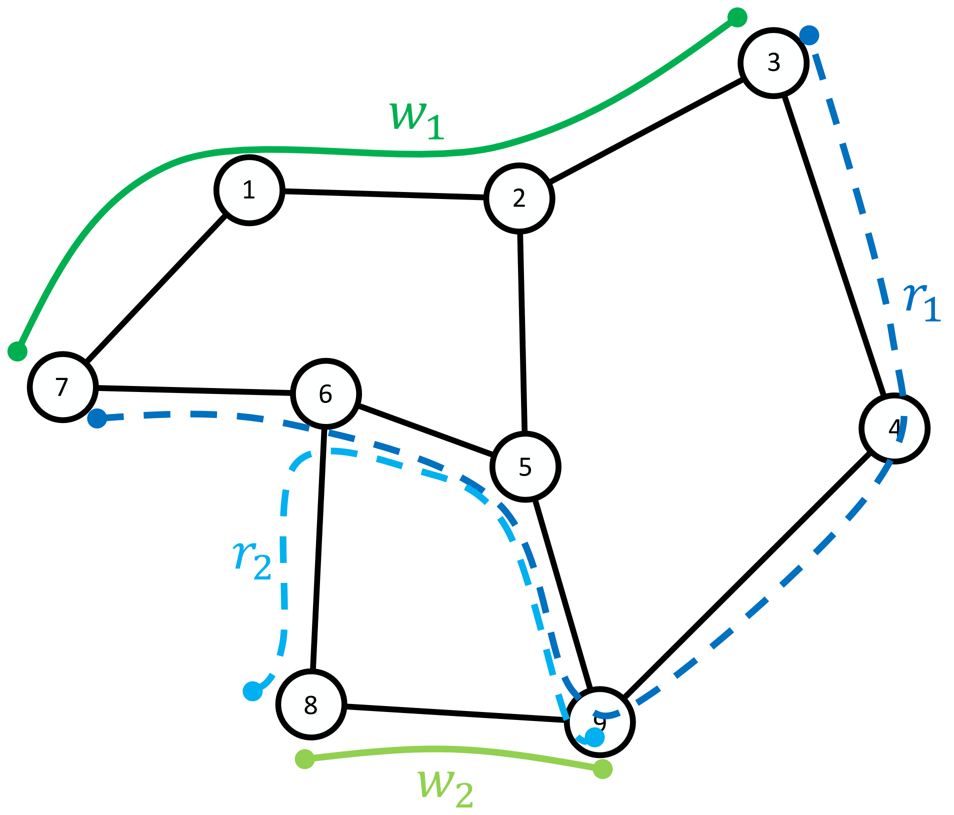
Shared mesh restoration

Basic idea

Two service demands of size $d_1$and d_2.

We want our services to be resilient to single link failures.

If traffic is sent on the alternative path only when the working path fails, then we need only $\max(d_1, d_2)$ bandwidth reservation on link *6-5* and *5-9*, not $d_2 + d_2$, because there is no link failure what effects both working paths.



Trade-off between resource requirement and recovery time: the restoration path need to be **activated** before traffic can be sent on it.

It is a distributed control procedure between the transport nodes along the restoration path what takes time.

Shared mesh restoration

Dimensioning problem

Given:

- undirected weighted link topology, linear link cost model:
  - link weight $c(l), l = 1 \dots L$: gives the cost of using the link for 1 unit of bandwidth
- set of demands: $\{(s(r), t(r), m(r))\},\ r = 1 \dots R$
- failure scenarios to protect against, indexed with $k = 1 \dots K$ ($\mathrm{K} = L$ in case of all single link failures)
- protect the demands with shared restoration

Find:

The p_r and q_r working and restoration paths for each demand r

Such that:

The total link cost, $\sum_{l=1}^{L}\left(\sum_{r|link(l)\in p_r} m_r + s_l\right) c(l)$ is minimal, where s_l denotes the ***minimum spare capacity*** required on the link.

Shared mesh restoration

Dimensioning problem

- If the demands were 1+1 protected, then the problem is easy – single-demand minimal cost path-pairs will be optimal
- In case of shared mesh restoration the problem is NP-complete (multi-commodity flow problem)
- Successive Survivable Routing * heuristic (SSR)
- SSR requires that the working paths already specified – use working path from the single-demand minimal cost disjoint path-pair

*Yu Liu, David Tipper, Peerapon Siripongwutikorn, *Approximating optimal spare capacity allocation by successive survivable routing*, IEEE/ACM Transactions on Networking, Vol. 13, No. 1, February 2005

Shared mesh restoration
SSR

- **Spare provision matrix** $G = \{g_{lk}\}, L \times K$, minimum spare capacity needed on link l in case of failure k
- $Q^T = \{q_{lr}^T\}, L \times R$ transpose of the restoration path link incidence matrix, column r describes the restoration path of demand r
- $U = \{u_{rk}\}, R \times K$, demand failure incidence matrix, 1 if failure k breaks the working path of demand r (or more generally: causes the activation of the restoration path – maximally disjoint cases…)
- $M = Diag(\{m_r\})$

$$G = Q^T M U \quad \Rightarrow \quad \max(G)$$

We are improving this sequentially demand by demand (Q^T)

Fixed (MU)

The column vector of row maximums, the min. spare capacity per link.

Shared mesh restoration

SSR: per-demand restoration path improvement step

- G^{-r}, G matrix as if demand r wouldn't have restoration path
- G^{r*}, the G matrix if we included all 'non-tabu' edges in q_r (it is not really a path). Tabu edges are in disjointedness conflict with w_r
- The **key step**: calculate link weight vector $v_r = \max(G^{r*}) - \max(G^{-r})$ and…

"What would be the minimum spare capacity increment on link l, if it would be part of q_r?"

… find q_r^{new} as shortest path with these weights.

- Accept new path if it is improving: $q_r = q_r^{new}, when\ v_r^T q_r > v_r^T q_r^{new}$

Shared mesh restoration

Using SSR per-demand restoration path improvement for dimensioning

- Start with empty Q
- **repeat** N times:
  1) Draw random order of the R demands
  2) Run restoration path improvement for each demand in this order
  3) **break** if there was no improvement for any of the demands

Repeat the whole process M times with different random seed

Haven’t talked about …

- ILP
- Bin packing: real link capacity is modular
  - Closely related: optimal equipment configuration
- Multi-layer
- Dual-homing, multi-domain
- Local protection
- Etc …

Summary

- Just a glimpse to some selected problems, hopefully interesting ones…
- Refer to Pióro's book*
- Real-life requirements → complex constrains → heuristics instead of exact methods
- Running time is also important due to human planner involvement – every algorithm is just a tool…

*Michal Pioro, Deepankar Medhi, "Routing, Flow, and Capacity Design in Communication and Computer Networks" Morgan Kaufmann Publishers